The ITB Journal

Volume 3 | Issue 2 Article 3

2002

The JAM Suite - a Voice-enabled Network-based Virtual Band Application

Brian Nolan

Martin Cummins

Irene Flanagan

Niall O'Brien

The JAM Suite - A Voice-enabled Network-based Virtual Band Application

Brian Nolan, Martin Cummins, Irene Flanagan, Niall O'Brien

Institute of Technology Blanchardstown

Abstract

The Java Audio Music-suite (J.A.M. for short) is a suite of applications that provides tools and audio processing utilities for musicians. It's core functionality includes a means whereby musicians who are geographically dispersed can play music together. An additional utility determines the actual musical notes from a piece of music using Fourier Transform techniques. All of these functions are complemented by a voice-enabled, animated help agent that takes in voice commands using speech recognition and reports errors via text-to-speech technology.

1.0 Introduction

The Java Audio Music-suite (J.A.M. for short) is a suite of applications that provides tools and audio processing utilities for musicians. Its core functionality includes a means whereby musicians who are geographically dispersed can play music together. An additional utility determines the actual musical notes from a piece of music using Fourier Transform techniques. All of these functions are complemented by a voice-enabled, animated help agent that takes in voice commands using speech recognition and reports errors via text-to-speech technology.

The basis of the application, the primary function, is a facility that allows geographically dispersed musicians to play live together over a network, or over the Internet using Java sockets with the Real Time Protocol (RTP).

The application has a number of functions, including the Fourier transform function. With this function, the user can input a melody into the system live, or as an audio file. This data is then passed through a Fourier algorithm, such that the data stream is converted into a group of frequencies. These frequencies are then be used to discern the musical notes that the frequencies represent. The notes found are then be used to get the key of the music and from that we get the accompaniment to be played with the music. This is both a valuable resource for songwriters who do not play instruments and a handy way for musicians to save time

while learning a new musical piece. At the moment we have implemented this function with slow mono-melodic audio streams (melodies with no harmonies). Differing degrees of audio compression and filters eliminate as much noise as possible so that the algorithm can be more effectively.

Another function is a help and error reporting system for the end user. We use Java speech technology for this. We complement this work with a macro-language defined in Backus-Naur Form (BNF) to underpin the voice activation system that allows a musician, whose hands are busy, to enter commands and make the choices presented by the help system. Error reports are returned through a speaking help agent.

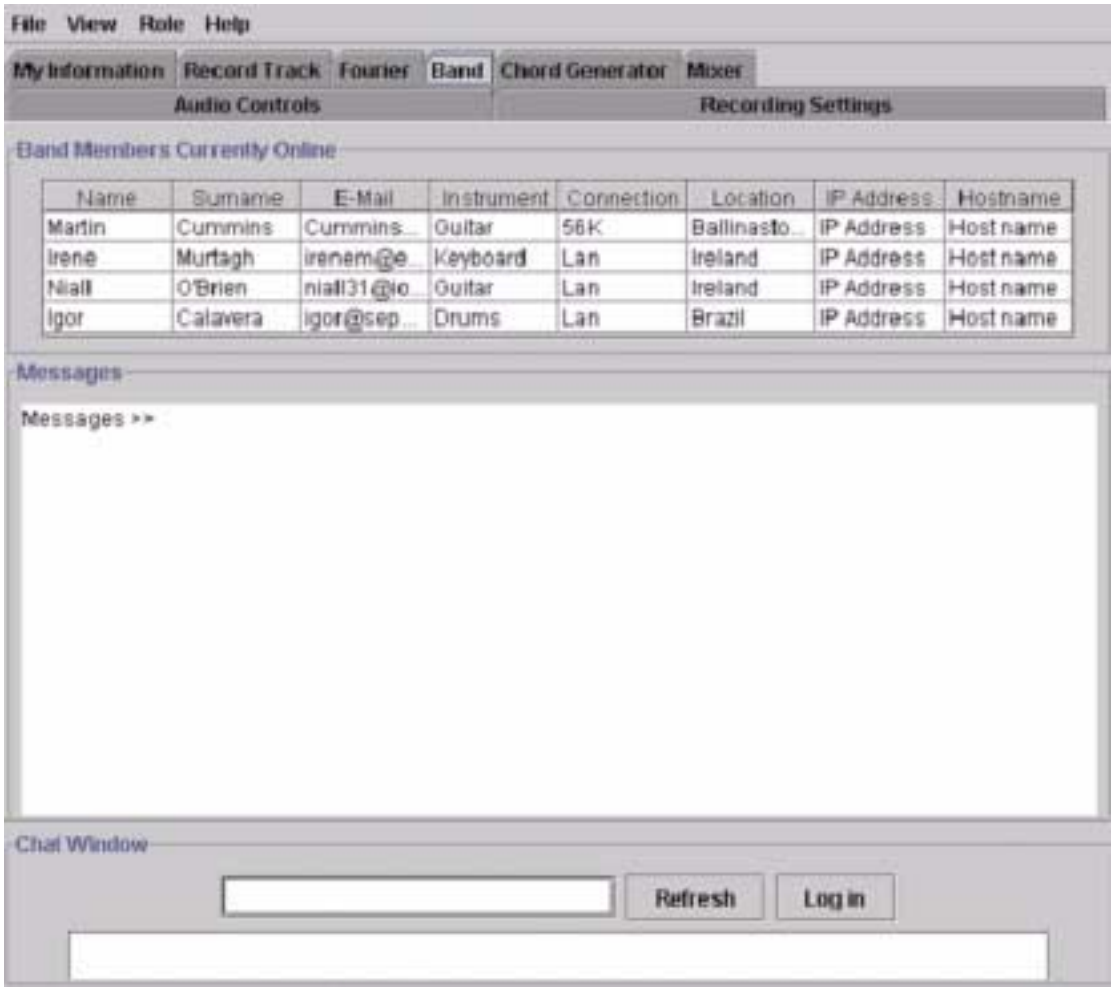

Figure 1: Screen shot of a user screen.

As a tool:

1. The software allows musicians who are geographically dispersed to play live together over a network or over the Internet. The audio data is synchronized so that the musicians are able to hear their own audio stream, in real time, but merged with the other musicians' audio stream.

2. Another user, who is not playing the music, is able to add sound effects and alter the respective volumes of the instruments remotely. This module, called the mixer module, has access to all of the audio streams and is able to manipulate the data for the users to hear. The mixer can record the data and manipulate it prior to the processing of the audio stream.
3. Each user has a basic version of the above on their own agent software that allows them to alter the sound and level of their own instrument.
4. We plan to include chord generators for guitars and keyboards (these will include a facility to hear the chords). The chords will be played from either MIDI data stored and reproduced through a sequencer, or from recorded audio.
5. A FFT (Fast Fourier Transforms) function discerns the notes from a melody and, from these notes, the key and chords to accompany that melody and the tempo of the music.
6. An intelligent help agent monitors the user's actions and provides help, suggestions and advice as appropriate. The user interacts with the agent by means of a voice-enabled speech recognition capability, and receives advice in the form of synthesized speech.

We describe some of the functions in more detail in the following sections.

2.0 The Virtual Band Function

2.1 Introduction

This function allows users who are geographically dispersed to play music live together over a LAN or through a suitable modem connection. This function is included as a core feature in the JAM suite. The primary technologies we utilise here are multi-threaded Java sockets, Java's Real Time protocol (RTP). Later we may use Java's Remote Method Invocation.

2.2 Difficulties for the Application

The primary difficulties are slow network speeds and difficulties synchronizing the users' input. This is further complicated by the time-critical nature of this kind of streaming multimedia application. We have considered a number of possible architectures and solutions to address these difficulties. We believe a client-server architecture to be the most effective because it helps to minimise the network traffic of the application. This minimisation is achieved by merging the audio streams received from the clients at a central point (the server)

so that the audio streams are merged before they are sent back as playback. These streams are captured and transmitted instantaneously using the Java Real Time Protocol (RTP).

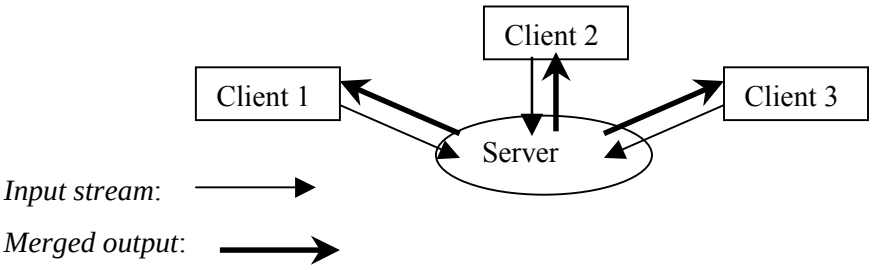

Figure 2: Overview of the Client-Server Architecture of the JAM suite

We have decided against the idea of using a *peer-to-peer* architecture because it would result in too much processing for each of the clients. For example, configured as a peer-to-peer topology with five users, each client would have to a), transmit its own stream and b), receive four others in real-time as well as c), capturing its own stream and d), playing all five.

2.3 The Buffer Idea

We have devised some techniques for handling slow connection speeds. One of these involves buffering the packets of audio data and numbering them on the transmitters side. We begin by sending only the first and fourth of the packets (for example). When their receipt is acknowledged we then send the second and third packet, but only if the server discerns that there is sufficient bandwidth to transmit the data. The server then has the option of deciding if the connection is fast enough at that time to warrant sending all four of the packets. If not, it will discard them and the process continues as previously. The client then has only the first and fourth packets. These are presented to the user with a "blank space" where packets two and three should be. This is obviously not ideal but it helps to ensure that there is minimal time lag in the playback of the audio streams on the receiver side. The data that was not sent in real-time could be sent to the user at a later stage and played as a backing track. The backing tracks are discussed shortly.

The basic steps underpinning the solution to the slow connection problem are:
Step 1: The audio stream is divided into a number of packets.
Step 2: Packets 1 and 4 are transmitted.
Step 3: An Acknowledgement is sent from the receiver on receipt of the "packets".
Step 4: The server discerns if it is viable to send the second and third "packets"

Step 5: Client leaves two blank spots (where packets 2 & 3 were) and plays one and four.

2.4 The "Overdub Idea"

Another idea we have devised is analogous to recording in a recording studio. It involves the users playing along to a backing track that is pre-recorded either in a previous session, or included as a default in the client. The users then agree between themselves on a backing track prior to the beginning of the recording session. They then all play along individually. The audio data is then sent to the server. These streams are amalgamated by the server and transmitted back to all the clients for their perusal. Clients then have the option of over-dubbing their parts afterwards. The server can keep copies of the original streams so that they may be included into later recordings of the same piece, or as backing tracks for later session.

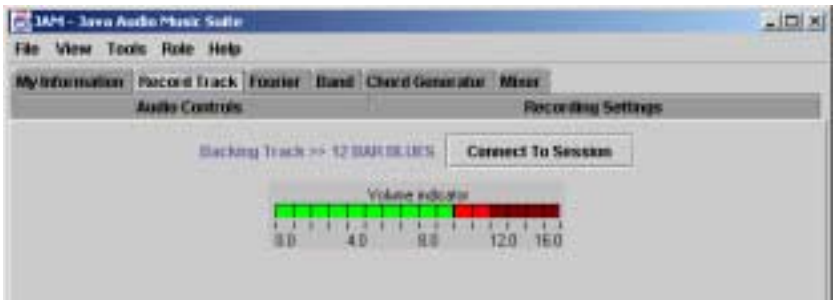

Figure 3: The figure shows the user recording a track accompanied by a backing track "12 bar Blues" .

3.0 The Fourier Transform

3.1 Introduction

In this section we describe the current applications for which Fourier Transforms are utilised. The Fourier transform decomposes or separates a waveform or function into sinusoids of different frequency which sum to the original waveform. It identifies or distinguishes the different frequency sinusoids and their respective amplitudes (Brigham: 4).

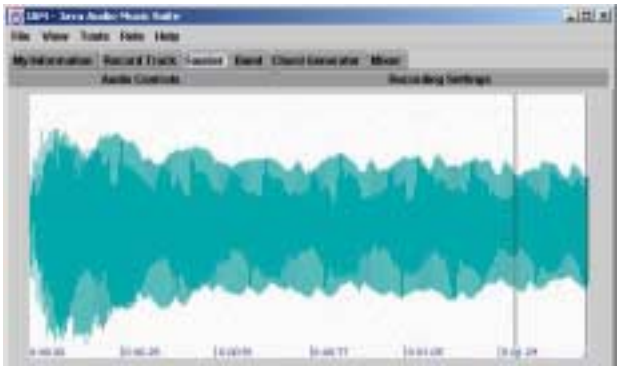

Figure 4: This figure shows the audio stream being inputted into the system prior to its transformation by the Fourier algorithm.

3.2 How We Use Fourier Transforms

We use Fourier transforms in the JAM suite to extract a precise frequency from an audio sample using spectral analysis. By comparing the discrete notes frequency and using a lookup table of note values and approximate frequencies, the application can then relay the musical notes to the user. This is done for all of the notes that are in the audio stream or sample.

3.3 Fourier Deployments

One of the deployments of the Fourier transforms is to provide the function performed by an electronic guitar tuner.

This involved:

- Creating a module to take in the audio samples in real time and place the audio data into a buffer.
- The sampled audio is then run through low pass IIR filters to clean it up before it is passed to the FFT transformation. The use of filters guarantee that only guitar range frequencies are involved in the spectrum and that the constituent noise in the sample is removed.
- The audio samples are processed with a Fast Fourier Transform to determine their spectral content.

Since the fundamental frequency of a guitar note contains the most energy, the FFT code is used to locate the frequency with the most energy from the buffer of samples. After all of the FFT data is examined, the frequency with the most power is be returned and this represents the current pitch of the guitar string.

3.4 Musical Pieces That Are Not Mono-Melodic

The next deployment using FFT builds on the guitar tuner application so that it functions over musical pieces that are not mono-melodic. This is achieved by processing an audio sample with a Fast Fourier Transform to determine its spectral content. However, this time we relay to the user the many musical notes in the audio sample. We achieve this by increasing the sampling rate and the number of transforms that will be required. This is

computationally expensive so at present we limit this function to operating over short pre-recorded musical pieces.

4.0 The Help Agent

4.1 Introduction

The purpose of the JAM Help agent is to provide users with context sensitive assistance at any stage during their utilisation of the program. It is also used for error management, providing the user with helpful information when an error has occurred, as opposed to the typical technical output of error information returned by the Java. User Interaction with the agent is by means of speech recognition and synthesis. The graphical interface for the help agent is designed in 3D Studio Max and Adobe Photoshop, and the plan is that the agent will react in accordance with the type of help that it is contextually required.

4.2 Contexts in Which the Help Agent will appear

The context in which help is provided will affect two areas:

4.2.1 Type Of Help

The JAM suite has many different stages of use with some steps being a necessary path in the achievement of other later steps. The help agent maintains a hierarchical representation of these steps and is context sensitive. If help is requested while the user is performing a step that leads to, for example, connecting to the mixer, then general mixer help will be made available to the user.

4.2.2 The Agents Appearance

We plan that the agent will have a set of pre-defined animations appropriate to the different types of help available. For instance, if the help were in relation to the Fourier Transform tool, then the agent would be holding a waveform in his hands as the tool is explained. We are still developing these sets of animations.

4.3 How We Implement The Speech Technology

The help agent interacts with the user by means of speech technology. This speech is provided through the use of IBM's implementation of the Java Speech API (JSAPI). The

speech is implemented through the use of the *runtimes* function provided by IBM's Via Voice. These *runtimes* contain the speech engines and data files that are integrated with this JAM suite application. The types of speech technology incorporated into the agent are speech recognition and speech synthesis.

4.3.1 Handling The Limitations Of Speech Recognition Technology

For the speech recognition part of the implementation we decided to limit the amount of commands available to the user to minimise he need for conversation level interpretation. The commands available to the user are basic discrete single words or phrases, examples of which include the simple "yes" and "no". We use a formal notation based on Backus-Naur Form to describe the syntax of a given set of commands. Due to the hierarchial nature of the commands and the context sensitivity provided by the agent we designed and built a small macro-language, and this is described in the same using the same BNF notation.

4.3.2 How We Will Implement Speech Synthesis

A decision had to be made as to whether we should use Java Speech Mark-up Language (JSML) for speech synthesis functionality included within our agent. JSML allows greater control and flexibility over Java speech by letting the programmer change the way in which words are said. We decided that the current version of the agent would not utilise this feature, but future versions might.

One of the attractions of Java is that utilising speech synthesis with the Java Speech API is relatively simple. One (merely) creates a synthesiser and invokes it's *speakPlainText*() method while handing it a String argument to be dictated.

4.4 The Graphical Side of the Help Agent

The character for the agent is initially based on the late Jimi Hendrix and is designed and rendered in 3D Studio Max through the use of Mesh Modelling. We design and export around 10 different AVI files, which are supported by the Java Media Framework (JMF v2.1.1). Through the use of JMF we create a player, which has a different AVI file running depending on the context of the help or during idle time. When the agent is speaking we also have a speech bubble onscreen that displays the text version of the help message being provided.

5.0 Conclusion

To conclude we feel that Java provides the ideal technology for this type of project because of the richness of the functionality within the language.

We feel that this application, as well as being an interesting and valuable learning experience for us, is a useful application of immediate interest to geographically dispersed musicians.

Java's cross platform compatibility is a very important for the success of the JAM suite, as not only will the users be from geographically dispersed areas but also they could be running the application on entirely different platforms. Java technology inherently allows for this eventuality.

Java's versatility and extensibility as a programming language proved very useful in the implementation of the JAM suite. Through the use of the Java Speech API and the Java Real Time Protocol, the benefits of developing with and deploying in Java become very apparent.

6.0 References

Lindsey, Craig A. (2000). *Digital Audio With Java*. Pearson Press UK.

// ***Interesting chat about Java sound and what it can do***
http://developer.java.sun.com/developer/community/chat/JavaLive/1999/jl1116.html
// ***Sound API programmers Guide***
http://java.sun.com/j2se/1.3/docs/guide/sound/prog_guide/title.fm.html
// ***Java Sound API Documents***
http://java.sun.com/products/java-media/sound/
// ***Interface Mixer taking audio lines and returning a single audio stream Clips***
// ***and real time***
http://www.chmsr.gatech.edu/java/api-js099/javax/sound/sampled/Mixer.html
// ***Java RTP (Real Time Protocol) page***
java.sun.com/products/java-media/jmf/2.1.1/solutions
// ***Fourier Transform Theory***
http://www.educatorscorner.com/experiments
// ***Brigham, Bracewell, Brault and White -Informative papers on Fourier Theory***
http://aurora.phys.utk.edu/~forrest/papers/fourier/
// ***Spectrum analysis using FFT***
http://www.dsptutor.freeuk.com/analyser/SpectrumAnalyser.html