Csound for Android

Steven YI and **Victor LAZZARINI**
National University of Ireland, Maynooth
{steven.yi.2012, victor.lazzarini}@nuim.ie

Abstract

The Csound computer music synthesis system has grown from its roots in 1986 on desktop Unix systems to today's many different desktop and embedded operating systems. With the growing popularity of the Linux-based Android operating system, Csound has been ported to this vibrant mobile platform. This paper will discuss using the Csound for Android platform, use cases, and possible future explorations.

Keywords

Csound, Android, Cross-Platform, Linux

1 Introduction

Csound is a computer music language of the MUSIC N type, developed originally at MIT for UNIX-like operating systems[Boulanger, 2000]. It is Free Software, released under the LGPL. In 2006, a major new version, Csound 5, was released, offering a completely re-engineered software, which is now used as a programming library with its own application programming interface (API). It can now be embedded and integrated into several systems, and it can be used from a variety of programming languages and environments (C/C++, Objective-C, Python, Java, Lua, Pure Data, Lisp, etc.). The API provides full control of Csound compilation and performance, software bus access to its control and audio signals, as well as hooks into various aspects of its internal data representation. Several frontends and composition systems have been developed to take advantage of these features. The Csound API has been described in a number of articles [Lazzarini, 2006], [Lazzarini and Piche, 2006] [Lazzarini and Walsh, 2007].

The increasing popularity of mobile devices for computing (in the form of mobile phones, tablets and netbooks), has brought to the fore new platforms for Computer Music. Csound has already been featured as the sound engine for one of the pioneer systems, the XO-based computer used in the One Laptop per Child (OLPC) project [Lazzarini, 2008]. This system, based on a Linux kernel with the Sugar user interface, was an excellent example of the possibilities allowed by the re-engineered Csound. It sparked the ideas for a Ubiquitous Csound, which is steadily coming to fruition with a number of parallel projects, collectively called the *Mobile Csound Platform* (MCP). One such project is the development of a software development kit (SDK) for Android platforms, which is embodied by the CsoundObj API, an extension to the underlying Csound 5 API.

Android [1] is a Linux-kernel-based, open-source operating system, which has been deployed on a number of mobile devices (phones and tablets). Although not providing a full GNU/Linux environment, Android nevertheless allows the development of Free software for various uses, one of which is audio and music. It is a platform with some good potential for musical applications, although at the moment, it has a severe problem for realtime use that is brought by a lack of support for low-latency audio.

In this article we will discuss Csound usage on Android. We will explore the CsoundObj API that has been created to ease developing Android applications with Csound, as well as demonstrate some use cases. Finally, we will look at what Csound uniquely brings to Android, with a look at the global Csound ecosystem and how mobile apps can be integrated into it.

[1] `http://www.android.com`

2 Csound for Android

The Csound for Android platform is made up of a native shared library (libCsoundandroid.so) built using the Android Native Development Kit (NDK)[2], as well as Java classes that are compilable with the more commonly used Android Dalvik compiler. The native library is linked using the the object files that are normally used to make up the libcsound, libcsnd, and libsndfile[3] libraries that are found part of the desktop version of Csound. The Java classes include those commonly found in the csnd.jar library used for desktop Java-based Csound development, as well as unique classes created for easing Csound development on Android.

The SWIG[4] wrapping used for Android contains all of the same classes as those used in the Java wrapping that is used for desktop Java development with Csound. Consequently, those users who are familiar with Csound and Java can transfer their knowledge when working on Android, and users who learn Csound development on Android can take their experience and work on desktop Java applications. However, the two platforms do differ in some areas such as classes for accessing hardware and different user interface libraries. To help ease development, a CsoundObj class was developed to provide out-of-the-box solutions for common tasks (such as routing audio from Csound to hardware output). Also, applications using CsoundObj can be more easily ported to other platforms where CsoundObj is implemented (i.e. iOS).[5]

One of the first issues arising in the development of Csound for Android was the question of plugin modules. Since the first release of Csound 5, the bulk of its unit generators (opcodes) were provided as dynamically-loaded libraries, which resided in a special location (the OPCODEDIR or OPCODEDIR64 directories) and were loaded by Csound at the orchestra compilation stage. However, due to the uncertain situation regarding dynamic libraries (not only in Android but also in other mobile platforms), it was decided that all modules without any dependencies or licensing issues could be moved to the main Csound library code. This was a major change (in Csound 5.15), which made the majority of opcodes part of the base system, about 1,500 of them, with the remaining 400 or so being left in plugin modules. The present release of Csound for Android includes only the internal unit generators. Another major internal change to Csound, which was needed to facilitate development for Android, was the move to use core (memory) files instead of temporary disk files in orchestra and score parsing.

Audio IO has been developed in two fronts: using pure Java code through the AudioTrack API provided by the Android SDK and, using C code, as a Csound IO module that uses the OpenSL API that is offered by the Android NDK. The latter was developed as a possible window into a future lower-latency mode, which is not available at the moment. It is built as a replacement for the usual Csound IO modules (PortAudio, ALSA, JACK, etc.), using the provided API hooks. The Csound input and output functions, called synchronously in its performance loop, pass a buffer of audio samples to the DAC/ADC using the OpenSL enqueue mechanism. This includes a callback that is used to notify when a new buffer needs to be enqueued. A double buffer is used, so that while one half is being written or read by Csound, the other is enqueued to be consumed or filled by the device. The code fragment below in listing 1 shows the output function and its associated callback. The OpenSL module is the default mode of IO in Csound for Android. Although it does not currently offer low-latency, it is a more efficient means of passing data to the audio device and it operates outside the influence of the Dalvik virtual machine garbage collector (which executes the Java application code).

The AudioTrack code offers an alternative means accessing the device. It pushes/retrieves input/output frames into/from the main processing buffers (spin/spout) of Csound synchronously at control cycle intervals. It is offered as an option to developers, which can be used for instance, in older versions of Android without OpenSL support.

[2] http://developer.android.com/sdk/ndk/index.html

[3] http://www.mega-nerd.com/libsndfile/

[4] http://www.swig.org

[5] There are plans to create CsoundObj implementations for other object-oriented desktop development languages/-platforms such as C++, Objective-C, Java, and Python, but at the time of this writing, CsoundObj is only available in Objective-C for iOS.

3 Application Development using CsoundObj

Developers using the CsoundObj API will essentially partition their codebase into three parts: application code, audio code, and glue code. The application code contains the standard Android code for creating applications, including such things as view controllers, views, database handling, and application logic. The audio code is a standard Csound CSD project that contains code written in Csound and will be run using a CsoundObj object. Finally, the glue code is what will bridge the user interface with Csound.

```
/* this callback handler is called every time a buffer finishes playing */
void bqPlayerCallback(SLAndroidSimpleBufferQueueItf bq, void *context)
{
   open_sl_params *params = (open_sl_params *) context;
   params->csound->NotifyThreadLock(params->clientLockOut);
}

/* put samples to DAC */
void androidrtplay_(CSOUND *csound, const MYFLT *buffer, int nbytes)
{
    open_sl_params *params;
    int     i = 0, samples = nbytes / (int) sizeof(MYFLT);
    short* openslBuffer;

    params = (open_sl_params *) *(csound->GetRtPlayUserData(csound));
    openslBuffer = params->outputBuffer[params->currentOutputBuffer];
    if (params == NULL)
      return;
    do {
      /* fill one of the double buffer halves */
      openslBuffer[params->currentOutputIndex++] = (short) (buffer[i]*CONV16BIT);
      if (params->currentOutputIndex >= params->outBufSamples) {
         /* wait for notification */
          csound->WaitThreadLock(params->clientLockOut, (size_t) 1000);
         /* enqueue audio data */
         (*params->bqPlayerBufferQueue)->Enqueue(params->bqPlayerBufferQueue,
                           openslBuffer,params->outBufSamples*sizeof(short));
         /* switch double buffer half */
         params->currentOutputBuffer = (params->currentOutputBuffer ?  0 : 1);
         params->currentOutputIndex = 0;
         openslBuffer = params->outputBuffer[params->currentOutputBuffer];
      }
    } while (++i < samples);
}
```

Listing 1: OpenSL module output C function and associated callback

```
public interface CsoundValueCacheable {
  public void setup(CsoundObj csoundObj);
  public void updateValuesToCsound();
  public void updateValuesFromCsound();
  public void cleanup();
}
```

Listing 2: CsoundValueCacheable Interface

```
String csd = getResourceFileAsString(R.raw.test);
File f = createTempFile(csd);
csoundObj.addSlider(fSlider, "slider", 0.0, 1.0);
csoundObj.startCsound(f);
```

Listing 3: Example CsoundObj usage

CsoundObj uses objects that implement the CsoundValueCacheable interface for reading value from and writing values to Csound (listing 2). Any number of cacheables can be used with CsoundObj. The design is flexible enough such that you can design your application to use one cacheable per user interface or hardware sensor element, or one can make a cacheable that reads and writes along many channels.

CsoundObj contains utility methods for binding Android Buttons and SeekBars to a Csound channel, as well as for a method for binding the hardware Accelerometer to preset Csound channels. These methods wrap the View or sensor objects with pre-made CsoundValueCacheables that come with the CsoundObj API. Since these are commonly used items that would be bound, the utility methods were added to CsoundObj as a built-in convenience to those using the API. Note that CsoundValueCacheables are run within the context of the audio processing thread; this was done intentionally so that the cacheable could copy any values it needed to from Csound, then continue to do processing in another thread and eventually post back to the main UI thread via a Handler.

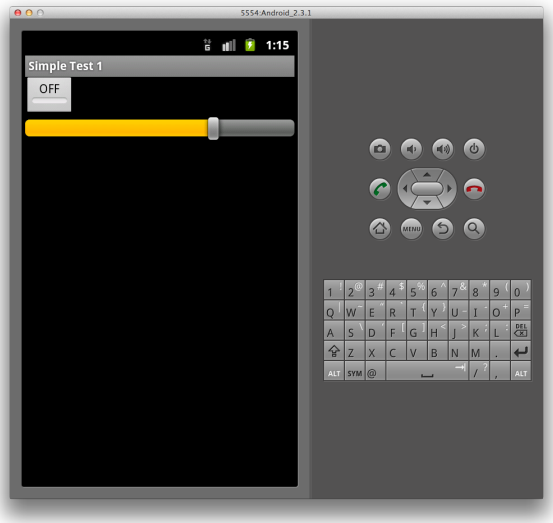

Figure 1: Android Emulator showing Simple Test 1 Activity

Listing 3 shows example code of using CsoundObj with a single slider, from the Simple Test 1 Activity, shown in Figure 1.The code above shows how a CSD file is read from the projects resources using the getResourceFileAsString utility method, saved as a temporary file, then used as an argument to CsoundObj's startCsound method. The 2nd to last line shows the addSlider method being used to bind fslider, an instance of a SeekBar, to Csound with a channel name of "slider" and a range from 0.0 to 1.0. When Csound is started, the values from that SeekBar will be read by the Csound project using the chnget opcode, which will be reading from the "slider" channel.

Figure 2 shows the relationships between different parts of the platform and different usage scenarios. An application may work with CsoundObj alone if they are only going to be starting and stopping a CSD. The application may also use CsoundValueCacheables for reading and writing values from either CsoundObj or the CsoundObject. Finally, an application may do additional interaction with the Csound object that the CsoundObj has as its member, taking advantage of the standard Csound API.

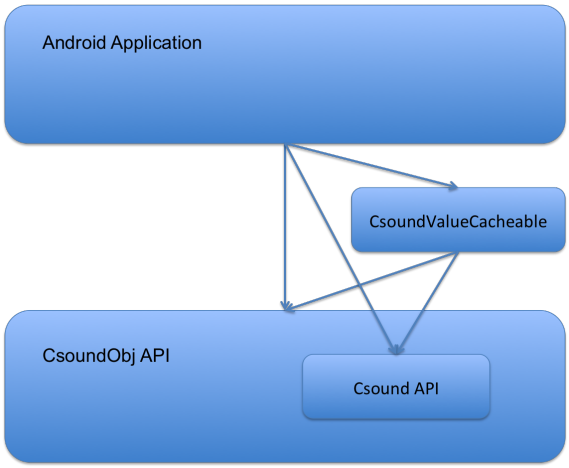

Figure 2: CsoundObj Usage Diagram

A Csound for Android examples project has been created the contains a number of different Csound example applications. These examples demonstrate different ways of using the CsoundObj API as well as different approaches to applications, such as realtime synthesis instruments and generative music. The examples were ported over from the Csound for iOS examples project and users can study the code to better understand both the CsoundObj API on Android as well as what is required to do cross-platform

development with Csound as an audio platform.

4 Benefits of using Csound on Android

Using Csound on Android provides many benefits. First, Csound contains one of the largest libraries of synthesis and signal processing routines. By leveraging what is available in Csound, the developer can spend more time working on the user interface and application code and rely on the Csound library for audio-related programming. The Csound code library is also tested and supported by a open-source community, meaning less testing work required for your project.

In addition to the productivity gain of using a library for audio, Csound projects–developed in text files with .csd extensions–can be developed on the desktop, and later moved to the Android application. Developing and testing on the desktop allows for a faster development process than testing in the Android emulator or on a device, as it removes the application compilation and deployment stage, which can be slow at times.

Having the audio-related code in a CSD file for a project also brings with it two benefits. First, development of an application can be split amongst multiple people; one can work on the audio code while the other focuses on developing other areas of the application. Second, developing an application based around Csound allows for moving that CSD to other platforms, such as iOS or desktop operating systems. The developer would then only have to develop the user-interface and glue code to work with that CSD on each platform.

Additionally, cleanly separating out the audio system of an application and enforcing a strict API (Application Programmer Interface) to that system is a good practice for application development. This helps to prevent tangled, hard to maintain code. This is of benefit to the beginning and advanced programmer alike.

5 Conclusions

From its roots in the Music N family of programs, Csound has grown over the years, continually expanding it features as a synthesis library as well as its usefulness as a music platform. With its availability on multiple operating systems, Csound offers a multi-platform option for developing musical applications. Current Csound 6 developments to enable realtime modification of the processing graph as well as other features will expand the types of applications that can be built with Csound. As Android is now supported within the core Csound repository, it will continue to be developed as a primary platform for deployment as part of the MCP distribution.

6 Availability

The Csound for Android platform and examples project are included in the main Csound GIT repository. Build files are included for those interested in building Csound with the Android Native Development Kit. Archives including a pre-compiled Csound as well as examples are available at `http://sourceforge.net/projects/csound/files/csound5/Android/`.

7 Acknowledgements

This research was partly funded by the Program of Research in Third Level Institutions (PRTLI 5) of the Higher Education Authority (HEA) of Ireland, through the Digital Arts and Humanities programme.

References

R. Boulanger, editor. 2000. *The Csound Book*. MIT Press, Cambridge, Mass.

V Lazzarini and J. Piche. 2006. Cecilia and tclcsound. In *Proc. of the 9th Int. Conf. on Digital Audio Effects (DAFX)*, pages 315–318, Montreal, Canada.

V Lazzarini and R. Walsh. 2007. Developing ladspa plugins with csound. In *Proceedings of 5th Linux Audio Developers Conference*, pages 30–36, Berlin, Germany.

V Lazzarini. 2006. Scripting csound 5. In *Proceedings of 4th Linux Audio Developers Conference*, pages 73–78, Karlsruhe, Germany.

V Lazzarini. 2008. A toolkit for audio and music applications in the xo computer. In *Proc. of the International Computer Music Conference 2008*, pages 62–65, Belfast, Northern Ireland.