SOUNDSCAPEGENERATOR: SOUNDSCAPE MODELLING AND SIMULATION

Marinos Koutsomichalis
CIRMA/StudiUm - Università di Torino
marinos.koutsomichalis@unito.it

Andrea Valle
CIRMA/StudiUm - Università di Torino
andrea.valle@unito.it

ABSTRACT

This paper describes SoundScapeGenerator, a generic system for modelling and simulating soundscapes both in real and non-real time. SoundsScapeGenerator features algorithms for three-dimensional sound-localisation and is able to model complex spaces. The system relies on abstract rule descriptions (generators) to simulate deterministic, "cartoonified" (loosely modelled) sequencing of sound events. It further supports virtual P.O.H. (point of hearing) and is able to render the final output in various channel configurations. Based on generative algorithms, the SoundScapeGenerator implementation allows real-time user interaction with ever-changing soundscapes, and it can easily communicate with other applications and devices. Finally, we introduce SoundScapeComposer, a higher level module developed for the SoDA project (dedicated to soundscape automatic generation), that has been built on top of SoundScapeGenerator, hiding most of the latter's details to the user.

1. INTRODUCTION

Manual or automatic soundscape generation has been the focus of several projects hitherto. Yet, to our knowledge no system has been implemented that enables sophisticated and full-scale modelling/simulation of complex soundscapes. From our point of view such system should have the following features:

- modelling of complex soundscapes that may consist of an arbitrary number of zones with different geographical, acoustical and sonic characteristics;
- 3-dimensional sound-localisation that takes into account listener's and source's positions, source's size, sound-zone's acoustic features (reverberation, dampening, resonance, etc);
- both deterministic and non-deterministic sequencing and localisation of individual sounds;
- generation of sound events out of atomic sounds and samples in both deterministic and non-deterministic ways;
- modelling of sound events that may move in deterministic or non-deterministic ways in 3D space;
- modelling and simulation of POH (Point Of Hearing) "soundwalks" within the soundscape using virtual listeners;
- multi-purpose audio decoding to various speaker configurations –e.g. mono/stereo/quadraphonic/5.1/etc;
- Real-Time and Non-Real-Time operation.

Before examining the specifics of the proposed implementation, it is interesting to briefly discuss existing solutions, which in some cases successfully address some of the previously introduced issues and propose plausible soundscape generation paradigms. Regarding automatic soundscape generation, at least four relevant research projects need to be cited. The European project *Listen* [1] coordinated by the Fraunhofer Institut für MedienKommunikation is focused on the generation and control of interactive soundscapes, although it is specifically targeted at innovative, media-oriented experimentation on augmented reality. Its main goal is to create a new medium: the immersive audio-augmented environment (IAAE). Nevertheless, Listen is not targeted at explicitly modelling the soundscape. *Tapestrea* [2] is intended to create "environmental audio" in real-time: however, it does not define any explicit relationship between sound and space, and it does not provide user interaction. *Physis* [1] is an industrial research project led by IRCAM that deals with the modelling and the synthesis of virtual soundscapes. Physis is exclusively oriented towards the game industry and implementation details have not been published yet. *GeoGraphy* [3] is designed for the real-time simulation of existing soundscapes, starting from a database containing sound materials and other information. It can work interactively and in real-time, and includes the modelling of a virtual listener. Nonetheless, the organisation of audio materials is based on specific data structures (the so-called "graphs") which are potentially very complex and thus hard to handle. A generation technique of realistic soundscapes by means of a higher level methodology based upon GeoGraphy has been proposed too [4].

[1] http://www.ircam.fr/305.html?&tx_ircamprojects_pi1%5BshowUid%5D=74&tx_ircamprojects_pi1%5BpType%5D=p&cHash=ed317fa8927e424c8700c020b1812a58&L=1 Retrieved June 14, 2014.

2. SOUNDSCAPEGENERATOR (SSG)

SoundScapeGenerator (SSG) is an autonomous soundscape synthesis engine capable of modelling soundscapes with a variable degree of realism. It has been designed trying to include all the features mentioned in Section 1, and has been prototyped using the SuperCollider programming environment. In SoundScapeGenerator the audio is generated by a Renderer that expects as arguments a SonicScape (the model of the soundscape, see Section 3), a Listener, which may be fixed or movable in the space, and a Decoder. The Decoder manages the desired output format. Internally, SSG relies on ambisonics spatialisation algorithms [5], so that, given the appropriate decoder, the same audio stream may be decoded (at least, theoretically) to any of the standard formats such as mono, stereo, 5.1, but also to a custom, arbitrarily speaker configuration in 2D or 3D space. The Renderer takes into account these three components to generate the final audio signal. It can operate both in real- and non-real time, thus making the system suitable for a wide range of applications. The SonicSpace is intended as a model of the desired space. A SonicSpace is built as an aggregation of an arbitrary number of individual "SoundZones" and with respect to their individual geographic, acoustic and sonic characteristics. SoundZone's geographical features refer to their spatial boundaries and their absolute positioning in 3D virtual space. Their acoustic features refer to modeled physical phenomena such as reverberation and resonance. Acoustic features are modelled independently and not with respect to the geometric properties of a modelled SoundZone—geographical informations are merely used to position/localise sounds. SSG also allows the user to model the acoustic properties of the boundaries (e.g. walls) that delimit the various SoundZones by means of various filtering. The sonic features of a SoundZone refer to the type of sound events that may occur inside a SoundZone and have to be specified using an arbitrary number of individual "SonicSources". SonicSources are conceived as containers for some sort of audio event which may or may not be repeated in time—their only differences lying in their spatial positioning and directionality. This means that a SonicSource consists of both the audio data to be reproduced and the information regarding timing and location for generation. In our model, audio data are intended as sound samples, as typically happens in real soundscape modelling. Timing is defined by means of a pattern-based mechanism (see next section); location depends also on the type of source. Figure 1 depicts the structure of a SonicSpace. SoundScapeGenerator features five different types of SonicSources:

- SonicAtmosphere: non-directional sonic ambience;
- FixedSound: directional and fixed in space;
- AmbulatorySound (2): governed by envelopes or by a callback function;
- SoundCloud: representing complex events, such as e.g. rain or crowds, that are characterised by multiple appearances of similar sonic sequences at ever-changing and random positions within a given cubic area and with respect to a density factor.

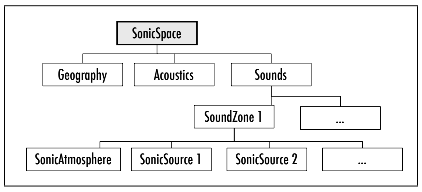

Figure 1. Structure of SonicSpace.

Each SoundZone has to be associated with at least one SonicAtmosphere, that functions as a background ambience and, therefore, provides a sort of canvas upon which individual SoundSources are positioned. The presence of such a background sound is useful also in order to mask unwanted artefacts in the sound samples, such as e.g. background noises in sound samples. The particular acoustic profile of a SoundZone will also add coherence to the scenery since all sound events within a given area will be processed in a similar way. The various directional SonicSources are localized using ambisonics algorithms thata have been further customized to allow for size-awareness (objects emitting sound from a larger area should have a broader spatial footprint than smaller ones) and listener-awareness (concomitant to the topological positioning and the listening radius of a virtual listener), as to be subsequently explained.The algorithm first encodes the audio source into an omnidirectional soundfiled (a four-channel b-format signal). Then the spherical coordinates of each SonicSource with respect to the virtual Listener's positioning in the SonicSpace are calculated as shown in equations 1, 2 and 3. In our implementation, the Listener's and Source's position coordinates are internally represented as three-channel control signals (to allow for ambulatory Listeners/Sources) and in Cartesian notation—channel 0 stands for the x, 1 for the y and 2 for the z dimensional axis. In the above equations, ρ stands for radius, ϕ for azimuth angle, θ for zenith, Δx, Δy and Δz for the difference between the Listener's and the SonicSource's coordinates for every dimensional axis. All coefficients are represented as functions of time (t) since they are audio signals. Note also that the equation to calculate the radius has been modified to account for the Source's size (R_s)—each SonicSource is assumed to be a spherical object that emits sound in all directions.

$$\rho(t) = \sqrt{\Delta x(t)^2 + \Delta y(t)^2 + \Delta z(t)^2} - \frac{R_s}{2} \tag{1}$$

$$\phi(t) = \arctan \frac{\Delta y(t)}{\Delta x(t)} \tag{2}$$

$$\theta(t) = \arccos \frac{\Delta z(t)}{\rho(t)} \tag{3}$$

Each SonicSource's corresponding audio signal is then localised according to equation 4 which demonstrates how a source soundfield is localised. Soundfields are notated as a standard b-format ambisonics signal (b-format signals comprise of 4 coefficients, namely w, x, y, z [5]). a_r

stands for relative amplitude and is a parameter associated with each SonicSource so that the algorithm takes into account the phenomenological amplitude characteristics of the Source (e.g. recordings of a space-rocket and a bird may be of the same amplitude yet the first is phenomenologically understood as much louder to the latter) which has to be taken into account for a dynamic and realistic SoundScape to be generated. P_r is a synthesis module used to simulate the proximity effect. P_u is also a synthesis module that 'pushes' the soundfield in a particular direction, taking into account a distortion angle (ω, calculated as in equation 5) and the azimuth and zenith angles as calculated in equations 2 and 3. A distortion angle of 0 stands for a non-directional soundfiled coming from all directions while one of $\frac{\pi}{2}$ stands for sound localised at a single spot. The proposed algorithm will result in 'pushing' all sounds that are outside a five meters range from the Listener's position to behave as nominal single-spot Sources while it will maintain a broader spatiality for those Sources located closer to it. Finally, the a_ρ represents a amplification factor which is calculated as an exponential mapping of the radius (ρ) (which represents the distance of a Source's to the Listener) to a range of $0, 1$ and with respect to the Listener's listening radius (ρ_l)—as shown in equation 6. The formula first produces a value between $1, 2$ and then subtracts 1, to compensate for the idiosyncrasies of the unit generators used internally in our implementation.

$$S_i \begin{bmatrix} w & x \\ y & z \end{bmatrix} (t) = a_r \times a_\rho(t) \times P_r(P_u(\begin{bmatrix} w & x \\ y & z \end{bmatrix} (t), \omega, \phi, \theta)) \tag{4}$$

$$\omega(t) = \begin{cases} \frac{\pi}{2} \times \frac{\rho(t)}{5} & \text{if } 0 \leq \rho(t) \leq 5 \\ \frac{\pi}{2} & \text{if } \rho(t) > 5 \end{cases} \tag{5}$$

$$a_\rho(t) = \begin{cases} ((\frac{1}{2})^{\frac{\rho}{\rho_l}} \times 2) - 1 & \text{if } \rho(t) \leq \rho_l(t) \\ 0 & \text{if } \rho(t) > \rho_l(t) \end{cases} \tag{6}$$

All SonicSources belonging to a SoundZone and the SonicAtmosphere associated with it are mixed and further processed with respect to the acoustic profile of the SoundZone, as shown in equation 8. Then, the audio output of all SoundZones (Z_{i-n}) is mixed and routed to the a Decoding module which will generate the SoundScape (S_f) in the desired format. Equation 7 demonstrates the decoding algorithm.

$$S_f = D(\sum_{i=1}^{n} Z_i(t)) \tag{7}$$

$$Z_i(t) = A_{coustics}(A_{tmo} + \sum_{i=1}^{n} S_i(t)) \tag{8}$$

Therefore, the proposed algorithm takes into account the Listener's relative three-dimensional positioning with respect to each SonicSource, their distance (also accounting for the the potential proximity effect), the size of the SonicSource and the acoustic profile of each SoundZone. Insofar as localisation of audio samples is concerned, it has to be kept in mind that sounds that already have spatial information registered within the recording should be treated differently when modelling a soundscape. Close-up of sound events, for example, should be positioned according to where the sound's source should be, e.g. in the sky in the case of a bird, while sound events recorded from a distance should be positioned somewhere next to the listener, since they already convey a listener's perspective of something occurring in a distance. When positioning such Sources, it should be also kept in mind that, as already explained the algorithm will progressively 'push' non-directional soundfields to directional ones within a five meters range from the Listener's positioning; therefore and for most cases, this means that such Sources should be positioned at a distance no less than 5 meters. Figure 2 demonstrates the flow of audio and control signals within our current implementation of SoundScapeGenerator. As already mentioned, the Listener's and the various Sources' positioning are represented as three-channel control-rate signals that feed into internal virtual buses. Audio is also streamed internally using four-channel virtual-buses.

2.1 Pattern-based Sequencing

One of the most difficult aspects in soundscape simulation and generation concerns the modelling of a source's behaviour in time. In real-life soundscapes, sound events may repeat themselves in irregular patterns. An interesting approach in relation to sound synthesis is cartoonification as devised by the Sounding Object project [6]; here sounds are described not in terms of the real mechanics of their production, but following a phenomenologically-compliant and physically-simplified approach. The idea at the base of cartoonification can be extended from sound synthesis to the organisation of sound sequences. In SoundScapeGenerator, time-organisation of sound events is thus cartoonified (loosely modelled) by means of specific data structures, namely generators [7]. A generator can be thought of as a rule for generating sequences. When executed, a generator will result in a stream of values of a certain length and with respect to some high-level rule. As the rule is specified rather than the actual data, the sequence can be of infinite length. In relation to SoundSources, a generator-based strategy allows for quick and efficient emulation of a variety of time behaviours: from one-shot to repetitive sound-events, from deterministic to stochastic sequences. Thus, in SoundScapeGenerator, all SoundSources have to be associated with a pattern that defines the exact time of their first appearance and their repetition scheme. Support for generators is native in the SuperCollider language by means of "Patterns" and "Streams" of data that result from their execution [8]. Hence on, we will use the term "pattern" and the relative SuperCollider notation. Built-in patterns include representations for linear sequences, random selections from lists, probability-based number generators, random walks, and other similar mathematical constructs that provide a conceptually straightforward way to model streams of values. Such patterns may be chained and/or nested recursively, allowing for a very compact notation of

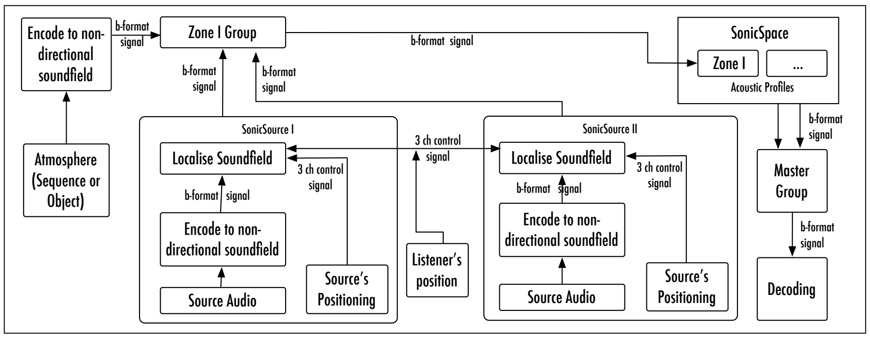

Figure 2. Audio flow within SSG.

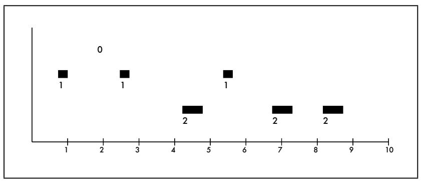

Figure 3. Barking Sequence.

very complex behaviour. Consider, for instance the following pattern:

$$Pseq([0,1],2)$$

This stands for a linear sequence of 0 and 1 repeated twice (2), representing time intervals. As a model of temporal behaviour, it will result in the corresponding SoundSource being reproduced 4 times: immediately once the Renderer is asked to play (0 seconds), 1 second after having finished playing back, immediately after this second appearance has finished (again, 0 seconds), and 1 second after the third appearance. As an example, it is easy to model a highly realistic, ever-permuting dog barking by means of simple patterns applied to just a few barks, like that:

$$Prand([a,b,s],inf)$$

$$Pwhite(0.5,2.0,rrand(3,10))$$

This two patterns will result in a virtual dog barking every now and then, as shown in Figure 3 in irregular patterns and having a varying duration. The first pattern defines the atomic sounds that will be used, with a and b representing two different bark atoms, s standing for silence and inf for infinity (it is upon the duration pattern to define the total number of atoms used). The second pattern defines their durations as random numbers between 0.5 and 2 seconds and will aggregate a random number of atoms between 3 and 10 (the $rrand$ function). Then, a Source which points to the aforementioned sequence object will cause it to generate a new audio sequence whenever needed and with respect to its particular repetition schemata. A graphical representation of a possible barking sequence (here made up of 7 sounds for $\approx$ 9 seconds) is shown in Figure 3, where each bark sound is given an index (1 and 2) and silence is represented by 0. To compensate for potential discontinuities when joining sound samples together and to seamlessly truncate audio files when needed, SSG uses parametrisable linear (cross)fades. The designer may select the appropriate cross-fade time with respect the idiosyncracies of the audio samples in use; e.g. joining individual footsteps together to form a larger sequence requires minimal or even not fade times between each sample, while joining city ambiences to construct a longer Atmosphere requires fade times of several seconds (even minutes). In any case, looping may be achieved using pattern sequences: the algorithm will simply create crossfades with a sound and a repetition of itself.

2.2 Composing soundscape as a hyper-narratives

As discussed, SoundScapeGenerator features different kinds of Listeners. Listeners are given a listening radius as well, outside of which no sound is audible. After having localised the sounds using the extended ambisonics techniques described before, the algorithm modulates their amplitude proportionally to their distance from the Listener's positioning to deliver a POH (Point of Hearing) interpretation of the SonicSpace. In that sense, SoundScapeGenerator follows a listener-oriented and phenomenological approach. The exported soundscape is analogous to a virtual listener's perspective of the SonicSpace and with respect to their spatial positioning, their listening radius and the particular acoustic features of the area in which they are positioned. Therefore, rendering with different Listeners may result in dramatically different versions of the very same SonicSpace.

Considered as a semiotics for the description of soundscape, SSG is able to output soundscapes as its utterances, depending on various factors. In contrast to a DAW-like purely "syntagmatic" (sequential) approach, SSG models a SoundScape as a "semiotics" where each Listener's particular trajectory through a SonicSpace can be seen as a realisation of virtual possibilities (its "paradigm"). In that sense, and from a user-perspective, SSG may be conceptualised as a tool to model a broader hyper-narrative (or maybe, a hyper-soundscape) that may yield very different outputs. Hyper-narratives are to be understood as the sum of the multiple trajectories through a paradigm [9] or, in this particular case, as the aggregate of all possible combinations of the elements in a SonicSpace. Each time SSG is asked to produce audio, a new narrative is generated, yet one which is always a subset of a broader hyper-narrative.

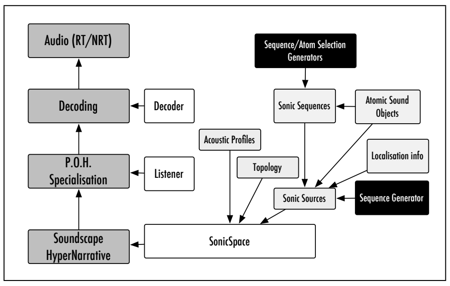

Figure 4. General structure go the SoundScapeGenerator Model.

From a practical point of view, using such an approach is highly advantageous in that coherency and consistency are easily achieved. A hyper-narrative remains identifiable as such and, if carefully designed, it outputs a unique, dynamic and unpredictable utterance–thus offering a highly personalised and dynamic experience that does not suffer from the problems normally associated with fixed soundscape recordings[10]. Figure 4 shows the complete architecture of SSG in four stages (dark grey): A "hyper-narrative" (designed by means of a SonicSpace) is to be "specialised" (with respect to a Listener object) and then "decoded" (with respect to an ambisonics decoder) so that audio is generated (in RT or NRT). Figure 4 further illustrates how a SonicSpace is designed by means of defining Acoustic, Topological and Sonic profiles (for each SoundZone). SonicSources are then shown to be depend on Generators (as already explained), localisation information and sound samples (be in an atomic sound object or in a sequence concatenated by generators).

3. THE SODA PROJECT: SOUNDSCAPECOMPOSER

SoundScapeGenerator has been designed as a generic system. While it is theoretically possible to directly embed it in third-party application, its current intended use is either as a stand-alone autonomous unit or as part of broader SuperCollider-based systems. Even in that form, however, it is quite straightforward to use SuperCollider's built-in communication channels (OSC, MIDI, etc) so that SSG can be controlled by other software or hardware applications. In this vein, SSG features no graphical user interface to avoid specialisation and keep the project generic [2] . Possible applications for SSG may include soundscape composition, acoustic ecology projects, virtual or augmented reality environments, interactive sound design. The system's NRT mode makes it a possible solution for audio-visual sound designers and composers looking for simulations of real or artificial environments to be used in other contexts. SSG's hyper-narrative paradigm in combination with its real-time capabilities makes it suitable for interactive or reactive navigation in virtual spaces. These include for example video games, animation, virtual reality or augmented reality applications, etc . We now describe the SoDA project for which a specialised module has been devised as a higher level interface to SSG.

[2] In any case, SuperCollider has a sophisticated support for designing GUIs and it is a trivial task to create a GUI according to one's needs.

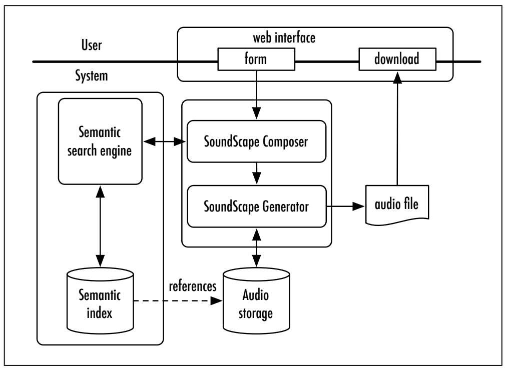

Figure 5. SoDA Architecture.

The SoDA (Sound Design Accelerator) project [3] aims at providing automatic soundscape composition by means of an ontologically annotated sound library [11]. SoDA requires a NRT sound engine which nevertheless should be automatically parametrisable according to the results of a semantical analysis engine. The idea is that the sound designer simply queries the system by inputting few keywords, and gets back an automatically generated soundscape. SoDA aims at providing –with respect to the creation of soundscapes– a twofold computational "acceleration" (hence its name): on the selection of relevant sound elements to be composited and on their organisation. Figure 5 shows SoDA's architecture and the various modules involved. The user is asked to provide a number of keywords through a web interface, then the semantical analysis engine is responsible for analysing the query so to return the sound files to be used in the generation step. Sound files have been previously annotated by a semantical analysis phase. Each of the audio files in the library is annotated so that technical as well as contextual information is associated, e.g. type of shot, relative amplitude, etc. SoDA then relies on an automated SoundScapeComposer. The latter is intended as a high level interface to SSG, as it provides SSG all the required data by taking into account the results of semantical analysis and by some ad hoc algorithms for compositing sound files. Once opportunely tuned, SSG is then responsible for the final audio rendering. Complying with SoDA's requirements for a fully automated soundscape composing paradigm, SoundScapeComposer (SSC) has being conceived as a bridge between SSG and the other components of SoDA. SSC has 4 tasks to address (6): parsing and interpreting the result of the semantical analysis engine; modelling a SonicSpace with the adequate features; populating it with SoundSources; placing a Listener within it. In the context of SoDA, a soundscape is intended as a background ambience with some possible moving sources: thus, a SonicSpace of a singleton Zone inhabited by a fixed Listener is enough. This is already an example of the specialisation achieved on SSG through SSC. SSC associates to the SonicSpace an ever-present Atmo-

[3] http://sodaproject.wix.com/soda

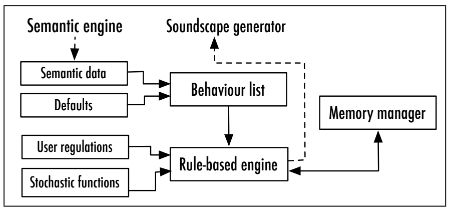

Figure 6. SoundScapeComposer Overview.

sphere, in order to provide a first background layer. One of the most difficult tasks of SSC is to figure out when and how individual atoms should be joined to form "meaningful" sequences. This is achieved by considering special meta-tags entries as well as the technical data retrieved from the files using a FEUtil –a special Feature Extraction Utility expressively designed for SoDA, which relies on machine listening algorithms to extract various spectral and psycho-acoustical information from audio files. To accomplish each task, SSC relies on three modules: an algorithmically generated Behaviour list; a Rule-based Engine, featuring general rules, stochastic and probability functions and user-defined regulations; a memory manager (Figure 6). The Behaviour list is intended as a description of the way a SonicSource behaves. It is built from the ontological data returned by the semantical engine, and provided with default states for everyday sounds: for example, animals or cars may occasionally move while architectural structures do not; speech is generally not to be repeated; cars move mostly on ground-level while elevators move vertically, etcetera. A behaviour list should be understood as numerical answers to certain properties, such as: the minimum/maximum allowed deviation in each spatial dimension for the localisation of a Sonic-Source, the minimum/maximum speed of their movement and their acceleration pattern, the maximum number of repetitions allowed for a SonicSource, and so on. The Rule-based Engine is, then, deputed to convert the data provided by the behaviour list into the required SonicSources and to provide them with their spatio-temporal features. SSC also features an intrinsic Memory module, initially empty, that is incrementally populated at runtime with the features of the generated SonicSources. Then, and unless the behaviour lists or the user-defined rules suggest otherwise, SSC attempts to accelerate variance by consulting the memory manager in order not to generate objects with almost identical features.
SSG is intended as a low-level, fine-tuneable engine. On the contrary, the purpose of SSC is to define an even higher level, by hiding the user the complexities involved in manually designing a soundscape with SSG. Interaction is intended to happen primarily through the various semantical filters s/he may apply to the query: the semantic engine becomes the main user interface.

4. CONCLUSIONS

The SoundScapeGenerator has been designed as a generic system suitable for generic soundscape simulation that involves sound samples. Its pattern-based logic and its hyper-narrative compositional paradigm are intended to provide a flexible, interactive and generative low-level engine. Indeed, it still requires the user a certain amount of work to be set up. But its modular nature allows to define higher level interfaces that provide automatic parametrisation and specialisation, as in the case of SoDA's SoundscapeComposer. In fact, the latter relies on a minimum set of SSG's features. SoundScapeGenerator's current implementation is stable and fully functional. SSG has been initially tested in a wide range of soundscape-generation scenarios. A rigorous user evaluation has not been carried on yet, and it will be the next step of the project. Also, there is still room for improvements (e.g. concerning fidelity, efficiency, flexibility and ease of use), that will become more evident by taking into account users' feedback.

5. REFERENCES

[1] O. Warusfel and G. Eckel, "Listen-augmenting everyday environments through interactive soundscapes," *Virtual Reality for Public Consumption, IEEE Virtual Reality 2004 Workshop*, vol. 27, 2004.

[2] A. Misra, R. Cook, and G. Wang, "Musical tapestry: Re-composing natural sounds," in *Proceedings of ICMC*, 2006.

[3] A. Valle, V. Lombardo, and M. Schirosa, "Simulating the soundscape through an analysis/resynthesis methodology," *CMMR/ICAD*, pp. 330–357, 2009.

[4] M. Schirosa, J. Janer, S. Kersten, and G. Roma, "A system for soundscape generation, composition and streaming," in *Prossime distanze. Atti del XVIII CIM*, pp. 115–121, 2011.

[5] D. Malham and A. Myatt, "3-d sound spatialization using ambisonic techniques," *Computer Music Journal*, vol. 19, pp. 58–70, 1995.

[6] D. Rocchesso and F. E. Fontana, *The Sounding Object*. Edizioni di Mondo Estremo, 2003.

[7] T. Budd, *A Little Smalltalk*. Reading, Mass.: Addison-Wesley, 1987.

[8] R. Kuivila, "Events and patterns," in *The SuperCollider Book* (S. Wilson, D. Cottle, and N. Collins, eds.), pp. 179–205, The MIT Press, 2011.

[9] L. Manovich, *The Language of New Media*. Leonardo (Series) (Cambridge, Mass.), MIT Press, 2001.

[10] M. Koutsomichalis, "On soundscapes, phonography and environmental sound art," *Journal of Sonic Studies*, vol. 4, [Online], 2013.

[11] A. Valle, P. Armao, M. Casu, and M. Koustomichalis, "Soda: A sound design accelerator for the automatic generation of soundscapes from an ontologically annotated sound library," in *Proceedings ICMC—SMC—2014*, (Athens), pp. 1610–1617, 2014.