AN EXPERIMENTAL ARCHITECTURE FOR INTERACTIVE WEB-BASED DSP EDUCATION

Martti Rahkila and Matti Karjalainen

Helsinki University of Technology
Laboratory of Acoustics and Audio Signal Processing
P.O. Box 3000
FIN-02015 HUT, Finland
Martti.Rahkila@hut.fi, Matti.Karjalainen@hut.fi
http://www.acoustics.hut.fi

ABSTRACT

This paper describes an experimental architecture for interactive education of DSP in the World Wide Web environment. The architecture is based on a client-server model providing means of distributing resources. The major design goal has been to combine interactivity and computational resources. The architecture itself is open in a sense that it does not specify implementation and it can be used for a variety of applications. Computer Based Education (CBE) of DSP is one area where it can be beneficially applied. An example of implementation for that purpose is presented. With our implementation, special attention has been paid to minimize the requirements of additional software for the user because of educational reasons. Thus only a Java-capable browser and audio support are needed at the user end.

1. INTRODUCTION

DSP is one of the natural target areas for Computer Based Education (CBE) because the main DSP tools are computers anyway. Thus one can ask: why not use computers for teaching DSP as well? At the Helsinki University of Technology, Laboratory of Acoustics and Audio Signal Processing, we have been doing research in Computer Based Education of DSP, digital audio and acoustics ever since 1994 [1]. During that time, the World Wide Web (WWW, Web) has become of major interest for distributing and developing CBE.

Despite the rapid growth and technological progress of the Web, many fundamental questions of CBE and DSP remain. For instance DSP still requires a lot of computational power. On the other hand interactivity is the key principle for CBE [2]. That is why the major goal for us has been to combine these two. At ICASSP'97 [3] we presented a tutorial level CBE application running on the Web (Figure 1). This time we go a bit further and present a technical solution for combining interactivity and computational power in a more general way.

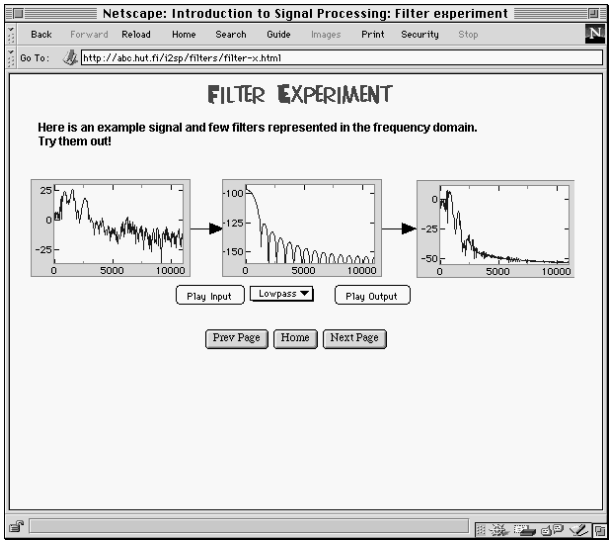

Figure 1: *An example of a simple, interactive exercise.*

The architecture itself is open in a sense that it does not specify the implementation. It is suitable for any task needing computational resources in a multi-user, network environment.

2. PRINCIPLES OF INTERACTIVITY IN THE WEB

The WWW is a powerful but easy-to-use hypermedia system. It was originally developed for documentation purposes providing ease of producing and distributing hypermedia documents. The basis for it lies in the HyperText Markup Language (HTML, *http://www.w3.org/MarkUp/*), a markup language fulfilling the SGML (Standard General-

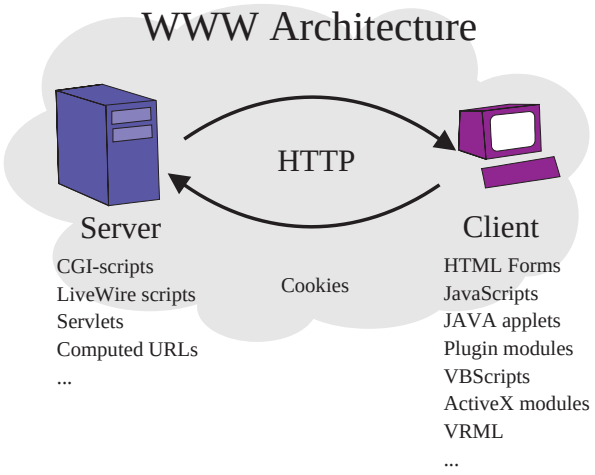

Figure 2: *Methods for creating interactivity in the WWW environment.*

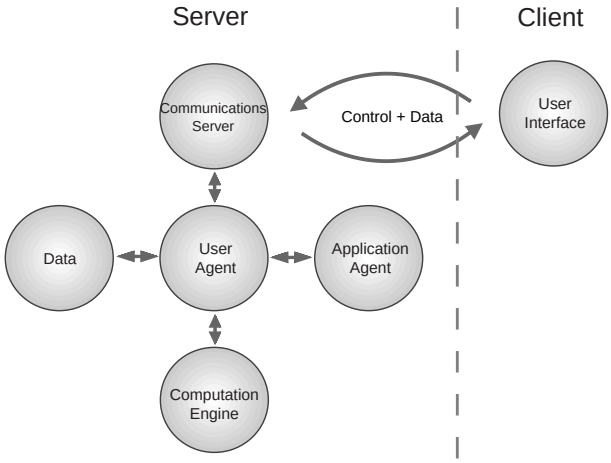

Figure 3: *The Logical structure of the architecture.*

ized Markup Language) standard [4]. HTML does not include programmability which has driven browser and server developers to come up with various solutions for producing interactive services. The most common ones are presented in Figure 2.

Common Gateway Interface (CGI, *http://hoohoo.ncsa.uiuc.edu/cgi/*) is one of the oldest solutions for creating on-line services. Together with HTML forms the CGI provides a method for getting input from the user and doing processing based on that input on-line at the server. Today CGI is widely used and supported. Later, variations and additions have appeared for server-side processing including, e.g. Computed URLs, advanced CGI-scripts in the CL-HTTP server.

On the client side there are plenty of added functionality to HTML. Plugins are programs that can interact with the WWW browser. Basically they are just like any other programs except that they fulfill the browser's API (Application Program Interface). Perhaps the most generic solution however is Java (*http://www.javasoft.com*), a platform independent programming language [5]. Modern WWW Browsers include a Java interpreter that can run special Java applications, called Applets, included into the HTML pages. For security reasons Applets received from the Web cannot utilize a local filesystem or other resources outside the browser.

Other interesting client-side solutions include scripting languages like JavaScript (*http://developer.netscape.com/library/documentation/javascript.html*), VBScript, and ActiveX (*http://www.microsoft.com/sitebuilder/*) components. All of these solutions have their pros and cons and it is hard to achieve browser independence. For complex functionality, one usually has to combine these possibilities. It is also remarkable that the fundamental concepts of the HTML language itself has remained pretty much the same.

3. OVERVIEW OF THE ARCHITECTURE

Figure 3 presents the logical structure of the architecture. It consists of six independent modules: User Interface at the client end and Communications Server, User Agent, Application Agent, Computation Engine and Data at the server side. Naturally there is also a protocol for information exchange between client and server.

The "visible" part of the architecture is the User Interface (UI). It is responsible for getting input from the user, forming a request based on that information, sending the request to the server, accepting and presenting results from the server to the user. The interface can vary from basic GUI functionality to complex data presentations.

The heart of the architecture, however, is the User Agent (UA). It takes care of processing the requests from the UI and returning appropriate data. Thus it controls all the other server side modules. It is also responsible for navigation. Actually the UA is a representative of the user at the server side: it follows the user's instructions when controlling the other modules. On the other hand, the UA can be thought of as a representative of the teacher: it provides the right information for the right user at the right time.

The Application Agent is an object providing information specific to the the application used. Its purpose is to serve User Agents with information of appropriate resources. It is not necessarily an active agent, it could also be a description of an application.

The Computation Engine takes care of the actual processing. In our example this is a DSP engine: a collection of signal processing routines. It could also be some other type of processing like a mathematics engine etc.

The Communications Server is responsible for exchanging information between the User Agent and the User Interface. The exchange is done via protocol specific to imple-

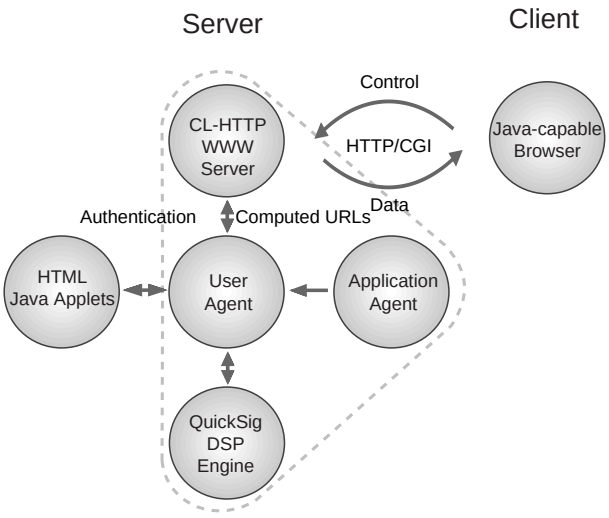

Figure 4: *An implementation of the architecture.*

mentation.

The Data module consists of the actual data used by Computation Engine and User Agent.

Basically the architecture is a model of distributing resources. The key idea is to divide the user interface and computation power in a client-server manner and control these with a user agent. This allows the user interface to be light but yet contain complex functionality when needed. The server side is flexible and extendable since it provides a multi-user, multi-application basis which is under total control of application developer and administrator. Evidently the server side is vulnerable to heavy load when several users request for computational operations. Nevertheless, the server power can easily be increased by adding capacity of the server and by limiting user access.

4. IMPLEMENTING INTERACTIVE DSP EDUCATION

Interactive, Web-based DSP education is one application area where this architecture can be used beneficially. We have implemented a tutorial-level CBE-application "Introduction to Signal Processing" [1] that demonstrates the architecture. Our implementation is presented in Figure 4.

The server side is constructed with two main components: the CL-HTTP [6] WWW-server and the QuickSig signal processing environment [7]. Both of them are object-oriented systems running on top of Common Lisp [8]. In the architecture, the QuickSig system is the Computation Engine and CL-HTTP the Communications Server. The CL-HTTP provides not only a standard WWW server functionality but also Computed URLs an advanced, CGI-like interface for on-line function calls. These calls are passed to User Agent which then in turn calls for QuickSig signal processing routines and finally returns the appropriate data.

The users are identified by standard authentication methods and a corresponding User Agent object is created for each user. The UA is also responsible for navigation and keeps track of request history. Application-specific information used by User Agent is obtained from Application Agent which is merely a description of the application.

Data consists of HTML-pages, Java Applets and additional data needed by QuickSig routines.

User Interface is built with HTML and Java Applets. Based on users' actions they send request and parameters to the server and the UA makes its decisions using that information. For example when on-line processing is needed, typically a Java Applet calls a computed URL, UA calls a QuickSig DSP routine and then returns the results as image or audio files. The Applet then shows these results accordingly. The advantage of this approach is a strict platform-independence at the user end. Also, users need no additional software, only a Java-capable WWW browser and audio support are needed.

An example (Figure 1): The user is presented with a spectrum of a speech signal and a filter that can be chosen from a popup menu. Selecting a filter invokes a request for a Computed URL. The Computed URL then passes this request to the User Agent which parses it, calls QuickSig to perform filtering and return the results as GIF images. The returned images are then updated to the screen by the Applet. Another example would be where users can design their own filters with sliders and choice items. In that case the User Agent would ask QuickSig to calculate the filter coefficients, perform the filtering and then return the results accordingly.

The server currently runs on a Macintosh computer. The reason for it is the QuickSig system. Currently the full QuickSig system runs only on a Macintosh and many of the routines have been optimized for that platform. The QuickSig DSP kernel is basically generic CLOS (Common Lisp Object System), just like CL-HTTP, but the graphics routines are specific to Macintosh Common Lisp (*http://www.digitool.com*).

Additional information on this project can be found from *http://www.acoustics.hut.fi/~mara/cbe/*.

5. CONCLUSIONS AND FUTURE WORK

This paper presents an experimental architecture for distributing resources on a multi-user, multi-purpose networked environment. The architecture is open in the sense that it does not specify implementation and experimental in the sense that it does not specify interfaces. It suggests a way to combine interactivity and computational power by distributing them over a client-server model. This can be utilized

in several application areas, one of which is CBE of DSP.

The fundamental elements in the architecture are User Interface at the client side and User Agent and Computation Engine at the server. This division allows the UI to be as flexible as needed, since computationally heavy operations such as signal processing are performed by the server. Thus there is room for UI to be simple or complex as desired. The "intelligency" is found in the User Agent. This object follows the user's requests and based on them calls for other resources. In other words, it represents the user for the server. On the other hand the same object controls which resources are available to the user and hence it also represents teacher or application developer!

An implementation for a WWW-based DSP tutorial is also described. This implementation uses CL-HTTP WWW server and QuickSig DSP system at the server side. The information exchange is done via an advanced, CGI-like interface Computed URLs and User Agent objects. Each user is represented by a UA object and standard authentication mechanism is used for identifying them. This solution makes it possible to serve multiple users with multiple applications.

The User Interface has been implemented with HTML and Java Applets. The Applets construct requests based on users choices and present the obtained data which are image and audio files. Some of the data is passed to the browser like new HTML pages. Unfortunately the Java language not yet supports any other audio formats except the SUN AU-format which is not suitable for broadband audio. The QuickSig supports AIFF format and thus the Applets let browser to take care of audio. In the next major release of Java, there should be support for AIFF files as well (*http://java.sun.com/ products/java-media/index.html*).

The virtue of our approach is its flexibility. It provides completely new possibilities for DSP education. For instance, an application could be designed to provide different layers for users with different background knowledge. Techically this could be easily achieved using authentication groups. On the other hand the UI can also contain rather complex functionality like a graphical block editor for DSP or listening tests.

Because the architecture itself is open for implementation, there are other interesting possibilities as well. In the future, there could be some other computational engines like for instance a Mathematics Engine. This kind of engine would give new possibilities for DSP education as well. The user interface could also be based on something else than Java, it could for instance be a plugin module. A very interesting approach could be a Virtual Reality Modeling Language (VRML, *http://www.sdsc.edu/vrml/*) based interface. VRML is a language for 3-D modeling and the models can also include activity which can be created with a scripting language or Java. A 3-D "study room" would be useful for room acoustics or 3-D sound and image processing related CBE-applications.

Of course, designing applications is somewhat difficult but when it comes to education, the most difficult questions are of contextual nature anyway.

6. REFERENCES

[1] Karjalainen M., Rahkila M., Learning signal processing concepts and psychoacoustics in the QuickSig DSP environment. In Proc. 1995 IEEE Int. Conf. Acoust., Speech, and Signal Processing (ICASSP'95), Detroit, Michigan, USA, vol 2, pp. 1125-1128, May 9-12, 1995.

[2] Rahkila M., A Computer Based Education System for Signal Processing. Helsinki University of Technology, Department of Electrical Engineering, Master's Thesis, 47 p. 1996. *http://www.hut.fi/HUT/Acoustics/ publications.html*

[3] Rahkila M., Karjalainen M., An Interactive DSP Tutorial in the Web. In Proc. 1997 IEEE Int. Conf. Acoust., Speech, and Signal Processing (ICASSP'97), Munich, Germany, 1997.

[4] ISO 8879. 1986. Information Processing – Text and Office Systems - Standard Generalized Markup Language (SGML). *http://www.iso.ch/cate/d16387.html*

[5] Lemay, L.; Perkins, C. L.; Morrison, M. 1996. Teach Yourself Java in 21 Days - Professional Reference Edition. Sams Publishing. USA. 1247 p.

[6] Mallery, J. C. 1994. A Common Lisp Hypermedia Server. In Proc. 1994 First International Conference on the World-Wide Web (WWW'94), CERN, Geneva, Switzerland, May 25-27 1994. *http://www.ai.mit.edu/projects/iiip/doc/cl-http/server-abstract.html*

[7] Karjalainen M., DSP software integration by object-oriented programming: a case study of QuickSig. IEEE ASSP Magazine, pp. 21-31, April, 1990.

[8] Steele, G. L. 1990. Common Lisp - The Language. 2nd ed. Digital Press.