Andante: A Mobile Musical Agents Infrastructure

Leo Kazuhiro Ueda[1]*, **Fabio Kon**[1]

[1]Department of Computer Science
Institute of Mathematics and Statistics (IME) – University of São Paulo
Rua do Matão, 1010 – 05508-090 São Paulo, SP – Brazil
http://gsd.ime.usp.br/andante

lku@ime.usp.br, kon@ime.usp.br

Abstract. *In the last decade, researchers in the area of distributed systems created the concept of mobile agents: active, autonomous objects capable of executing computation in a computer network, migrating from node to node. We propose the use of this new concept and related technology for the creation and performance of music within a distributed computing environment.*
In this paper, we introduce Andante, an open-source infrastructure for the construction of distributed applications for music composition and performance based on mobile musical agents. We describe a prototype implementation of this infrastructure based on Aglets, a Java mobile agent platform, as well as a sample application built on top of this infrastructure.

1. Introduction

Composers have always looked at contemporary scientific achievements to devise new forms of producing their art. Over the past decades, we have witnessed an astonishing development of Computer Science that led to an intensification of its relationship to Music [Miranda, 2001, Roads, 1996]. Nowadays, desktop computers can be equipped with inexpensive hardware and software and produce high quality sound. The Internet has also brought many new possibilities for music making [Kon and Iazzetta, 1998].

In this context, we are interested in discovering how an advanced concept in Computer Science, namely, *Mobile Agents*, can be applied to introduce new forms of musical composition and performance.

We introduce in this paper the Andante project, which offers an infrastructure to create distributed applications that use *Mobile Musical Agents* to compose and perform music. Using Andante, programmers can build their own agents to build such applications. We now wish to attract the interest of composers and researchers to use this infrastructure to conduct interesting musical experiments.

2. Mobile Agents

A *mobile agent* is a computer program that can interrupt its execution on a host, migrate to another host travelling through a network, and resume its execution on the new

*Supported by CAPES.

host [Kotz and Gray, 1999]. It is an autonomous program in the sense that it can decide itself to migrate and it may react to changes on the host environment.

This concept brings advantages [Lange and Oshima, 1999] over the traditional client/server paradigm. Since the agent migrates to a foreign host, remote commands are actually executed locally, so the latency caused by network communications is eliminated and the traffic is reduced. Applications that involve the processing of large amounts of distributed data can benefit from the use of mobile agents.

Our infrastructure is built on top of the *Aglets Software Development Kit* (ASDK). *Aglets* [Lange and Oshima, 1998] is a mobile agent system, written in Java and originally developed by IBM (see `http://www.trl.ibm.com/aglets`). It is currently an open source project (see `http://aglets.sourceforge.net`), offering libraries and applications to implement and deploy Java mobile agents.

The choice of Aglets has more to do with the benefits of Java as a middleware platform. Java provides a good solution in terms of platform independence. This is a major concern in the Andante project, considering that we want both musicians and programmers to use the system in different environments. To state more precisely, we want the system to run on distinct operating systems such as Windows, MacOS, and Linux. Moreover, the Java Swing is a solid library for the construction of platform-independent graphical interfaces and the Java Sound API (see `http://java.sun.com/products/java-media/sound`) simplifies the use of audio devices.

3. Mobile Musical Agents

We define *mobile musical agent* (simply *agent*, from now on) as a *mobile agent* which participates in a musical process. It may do so by performing one or more of the following.

Encapsulating an algorithm. As a computer program, an agent can carry algorithms, in particular composition algorithms [Miranda, 2001, Roads, 1996, Rowe, 1993], allowing it to produce music. These algorithms may also require input data that may be carried with the agent.

Interacting and exchanging information with other agents. Similarly to a situation where real musicians play together on a stage, several agents can interact with each other.

Interacting with real musicians. An agent can receive commands or audio/musical data from a real musician. This interaction could be as simple as reproducing the notes played by the musician on a keyboard. The musician could also dynamically control parameters of an algorithm executed by an agent, for example, through a MIDI keyboard.

Reacting to sensors. Agents can receive commands from other non-agent programs. These commands could be triggered by sensors so that the agents could react to events in the real, physical world.

Migrating. A migration process can be set off by the above actions. In other words, the agent can decide to migrate (1) stochastically or deterministically, based on an algorithm; (2) based on the interaction with other agents; (3) based on the interaction with musicians; and (4) by reacting to sensors. An agent that migrates resumes its performance when it arrives to its destination.

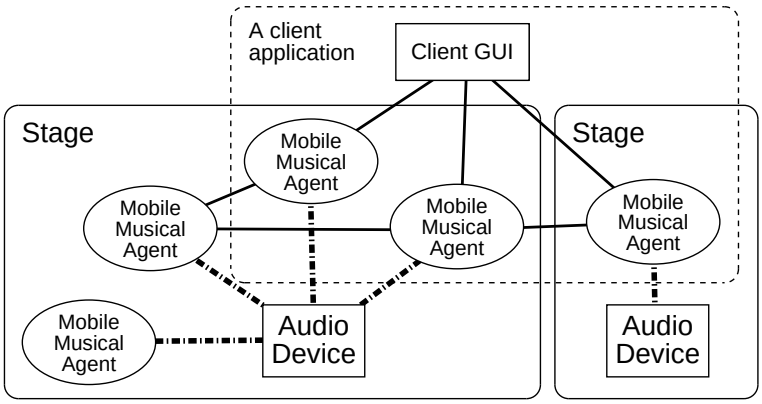

Figure 1: Architectural overview

As an example, consider a museum or exhibit hall equipped with several computers connected by a network. Each computer could be equipped with motion sensors and host a few agents. The agents would communicate to each other and play a distributed music piece in a synchronized manner. A specific agent could receive information from motion sensors in order to follow a person who walks around the room (using its ability to migrate). The listener perception would be that part of the music is following him. Another part of the music, on the other hand, could run away from the listener migrating to computers far from where the listener is.

4. Andante Architecture

An agent performs its actions in a heterogeneous computer network environment. The computers in this network must run a host software that we call *Stage*. A Stage can host multiple agents. Agents in a Stage can perform their actions, including interact with each other and play music.

To produce sound, the Stage maintains an *Audio Device*, which in turn provides musical instruments for the agents to "play".

We have thus defined three key elements of the Andante architecture: the (mobile musical) agents, the Stage, and the Audio Device. Figure 1 gives an abstract overview of the architecture. An additional fourth element, the Client GUI, is not necessarily part of the Andante infrastructure, but it plays the important role of supporting human interaction with agents.

4.1. Implementation

A UML [Booch et al., 1998] class diagram for the abstract overview would look like Figure 2.

Our initial prototype is completely based on Java. However, since we would like to allow parts of the system to interact with components written in other programming languages, all the communication among the components are performed via CORBA [OMG, 2002]. The CORBA middleware allows programs written in different languages and running on different operating systems to communicate to each other seam-

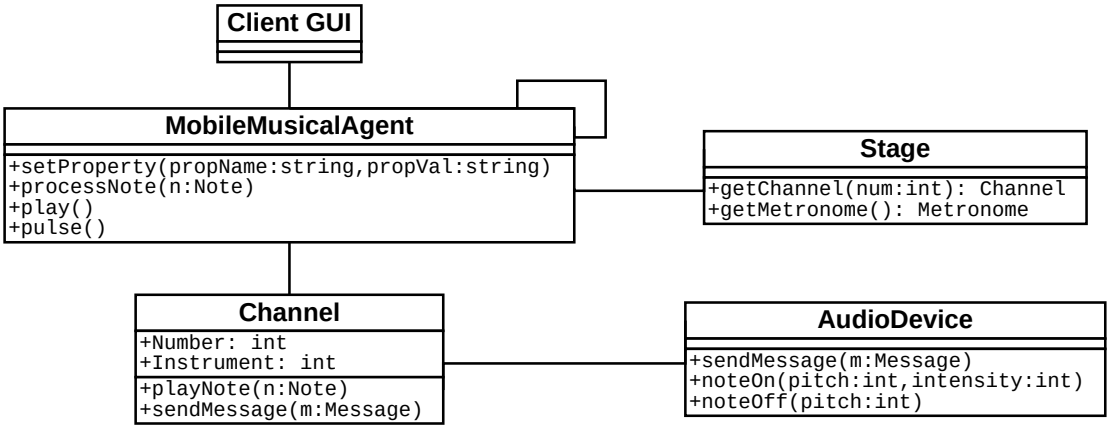

Figure 2: Class diagram of the abstract overview

lessly. We will need this flexibility in the future when we implement a variety of audio devices such as a CSound [Boulanger, 2000] device.

As mentioned in Section 2, the infrastructure is based on the ASDK. The `MobileMusicalAgent` is a subclass of Aglet so it can migrate from one machine to the other carrying its code and state. The `Stage` class is capable of hosting Aglets (again, with the help of the ASDK). Thus, `MobileMusicalAgent` and `Stage` must be written in Java.

Lastly, the `MobileMusicalAgent` class offers methods to create instances of itself and to send them to a Stage.

4.2. An Example Scenario

We now describe a hypothetical situation that illustrates a possible use of the described architecture.

- *Agent implementation*: Assume we have implemented the `SampleAgent` class (a subclass of `MobileMusicalAgent`). For simplicity, we will use only one instance of this agent in this example, although it is possible to implement and use several different kinds of agents. Let us call this instance *sampleAgent*.
- *Dispatching*: after being instantiated, *sampleAgent* needs to be dispatched to a host Stage. The Stage is an instance of the `Stage` class, let us call the instance *stage*.
- *Getting a Channel*: to be able to play something, *sampleAgent* has to gain access to the Audio Device. The *stage* has an instance of the `AudioDevice` class, let us call it *audioDevice*. The *sampleAgent* can then access the *audioDevice* by getting a Channel from the *stage*. The *stage* actually provides an instance of the `Channel` class, which we shall call *channel*.
- *Playing notes*: *sampleAgent* can now play notes using *channel*.
- *GUI control*: *sampleAgent* may receive commands from a graphical user interface (GUI) while it is playing.

Figure 3 depicts this scenario more precisely (with a UML sequence diagram).

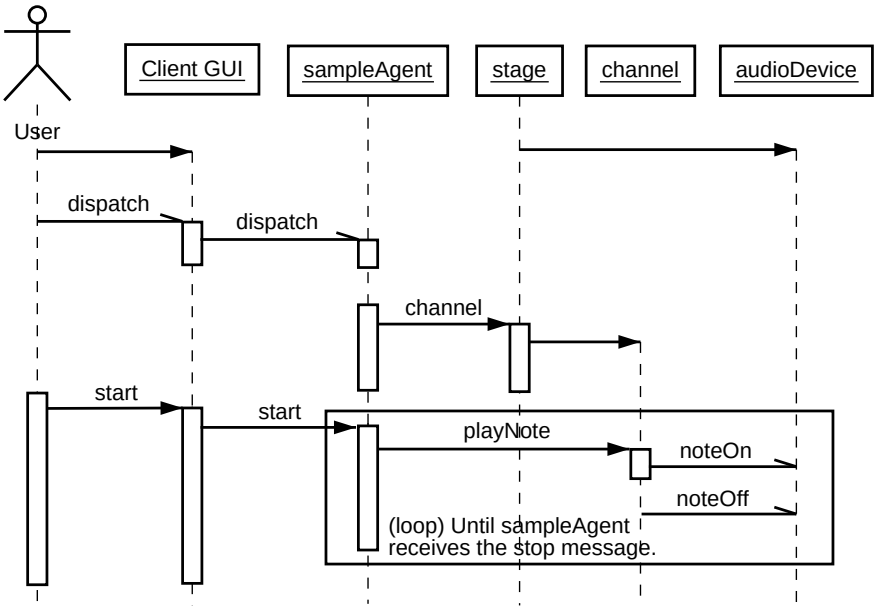

Figure 3: UML sequence diagram for the example scenario

4.3. Technical Issues

Our current implementation of the Andante infrastructure carries a number of simplifications. The use of the Java Sound API to generate sound is one of them.

We implemented the `AudioDevice` class using the MIDI classes provided by the Java Sound API. As a result, the interfaces of the Andante infrastructure are based on the MIDI protocol. On the other hand, we were careful not to get too influenced by this protocol so that we could experiment with other alternatives for sound generation in the future. In fact, as we already mentioned, we plan to use the CSound package to implement another Audio Device in the near future.

In our experiments we found that the Java Sound API has a very significant latency and jitter problem in its MIDI synthesizer class. To overcome this, we used an alternative platform-dependent implementation of the Java Sound API called Tritonus (see `http://www.tritonus.org`). The Andante infrastructure can still use the official Java Sound API implementation as a platform-independent solution.

5. Sample Application: NoiseWeaver

To give a more concrete demonstration of the architecture viability, we built a sample application that takes advantage of the Andante infrastructure. This application, which we call the *NoiseWeaver*, composes and plays stochastic music in real-time.

It implements only one kind of agent: the *NoiseAgent*. This agent generates a single melody in real-time. In the generated melody, simulations of selected types of noise determine the pitch, intensity, and duration of notes[1]. For example, a certain NoiseAgent could play a melody in which the pitch of the notes is determined by a sequence of num-

[1]See the *fractal algorithm* description in [Roads, 1996] (fourth printing, pages 880–886).

bers that simulates a pink noise. This same agent could have a brownian noise sequence to determine the duration of the notes, and a white noise sequence for the intensity.

The Stage Metronome service is used to synchronize the NoiseAgents. Each NoiseAgent registers itself with the Metronome service so the Metronome can send `pulse` messages to all registered agents at regular times. The time interval between pulses is determined by the Metronome time signature and tempo properties, and all the registered agents receive the pulse at (almost) the same time.

The NoiseWeaver provides a GUI to control several hosted NoiseAgents. It can find the NoiseAgents using the CORBA Naming Service [OMG, 1998].

A NoiseAgent holds various properties that influence the generation of its melody and the GUI lets a user change these properties, even while the agent is playing on remote nodes. Figure 4 shows a screen shot of the GUI.

The *Find* tab lists the NoiseAgents found in the distributed system composed of the machines collaborating to generate the musical piece. The *Control* tab acts as a control panel for each of the NoiseAgents listed in the *Find* tab. On the *Commands* pane, we have:

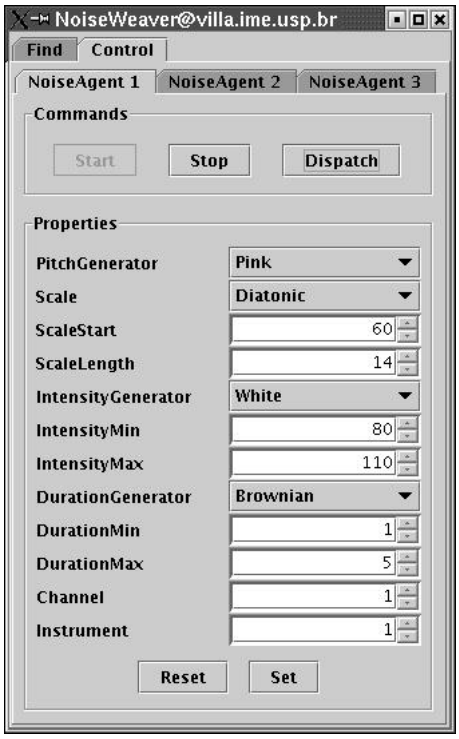

Figure 4: The NoiseWeaver GUI

Start: tells the agent to start (or resume) playing.

Stop: tells the agent to stop playing.

Dispatch: tells the agent to migrate to another Stage.

And on the *Properties* pane we have:

Pitch: *PitchGenerator* defines the type of noise which generates the pitch of the notes. The selected noise is used to generate integer numbers which are then mapped to notes in a certain musical scale. The *Scale* property determines the scale to which the generated pitch refers. *ScaleStart* is the point where the selected scale starts. The value 60 is equivalent to the middle C (as in the MIDI protocol). The *ScaleLength* determines the length (number of notes) of the chosen scale.

Intensity: *IntensityGenerator* defines the type of noise used to generate note intensity (velocity, in MIDI terminology). The intensity is an integer between 0 and 127, *IntensityMin*/*Max* determine the interval of possible intensity values.

Duration: similar to the intensity settings. The duration values are related to the Metronome time signature.

Channel: the channel to which the agent will output its melody.

Instrument: the instrument (patch) to be used to play the melody.

The possible values for the *Generator* properties are *Constant*, *White*, *Pink*, and *Brownian*. For *Scale*, we have *Diatonic*, *WholeTone*, *Chromatic*, *HarmonicMinor*, *HarmonicNatural*, *Pentatonic*, *Blues*, *PentaBlues*, and *Hirajoshi*.

With NoiseWeaver, it is possible to have several NoiseAgents playing a noise-based melody each, and control the way all these melodies are generated using its GUI.

6. Related Work

In *DASE* (Distributed Audio SEquencer, `http://www.soundbyte.org`), users running client applications work with a sequencer to build musical loops. These loops are sent to a centralized server, where they are all played together.

Musical Agents [Fonseka, 2000] is a distributed algorithmic composition system where musical agents collaborate with each other to create a musical performance. This system defines a scripting language designed for musicians to write algorithmic compositions. Here, a musical agent is an application capable of executing scripts in this language, and an agent can communicate with similar agents over a network. The sound is generated by MIDI instruments.

Similarly to the Andante project, both DASE and Musical Agents have a collaboration aspect, where music is created in a distributed fashion. However, DASE is not agent-based (the loops are sent and presumably built by humans) and the Musical Agents are not mobile. Another fundamental difference is that the Andante's proposal is to offer a generic infrastructure on top of which programmers and composers will build musical applications/compositions.

In the field of Artificial Life, multi-agent systems may be used to model biological phenomena. Miranda [Miranda, 2002] presents "*a simulation where a society of autonomous agents evolves a common repertoire of intonations from scratch by interacting with one another.*" His goal is to study the relationship between the evolution of music and language. The Andante infrastructure would be a very suitable environment for experimentations with such multi-agent simulations.

7. Ongoing and Future Work

We are currently working on new applications and musical pieces created over the Andante infrastructure. Our ongoing and future work include the following.

- The *NoiseMaestro*, a scripted version of the NoiseWeaver. Instead of being controlled by a GUI, the NoiseAgents will be guided by the NoiseMaestro, which is controlled by a script (analogous to a music score). This script defines time-stamped changes in properties of specific NoiseAgents. So a script would be a stochastic musical piece with a fixed structure.
- An agent that receives notes from a real musician. The notes are inputed through a musical keyboard and processed by an agent which generates music based on the musician's input, resembling existing interactive music systems [Rowe, 1993].
- Agents that stream audio data could allow musicians playing conventional instruments to participate in a distributed Andante performance with musicians in different locations.

- The use of the CSound system for sound synthesis. We will then have other possibilities for sound generation and, therefore, new ideas for making musical.

We are improving the usability of the Andante user interfaces, as it is currently not friendly to lay users. Although the Andante infrastructure is already functioning well, we will continue to evolve its architecture, adapting it to the needs of new users and applications.

8. Conclusions

We are looking for composers willing to experiment with the Andante infrastructure and applications. Their feedback will be of great value for the refinement of our system.

We want Andante to be an open community of art and technology lovers who collaborate to create musical ideas, mobile musical agents, and to develop the enabling software infrastructure. We have created a Web site through which programmers, musicians, researchers, and composers can collaborate with the project.

References

Booch, G., Rumbaugh, J., and Jacobson, I. (1998). *The Unified Modeling Language User Guide*. Addison-Wesley.

Boulanger, R., editor (2000). *The CSound Book*. The MIT Press.

Fonseka, J. R. (2000). Musical agents. Honours thesis, Monash University.

Kon, F. and Iazzetta, F. (1998). Internet music: Dream or (virtual) reality. In *Proceedings of the 5th Brazilian Symposium on Computer Music*, Belo Horizonte, Brazil.

Kotz, D. and Gray, R. S. (1999). Mobile agents and the future of the internet. *ACM Operating Systems Review*, pages 7–13.

Lange, D. B. and Oshima, M. (1998). *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley.

Lange, D. B. and Oshima, M. (1999). Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89.

Miranda, E. (2001). *Composing Music with Computers*. Music Technology Series. Oxford (UK): Focal Press.

Miranda, E. R. (2002). Mimetic development of intonation. In *Proceedings of the 2nd International Conference on Music and Artificial Intelligence (ICMAI 2002)*. Springer Verlag - Lecture Notes on Artificial Intelligence.

OMG (1998). *CORBAservices: Common Object Services Specification*. Object Management Group, Framingham, MA. OMG Document 98-12-09.

OMG (2002). *CORBA v3.0 Specification*. Object Management Group, Needham, MA. OMG Document 02-06-33.

Roads, C. (1996). *The Computer Music Tutorial*. The MIT Press.

Rowe, R. (1993). *Interactive Music Systems*. The MIT Press.