A FRAMEWORK FOR MUSICAL MULTIAGENT SYSTEMS

Leandro Ferrari Thomaz and Marcelo Queiroz
Department of Computer Science - University of São Paulo
{lfthomaz | mqz}@ime.usp.br

ABSTRACT

Multiagent system technology is a promising new venue for interactive musical performance. In recent works, this technology has been tailored to solve specific, limited scope musical problems, such as pulse detection, instrument simulation or automatic accompaniment. In this paper, we present a taxonomy of such musical multiagent systems, and an implementation of a computational framework that subsumes previous works and addresses general-interest low-level problems such as real-time synchronization, sound communication and spatial agent mobility. By using it, a user may develop a musical multiagent system focusing primarily in his/her musical needs, while leaving most of the technical problems to the framework. To validate this framework, we implemented and discussed two cases studies that explored several aspects of musical multiagent systems, such as MIDI and audio communication, spatial trajectories and acoustical simulation, and artificial life constructs like genetic codes and reproduction, thus indicating the usefulness of this framework in a variety of musical applications.

1 INTRODUCTION

Agent technology is particularly suited for musical applications due to the possibility of associating a computational agent with the role of a singer or instrumentalist. With these associations one can map features such as performance, perception, adaptation and improvisation on one side, and artificial processes on the other. Moreover, it is possible to define forms of social interrelationship between agents, which brings this technology even closer to collaborative musical performance. By focusing the discussion on multiagent systems we purposely leave aside any aesthetic issues specific to the choice of compositional algorithms for each agent, and concentrate on the study of communication and interaction (i.e., sociology) of musical agents in the collective musical production context [1, 2].

Most of the previous work involving computer music and multiagent systems is generally very limited in its scope, dealing with problems such as generation and evaluation of melodies [3], pulse detection [4], simulation of specific instruments [5] or automatic accompaniment and collective performance in a tonal context [6, 7]. A more ellaborate example may be found in Living Melodies [8], in which the authors build an artificial world of music players with a well-defined spatial structure (including sound propagation), as well as rules for walking, interacting and music-making. Works that aim at more general settings are the MAMA architecture [9], where agents communicate using a symbolic representation of the piece being performed, and the Andante project [10], that allows agents to migrate between machines in a distributed computer environment.

This work aims at both a definition and an implementation of a general framework for musical multiagent systems, as well as a study of their inherent problems. We extend on [9] by allowing synchronous and asynchronous communication of various sorts (audio signals, symbolic music streams, video signals and other forms), artificial life concepts (life, death and reproduction of agents) and physical simulation (sound propagation and motion of agents). Our musical agents can also benefit from the migrating abilities of Andante's agents [10], since both systems are written in Java.

A central issue in this project is the treatment of space, a musical attribute of utmost importance in contemporary musical composition, which has been superficially explored in the context of multiagent systems in previous works [8]. Thus, one of the main goals of our framework is to allow the simulation of sound propagation in a virtual environment, automatically adjusting the sound information received by each agent, depending on its position and listening conditions.

Through the observation of common elements among existing musical multiagent systems and by proposing new features and tools, we intend to put forward both a conceptual and a computational framework that eases the task of implementing a musical multiagent system that best fits a given musical application. On one hand, simulation of existing musical multiagent applications should be straightforward by using the framework with its basic components; on the other hand, extending the framework by adding new features such as compositional algorithms, music and sound analysers and synthesisers, or rules for handling artificial life aspects of the virtual musical agent society, should be made easy by freeing the user from low-level implemen-

SMC 2009, July 23-25, Porto, Portugal

tation details such as synchronization and communication protocols, and allowing him/her to concentrate on the conceptual level of the musical project.

2 A TAXONOMY FOR MUSICAL AGENTS SYSTEMS

This section presents a preliminary taxonomy of musical multiagent systems that has been generalized from previous musical multiagent works [8, 9, 10, 5, 6, 11, 7, 4], as well as other works that deal with related issues [3, 12].

The higher level categories considered in a musical multiagent system are the musical agent, the virtual environment, and interactions (both among musical agents and between musical agent and virtual environment). Figure 1 summarizes these categories and their components.

2.1 Virtual Environment

An environment in the context of a musical multiagent system can be defined as a virtual space in which computational agents are immersed and interact with it by means of sensors and actuators.

Physical Representation. An environment's physical representation includes the definition of a virtual world together with every physical information that is relevant to the musical application. These may include space representation (dimensionality, connectedness), sound representation (MIDI, audio) and propagation (including acoustical effects such as air absorption, reflection, diffraction or Doppler effect), and mechanical laws (gravity, inertia, attraction and collisions).

Ecological Representation. Systems inspired in Artificial Life [8, 3] often use some representation for energy to control a few aspects in an agent's life, such as eating, walking, growing old, among others. Energy may be defined and used to control which actions (requiring some amount of energy) an agent may perform, if it needs refueling (through food consumption, for instance) or rest, or if it is able to reproduce.

2.2 Musical Agent

The musical agent is a computational agent specialized in processing sound and musical information. Typically, this agent is capable of analysing incoming sound, performing some sort of musical reasoning, and creating a response by means of sound processing and synthesis. Figure 2 shows a musical agent and its components, described below, immersed in a virtual environment.

Knowledge Base. The knowledge base is a storage area where the agent keeps everything related to its know-how in music-making as well as its memory of past events. This includes algorithms for composition, music theories, non-musical facts about the environment and other agents, techniques for music or sound encoding and processing, linguistics, ontology, among others.

Reasoning. We denote by Reasoning the set of mechanisms an agent uses to decide its future actions, musical or otherwise. An agent's output may combine several compositional strategies, such as playback of recorded fragments, context-based reactions to musical input, or analysis and synthesis tools within a structural plan to achieve a compositional goal.

Sensors and Actuators. Sensors capture information from the environment and forward it to the cognitive center (Knowledge Base + Reasoning) of the agent. They are usually specialized in receiveing a particular type of sensorial information, for instance auditory sensors, visual sensors or tactile sensors. An agent may have several similar sensors (e.g. two ears, two eyes) to aid its cognitive capabilites (e.g. to perceive direction or distance of a sound source).

Actuators are the active counterparts to sensors, and are responsible for affecting the environment, driven by the agent's reasoning. They produce sound, perform movement, and generate events that may change the way the world is perceived by other agents.

Analysis and Synthesis. In addition to any kind of analysis automatically done by an agent's sensors, higher level analysis may be requested by the agent's reasoning in order to achieve its musical goals. These may include contextual analysis (comparing instantaneous inputs to previous data) and planning of future events (comparing possible outputs to expected outputs by other agents), and may include a number of specific techniques of signal processing such as temporal and spectral analysis, analysis of psychoacoustical phenomena (such as pitch detection or perceived loudness) and musical analyses of all sorts (rhythmic, melodic and harmonic analysis, or analysis of genre, style and expression).

Synthesis may be regarded as the most fundamental part of a musical agent, since through it an agent partakes musically in a collective performance. It includes symbolic synthesis of high-level events (as in MIDI or MusicXML), sound synthesis and signal processing techniques, but also non-musical information such as spatial trajectories or graphical output (to communicate visually with other agents or as a visual counterpart to the musical performance).

2.3 Interactions in the Environment

In a collaborative musical performance there are many kinds of interaction that may be simulated in a musical multiagent system. Besides the obvious exchange of sound information during performance (through hearing and playing together), other information may be exchanged, such as gestures for guiding the performance (for instance in slowing

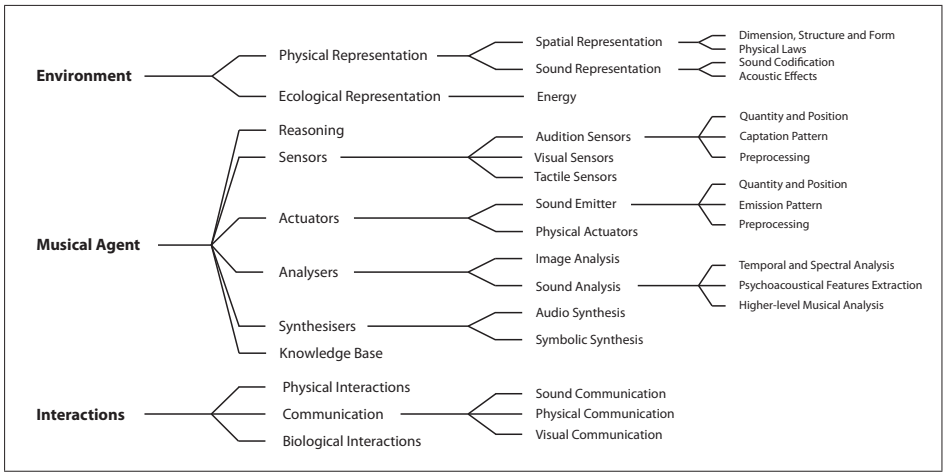

Figure 1. Taxonomy of Musical Multiagent Systems

the tempo down), or may be agreed upon before the actual performance starts (such as a musical score or a general plan for the composition).

When the simulation includes physical or biological elements, other interactions may be observed, among agents or between agent and environment, that affect their states. These include mobility of agents through external forces or free-will, feeding and reproduction, to name a few.

3 FRAMEWORK'S ARCHITECTURE AND IMPLEMENTATION

The present architecture was designed with plugability in mind, that is, it should be possible for a user to develop new components which are easily added to the original framework, and also connect component instances to an agent in execution time. For instance, a running system might consist of simple agents with one sensor for receiving sound and one actuator for emitting sound; later on, the user might want to change those into anthropomorphic musical agents, by plugging another sound sensor at a different position in the agent's body, and creating a component that analyses incoming sounds received by both sensors and infer sound source positions.

The framework presuposes the utilization of a multiagent system middleware that provides the required infrastructure for maintaining agent execution and controlling messages exchanged among agents. The current implementation uses JADE 3.6 [1] with the Java 6 programming language.

3.1 Framework's Actors

Relying on the taxonomy described in the last section, a *musical agent* is modeled as an aggregate of linked components, such as reasoning, knowledge base, sensors, actuators, analysers and synthesisers. In order to create a musical agent, one defines its components either by reusing existing ones, or by extending basic classes to create new components.

The virtual environment is represented by a unique agent called *environment agent*. This agent is composed by a physical representation of the virtual world in which the agents live, and by event server components. *Interactions* are represented by events that flow between musical agents and the environment agent, where each event type represents a particular kind of interaction, such as sound exchange, verbal messages, gestures, and so on.

An *external agent* is a human or an outside system that interacts with the framework in runtime. It is embodied in the virtual world by a regular musical agent, so that it interacts with the environment and other musical agents using the default mechanisms. For instance, a compositional algorithm implemented in Pure Data [2] might communicate with the framework using sockets, by sending a stream of notes to be played through the agent's actuator.

3.2 Virtual Time

Time in the virtual environment is controlled by a virtual clock that allows agents to obtain the current time and schedule tasks. This virtual clock may be managed in two different ways: *by the internal clock*, which means that the virtual clock is tied to the computer's internal clock and so to the

[1] available at `http://jade.tilab.com/`. 13 april 2009.

[2] available at `http://crca.ucsd.edu/~msp/software.html`. 13 april 2009.

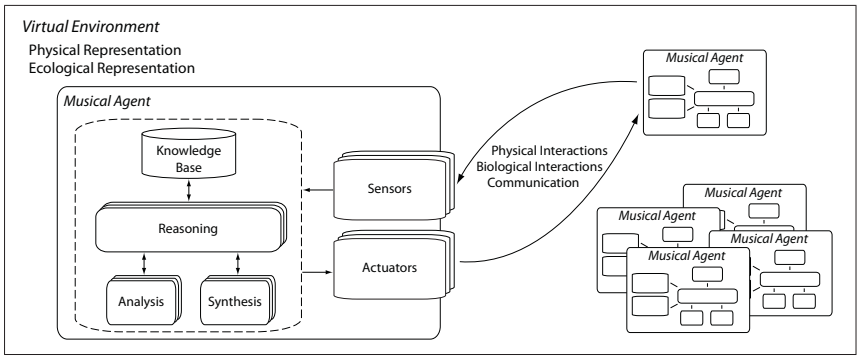

Figure 2. Musical Agents in the Virtual Environment

external flow of time; and *by the user*, which means that the user, usually through the environment agent, updates the clock, giving rise to non-homogeneous time measurements.

3.3 Communications

Communication in the multiagent middleware serves two different purposes: to enable the operation and control of the framework, and to implement the aforementioned interactions in the virtual world. The first purpose is addressed by commands, while the latter correspond to communication via events.

Commands. A command is a message passed between agents that controls the framework's internal functioning. Every agent has a dedicated asynchronous communication channel that is able to send and receive commands, and is used to pass control messages such as registering an agent, informing a change of turn (see 3.4), or updating a public fact in its knowledge base.

Events. Interactions between musical agents and the environment is done by means of exchange of events, which is always controlled by the environment's event servers. Events are used to model all sorts of interactions in the virtual world, such as exchange of audio chunks, an agent's collision with an obstacle, non-musical messages between agents, among others.

There are two exchange modes for events: *sporadic* events, such as gestures or verbal messages, can be sent at any rate and instant; and *periodic* events, such as audio chunks, which must obey a frequency of exchange that is previously agreed upon by all components that use that event type.

The periodic exchange mode is a synchronous communication process involving a set of sensors, actuators and an event server. In this case, virtual time is divided in frames of the same period as the periodic event, and processing is always done ahead of time, i.e. while an event is happening in the environment, reasonings and actuators are working

	Deadline	**Description**
t_0	*frameTime*	Frame start time.
t_1	*needAction*	In this instant, the actuator automatically warns its registered reasonings that an action must be taken to produce the next frame.
t_2	*agentDeadline*	Deadline for actuators to send frames to the event server.
t_3	*receiveDeadline*	Deadline for events sent by actuators to arrive at the event server; after this time, arriving events are discarded.
t_4	*sendDeadline*	Deadline for the event server to send events back to registered sensors.
t_5	*period*	End of current frame, and simultaneous start of the next one.

Table 1. Deadlines for periodic exchange mode

to send the event corresponding to the next frame. Actuators and event servers have state machines that tell them to work or to wait for some response from other agents, and their internal states are affected by scheduled tasks which are triggered at user-defined times. The deadlines for agents and event server is detailed in table 1.

The communication interface, present in every sensor, actuator and event server, is responsible for sending and receiving events by means of the callback methods *send()* and *receive()*, respectively. Two interfaces were implemented and compared: *communication by messages*, that uses the message passing system native to JADE, based on FIPA-ACL messages; it works by encapsulating an event inside a message and relaying its delivery to JADE; this kind of communication can be used in networked systems, but it may be slow and therefore not suited to time-sensitive events; and *communication by direct calls*, implemented as a single JADE service instance that can be accessed by every agent; a component must register its access point in this service, which will be used by the service to deliver the event by a direct call to the interface's *sense()* method; this implementation has the advantage of being much faster than the traditional message passing method, but on the other hand it blocks the service process while *sense()* does not return, so

it must be carefully and thoughtfully used.

3.4 Execution Mode

There are two possible execution modes: *Batch mode*, where time is discrete and divided in turns, and is updated by the environment agent only after every agent responds with whatever it is supposed to produce in that turn, with no time constraints whatsoever. It can be used when there is a need to control the sequence of actions, or it can be used in computationally intense non-interative processes to create a musical output for later appreciation; and *Real-Time mode*, where time is controlled by the internal clock, and every agent is responsible for producing their outputs ahead of time, or else they will be silent, i.e. the environment agent does not wait every agent to complete its action, and only forwards events that arrived up to each deadline. Agents may rely on some fallback strategy that sends previously computed outputs whenever a certain real-time deadline cannot be met.

4 CASE STUDY

To validate the current stage of the framework development, two different systems were implemented, to test the framework for user programmability, robustness and computational performance. These two systems were chosen in order to explore the use of both symbolic and audio communication streams, both batch mode and real-time execution modes, and artificial life concepts.

4.1 Living Melodies

The Living Melodies [8] system is a complex example of musical multiagent system having various kinds of sensors and actuators that deal with sound, energy, life cycle, movement, and contact. Agents sing, walk, reproduce and die, and their physiology includes not only hearing and singing devices, but also tactile sensors, genetic codes, preference rules for reproduction and energy management. The system uses a symbolic codification of sound as musical events, which are propagated through a bidimensional discrete space in a manner similar to wave fronts.

The simulation of the Living Melodies system was based on an article [8] and a software [3] . Although not explicitly said by the authors, it seems to use a monolithic non-real-time architecture, in which the processing of agents' actions is made in a round-robin fashion by a single thread. The mapping to the framework required the use of the batch execution mode.

The re-implementation of this system within our framework allows the user to fine-tune many parameters of the simulation, including the number of agents, genetical material, agent's age limits, sound absorption by the air, world size, among many others. It also features a graphical user interface that shows the spatial distribution of the agents in the world, as well as wavefronts of sound propagation.

4.2 Audio Exchange and Acoustical Simulation

A simple audio system was conceived to test the framework capabilites in dealing with real-time audio exchange. Musical agents are positioned in a virtual two-dimensional space, and produce audio streams that are constantly broadcasted to other agents. Each agent receive a mixture of the sound in the environment, considering the effects of delay and attenuation of every other signal according to relative distances between agents. The user may be immersed in the virtual environment, disguised as a special musical agent, who captures all sound received at its virtual position, and renders it through an audio interface. As an example of application, this system might be used for placing several human musicians in a virtual environment, and letting them play together in real-time, hearing each other as they would in the virtual acoustic space.

In real-time terminology, this kind of audio exchange is classified as soft real-time since the loss of a deadline is not catastrophic for the system. Such a loss simply implies that an agent will be mute during a time frame. On the other hand, the loss of a deadline by the event server is more serious because it will cause all agents to be mute for a frame. To test for robustness and performance of the system, we measured the number of successful fragments delivered to the event server, as a function of both the number of agents and the size of the audio chunks. Table 2 shows the results over 25 seconds of digital audio delivered by each agent [4] .

Samples	Period	# Musical Agents					
	(ms)	1	5	10	25	50	100
44100	*1000*	100	100	100	100	100	100
22050	*500*	100	100	100	100	98	77
11025	*250*	100	100	100	97	75	*
4096	*90*	100	100	100	86	*	*
1024	*23*	100	100	93	*	*	*
512	*11*	100	100	67	*	*	*
256	*5*	100	90	*	*	*	*

Table 2. Percentage of fragments received by the Event Server. Stars represent simulations that could not keep a constant audio exchange rate.

As expected, there is a tradeoff between the number of agents and the size of audio chunks, in order to keep full functionality. When the number of agents increases past a certain point, the system performance lowers due to the increased number of threads and, consequently, of process time and memory. Also, very small chunks increase the

[3] available at `http://www.ituniv.se/~palle/projects/living-melodies/`. 13 april 2009.

[4] Using a Intel Core 2 Quad 2.40 GHz, 4 GB of and Windows Vista.

rate of loss because there is less processing time available to agents within each time frame. Since this implementation uses a temporal resolution to schedule tasks of the order of miliseconds, and the deadlines are too close to each other, this lack of precision can sometimes desynchronize the agents' state machines.

5 DISCUSSION AND FUTURE WORK

The implementation of two systems (Living Melodies and Audio Exchange and Acoustical Simulation) has shown that the current state of framework development is capable of covering systems that have different demands, such as real-time digital audio and exchange of symbolic sound information. Features made available by the framework allow the user to concentrate in solving his/her specific musical problem without worrying about lower level problems, such as communication between agents and synchronization.

As for the exchange of periodic events, tests have shown that it is possible to accomplish real-time audio exchange between agents, keeping a constant audio rate, by carefully choosing the size of the audio chunks as a function of the number of agents needed. However, since audio chunk size determines the overall latency in agents' perception of each other's performance, it is desirable to improve the communication mechanism in order to allow for smaller audio chunks with larger number of agents.

Apart from refinements in the current implementation of the framework that might lead to a better performance, another more structural approach to this problem is to use the Real-Time Java Specification [5] , which provides a high resolution time clock (in nanoseconds), and the possibility to do real-time scheduling with this resolution. Although using a real-time operating system and a real-time Java infrastructure to execute the application should produce better results, we intend to keep both implementations available, so as not to impose complicate system requirements on the common user.

From the user interface point-of-view, we intend to provide access for users of various levels of expertise. Users with no programming experience or who only need existing components can define a musical multiagent system simply by writing a text description, which is interpreted and executed by the framework. This script only points out which components (names, quantity and parameters) belong to each kind of musical agent and environment agent, and the global parameters of the simulation. This might also be done through a graphic interface, where one manipulates existing components, their interconnections and parameters. During the execution of the system, it would be possible to visualize a representation of the virtual environment with its agents, and to plug-in new components in runtime. Advanced users with programming skills may extend the framework by programming new components with special features using Java, either adapting existing codes to their needs or writing out new components and maybe new features for the architecture.

[5] available at `http://java.sun.com/javase/technologies/realtime/index.jsp`. 13 april 2009.

6 REFERENCES

[1] H. J. Levesque, P. R. Cohen, and J. H. T. Nunes. On acting together. *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, page 9499, 1990.

[2] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[3] E. R. Miranda. Emergent sound repertoires in virtual societies. *Computer Music Journal*, 26(2):77–90, 2002.

[4] S. Dixon. A lightweight multi-agent musical beat tracking system. *PRICAI 2000: Proceedings of the Pacific Rim International Conference on Artificial Intelligence*, page 778788, 2000.

[5] L.L. Costalonga, R.M. Vicari, and E.M. Miletto. Agent-based guitar performance simulation. *Journal of the Brazilian Computer Society*, 14:19–29, 2008.

[6] G. L. Ramalho, P. Y. Rolland, and J. G. Ganascia. An artificially intelligent jazz performer. *Journal of New Music Research*, 28(2):105–129, 1999.

[7] R. D. Wulfhorst, L. Nakayama, and R. M. Vicari. A multiagent approach for musical interactive systems. *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 584–591, 2003.

[8] P. Dahlstedt and M. G. Nordahl. Living melodies: Coevolution of sonic communication. *Leonardo*, 34(3):243–248, 2001.

[9] D. Murray-Rust, A. Smaill, and M. Edwards. Mama: An architecture for interactive musical agents. In *Ecai 2006*, 2006.

[10] L.K. Ueda and F. Kon. Mobile musical agents: the andante project. In *Conference on Object Oriented Programming Systems Languages and Applications*, pages 206–207. ACM New York, NY, USA, 2004.

[11] P.A. Sampaio, G. Ramalho, and P. Tedesco. CinBalada: a multiagent rhythm factory. *Journal of the Brazilian Computer Society*, 14:31–49, 2008.

[12] P. M. Todd and G. M. Werner. Frankensteinian methods for evolutionary music composition. *Musical Networks: Parallel Distributed Perception and Performance*, 1999.