Emergent Rhythms Through Multi-agency in Max/MSP

Arne Eigenfeldt

School for the Contemporary Arts
Simon Fraser University
Burnaby, BC
Canada
arne_e@sfu.ca

Abstract. This paper presents a multiple-agent architecture created in Max/MSP that generates polyphonic rhythmic patterns which continuously evolve and develop in a musically intelligent manner. Agent-based software offers a new method for real-time composition that allows for complex interactions between individual voices while requiring very little user interaction or supervision. The system described, *Kinetic Engine* is an environment in which networked computers, using individual software agents, emulate drummers improvising within a percussion ensemble. Player agents assume roles and personalities within the ensemble, and communicate with one another to create complex rhythmic interactions. The software has been premiered in a recent work, *Drum Circle*, which is briefly described.

Keywords: Multi-agents, evolutionary rhythm, interactive systems

1 Introduction

The promise of agent-based composition in musical real-time interactive systems has already been suggested [13], [17], specifically in their potential for emulating human-performer interaction. Agents have been defined as autonomous, social, reactive, and proactive [16], similar attributes required of performers in improvisation ensembles.

Kinetic Engine [6], created in Max/MSP, arose out of a desire to move away from constrained random choices within real-time interactive software, and utilize more musically intelligent decision-making processes. Agents are used to create complex, polyphonic rhythms that evolve over time, similar to how actual drummers might improvise in response to one another. A conductor agent loosely co-ordinates the player agents, and manages the high-level performance parameters, specifically *density*: the number of notes played by all agents.

The software is written by a composer with compositional, rather than research, objectives, and is the first stage in a long-term investigation of encoding musical knowledge in software. As such, the encoded knowledge is my own; my experience as a composer suggests that I have some knowledge as to what determines interesting music, so I am relying upon that knowledge. No attempt has been made to create a comprehensive compositional system that can reproduce specific styles or genres; the system is rule-based, rather than data-driven, and the rules and logic within *Kinetic Engine* are derived from auto-ethnographic examination.

This paper will describe the implementation of multi-agents in *Kinetic Engine*. Section 2 gives an overview of existing research into multi-agent systems and rhythm generation. Section 3 describes the specific implementation of agents. Section 4 describes how agents activate themselves. Section 5 discusses how rhythms are generated and checked. Section 6 describes the social behaviour of agents. Section 5 describes how messaging between agents operates. Section 8 describes how agents learn and evolve. Section 9 offers conclusions and future directions.

2 Overview of Existing Research

2.1 Multi-agent systems

Multiple-agent architectures have been used to track beats within acoustic signals [5], [9] in which agents operate in parallel to explore alternative solutions. Agents have also been used in real-time composition: Burtner [3] created a multi-agent, multi-performer system; Dahlstedt and McBurney [4] developed a multi-agent model based upon Dahlstedt's reflections on his own compositional processes; Wulfhurst et.al. created a multi-agent system where software agents employ beat-tracking algorithms to match their pulse to that of human performers.

Many of these systems incorporate improvisatory elements. As already noted, agents seem to suggest the same sorts of specifications required of human improvisers. Benson suggests that there are many shades of improvisation in music, ranging from standard performance – in which musicians fill in certain details which are not specified by the score – to complete melodic and harmonic freedom; as such, the role agents could play in such works is widely varying.

Murray-Rust and Smaill [13] create a theory of Musical Acts, an expansion of Speech Act Theory, to describe the actions of musicians (represented as agents) engaged in improvisatory ensemble playing. However, the authors are interested in creating a system that will "enable a wider range of people to create music," provide a "new approach to musical composition," and facilitate "the interaction of geographically diverse musicians," none of which are motivating forces behind *Kinetic Engine*.

2.2 Rhythm Generation

The generation of rhythm through software processes has been explored through a variety of methods, including genetic algorithms [10], cellular automata [2], neural networks [11] and multi-agents [8]. Brown suggests that CA provides “a great deal of complexity and interest from quite a simple initial setup”; while this may be the case, he also comments that his generated rhythms “often result in a lack of pulse or metre. While this might be intellectually fascinating, it is only occasionally successful from the perspective of a common aesthetic.” He concludes that musical knowledge is required within the rule representation system in order for the system to be *musically* successful.

Gimenes explores a memetic approach that creates stylistic learning methods for rhythm generation. *RGeme* “generates rhythm streams and serves as a tool to observe how different rhythm styles can originate and evolve in an artificial society of software agents.” Using an algorithm devised by Martins et.al. for comparing similar rhythms, agents choose rhythmic memes from existing compositions and generate new streams. The highest scoring memes, however, proved to be of questionable rhythmic interest.[1]

Pachet [14] proposes an evolutionary approach for modelling musical rhythm. Agents are given an initial rhythm and a set of transformation rules from a shared rule library; the resulting rhythm is “the result of ongoing play between these co-evolving agents.” The agents do not actually communicate, and the rules are extremely simple: i.e. add a random note, remove a random note, move a random note. The system is more of a proof of concept than a performance tool; seemingly, it developed into the much more powerful *Continuator* [15], which is a real-time stylistic analyzer and variation generator.

Finally, Miranda [12] describes an unnamed rhythm generator in which agents produce rhythms that are played back and forth between agents. Successful rhythms (those that are played back correctly) are stored, and unsuccessful ones are eventually deleted, while rhythms that are too close to each other are merged by means of a quantiser mechanism. A repertoire of rhythms eventually emerges, which Miranda suggests is a cultural agreement between agents. This suggests an interesting possibility for evaluating rhythms outside of a database.

3 Agents in Kinetic Engine

Agent-based systems allow for limited user interaction or supervision. While this may seem like a limitation, this allows for more higher-level decisions to be made within

[1] The two highest scoring memes were [11111111] and [01111111], where 1 is a note, and 0 a rest, in a constant rhythm (i.e. one measure of eighth notes).

software. This models interactions between intelligent improvising musicians, with a conductor shaping and influencing the music, rather than specifying what each musician/agent plays.

Kinetic Engine can run as a distributed network, in which each computer operates as a separate agent, or internally within a single computer. *Drum Circle*, an installation/performance using *Kinetic Engine*, was premiered with one central computer operating as a conductor agent, and nine networked computers operating as player agents.

In *Kinetic Engine* v.2, there are two agent classes: a conductor and an indefinite number of players.

3.1 The Conductor Agent

The conductor agent (hereafter simply referred to as "the conductor") has three main functions: firstly, to handle user interaction; secondly, to manage (some) high-level organization; thirdly, to send a global pulse.

Kinetic Engine is essentially a generative system, with user interaction being limited to controlling *density* – the relative number of notes played by all agents. This value can be set directly via a graphic slider or an external controller. The user can also influence the system by scaling agent parameters (see section 3.2).

Metre, tempo, and subdivision are set prior to performance by the conductor; these values remain constant for the duration of a *composition*. The user can force a new composition, which involves new choices for these values. Each of these values is dependent upon previous choices using methods of fuzzy logic; for example, if the first tempo was 120 BPM, the next cannot be 116, 120, or 126 (which would be deemed to be "too close" to be considered new). If a subsequent tempo is considered "close" to the previous (i.e. 108/112 or 132/138), then the *next* tempo would have to be significantly different.

The conductor also manages the initialization routine, in which agents register and are assigned unique IDs. A more truly evolutionary model eventually could be used, in which agents are created and destroyed during the performance, modeling the notion of musicians entering and leaving the ensemble.

The conductor also sends a global pulse, to which all player agents synchronize.

3.2 The Player Agents

Player agents are instances of a single Max patcher running on separate machines. Upon initialization, agents "report in" to the conductor with their instance number, and are assigned a unique ID, which is stored in the agent as a local value (see figure 1).

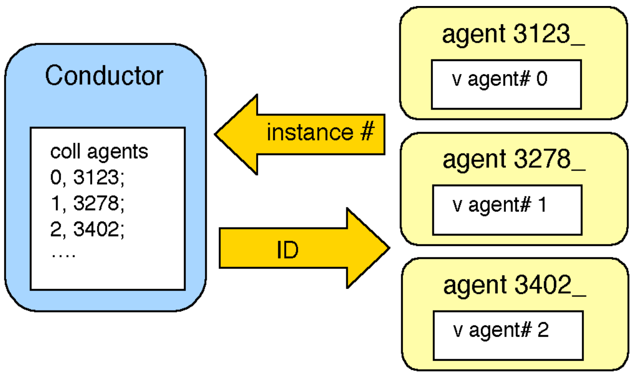


Fig. 1. Initialization of agents.

The conductor also counts the number of agents online, and sends this information out: agents adjust their internal data arrays accordingly.

Upon initialization, player agents (hereafter referred to simply as “agents”) also read a file from disk that determines several important aspects about their behaviour; namely their *type* and their *personality*.

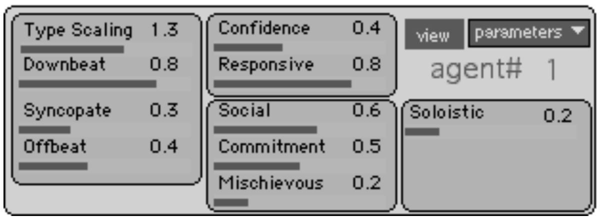
Type can be loosely associated with the instrument an agent plays, and the role such an instrument would have within the ensemble. See Table 1 for a description of how type influences behavior.

Table 1. Agent *types* and their influence upon agent behaviour.

	Type *Low*	Type *Mid*	Type *High*
Timbre	low frequency: • bass drums	midrange frequency: • most drums	high frequency: • rattles, • shakers, • cymbals
Density	lower than average	average	higher than average
Variation	less often	average	more often

The stored personality traits include *Downbeat* (preference given to notes on the first beat), *Offbeat* (propensity for playing off the beat), *Syncopation* (at the subdivision level), *Confidence* (number of notes with which to enter), *Responsiveness* (how responsive an agent is to global parameter changes), *Social* (how willing an agent is to interact with other agents), *Commitment* (how long an agent will engage in a social interaction), and *Mischievous* (how willing an agent is to upset a stable system). A further personality trait is *Type-scaling*, which allows for agents to be less restricted to their specific types[2]. See figure 2 for a display of all personality parameters.

[2] For example, low agents will tend to have lower densities than other types, but a low agent with a high type-scaling will have higher than usual densities for its type.



Type Scaling 1.3
Downbeat 0.8
Syncopate 0.3
Offbeat 0.4
Confidence 0.4
Responsive 0.8
Social 0.6
Commitment 0.5
Mischievous 0.2
Soloistic 0.2

Fig. 2. Personality parameters for a player agent.

4 Agent activation

A performance begins once the conductor starts "beating time" by sending out pulses on each beat. Agents independently decide when to activate themselves by using fuzzy logic to "wait a bit". Once these pulses begin, agents do not respond immediately, nor synchronously; instead, agents react every few beats – a *checkbeat* - using such a fuzzy counter. Each beat is tested[3], and agents wait on average between 3 and 6 beats before passing a checkbeat. This amount is scaled by the agent's responsiveness parameter, as well as the overall system responsiveness; less responsive agents will take longer to react to the conductor's demands (see Figure 3 to see how probabilities increase with each test).

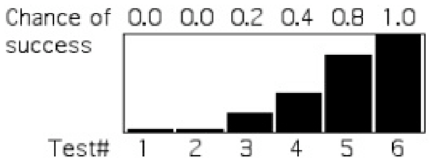


Fig. 3. Using fuzzy logic to "wait a bit" by controlling chance of success for each test.

The system responsiveness parameter scales the test number, whereas the agent responsiveness parameter scales the chance of success. The former thus controls how quickly success is possible - allowing for all agents to react immediately - whereas the latter controls how soon success is reached after the initial wait of three beats.

When an agent becomes active, it determines its density.

[3] A random value between 0.0 ad 1.0 is generated, and compared to an increasing "chance of success" parameter in a Boolean test.

4.1 Fuzzy Logic Ratings

Kinetic Engine attempts to model human approximation through the use of fuzzy logic to judge success. In the case of density, agents are unaware of the exact global density required. Instead, the conductor rates the global density as “very low”, “low”, “medium”, or “high” and broadcasts this rating.

Agents know the average number of notes in a pattern based upon this rating, which is scaled by the agent’s type and type-scaling parameter. Agents generate individual densities after applying a Gaussian-type curve to this number (see Figure 4 for the Gaussian curve in Max’s **table** object), and broadcast their density.

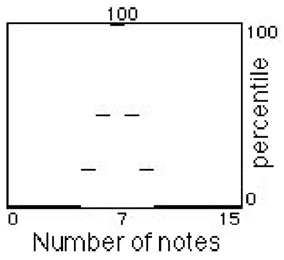


Fig. 4. A Gaussian curve in Max’s **table** object,

The conductor collects all agent densities, and determines whether the accumulated densities are “way too low/high”, “too low/high”, or “close enough” in comparison to the global density, and broadcasts this success rating.

• if the accumulated density is “way too low”, non-active agents can activate themselves and generate new densities (or conversely, active agents can deactivate if the density is “way to high”).

• if the accumulated density is “too low”, active agents can add notes (or subtract them if the density is “too high”).

• if the accumulated density is judged to be “close enough”, agent densities are considered stable.

5 Generating Rhythms

5.1 Density Spread

An agent’s density is spread across the available beats using fuzzy logic to determine probabilities, influenced by the agent’s downbeat and offbeat parameters (see Figure 5 for an example of probability weightings spread across four beats).

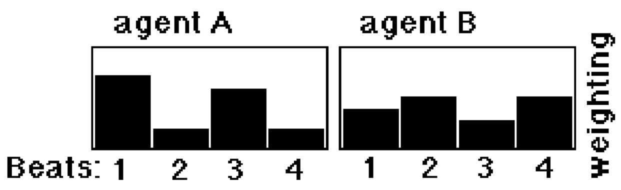


Fig. 5. Example density spread weightings for two agents, 4/4 time with different downbeat and offbeat parameter values.

Agents determine the placement of the notes within the beat using a similar technique, but influenced by the agent's syncopation parameter.

Notes are represented using simple binary values over a grid of potential notes within a measure, given the time signature and subdivision. Thus, in 2/4 metre with a subdivision of 4, (0 0 0 0 0 0 0 0) represents an empty measure, whereas (1 0 0 0 1 0 0 0) represents a quarter note on each beat (durations are not represented).

5.2 Pattern Checking

After an initial placement of notes within a pattern has been accomplished, *pattern checking* commences. Each beat is evaluated against its predecessor and compared to a set of rules in order to avoid certain patterns and encourage others.

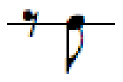
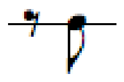
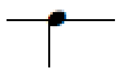
Previous beat	Pattern A	Pattern B
		
	30%	90%

Fig. 6. Example pattern check: given a previous beat's rhythm, with one note required for the current beat, two "preferred" patterns for the current beat.

In the above example, pattern A is tested first, and there is a .3 percentile chance that this pattern will result. Failing that, pattern B is tested, and there is then a .9 percentile chance that this pattern will result. If this last test fails, the original rhythm is allowed to remain.

6 Social Behaviour

Once all agents have achieved a stable density and have generated rhythmic patterns based upon this density, agents can begin social interactions. These interactions involve potentially endless alterations of agent patterns in relation to other agents; these interactions continue as long as the agents have a *social bond*, which is broken when testing an agent's social commitment parameter fails[4].

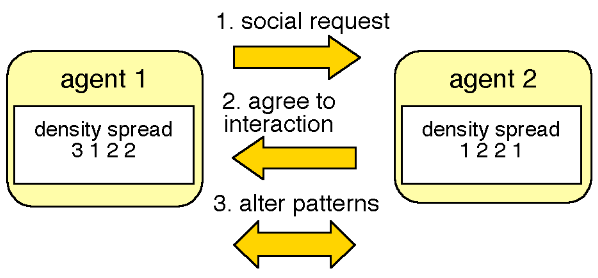
Social interaction emulates how musicians within an improvising ensemble listen to one another, make eye contact, then interact by adjusting and altering their own rhythmic pattern in various ways. In order to determine which agent to interact with, agents evaluate[5] other agent's *density spreads* - an agent's density distributed over the number of beats available, given the composition's metre.

Table 2. Example density spreads in 4/4: comparing agent 1 with agents 2 and 3.

Agent #	1	2	3
Density Spread	3 1 2 2	1 2 2 1	2 3 3 3
Similarity rating		0.53	0.48
Dissimilarity rating		0.42	0.33

An agent generates a *similarity* and *dissimilarity* rating between its density spread and that of every other active agent. The highest overall rating will determine the type of interaction[6]: a dissimilarity rating results in rhythmic polyphony (interlocking), while a similarity rating results in rhythmic heterophony (expansion).

Once another agent has been selected for social interaction, the agent attempts to "make eye contact" by messaging that agent. If the other agent does not acknowledge the message (its own social parameter may not be very high), the social bond fails, and the agent will look for other agents with which to interact.



[4] This test is done every "once in a while", another fuzzy counter.

[5] Evaluation methods include comparing density spread averages and weighted means, both of which are fuzzy tests.

[6] Interlocking interactions (dissimilarities) are actually encouraged through weightings.

Fig. 7. Social messaging between agents

6.1 Interaction types: Polyphonic

In polyphonic interaction, agents attempt to “avoid” partner notes, both at the beat and pattern level. For example, given a density spread of (3 1 2 2) and a partner spread of (1 2 2 1), both agents would attempt to move their notes to where their partner’s rests occur[7] (see Figure 8).

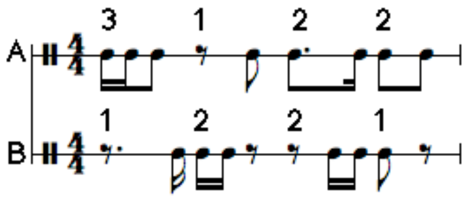


Fig. 8. Example polyphonic interaction between agents A and B, with density spreads of (3 1 2 2) and (1 2 2 1). Note that not all notes need to successfully avoid one another (beats 3 and 4).

6.2 Interaction types: Heterophonic

In heterophonic interaction, agents alter their own density spread to more closely resemble that of their partner, but no attempt is made to match the actual note patterns (see Figure 9).

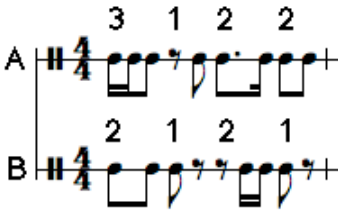


Fig. 9. Example heterophonic interaction between agents A and B, with density spreads of (3 1 2 2) and (2 1 2 1). Agent B had an initial spread of (1 2 2 1).

7 Messaging

Through the use of instances in Max, agents hide their data within local values and colls, two of Max’s data objects (see Figure 10).

[7] Because both agents are continually adjusting their patterns, stability is actually difficult to achieve.

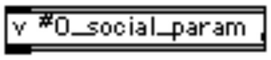


Fig. 10. The value object as a local variable.

Certain variables are required to be global: they can be accessed by any agent, but are only altered by the conductor agent (see Figure 11). When the conductor alters a global variable, it broadcasts this to the network, and agents update their internal values.

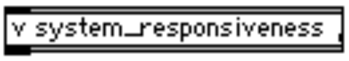


Fig. 11. A global variable.

Data that is shared between agents – i.e. an agent's note density – is stored as an array within every agent. Each time an agent alters its internal value, it broadcasts it to the network (see Figure 12).

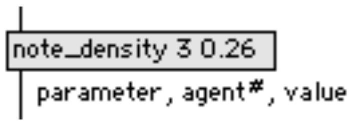


Fig. 12. Broadcasting new values to the network.

Agents receive the new value(s), and store them in their own arrays, using the agent number as an index.

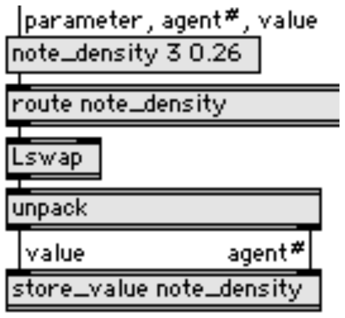


Fig. 13. Storing data from other agents.

Example data-handling abstractions, such as "store_value" in Figure 13, are given in [Eigenfeldt 2007].

8 Evolution of Agents

Agents adapt and evolve their personalities over several performances, and within the performance itself. After each composition (within the performance), agents evaluate their operation in comparison to their personality parameters. For example, an agent that was particularly active (which relates to both the responsiveness and confidence parameters) during one composition, might decide to "take a rest" for the next composition by temporarily lowering these parameters.

Agents also judge their accumulated behaviours over all compositions in a performance in relation to their preferred behaviour (as initially read from disk), and make adjustments in an attempt to "average out" to the latter. At the end of the performance (of several compositions), the user can decide whether to evolve from that performance. Comparing the original parameter with the final accumulated history, an exponential probability curve is generated between the two values, and a new personality parameter – close to the original, but influenced by the past performance – is chosen and written to disk, to be used next performance.

9 Conclusion and future work

This paper presented methods of using multi-agents within Max/MSP to create complex polyphonic rhythmic interactions that evolve in unpredictable, yet musically intelligent ways.

The software has already been premiered in the performance piece *Drum Circle*, demonstrating the potential for Max/MSP to create music that explores "groove-based" rhythm through such complex methods, music that can be described as displaying emergent properties.

There are several planned strategies for improving the machine musicianship of *Kinetic Engine*, including the use of a dynamic rule base to avoid a homogeneity of rhythms, the ability to incorporate predefined (scored) ideas, and the ability to interact with human performers.

Example music created by *Kinetic Engine* is available at www.sfu.ca/~eigenfel/research.html. The software is also available at this URL as applications (Max OSX only).

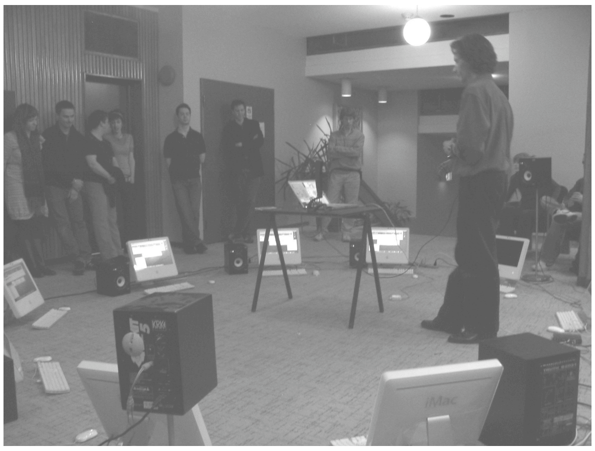


Fig. 14. The premiere of *Drum Circle* in Vancouver, April 2007, with the author controlling the system using a P5 Dataglove (photo R. Bader).

10 REFERENCES

1. Benson, B. E.: The Improvisation of Musical Dialogue. Cambridge University Press (2003)
2. Brown, A.: Exploring Rhythmic Automata. In: Applications On Evolutionary Computing. vol. 3449, pp. 551--556 (2005)
3. Burtner, M.: Perturbation Techniques for Multi-Agent and Multi-Performer Interactive Musical Interfaces. In: NIME 2006. Paris, France (2006)
4. Dahlstedt, P., McBurney, P. : Musical agents. In: Leonardo. vol. 39, no. 5, pp. 469--470 (2006)
5. Dixon, S.: A lightweight multi-agent musical beat tracking system. In: Pacific Rim International Conference on Artificial Intelligence. pp. 778–-788 (2000)
6. Eigenfeldt, A.: Kinetic Engine: Toward an Intelligent Improvising Instrument. In: Proceedings of the 2006 Sound and Music Computing Conference. Marseilles, France (2006)
7. Eigenfeldt, A.: Managing Complex Patches in Max, www.cycling74.com/story/2007/2/5/142639/8843 (2007)
8. Gimenes, M., Miranda, E.R., Johnson, C.: Towards an intelligent rhythmic generator based on given examples: a memetic approach. In: Digital Music Research Network Summer Conference (2005)
9. Goto, M., Muraoka, Y.: Beat Tracking based on Multiple-agent Architecture - A Real-time Beat Tracking System for Audio Signals. In: Proceedings of The Second International Conference on Multi-agent Systems. pp. 103--110 (1996)
10. Horowitz, D.: Generating rhythms with genetic algorithms. In: Proceedings of the International Computer Music Conference. Aarhus, Denmark (1994)
11. Martins, J., Miranda, E.R.: A Connectionist Architecture for the Evolution of Rhythms. In: Proceedings of EvoWorkshops 2006. Lecture Notes in Computer Science, Berlin: Springer-Verlag, Budapest (2006)
12. Miranda, E.R.: On the Music of Emergent Behaviour. What can Evolutionary Computation bring to the Musician? In: Leonardo. vol. 6 no. 1 (2003)

13. Murray-Rust, D., Smaill, A.: “MAMA: An architecture for interactive musical agents. In: Frontiers in Artificial Intelligence and Applications. vol. 141, ECAI 2006, 17th European Conference on Artificial Intelligence (2006)
14. Pachet, F.: Rhythms as emerging structures. In: Proceedings of the 2000 International Computer Music Conference. Berlin, ICMA (2000)
15. Pachet, F.: The Continuator: Musical Interaction With Style. In: Journal of New Music Research. vol. 32, no. 3 pp. 333--341 (2003)
16. Woolridge, M., Jennings, N. R.: Intelligent agents: theory and practice. In: Knowledge Engineering Review. vol. 10 no. 2 pp. 115--152 (1995)
17. Wulfhorst, R.D., Flores, L.V., Flores, L.N., Alvares, L.O., Vicari, R.M.: A multi-agent approach for musical interactive systems. In: Proceedings of the second international joint conference on Autonomous agents and multi-agent systems. pp. 584–-591 (2003)