The Internet of Machines - Technological Synergy and Computer Music

Eugene Cherny
Saint-Petersburg State University of
Information Technologies, Mechanics and Optics
Saint-Petersburg, Russia
eugene.cherny@niuitmo.ru

Gleb Rogozinsky
The Bonch-Bruevich Saint-Petersburg
State University of Telecommunications
Saint-Petersburg, Russia
gleb.rogozinsky@gmail.com

***Abstract*—The Internet of Machines radically changes human culture. The upcoming shapes of post-NGN world philosophy inspire the ultra modern approaches to the creativity phenomenon. The novel forms of art exploit the new possibilities of network environments, collaborating with human beings and without them. The paper describes the model of intellectual system for computer music generation based on a network of autonomous computational agents. The proposed model uses OWL ontology for representing knowledge of the agents in network.**

I. Introduction

The rise of the Internet of Things (IoT) due to an increasing number of online devices, as well as the development of inter-machine communication provides a new ideological basement for the new forms of collaborative computer art. Such systems should create original art forms by the means of interaction between number of participants, where human beings go together with intellectual agents.

During the last decade informational telecommunications has become "the technology of technologies", rapidly transforming the face of modern life. Nowadays such changes have infiltrated almost every part of human activity. They are rapidly evolving from fundamental technological basements of present life towards the new forms of art [1]. The latter in turn have always been bounded with technologies. The result of application of computers for musical composition in the 50's gave birth to the discovery of the so-called computer music and algorithmic composition [2], where composer becomes a programmer of the machine which calculates the music score according to the preliminary set of rules and algorithms. The discussed approach could hardly be applied to the traditional music, i.e. song writing, but it is useful for creating long-scale soundscapes, sound design and experimental art. The algorithmic composition system achieves a new plane of possibilities if it becomes distributed. The modern world of the Internet of Things provides an excellent base for experiments in this field of art.

The IoT world features the great number of heterogeneous devices, working together in the same network [3]. Due to heterogeneity of connected devices IoT implementation should provide interoperability functions, such as those based on the universal method for data representation and exchange. This method should allow to integrate the data produced in the network. One of the most promising technologies for implementing these functions is the Semantic Web Stack [4], which includes a universal data model (Resource Description Framework, RDF) with the basic vocabulary of terms (RDF Schema, Web Ontology Language – OWL) for high-level concepts description.

The process of creativity automation (generative art, etc.) has always been on the edge of computer music research, where computational methods were used to formalize parts of the creative process. Such generative systems often include the compositional model: a set of algorithms, implementing specific rules, e.g. the rules describing the melodies, harmonic structures, rhythmic patterns, etc. Thus, the compositional model represents semantics of the musical piece by defining a set of the music-related facts (rules) and links between them (how rules are used). Such models are static by their design and could not be applied to the creativity simulation due to its dynamic nature. For instance, the person's taste changes over time, as well as their creative activities. In the case of music such changes affect the musical composition paradigm the person exploits. Thus, the compositional model should be able to change itself to modify the musical output, adding or removing facts in the model and changing the links between them.

Distribution and heterogeneity of the IoT imply that there is no single entity to possess the all available knowledge, thus every entity is potentially capable for self-development by obtaining the new knowledge. The possibilities of IoT-based creativity model are limited only by the diversity of the real world processes.

The IoT will radically change the face of the global network [5]. Together with the evolution of traditional services the novel ones appear, characterized by fundamentally different relations between human and machine. One of the most important components of the network traffic is the multimedia. Thus, the research of the new IoT-based multimedia services becomes significant. Authors explore the possibilities of IoT technologies for creation of new IoT-based multimedia solutions. In this paper we propose the system for computer music generation, based on the multi-agent architecture and OWL, as an example of the new service.

II. Related work

A. Electric Sheep

One of the most remarkable and self-explaining examples of the distributed systems of collective art is a popular screen-

saver program for laptops and mobile devices called Electric Sheep [6], which could be depicted as an abstract distributed art piece, generated by thousands of computers and mobile devices across the world web. The program is based on genetic algorithm, which controls the mutation of initial fractal-like visual primitives. In the beginning of its work Electric Sheep downloads a set of initial visuals from the remote server, which are used further for obtaining new visuals. Any user could vote for the visuals he/she likes, gaining the current offspring's chances for the new cycle of genetic selection or exclude it from the evolution process. The offsprings of initial visual set travel from computer to computer, interbreeding and creating new combinations of visuals.

B. Experiments in Musical Intelligence

Similar approach was explored by David Cope in his work Experiments in Musical Intelligence (EMI) [7]. The new music is generated according to the results of previously analyzed compositions. The proceeds of composing by itself consists of the three parts: deconstruction (that is analysis and segmentation), authors' "sign" analysis (that defines style) and combination of all fragments together into one piece. This model describes not only the sequences of notes and chords, but also provides common form of piece and its structure in general. Using this approach the result of the work is always depends on compositions to be analyzed (the training set).

The scaling of EMI in the multi-agent domain is possible, if the context of agent is thought as the pre-existing base of source musical material.

The EMIs main flaw is its demand for exact music scores which in some form should exist in the database, providing "ideas" for the model. It disallows to perform the intellectual selection of material for composition, obstructing the usage of algorithms for automated combining of pieces by their features. On the other hand, the knowledge representation using ontologies could provide an easy solution to apply the automated aggregating of composed material according to AI decisions.

C. The Listening Machine

The Listening Machine project (http://thelisteningmachine.org/) analyzed activity of 500 Twitter users in UK. With the use of machine learning and natural language processing, the system analyzed tweets and generated music in real-time. The project remained active for nine months from May 2012 to January 2013, continuously creating algorithmic music and broadcasting it through the Internet.

D. #tweetscapes

Another project worth mentioning here is #tweetscapes (http://heavylistening.com/tweetscapes/). It converts tweets of German users according to their content into sounds and visuals. The algorithm uses as input such parameters as user name, geoposition, hash-tags and other information are used. As the result, each tweet creates sound in stereo, accompanied by specific visualization.

III. Idea

We propose a model for automated music generation, based on distributed network of autonomous agents. In the discussed model each agent is able to carry some knowledge or state of compositional model, that is used for music generation. Every agent is able to uphold some number of connections with others. Under the *context* we term the aggregate of all connections of agent. The context one way or another affects the internal state of agent, changing its compositional model used to generate music. Thus the process of music material generation becomes the product of analysis and processing of existing musical pieces. The knowledge extracted from the analysis stage is used to create a score for the sound synthesizers. Each agent could be understood as a single sound synthesis entity in the global orchestra of machines, making music through the interchanging of ideas and sound samples across the network.

We using the *Web Ontology Language* for knowledge representation and *Csound audio programming language* [8] for sound synthesis.

IV. Implementation overview

In the present state of research the system possesses simplified design. To gather information from surround world the system uses a set of agents, which is capable of interact with the user. Currently, we have two implementations of the user interaction: an on-screen GUI with XY-controller and a video tracking.

Each agent in the system consists of the following modules:

- *UI* gathers information from the sensors (XY controller and video) and processes it to provide high-level characteristics of the interaction (quantity of motion, etc.).
- *Network Core* provides a set of abstractions over OS network APIs. This module allows to discover agents in the network and to perform SPARQL-requests to the agents for getting their state.
- *Knowledge Base* holds all information about current agent and its neighbors. It is used by the Network Core for maintaining knowledge about neighbors and by the Algorithmic Core to set the parameters for compositional model.
- *Decision Core* is responsible for implementing the business logic of the applications. It provides communication between modules.
- *Algorithmic Core* implements compositional model. It utilizes knowledge taken from the Knowledge Base for configuring the compositional model and generates a set of commands for Csound Core.
- *Csound Core* is responsible for sound synthesis. It receives commands from the Algorithmic Core and triggers the sound events.

The architecture of agent is presented on the Fig. 1.

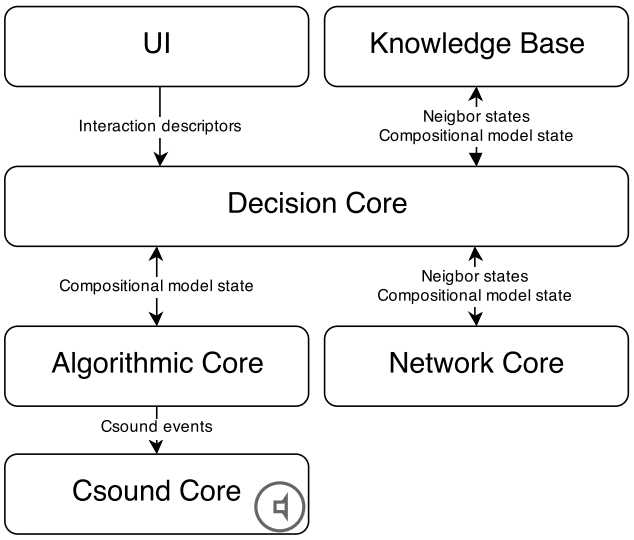

Fig. 1. Architecture of the agent

V. User interactions

Human interaction capabilities, as we will see further, provides the way to create diverse variations of soundscapes by augmenting strict algorithms of the program with human's unpredictable actions. The system will try to stabilize itself when no interaction happened.

Current implementation has two mechanisms for gathering information from the real world: XY-controller on the device screen and video tracking. This is due to availability of sensors which depends on the exact device features: it is very rarely when laptops have multi-touch display, but smart-phones almost always have one. We decided to use web-camera video tracking on laptops and XY- controller on smart-phones for user interaction to provide integration of different devices to the single network application.

UI was module designed such way, that the actual sensors can be interchangeable in the application. This was achieved by delegation of interaction analysis functions to this module. The software can be enabled to use video camera or multi-touch sensor screen with the configuration script.

A. Interaction analysis

Every UI module provides two interaction descriptors: *average activity* and *variance*. The former gives us the idea of how intensive actions are, and its implementation depends on the type of sensor module using. The latter characterizes variation of actions' intensity and implemented as a statistical variance estimator.

Activity in the video tracking UI module is calculated using traditional technique by thresholding pixel-by-pixel difference of two successive image frames:

$$A_i = \frac{\sum_{x=0}^{w-1}\sum_{y=0}^{h-1} thresh(P_i(x,y) - P_{i-1}(x,y))}{w \cdot h},$$

where i — index of current frame in the video stream;
$P_i(x,y)$ — pixel value (integer in range $0 \dots 255$) of i-th frame at coordinate x, y;
$thresh()$ — function, which outputs 1 if input value is more than a threshold parameter or 0 otherwise;
w, h — width and height of incoming video in pixels respectively.

Division by $w \cdot h$ is made to normalize parameter A to the $0 \dots 1$ range.

To lower the CPU usage we grab the video with 320×240 resolution at 20 FPS rate.

XY-controller position is captured with the 50 Hz frequency. The activity in its case is the length of the displacement of controller position in two successive position samples with normalized coordinates:

$$A_i = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2},$$

where i is the index of the current sample and x, y are the coordinates of XY-controller position mapped to the $0 \dots 1$ range.

To calculate the *average activity* and *variance* parameters, the UI organizes activity values in the array of size 32 and then the following calculations are applied:

$$\mu_A = \frac{\sum_{i=0}^{31} A_i}{32};$$

$$\sigma_A^2 = \frac{1}{32}\sum_{i=0}^{31}(A_i - \mu_A)^2,$$

where μ_A — average activity, A_i — activity parameter, stored in the array at index i, σ_A^2 — activity variance.

This calculations are performed every time UI module collects 16 new activity samples from the input.

The analysis results are sent to the *Decision Core* as a pair, defined as $UI_D(\mu_A, \sigma_A^2)$.

B. Contact event

Besides the UI_D event, UI module also sends to the Decision Core the $UI_C(true|false)$ (contact) event. It represents the start and the end of interaction process with the user. When user touches XY-controller or moves inside the camera visibility zone, UI module sends $UI_C(true)$ message. $UI_C(false)$ is sent when no interaction happens in 10 seconds.

C. Video tracking implementation details

Currently, the video tracking is implemented as Max/MSP [9] patch, and tracking data is sent to the main application in a form of OSC [10] message via UDP. Max/MSP is the visual programming language for creating real-time audio-visual applications. We used it for rapid prototyping of the minor features, such as video analysis.

VI. SOUND GENERATION

For the sound generation and processing purposes the proposed system uses Csound, a high-level computer music programming language. It provides great flexibility for implementing custom sound synthesis and processing modules. It also has a Java API, which is useful for controlling the sound synthesis process from the compositional model. Csound is a cross-platform solution and aside from Windows, Mac OS X and GNU/Linux, it supports Android and iOS. Also, Csound is able to work with Raspberry Pi, BeagleBone and Arduino platforms. Csound was originally developed by Barry Vercoe in 1985 at MIT Media Lab. Since the 90's, it has been developed by a group of core developers, and a wider community of volunteers which contributes examples, documentation, articles, and takes part of the Csound development with bug reports, feature requests and discussions with the core development team.

Although Csound has a strong tradition as a tool for composing electro-acoustic pieces, it is used by composers and musicians for any kind of music that can be made with the help of the computer. Csound can also be used in real-time and interactive contexts, on mobile devices or in combination with the other programming languages.

Csound's working principle is based on the modular structure. It means that the sound synthesis algorithms and sound processing techniques are implemented through connecting multiple primitives, e.g. sound generators, adders, filters etc. During its work some modules could be substituted for others, so sound synthesis algorithms could be altered, in particular as a result of collaborative decision between several agents. Typical Csound code consists of two parts: an *orchestra*, where all instrument structures are defined, and a *score*, which provides event triggering data for instruments defined in the orchestra.

A. Synthesizer structure

Our main instrument in the present implementation of sound synthesis part is an ambient drone generator. We use it to create wide, ambient, reverberating soundscapes with the multiple variations in timbre.

The instrument consists of number of voices, which differ from each other only by parameters of the elements, and the structure of each voice remains the same. The voice structure is given on Fig 2. It consists of white noise generator (WG), simple two-stage envelope generator (EG), low frequency oscillator (LFO) and a 12 dB/octave band-pass filter (BP). The mix of all voices is fed through delay (DEL) and reverberation unit (REV). The audio signal path is marked by continuous line, while the control signal line is dotted.

Envelope generators of each voice are randomly retriggered, creating the sense of rain drops when attack and release parameters are both fast, and evolving layers of sound otherwise. The output of envelope generator is also fed to control the filter's frequency. The central frequencies of all filters are tuned to cover the whole spectral range. By default, they are spread harmonically, although some fine adjustments are possible.

The bandwidth of band-pass filters could also change the timbre of sound. When the band width is narrow, the sound is close to sine wave. Since the filter frequencies are modulated, it is possible to achieve vibrating sound structures. LFO unit provides another way to periodically alter the frequencies of filters. The effect part makes the mix wider and denser, adding the sense of space and distance.

All described parameters of the instrument could be organized in an array of parameters, which could be altered during the working cycle of the system.

VII. SCORE GENERATION

Algorithmic Core implements compositional model — the set of algorithms, responsible for musical events generation. Before moving to the implementation we will describe some basic concepts behind it.

- The chosen synthesis method, described in the previous section, imposes certain restrictions on the compositional model. Particularly, the synthesizer can not be controlled with note events. The continuous control mechanism should be implementing for smooth control of the drone-synthesizer parameters.
- The compositional model should use the interaction between several agents to control the sound synthesis parameters.
- User interaction represents the concept of external knowledge, that network of agents receives from the real world, and which it will try to preserve. Every agent will try to share this new knowledge to its neighbors, but every time the knowledge is shared, it reliability decreases until some agent will simply ignore this knowledge with low reliability.

A. Implementation

We introduce following notation for describing the compositional model implementation:

- $P = Pitch, FM, D$ — the set of synthesizer parameters (pitch, frequency modulation coefficient and density respectively).
- P_0 — synthesizer parameters of the current node.
- P_n — mean synthesizer parameters of all connected nodes.
- K_0 — reliability factor of the current node. This factor shows, how relevant the knowledge that node holds is.
- K_n — mean reliability factor of all connected nodes.
- K_u — novelty coefficient. It shows the distance between reliability of current node and its neighbors.
- UI_C — "event happened" parameter. Can be $true$ or $false$.

The compositional model starts with random parameters:

$$P_0 = \{random(Pitch_{min}, Pitch_{max}), \\ random(FM_{min}, FM_{max}), random(D_{min}, D_{max})\},$$

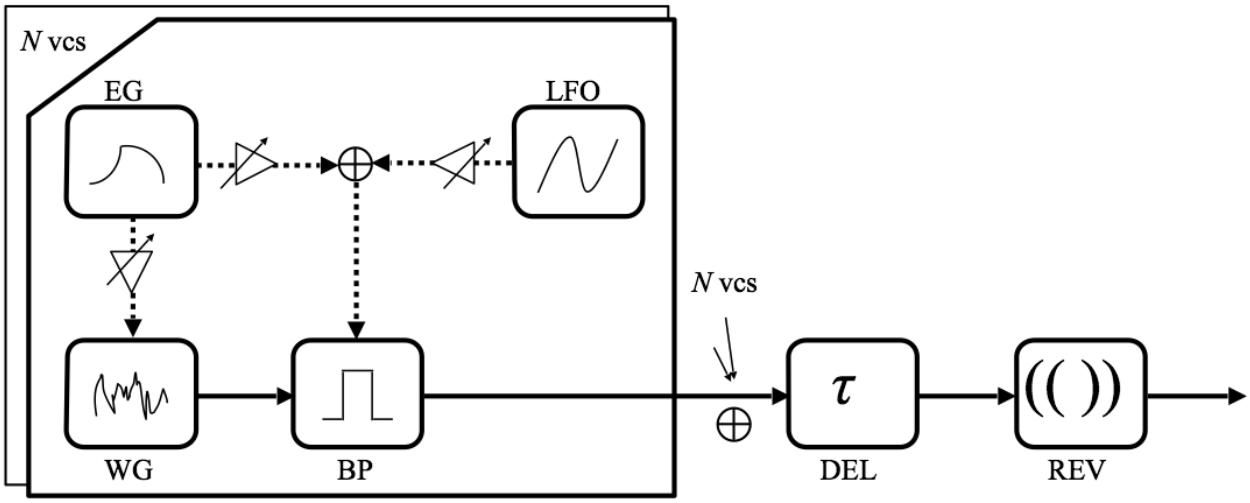

Fig. 2. Synthesizer voice structure

where $random(a, b)$ is the function that generates a new random value between a and b. $Pitch_min$, $Pitch_max$, FM_min, FM_max, D_min, D_max are the minimum and maximum parameter values defined in the configuration file.

After initialization we start the main loop.

First, the parameters analyzed in UI are scaled and used to update the FM and Density parameters of the synthesizer, but only if UI_C is $true$ (interaction happened):

$$FM = scale(\mu_A, FM_{min}, FM_{max}) \mid UI_C = true$$
$$D = scale(\sigma_A^2, D_{min}, D_{max}) \mid UI_C = true$$

If interaction happened, reliability factor of the current node should be 1:

$$K_0 = 1 \mid UI_C = true$$

We need to update the novelty coefficient for the later use. If the *mean reliability factor of connected nodes* (K_n) is bigger than current (K_0), the *novelty coefficient* is defined as difference between K_n and K_0, otherwise zero:

$$K_u = (K_n - K_0, 0 \mid K_n > K_0)$$

Now we are ready to update the current node's synthesizer parameters. If current $K_0 = 0$ we update synthesizer state with the random parameters, otherwise we do it with the result of interpolation between current synthesizer's state and the neighbor states. If reliability of the neighbor's knowledge is less than of current node ($K_u = 0$), we do not modify the parameters of current node. Otherwise, we interpolate between parameters of the current node and its neighbors ($delta$ function).

$$P_0(t+1) = (interp(P_0(t), delta(t), K_u(t)),$$
$$random(P0) \mid K_0(t)! = 0)$$

$interp(a, b, f)$ — interpolation between values a and b with the factor f. If $f = 0$, then this function outputs a; when $f = 1$, the output will be b.

$random(P0)$ generates new random synthesizer parameters.

$delta(t)$ uses weighted neighbors synthesizer and random parameters to generate new ones. It adds some randomness to the mean synthesizer parameters of the neighbors.

$$delta(t) = random(P0) * (1 - K_0(t)) +$$
$$+ (K_n(t), 0 \mid K_n(t) > K_0(t)) * P_n(t)$$

Finally after the update of synthesizer parameters, the K_0 can be updated:

$$K_0(t+1) = (interp(K_0, K_n, K_u), K_0 \mid K_n > K_0)$$

After that the algorithm returns to the beginning of the main loop.

VIII. KNOWLEDGE BASE

Compositional model representation should be based on an unified extensible knowledge model, to be easily understood by other agents in the network. For this case we use Web Ontology Language (OWL) as the base for implementing agent knowledge ontology. The OWL was designed for representing a structural knowledge in the global network. Due to the distributed nature of the discussed system, the OWL seems to be a logical choice for using in such system, because by its design it allows: 1) to describe connections between entities as facts, 2) to distribute knowledge and 3) to describe entity classes.

In the present state of research the system uses rather trivial ontology. Its main aim — to hold the information about state of the compositional model and neighbors. Current ontology contains three classes: *NodeID*, *NodeState*, *SynthState*. *NodeID* holds IP address of the node and *NodeState*. *NodeState* has

links to one or more instances of *NodeID* classes (representing neighbor nodes) reliability factor as float and reference to *SynthState*, associated with node. The latter holds the parameters for synthesizer: base pitch, frequency modulation index and density.

Agents are communicating with each other by SPARQL queries. As each agent has its own triple-store, to avoid URI conflicts we defined the following rules for using URIs (prefix "wm" stands for *http://wintermuse.net/ontology/demo/0.1/*):

1) In every triple-store, instance of the *NodeID* class, representing state of the current agent, has the following URI: *wm:node#this*.
2) *NodeID*, that represents neighbor agent's state has the URI, generated as follows: *wm:node# + IP address*, e.g. *wm:node#10.0.2*.

Due to the first rule agent can always get state of his neighbor with appropriate SPARQL-request as it know the URI of the instance to refer. Second rule ensures that neighbors will never have the same URI.

In the current implementation the agents use Sesame's [11] memory store for storing triples, as we do not need a persistence for our purposes.

IX. Conclusions and future work

The present state of the developed system gives rise to number of different problems and tasks related to different inter-crossing fields of science. The working model by itself creates a framework under which those tasks should be solved. The aesthetic problem is derived from the typical algorithmic composition problematic field, i.e. how to create the music which could be decided as written by the human being or how to create system being able to compose "interesting" music, which could attract people's attention. The solution lays on the thin edge between chaos and determinacy.

The possible applications of the system: adaptive context-aware soundscape generation (e.g. in public spaces), composer's instrument, framework for modeling and exploring the creativity processes.

Together with humanity related problems also arise technical ones. The developed service should be scalable as well as self-organizing and should allow to introduce new functionalities to the agents by sharing knowledge and software modules between them. The base ontology should be developed to be as flexible as possible to facilitate an automated art-related knowledge engineering inside the distributed system. All of these topics require further research. To create a long-lasting, non-stationary, continuously transmogrifying musical compositions and soundscapes the system should be able to acquire data from multiple sources and to map it into new creativity ideas. Several databases (e.g. DBpedia or Europeana) could be used for that purpose, providing data to agents. Also, agents could use user's shared information or sensor data to spread it later as a new data patterns across the swarm of agents. The data could include rhythmic patterns, new timbre, novel composition algorithms and synthesis code modifications, as well as audio fragments of speech and music, combined with other media e.g. photos and texts, to extract ideas form.

References

[1] S. Nora and A. Minc, *Computerization of Society*. MIT Press, 1980.

[2] P. Doornbusch, "Computer sound synthesis in 1951: The music of CSIRAC", *Computer Music Journal*, vol.28.1, 2004, pp. 10-25.

[3] L. Atzori and A. Iera and G. Morabito, "The internet of things: A survey", *Computer networks*, vol.54.15, 2010, pp. 2787-2805.

[4] I. Horrocks et al., "Semantic web architecture: Stack or two towers?", *in Proc. of Principles and practice of semantic web reasoning*, 2005, pp. 37-41.

[5] "Vision and challenges for realising the Internet of Things", European Commission, 2010.

[6] S. Draves, "The electric sheep screen-saver: A case study in aesthetic evolution", *in Proc. of EC'05 Proceedings of the 3rd European conference on Applications of Evolutionary Computing*, 2005, pp. 458-467.

[7] D. Cope, *Computer models of musical creativity*, MIT Press, 2005.

[8] R.C. Boulanger, *The Csound book: perspectives in software synthesis, sound design, signal processing, and programming*, MIT press, 2000.

[9] Cycling' 74 official website, Max visual programming language, Web: http://cycling74.com/products/max/

[10] M. Wright, "Open Sound Control-A New Protocol for Communicationg with Sound Synthesizers", *in Proc. of the 1997 International Computer Music Conference*, 1997, pp. 101-104.

[11] OpenRDF Sesame's official website, Main page, Web: http://www.openrdf.org/