Web-enabled Audio-synthesis Development Environment

Rory Walsh
Dundalk Institute of Technology,
Dundalk, Co. Louth
Ireland
rory.walsh@dkit.ie

Conor Dempsey
Dundalk Institute of Technology,
Dundalk, Co. Louth
Ireland
conor.dempsey@gmail.com

Abstract

Collaboration and education in the world of digital audio synthesis has been facilitated in many ways by the internet and its World Wide Web. Numerous projects have endeavoured to progress beyond text-based applications to facilitate wider modes of collaborative activity such as, network musical performance and composition. When developing a software application one is faced with technology decisions which limit the scope for future development. Similarly, the choice of one audio synthesis language or engine over another can limit the potential user base of the application. This paper will describe how, through the WADE system architecture it is possible to integrate existing software in a highly scalable and deployable way. The current incarnation of the WADE system, as depicted in the WADE Examples section below, uses the Csound synthesis engine and audio synthesis language as its audio server.

Keywords

Audio-synthesis, collaboration, education, web-based, OSGi

1 Introduction

The Web-enabled Audio-synthesis Development Environment, WADE, is a proof of concept application designed to facilitate collaboration and education in audio synthesis development. While the initial goal of this project was to investigate the creation of a browser based user interface for the Csound[1] synthesis engine, research and technology decisions lead to the outlining of a possible software architecture which was not tied to any specific synthesis engine, or even to audio synthesis development itself. The application was developed using the Eclipse Rich Client Platform[2], Equinox OSGi[3], Java Servlets, HTML[4] and javaScript[5].

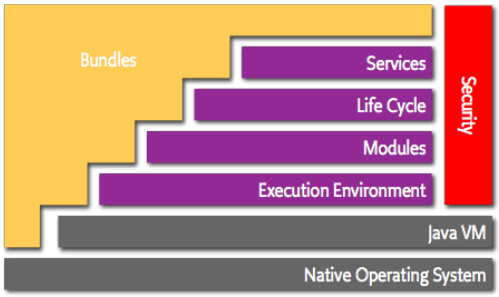
1.1 Web-enabled Audio Synthesis

In 1995 Ossenbruggen and Elıens proposed the use of client-side sound synthesis techniques in order to reduce the amount of resources needed for high quality music on the web[6]. Their project wrapped Csound for use with a Tcl/TK (scripting language and its graphical widget library) based browser or Tcl browser plugin. This allowed Csound to be used as a synthesis engine on the client side, thus removing the amount of data being transferred from client to server and moving the more processor intensive task, of synthesising a Csound Score file, to the client.

This work was closely followed by a network enabled sound synthesis system created by James McCartney, (1996) in the guise of his Supercollider synthesis project. More recent work done on web enabling existing sound synthesis engines has been carried out by Jim Hearon[7] and Victor Lazzarini[8] (independently), using Csound and Alonso, et al. in their creation of a Pure Data browser plug-in [9]. The need for web-enabled audio synthesis as a pedagogical tool or as a means of offering high quality audio across low bandwidth networks can now be answered in new and interesting ways which can lead to unforeseen future development projects.

1.2 Open Service Gateway Initiative (OSGi)

The OSGi component architecture is a framework which sits on top of the Java virtual machine. It is specifically designed to deal with many of the pitfalls of OOP and the way in which software applications have historically been constructed.



(fig.1 OSGi Model)

At the heart of the OSGi *(fig.1)* ideology is the bundle; a bundle provides a specific piece of functionality and advertises this functionality as a service to other bundles through a well defined interface. As well as specifying the services they provide, bundles must specify their dependencies on other services/ bundles. This allows the OSGi service registry to determine whether a particular bundle can be started. In the same vein, a bundle can specify what functionality it can provide in circumstances when only some of its dependencies are satisfied. It is also possible for bundles to provide extension points, points at which one bundle can extend or override the functionality of another bundle.

This tolerance for dynamic service availability makes OSGi well suited for developing applications seeking to utilise web services and diverse hardware. In this specific project one of the underlying requirements is the reuse of existing software applications i.e. Csound and the Jetty http server; the ability to register these as separate services within the OSGi framework means that they can be implemented independently of the overall system, e.g. if future implementations wish to use a different audio synthesis engine they would simply have to provide an OSGi bundle for that synthesis engine, its complementary web-interface bundle, and choose to use that engine over any pre-existing one.

1.3 Eclipse RCP

The Eclipse Rich Client Platform is a collection of OSGi bundles called a target platform. The RCP specific bundles allow developers to make contributions to an Eclipse GUI through extension points, advertised by the Eclipse runtime's extension registry.

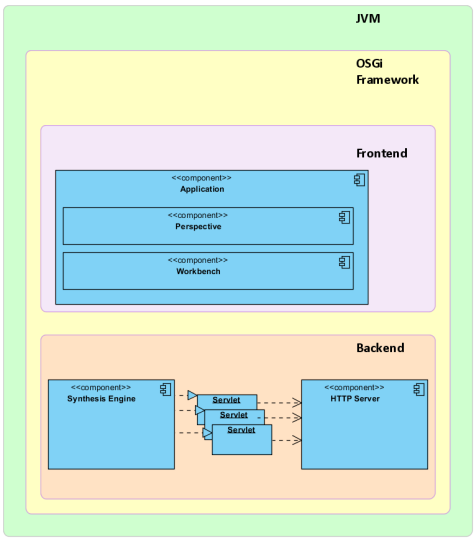
1.4 HTML + JavaScript

Hyper Text Mark-up Language is the language used by web developers and web browsers to layout web-page content. While book publishing houses have been using mark-up annotations for many years, they do not have to contend with dynamic content changes such as those seen in web pages. JavaScript or ECMAScript is a scripting language supported by most web browsers that allows web developers to create more dynamic and interactive web pages. This project extends the CodeMirror JavaScript code editor[10] to enable parsing and syntax highlighting of Csound CSD files within the web browser. The JQuery [11] library and JQueryUI JavaScript libraries were used to create the web page user interface.

1.5 Java Servlets

Java Servlets are Java classes which can be used to create and extend web based applications. They have access to a range of http calls and also the Java APIs. In this particular application they are used to provide the web facing interface for the Csound synthesis engine.

2 WADE Architecture



(fig.2 WADE Architecture)

In explanation of the fig. 2 above; the program runs within the JVM, on top of this runs the OSGi framework which manages the bundle configuration that provides functionality for both the frontend and the backend of the system. As all bundles are registered with the OSGi service registry it is possible for any bundle to request the

functionality of another bundle. As such, the frontend and backend sections of the diagram are simply logical compartmentalisations for the purposes of design thinking.

The frontend contains the specific bundles required to manage the GUI features of the application and the backend contains the functionality desired by the project, e.g. the synthesis engine, its servlet interface and the servlet container which will serve the servlets when requested by the web browser. A benefit of this configuration is that the applications functionality can be extended independently of the user interface.

OSGi enables dynamic service composition through a number of mechanisms. One of these is Declarative Services, which can be seen in the PlayServlet class, where there is no Equinox specific code required. The component.xml file and the ServletComponent class are used by the Equinox DS bundle to weave the services together at runtime.

```
public class PlayServlet extends HttpServlet{...
```

The HTTP POST method needs to be implemented in order to accept the incoming Csound file from the web editor.

```
public void doPost(
        HttpServletRequest req,
        HttpServletResponse resp){
String csdString = (String)
req.getParameter("csdElement");
try {
  if((csnd!=null)&&(!csdString.equalsIgnoreCase(""))){
        csnd.setCSDString(csdString);
        }
  else{
   resp.getWriter().write(csnd.getPlaying()?"true":"false");
      }
   }
catch(IOException e){
    e.printStackTrace();
   }
}
```

2.1 Csound API

As can be seen in the code excerpt above, the csnd object is used by the servlet. This object is created using the CppCsound interface and uses the CsoundPerformanceThread and CsoundCallbackWrapper classes to control real-time operation of Csound.

The code shown below is from the Csound API service bundle; it creates the aforementioned objects, passing the CppSound object to the CsoundPerformanceThread object and setting up the callback object for channel input and retrieving console output messages.

```
csoundObj= new CppSound();
chanCaller = new CsChansetCB(csoundObj);
chanCaller.SetOutputValueCallback();
chanCaller.SetMessageCallback();
csndSingleThread = new Thread(this);
csndSingleThread.start();
```

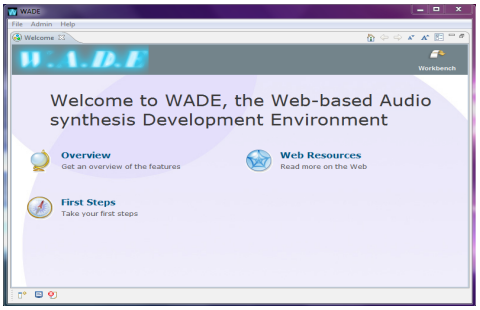
When the servlet retrieves the csdElement parameter from the HttpServletRequest object it passes this string value to the csnd object via the setCSDString function. Ultimately the createCSDFile function is called to create the temporary .orc and .sco files from the csd string and prepares the Csound object to run these.

```
private void createCSDFile(String csdString){
if(csoundObj!=null && (!csdString.equalsIgnoreCase("")))
        {
        CsoundFile csFile = csoundObj.getCsoundFile();
        csFile.setCSD(csdString);
        csFile.setCommand(csOptionsString);
        csoundObj.PreCompile();
        csFile.exportForPerformance();
        csoundObj.compile();
        csdFileString="";
        csdFileCreated = true;
        }
        else{
            csdFileCreated = false;
            }
}
```

While this prototype uses Csound as it's synthesis engine, it is entirely possible to add OSC to allow control of any OSC aware synthesis engine such as, Pure Data or SuperCollider.

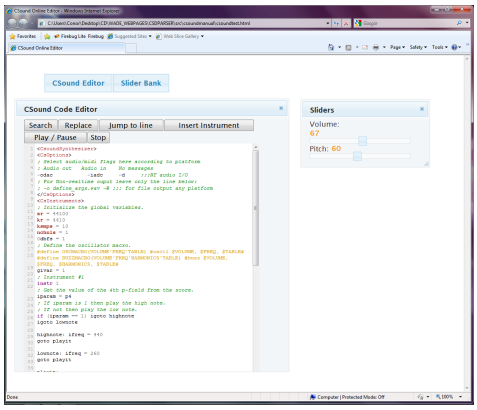
3 WADE Examples

The current version of this application is being tested with the Ubuntu 10.10 Linux distribution and the Google Chrome web browser.



(fig.3 WADE Application window)

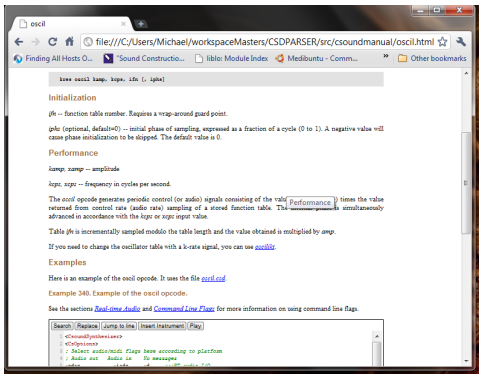
Once the desktop application has started, the "Welcome" view will be displayed, providing links to a number of pages informing you about the WADE application. Along the top of the application window you will see the obligatory main toolbar, from which you can access the preferences and console view.



(fig. 4 WADE Browser-based Interface)

Next, access the web-based code editor with slider bank in your web browser (as you would any web page). Once the page has loaded, click the "Csound Editor" and "Slider Bank" buttons to show the editor and associated faders. You will note that Instrument 2 in the CSD file has two channels "volChan" and "pitchChan"; these are controlled by the faders in the slider bank window. Press the "Play / Pause" button to send the CSD file to the WADE desktop application for rendering. It is possible to send live control information to the WADE desktop application via the faders in the slider bank.

A pedagogical application of this system, could see an interactive Csound manual created, or a large database of interactive Csound instruments made available to the sound synthesising community.



(fig.5 WADE browser-based editor in Csound manual)

Above is an example of the oscil opcode reference page from the Csound manual in HTML format.

4 Future Developments

Current refactoring efforts are underway to resolve issues in line with a first public release of the system. Due to the integration of different open source technologies, the completed system and source code will likely be made available under the LGP licence, with the obvious caveat when integrating other technologies; that their respective licences are adhered to and that the use of these projects is acknowledged. The project releases and source code will be available from the WADE project page on Sourceforge: http://wadesys.sourceforge.net/. The features being assessed are as follows, the dynamic generation of a RESTful OSC [12] API on a per instrument basis, using Apache CXF [13]; dynamic GUI slider bank generation; the inclusion of a HyperSQL database which could be used to store OSC packets for replaying a live performance; XMPP [14] chat client for real-time communication with other developers; XML specification for instruments, including what graphical widgets should be used to display the instrument.

The provision for extensibility and deployment options afforded by OSGi and the Eclipse Rich Client Platform could lead to the incorporation of features in the areas of: networked musical performance, sound installation frameworks, visual art development, cloud based audio synthesis and even pseudo-embedded synthesis systems. In short, it is possible that future iterations of this project will be deployed on small

form factor devices such as the BeagleBoard[15] or PandaBoard[16], to create a Csound based effects pedal, or across a number of large servers to provide a cloud synthesis solution.

5 Conclusion

The question of how to integrate these diverse technologies lead to the identification of the OSGi framework, which in turn lead to a much greater consideration of the software architecture of the project. While it can be shown that systems designed for a specific task are more likely to be less bloated and in many cases better suited to that task than larger programs designed to address numerous concerns[17] it was concluded that by designing a system which facilitated future expansion and development, the long term goal of creating a system capable of delivering a complete collaborative environment for audio synthesis development, learning and performance would be best satisfied.

6 Acknowledgements

I would like to acknowledge the help I received from the Csound mailing list, in particular I would like to thank Jacob Joaquin, Michael Goggins, Steven Yi and Victor Lazarini(Csound Mailing List); Dundalk Institute of Technology for their support of my research and finally, I wish to thank the contributors of the Eclipse help wiki and Lars Vogel for his Eclipse RCP programming tutorials.

References

[1] Vercoe B. et al. 2005. The Csound Reference Manual.
http://www.csounds.com/manual/html/index.html

[2] Eclipsepedia. 2010. Rich Client Platform. http://wiki.eclipse.org/index.php/Rich_Client_Platform

[3] OSGi Alliance. 2010. Benefits of Using OSGi http://www.osgi.org/About/WhyOSGi

[4]W3C. 1999. HTML4.01 Specification. http://www.w3.org/TR/html401/

[5] ECMAScript: The language of the web. 2010. http://www.ecmascript.org/index.php

[6] van Ossenbruggen, J. and Eliens, A. 1995. Bringing music to the web. In Proceedings of the Fourth International World Wide Web Conference, The Web Revolution, pages 309–314. O'Reilly and Associates, Inc

[7] Hearon J. 2009. Processing and csnd.jar. *Csound Journal*. 11 (5) -no page numbers. http://www.csounds.com/journal/issue11/Processing.html

[8] Lazzarini, V. 2006, 27-30 April. Scripting Csound 5. Proceedings of the 4th Linux Audio Conference. http://lac.zkm.de/2006/papers/lac2006_victor_lazzarini.pdf

[9] Alonso, Marcos and Geiger, Gunter and Jorda, Sergi. "An Internet Browser Plug-in for Real-time Sound Synthesis using Pure Data." In Proceedings, International Computer Music Conference, Miami, USA. International Computer Music Association. 2004 http://www.dtic.upf.edu/~malonso/pdplugin/pdpluginICMC.pdf

[10] CodeMirror. 2010. http://codemirror.net/

[11] JQuery JavaScript Library. 2010. http://jquery.com/

[12] Freed, Schmeder and Zbyszynski. 2007. Open Sound Control: A flexible protocol for sensor networking. *Presented at Maker Faire, San Mateo, CA, USA, 20th October 2007* http://opensoundcontrol.org/files/OSC-Demo.pdf

[13] Apache CXF. 2010. CXF User's Guide http://cxf.apache.org/docs/index.html

[14] Ignite Realtime. 2008. Smack API. http://www.igniterealtime.org/projects/smack/

[15] BeagleBoard.org. 2010 http://beagleboard.org/

[16] PandaBoard.org. 2010 http://pandaboard.org/

[17] Samaai S. and Barnes J. 2007. Investigating the effect of software bloat on users. http://dk.cput.ac.za/cgi/viewcontent.cgi?filename=17&article=1004&context=inf_papers&type=additional