AALIVENET: An agent based distributed interactive composition environment.

Michael Spicer
School of Info-communications Technology, Singapore Polytechnic
mspicer@sp.edu.sg

Abstract
An approach to creating distributed interactive composition environments is presented. This approach utilizes autonomous agents running on their local clients to overcome network latency issues. A prototype implementation, written in Java, is discussed.

1 Introduction
In recent years there has been an interest in developing systems that facilitate improvisatory ensemble performance over computer networks, such as the Internet. Several of systems of this type were presented at I.C.M.C3003. It should be noted that this type of system has distinctly different goals when compared to systems that facilitate collaboration by file sharing, such as resRocket, or those that try to use some sort of score following to facilitate synchronization of notated performances at different locations.

The central issue surrounding distributed ensemble performance is the management of network latency. There are several common approaches to handing this problem. One approach is to ignore latency and let the performers accommodate it. An example of this approach is the Quintet system (Hajdu 2003). Another approach acknowledges latency as an inherent aspect of the medium, and incorporates it as a feature of the performance. peerSynth (Stelkens 2003) takes this approach.

This paper presents AALIVENET, which uses a client/server based autonomous agent approach to try and sidestep the issue. Each client machine independently generates a local rendition of the music, based on the latest information it has received from the human performers. The server acts as an information hub and echoes the performers intentions to the various client machines. All of the local machines will converge on some state that will reflect the various performers intents, but the music produced at each site will vary in the details.

2 Ensembles of Agents
AALIVENET builds on the approach used in the AALLIVE system that I developed for interactive composition (Spicer, Tan and Tan, 2003). Both systems are designed around the idea of goal driven artificial performers, implemented as agents, which are controlled by human performers. Agents can be thought of as software entities that exhibit a degree of autonomous behavior. They exist in an environment, which they can perceive and act upon in some way. For more information on Autonomous Agents see (Russell and Norvig 2003) or (Wooldridge 2002).

My first agent-based system, AALIVE, runs stand-alone. One human performer specifies high-level musical goals that the complete ensemble of agents tries to realize. In contrast, the new system, AALIVENET, is designed to allow several human performers, in different locations, to interact with each other, via the agents under their control.

Each human performer specifies the musical goals for one member of the virtual ensemble. This agent acts as a kind of proxy for the human performer on all the other client systems. Thus, each human performer hears an ensemble of agents running on his local machine, and these local agents autonomously try to achieve the musical goals that their human performer has specified. As a result, each local instance of the ensemble will be producing music with the same kind of general characteristics, but the details will vary from machine to machine. peerSynth uses a similar approach.

3 Server Implementation
AALIVENET is implemented in Java using a client/server approach. The server is very basic. It simply manages clients and echoes the commands from the human performers to all the other clients logged in. There is no server side manipulation of the data; the server merely acts as a distribution hub. The commands consist of strings that are used to specify the goals for each of the virtual performer agents running on the client computers and are sent using the U.D.P. protocol.
This system can be thought of as a virtual “studio” where people can perform music, in the same way an I.R.C. chat room can be thought of as a place where people can talk. (An I.R.C. chat system is included in AALIVENET to allow the human performers to communicate by typing messages to each other.)

4 Client Implementation

The client side of AALIVENET where all the real work is done. The AALIVENET client is implemented as a Java applet, and makes use of the JavaSound default synthesizer to produce the music. The applet acts as a container for the local agents to run on the client machine. In addition it provides a graphical user interface for the human performers and the U.D.P. network “plumbing”.

The agents’ are contained in Java object that represents the “virtual studio”, called CSystem. This abstraction of the system makes it easy to implement other AALIVENET client applications with different user interfaces. The current interface is implemented using standard “Swing” components, but I am doing some experiments with an alternative interface written in OpenGL.

4.1 Internal design of a basic agent

As stated above, each virtual musical performer on the client is implemented as an agent. The agent design is based on “Sample and Hold” approach used in AALIVE, and is implemented in as a Java class called CAgent. This stores the pitch and note duration contours in digital wavetable oscillator objects. (The oscillator objects essentially encapsulate an array, to store the waveform samples, and methods for accessing and manipulating the array contents.) A master clock, locked to the tempo of the music, synchronously clocks the agents’ two oscillators. (The system uses 24 ticks per quarter note. This very low time resolution seems to be helpful in keeping the musical output “in time” in the very unpredictable time constraints of the Java environment.)

When an agent plays a note, the value at current position in the duration oscillators wavetable is used to determine how long the note will play, and consequently, where in the pitch oscillator the next pitch value will be sampled. This combination of oscillators can be thought of as a kind of Finite State Machine, where the current pitch is the current state, and the duration oscillator determines the state transition.

This approach is very intuitive to use, especially when a plot of the oscillator waveforms is displayed. Novice users can quickly grasp how the wave shapes affect the melodic lines that are created. The waveforms stored in the two oscillators can be generated using a number of standard algorithms. Examples include:

- Standard synthesizer waveforms (sin, pulse, sawtooth, triangle etc.)
- Probability distributions (uniform, gauss, exponential etc.)
- Chaotic functions (logistic and 1/f noise)
- Fractals

In addition there are several common melodic transformational processes that can be applied, such as:

- Inversion
- Retrograde
- Scaling (intervallic augmentation/diminution)

The above agent design was chosen for its ease of use and execution speed. Because the CAgent class is an abstraction, other agents can easily be integrated into the system that uses completely different approaches. I have done some preliminary work on agents implemented using neural networks (in particular, a multiplayer perception that was trained using pitch contours derived from Irish jigs).

4.2 Specialized Agent performers

There are a number of specialized agent performer classes that are descended from the CAgent base class. Some of these are adapted to produce the idiomatic roles of ensemble performers. These include agents that play the role of:

- Bass player
- Kick and snare drum player
- Arpeggio player

These agents essentially use slightly different mappings of the pitch oscillator data onto MIDI note numbers, as well as adding extra methods that may fulfill useful idiomatic musical needs. Examples of this are methods that implement behaviors like “always play the kick drum on beat one”, “only play notes consonant with the current chord” etc.

4.3 Sub-ensembles of agents

There are some agent performers that are designed to work together as a sub-ensemble. These utilize an important aspect of the agent approach, which is that agents can perceive their environment, and part of that environment is other agents. The sub-ensemble agent groupings consist of a lead agent, which is controlled human performer, and one or two other agents that adjust their internal parameters in response to the state of their lead agent. These agents make use of the tracking and evasion techniques common in computer games. Example applications of this technique include:

- Tracking the pitch of the lead agent performer and displacing by a fixed interval. This can produce block harmony parts.
- Using evasion on the duration parameter to produce interlocking rhythm patterns between players.
4.4 Musical goals for the agent

The human performers interact with the agents in two main ways:

- Specifying the initial pitch and duration contours, and
- Providing target values for the average pitch and average duration of their agent performer (or group of performers, in some cases).

Each agent on the client machines adjusts itself to achieve that target in its host environment. In this way, all the client environments will converge to produce music with similar characteristics. The adjustments of the pitch and duration are carried out using a gradient descent learning approach in both systems. A small bias is applied to the values in the relevant wavetable so as to bring the average value in the table closer to the target value.

This is similar to the older AALIVE system, but AALIVE used the average pitch and duration values of the entire ensemble in its error calculation, rather than working at the level of individual agents.

4.5 Performance Interface

Because of the design of CAgent, making use oscillators with a fixed set of waveforms, and the setting target values, the current user interface is reminiscent of an analog synthesizer. In fact the experience of performing with the system feels very much like performing on an old Moog or Serge system. The current interface is shown below.

Currently there are no other ways for the user to interact with the system, but an agent is planned that will set its pitch and duration wavetables by listening to a MIDI controller, enabling a more conventional performance paradigm to be used. This data will be transferred to the corresponding agents on the other client applets. Similar agents can be produced that extract information from performance on acoustic instruments, although the realization of this performance on other client sites will obviously synthesized.

5 Applications

5.1 Virtual concerts

There are several areas where AALIVENET can be usefully deployed, the most obvious being the “distributed concert”. In addition to the performer clients, that allow the user to interact with the agents, it is possible to produce passive clients, which would allow a user to simply listen to a performance. This is implemented on the quintet.net system.

5.2 Stand alone

Although designed as a distributed ensemble tool, AALIVENET can be used stand alone, with no network. This is useful for learning how to the system, and also good for producing a solo “ensemble” performance! In this mode, the user, selects which agent to control, sets the appropriate goals, and then repeats this process for the other agents in the ensemble.

5.3 Virtual teaching labs

One of the powerful aspects of using interactive composition environments is the possibility that “non-musicians” can be effectively involved in the music making process. The system will be used as kind of virtual improvisation laboratory, which I can use in an introductory music course. In my teaching situation, resource and timetable constraints make it impractical to use ensemble improvisation to learn about fundamental musical concepts. This software makes it possible to have a large number of small groups of students interacting with each other in their own virtual spaces, while being physically in the music lab all at the same time.

5.4 Mobile applications (one day soon!)

The AALIVENET client is very small, less than 100 kilobytes, and it can be implemented without floating point arithmetic. This means that it should be able to run on mobile devices. I have visions that in the not too distant future, the teenagers that sit in the garden outside my apartment in the evenings will be able to play music together on their 3G (or some other packet based system) hand phones, instead of simply triggering back their ring tones, like they do now!
6 Conclusion
The AALIVENET system effectively demonstrates the viability of using autonomous agents as virtual performers in network based interactive composition environments. This will enable people (trained in music, or not) to perform together in a number of situations that are currently impractical.

References