A new implementation of Spat in Max

Thibaut Carpentier
STMS (UMR 9912) — IRCAM — CNRS — Sorbonne Université
1, place Igor Stravinsky, 75004 Paris
thibaut.carpentier@ircam.fr

ABSTRACT

Ircam spat∼ is a real-time audio engine dedicated to sound spatialization, artificial reverberation, and sound diffusion. This paper introduces a new major revision of the software package (spat∼ 5), and its integration in the Max environment. First, we present the newly adopted OSC interface that is used throughout the library for controlling the processors; we discuss the motivations for this choice, the syntax in use, and the potential benefits in terms of usability, performances, customization, etc. Then we give an overview of new features introduced in this release, covering Higher Order Ambisonics processing, object-based audio production, enhanced inter-operability with VR or graphics frameworks, etc.

1. INTRODUCTION

Ircam's Spatialisateur [1, 2], frequently dubbed spat∼, is a real-time audio engine dedicated to sound spatialization, artificial reverberation, and sound diffusion. It has been developed at Ircam since the early 1990s, and it primirarly operates in the Max [3] environment. It is packaged as a large library of processors, and structured around a feedback delay network reverberation unit [4], and panning modules. These processors can be parameterized by a high-level control interface; this allows to specify and modulate the acoustical quality of the synthesized room effect according to perceptually relevant criteria [5, 6]. spat∼ has applications in various fields, such as concerts, mixing, post-production, virtual reality, sonic installations, or sound design.

The software suite is developed through Agile methods [7], and it continuously integrates the research outcomes of the Acoustics and Cognition Team (formerly Room Acoustics Team). It has therefore evolved significantly over the years, and the latest major revision, spat∼ 4, was released in 2009 and presented in [8]. This paper introduces spat∼ 5, a new revision of the environment. As the application is built as a long-term project, its development roadmap tackles multiple concerns: improvement of existing features, implementation of new features, maintenance (adaptive, corrective, and preventive), optimization, architecture refactoring (to improve the maintainability and extensibility of the code base), etc. This paper first discusses a major refactoring of the environment (sections 2 and 3), and then introduces some of the newly added features (section 4).

Historically, spat∼ relies on an object-based internal model wherein the room effect consists of four temporal sections that are filtered and panned independently: direct sound, early reflections, late diffuse reflections, and reverberation tail (see section 3 in [8] for further details). This framework has been found useful and relevant in many situations, and it is preserved in the newly presented version. However, one significant change of spat∼ 5 concerns the software interface with its host environment.

2. OSC INTERFACE

2.1 Motivations

Released in 2008, Max 5 first introduces the concept of "attributes". Using the analogy of object-oriented programming, one can say that each *external* object in Max is a class instance, and attributes are member variables of this class. The Max API provides functionalities to publicly (i.e. to the end user) expose attributes, and to manipulate them via *getter* and *setter* methods. spat∼ 4 intensively uses this mechanism, as most control parameters of spat∼ are exposed as attributes. Such design choice has several advantages, and also, in retrospect, some drawbacks. The advantages are clear: attributes are very well integrated in Max, they can be easily accessed (attrui, getattr, *inspector*), and stored (pattr, pattrstorage); they provide an explicit syntax (unlike arguments). From a developer perspective, they are easy to integrate, through a comprehensive API. For the very specific context of spat∼, however, it turns out that attributes are somewhat inappropriate:

- spat∼ externals are inherently multichannel processors, and they often come with a large number of attributes (typically proportional to the number of channels). This may lead to a slow or inconvenient navigation throughout Max context menus. Furthermore, Max attributes are stored as arrays which is inappropriate for spat∼ parameters; a tree structure would be better suited so as to reflect the data hierarchy [1] . Also, storing attributes as lists can lead to poor performances: for instance, pattr (storage

[1] It is theoretically possible to create attributes of attributes in order to generate a hierarchical tree; however this does not really solve the problem, and does not serve simplicity.

mechanism for attributes) needs to copy the whole list as soon as one element is modified.

- `spat~` objects are often polymorphic: the type and number of exposed parameters may change over time. One example is the `spat.pan~` object which exhibits different properties whether it operates in stereo, binaural, Ambisonics, etc. Max attributes are somewhat inadequate to reflect this polymorphism [2].

- Attributes are strictly specific to the Max API. The `spat~` software library is integrated in many environments: Matlab [3], Spat Revolution [4], Ircam Tools plugins [5], Open Music [9], Pure Data [6], etc. Each binding requires some glue code to interoperate with the host application. From a developer perspective, it is important to minimize the workload and maintenance for such glue code; from a user perspective, it is efficient to use a similar interface (e.g. same syntax) in various hosts.

We have therefore shifted our ground, and adopted for `spat~` 5 a new protocol interface and syntax based on Open Sound Control (OSC [10]). Attributes are no longer used.

OSC was somehow an obvious choice: this protocol is widely used and accepted in the audio community, it eases inter-application communication, allows for interoperability with remote devices (e.g. via UDP/IP), etc. It can be easily implemented, and several libraries and language bindings are available. Its URI-style symbolic naming scheme is well suited for many computer music applications such as `spat~`. In addition to conventional messages, the protocol supports "bundles" which encapsulate several simultaneous messages, simplifying the transmission of a large amount of synchronous events. Finally it supports pattern matching language in order to dispatch a single message to multiple recipients.

2.2 Integration in Max

2.2.1 Syntax

`spat~` 5 externals are controlled via OSC syntax. At the Max interface level, messages are converted to/from Max native format: `atoms`. Such conversion is trivial as atoms and OSC arguments have very similar data type (int, float, symbols, etc.). OSC bundles are transmitted as `FullPacket` that only convey a pointer to a memory address (similarly to Max dictionaries). This allows for the very efficient transmission of large amount of data (the Max scheduler service is triggered only once per bundle, and not for each individual message contained in the bundle).

The syntax we have adopted is inspired from the REST (Representational State Transfer) style [11], and it happens to be quite close to the `spat~` 4 syntax (with the addition of the `/` separator). For instance, to control the Cartesian position of a sound source in `spat~` 5, one can use the following message:

```
/source/1/xy [float][float]
```

External objects support pattern matching semantics, which facilitate the grouping of multiple elements:

```
/source/*/mute [boolean]
/source/[2-5]/mute [boolean]
/source/{3,6,7}/mute [boolean]
```

Routing and dispatching OSC address patterns in Max may require the manipulation of regular expressions (`regexp`), which is usually inconvenient and inefficient; we have thus developed a toolbox of handy objects (approximately 25 externals) to simplify usual operations (see Figure 1).

The most frequently used OSC address patterns are stored in a hash table at compile-time. This avoids CPU-intensive string operations during runtime, and guarantees efficient dynamic lookup (similar to Max static symbol tables).

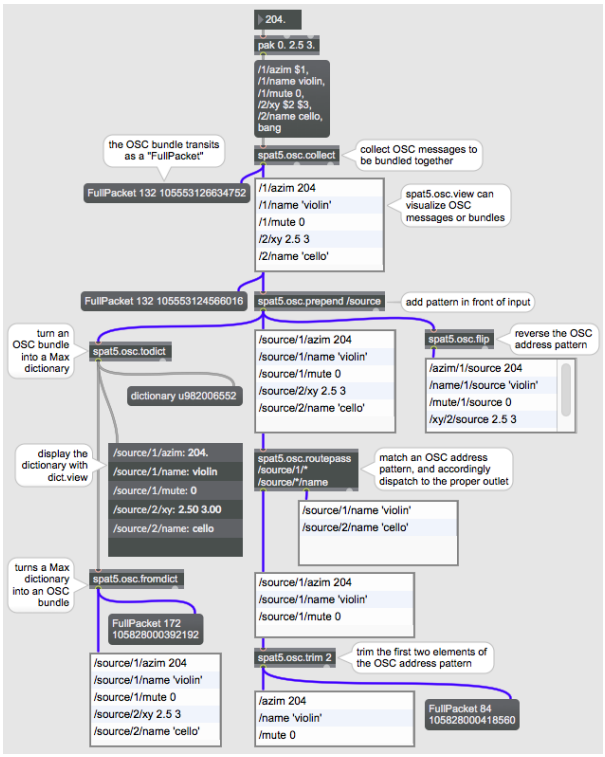

Figure 1. Examples of usual OSC manipulations. OSC bundles are conveyed (as `FullPacket`) through the blue-colored *patchcords*.

2.2.2 Inter-operability and compatibility

One potential advantage of the OSC interface is that users can benefit from existing tools and libraries, such as:

- the `odot` package [12], which provides a powerful expression language for the manipulation of OSC bundles in a variety of programming paradigms. Typically, `odot` might be used for the algorithmic generation and transformation of spatialization data (trajectories).

[2] Again, it is possible, with the API, to create object attributes – as opposed to class attributes – that can be dynamically added or removed, but this does not serve simplicity.

[3] Not publicly released at the time of writing this article.

[4] `www.spatrevolution.com`

[5] `www.ircamtools.com`

[6] Not publicly released at the time of writing this article.

- ToscA [13], a DAW plugin that allows the transmission of automation data over OSC. Although ToscA is designed as a completely generic tool, it has been thought, from its inception, for the remote control of object-based spatialization processors such as `spat~`.

- IanniX [14], Holo-Edit [15], `o7` [16], Antescofo [17], etc., are other examples of OSC-compatible software tools with powerful features for generative compositional processes.

However, it must be noted that OSC syntax breaks backward compatibility with previous version `spat~` 4. This is an important point as several hundreds of musical pieces currently rely on `spat~` 4. At the moment, there is no automatic way to "port" a patcher from `spat~` 4 to `spat~` 5; however, we believe that in most cases the transition should be fairly smooth as it mostly consists in minor syntactic adjustments. The package also provides examples for porting canonical patchers.
Finally, it should be noted that `spat~` 4 and `spat~` 5 can run simultaneously without conflict, as all `spat~` 5 externals are located in a dedicated namespace (prefix `spat5.*`). This also allows to progressively and iteratively port existing work.

3. REFACTORING THE ENVIRONMENT

The refactoring of the `spat~` 5 environment not only impacts the control syntax of the objects, but it also affects other aspects of the software library: overall usability, audio processors, and graphical user interfaces.

3.1 Usability

As mentioned in section 2, `spat~` 5 external objects no longer use attributes. As a consequence, they can not benefit from Max built-in features such as the `inspector` or automatic documentation hints. We have thus introduced new mechanisms that serve as a replacement: each object has its own `status` and `help` window (see Figure 2). The status window displays the current state of the object, similar to the Max inspector; it comes with a search filter, and one can copy/paste messages from this window to the patcher. The help window displays a text description of all supported OSC messages. Reference pages are also proposed and can be accessed via the standard Max documentation browser.

One can grasp the benefit of this new infrastructure by comparing against the attribute inspector in `spat~` 4 (Figure 3): the status window offers a hierarchical view of the parameters that is more readable than the list array of values in the inspector; furthermore, this allows to operate on the parameter with a finer granularity: for instance, the `sourceseditable [boolean]` attribute in `spat~` 4 (that enables the edition of source elements) is applied globally to all sources; in `spat~` 5, each element can be accessed independently:
`/source/i/editable [boolean]`

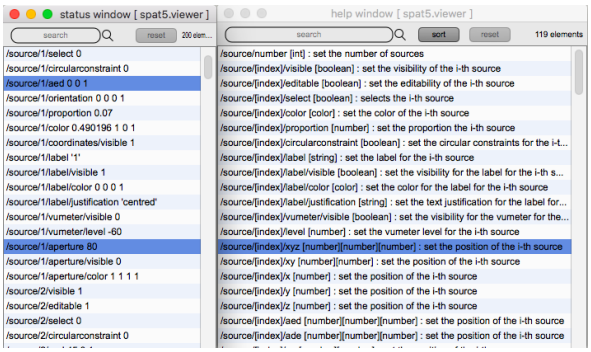

Figure 2. Status window (left) and help window (right) for `spat5.viewer`.

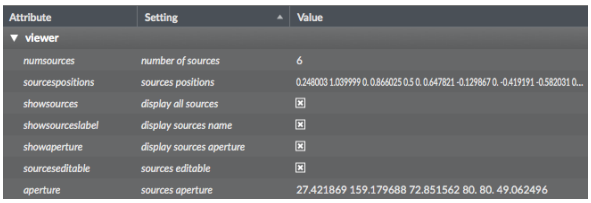

Figure 3. `spat.viewer` inspector in `spat~` 4.

3.2 Scheduling and thread-safety

Compared to attributes, the encapsulation of events inside OSC messages greatly simplified the programming of thread-safe queues for the synchronization of data shared in the various Max threads (audio thread, message thread, high-priority events thread, etc.). This provides better code hygiene, and significantly reduces the risk of bugs: incoming events (OSC messages or bundles) are stored in a thread-safe non-blocking FIFO queue, and later processed, in due time and in the appropriate thread (in `spat~` 4, only a few objects were guaranteed to be thread-safe).
For DSP objects, the FIFO is dequeued in the audio thread, at the beginning of the processing callback (see Figure 4). Such behavior is similar to the Max *scheduler in audio interrupt* mechanism. By default, all `spat~` 5 audio objects operate as such, regardless of the *overdrive* or *interrupt* settings of the host application.

3.3 Control interfaces

The `spat~` package contains more than twenty graphical user interface (GUI) control objects (e.g., see Figure 5). All these GUIs have been revamped for improved clarity and usability, and many new tweaking options have been added. A number of keyboard shortcuts have also been implemented for fast access to the most usual operations; these shortcuts can further be customized (Figure 6).

These GUI externals also benefit from the efficient thread-safe queue discussed in the previous section. This allowed to implement the GUI without the need for *deferlow* (deferring execution to the low priority thread), thus providing improved reactivity compared to `spat~` 4 (wherein *defer-*

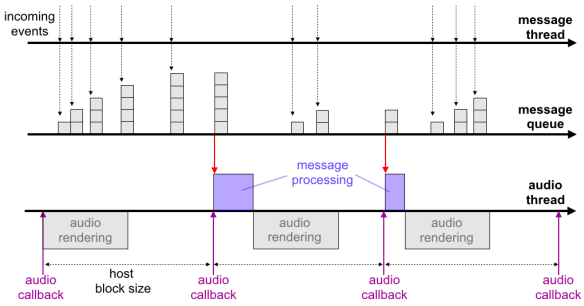

Figure 4. Scheduling of events according to the *scheduler in audio interrupt* procedure.

low was used, resulting in poor performances and backlog of the scheduler service, especially when the event rate is high).

The complete status of a GUI external is represented as an OSC bundle (see for instance Figure 2). This bundle can be loaded from and exported to a text file (human-editable). This provides a simple mechanism for creating and handling presets. The bundle can also be stored (embedded) into the patcher, or via the Max *snapshots* window ("Parameter Enable Mode"); in these cases, the OSC bundle is converted to a binary blob, and saved within the patcher file.

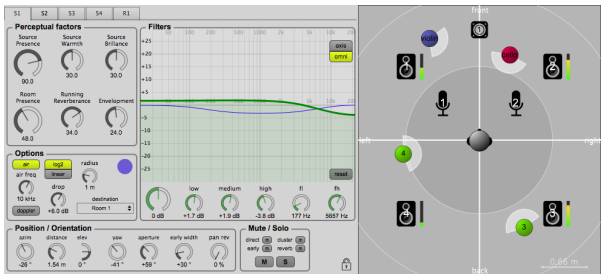

Figure 5. Graphical user interface for `spat5.oper` (high-level perceptual control for `spat~`). Perceptual factors for controlling room effect (left); filtering (centre); 2D view of the sound scene (right).

3.4 Software development

The code of the underlying `C++` libraries has been considerably modernized, now utilizing new programmation idioms as proposed by the `C++11` and `C++17` standards [7]: compile-time constant expressions (`constexpr`), automatic type deduction (`auto`), *lambda* functions, *range-based for loop*, etc. These new paradigms helped sanitizing the code, reducing bug probability, and incidentally shrinking the size of the code base; for instance, the glue code required for binding to the Max API (see section 2) has been reduced by 80%.

Signal processing algorithms, previously optimized and vectorized with the `Accelerate` [8] framework, have further been enhanced by using the Intel® Integrated Performance Primitives (IPP) [9].

[7] ISO International Standard ISO/IEC 14882:2017(E) – Programming Language `C++`

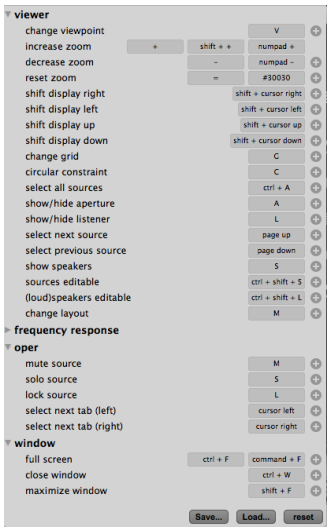

Figure 6. Key-mapping window for the customization of keyboard shortcuts `spat5.oper`.

4. NEW FEATURES

This section presents a few significant new features of the `spat~` 5 package.

4.1 Higher Order Ambisonics

All audio processors for the analysis/synthesis of spatial sound scenes have been improved, in many ways, and several new features have been added to the `spat~` 5 libraries. It would be cumbersome to enumerate all enhancements. Nonetheless, research in the last few years has particularly focused on the Higher Order Ambisonics (HOA [18]) technique, and we list below some of the related new features added to `spat~` 5:

- The various normalization schemes used for HOA (FuMa, MaxN, SN3D, N3D, etc.) are a frequent source of confusion for the users, and they may lead to compatibility issues between rendering tools. A formalization effort has been proposed [19], and the `spat~` documentation has been significantly improved in order to clarify the impact of normalization schemes in the Ambisonics production workflow, and to ease inter-operability.

- Several decoding strategies are proposed in `spat~` (see [8]). In `spat~` 5, we have further added the so-called "all-rad" (All-Round Ambisonic Panning and Decoding [20]) and the Constant Angular Spread [21]

[8] `http://developer.apple.com`
[9] `http://software.intel.com`

decoders. They are based on regular HOA decoding over a virtual t-design sphere, later projected via VBAP or MDIP onto the physical loudspeaker layout.

- The different available HOA decoders (*sampling decoder*, *mode-matching* [18], *energy-preserving* [22], *all-rad* [20], *constant-spread* [21]) may generate sound fields with significantly different loudness (up to a dozen dB, depending on the configuration). This prevents any comparative listening/study of the decoding strategies; we have thus introduced an energy compensation technique which allows to calibrate all decoders, according to an arbitrary reference. The method is based on the estimation of the delivered energy, under diffuse-field condition (see for instance section 4.4 in [22]).

- `spat5.hoa.blur~` is a new tool for manipulating the "spatial resolution" of an encoded HOA field. It allows to continuously vary the order of the HOA stream (i.e. simulating fractional orders), while preserving the overall energy [23]. It can be used to adapt the order of existing content, or as a creative FX (typically by varying the "blur" factor dynamically).

- `spat5.hoa.focus~` is another effect operating in the HOA domain. Inspired from [24], it allows to synthesize virtual directivity patterns and apply them to a HOA stream. Orientation and selectivity of the pattern(s) can be edited in an intuitive graphical user interface (Figure 7). The tool is most useful during post-production stage, i.e. when applied to recorded HOA fields, as it allows to directionally "zoom" into the sound scene.

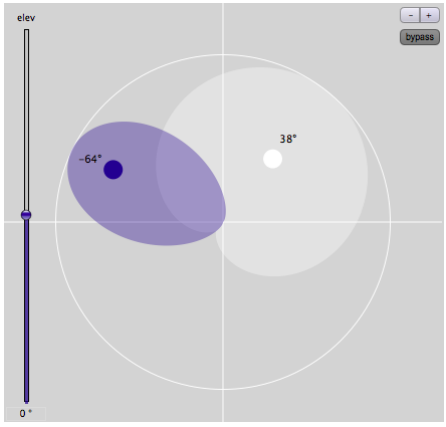

Figure 7. `spat5.hoa.focus` interface for synthesizing virtual patterns in the HOA domain.

4.2 Object-based audio

In recent years, there has been renewed interest in the object-based paradigm for producing and broadcasting multichannel audio. Several inter-exchange formats have been proposed; in particular, the Audio Definition Model (ADM [25]) is an open standard, published by the ITU and EBU [10], for the description of object-oriented media encapsulated in a Broadcast Wave Format (BWF) container. ADM prescribes a set of metadata (such as time-varying position and gain of audio objects) encoded in a XML chunk. `spat~` is one of the first toolbox offering a complete production chain for BWF-ADM files: `spat5.adm.record~` allows for the creation of BWF file with embedded spatialization metadata, and `spat5.adm.renderer~` copes with the real-time rendering of ADM media over an arbitrary reproduction setup (headphones or loudspeaker layout); other externals also allow to handle objects' interactivity. These externals are presented in greater details in [26]. Note however that only a subset of the ADM specifications is currently supported (although covering most typical usages), and a tighter integration of the format within the `spat~` architecture remains to be done (e.g. direct import/export of ADM files from processors such as `spat5.spat~`). This is part of on-going development work.

4.3 Panoramix

`panoramix` is a workstation for sound spatialization and artificial reverberation (Figure 8), intended for 3D mixing and post-production scenarios. The tool has been presented in previous publications [27, 28]. It builds on the `spat~` C++ librairies, and its rapid development was possible thanks to the in-depth refactoring discussed in sections 2 and 3. `panoramix` offers essentially the same functionalities as `spat5.oper` and `spat5.spat~`, however with an ad-hoc front-end, especially designed for mixing heterogenous multichannel content (seamlessly combining object-, scene-, and channel-based paradigms). So far distributed independently as a standalone application, `panoramix` is now also included as external objects part of the `spat~` 5 package; this will ease the integration of the tool in larger Max projects, coping with non-conventional mixing scenarios. The interested reader can refer to [27, 28] for further details about the design and usage of `panoramix`.

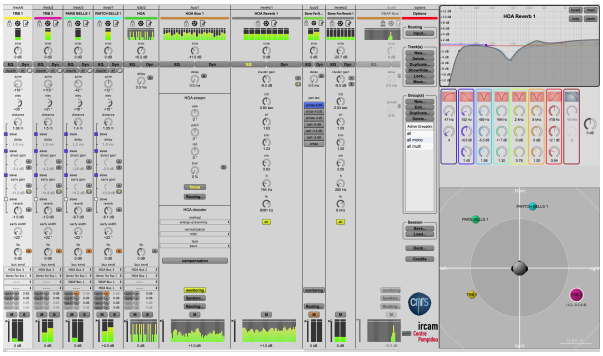

Figure 8. Overview of the `spat5.panoramix` mixing workstation.

[10] International Telecommunication Union and European Broadcasting Union

4.4 Quaternions

With the democratization of VR devices, spat∼ is more and more used for rendering audio scenes in immersive multimedia applications, typically presented with binaural over headphones. These virtual environments require the manipulation of 3D geometrical data. In particular, controlling the orientation of entities (either audio objects or the listener in the scene) is a frequent source of confusion for the users, as various conventions are being used (and they are not intercompatible). To solve this problem, we have developed a library of externals for the manipulation of quaternions, Euler angles, and 3D rotation matrices. These tools allows for converting between the different representations. The spat∼ audio engines (for instance spat5.binaural∼) can be controlled by either quaternions or Euler angles, therefore facilitating the cross-operability with VR SDKs.

4.5 Time Code

spat∼ is also frequently used for audio-visual productions; in such contexts, it is necessary to synchronize the audio and video streams. One of the most popular technique to do so, is to use a *Linear Timecode* (LTC [29]) which encodes SMPTE frames, and is transmitted as a longitudinal audio signal. Unfortunately, this standard is not natively supported in Max. We have therefore developed tools for receiving (spat5.ltc.decode∼) and generating (spat5.ltc.encode∼) linear time codes. Furthermore, the spat5.ltc.trigger∼ external can be used as a *cue manager* as it triggers actions at specific (user-defined) time stamps; the temporal granularity is rather low (typically 30 fps ≈ 33 milliseconds), but sufficient for most spatialization use cases.

4.6 Integration in Open Music

Sound spatialization is not only involved during live performance of a musical piece, but it should also be grasped during the early compositional phases. It is thus meaningful to bind the spat∼ library with computer-aided composition frameworks. As discussed in section 2, one motivation for adopting an OSC interface is to encourage host-independency (i.e. not be tied to Max), and to foster and accelerate the integration of spat∼ in other environments, while sharing a similar syntax. Thanks to the new OSC-based architecture, several spat∼ modules have been successfully inserted into the Open Music [30] and o7 [16] frameworks. The first results of this integration have been presented in [9, 16, 31], and they open the door to new spatialization effects and workflows, that are currently being investigated as part of artistic residencies at Ircam.

5. CONCLUSIONS AND PERSPECTIVES

We have presented spat∼ 5, a new major revision of the spat∼ framework for sound spatialization and reverberation, implemented in the Max environment. The library contains more than 200 external objects, covering a broad range of multichannel activities, as well as comprehensive documentation and tutorials. Compared to previous versions, this implementation offers a new interface and syntax, based on the OSC protocol. Besides syntactic considerations, the new OSC-based architecture comes with an in-depth refactoring of the library, aiming at improved usability, stability, performance, and inter-operability. The spat∼ 5 package also includes many new objects (control, DSP, and GUI) which further widen the scope of possibilities. Note also that several "subsets" of the software package (e.g. OSC tools, LTC, quaternions, etc.) are completely independent of the spat∼ paradigm, and could be useful to the broader computer music community.

Driven by research outcomes, technological innovations, and artistic challenges, spat∼ remains an ever-changing environment. The short-term prospects concern:

- the compatibility with OSC discovery and reflection protocol. A number of proposals have been made for querying an OSC namespace (OSC Query [32], Minuit [11] , libmapper [12] , OSNIP [13] , OSCQueryProposal [14]), however there is yet no consensus in the community. The integration of such protocol in spat∼ will be investigated.
- the tighter integration of object-based formats, especially ADM, in external objets, as discussed in paragraph 4.2.
- the compatibility with Max multichannel signals mc. Still in beta version at the time of writing this article, mc is a new type of patchcord in Max, which carries multichannel audio signals in a single connection. Such feature will tremendously simplify and improve the spat∼ workflow.
- continuing the integration of spat∼ modules in the o7 computer-aided composition environment, for extended spatial sound synthesis applications.

6. REFERENCES

[1] J.-M. Jot, "Real-time spatial processing of sounds for music, multimedia and interactive human-computer interfaces," *ACM Multimedia Systems Journal (Special issue on Audio and Multimedia)*, vol. 7, no. 1, pp. 55 – 69, 1999.

[2] J.-M. Jot and O. Warusfel, "A real-time spatial sound processor for music and virtual reality applications," in *Proc. of the International Computer Music Conference (ICMC)*, Banff, Canada, 1995, pp. 294 – 295.

[3] M. Puckette, "The Patcher," in *Proc. of the International Computer Music Conference (ICMC)*, San Francisco, CA, USA, 1988, pp. 420 – 429.

[4] J.-M. Jot and A. Chaigne, "Digital delay networks for designing artificial reverberators," in *Proc. of the 90th Convention of the Audio Engineering Society (AES)*, Paris, France, Feb 1991.

[11] https://github.com/Minuit
[12] http://libmapper.github.io
[13] https://github.com/jamoma/osnip/wiki
[14] https://github.com/mrRay/OSCQueryProposal

[5] J.-P. Jullien, "Structured model for the representation and the control of room acoustic quality," in *Proc. of the 15th International Congress on Acoustics (ICA)*, Trondheim, Norway, June 1995, pp. 517 – 520.

[6] E. Kahle and J.-P. Jullien, "Subjective listening tests in concert halls: Methodology and results," in *Proc. of the 15th International Congress on Acoustics (ICA)*, Trondheim, Norway, June 1995, pp. 521 – 524.

[7] C. Larman, *Agile and Iterative Development: A Manager's Guide*. Addison Wesley, 2003.

[8] T. Carpentier, M. Noisternig, and O. Warusfel, "Twenty Years of Ircam Spat: Looking Back, Looking Forward," in *Proc. of the 41st International Computer Music Conference (ICMC)*, Denton, TX, USA, Sept. 2015, pp. 270 – 277.

[9] J. Garcia, T. Carpentier, and J. Bresson, "Interactive-compositional authoring of sound spatialization," *Journal of New Music Research – Special Issue on Interactive Composition*, vol. 46, no. 1, pp. 74 – 86, 2017.

[10] M. Wright, "Open Sound Control: an enabling technology for musical networking," *Organised Sound*, vol. 10, no. 3, pp. 193 – 200, Dec 2005.

[11] A. Schmeder, A. Freed, and D. Wessel, "Best Practices for Open Sound Control," in *Proc. of the Linux Audio Conference (LAC)*, Utrecht, Netherlands, May 2010.

[12] A. Freed, J. MacCallum, and A. Schmeder, "Dynamic, instance-based, object-oriented programming in Max/MSP using Open Sound Control message delegation," in *Proc. of the 37th International Computer Music Conference (ICMC)*, Huddersfield, Aug. 2011, pp. 491 – 498.

[13] T. Carpentier, "ToscA: An OSC Communication Plugin for Object-Oriented Spatialization Authoring," in *Proc. of the 41st International Computer Music Conference (ICMC)*, Denton, TX, USA, Sept. 2015, pp. 368 – 371.

[14] T. Coduys and G. Ferry, "Iannix – Aesthetical/Symbolic visualisations for hypermedia composition," in *Proc. of the of the 1st Sound and Music Computing Conference (SMC)*, Paris, France, Oct 2004.

[15] C. Bascou, "Adaptive spatialization and scripting capabilities in the spatial trajectory editor Holo-Edit," in *Proc. of the 7th Sound and Music Computing Conference (SMC)*, Barcelona, Spain, July 2010, pp. 21 – 24.

[16] J. Bresson, D. Bouche, T. Carpentier, D. Schwarz, and J. Garcia, "Next-generation Computer-aided Composition Environment: A New Implementation of OpenMusic," in *Proc. of the International Computer Music Conference (ICMC)*, Shanghai, China, Oct 2017.

[17] A. Cont, "Antescofo: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music," in *Proc. of the 34th International Computer Music Conference (ICMC)*, Belfast, Ireland, Aug 2008, pp. 33 – 40.

[18] J. Daniel, "Représentation de champs acoustiques, application à la transmission et à la reproduction de scènes sonores complexes dans un contexte multimédia," Ph.D. dissertation, Université de Paris VI, 2001.

[19] T. Carpentier, "Normalization schemes in Ambisonic: does it matter?" in *Proc of the 142nd Convention of the Audio Engineering Society (AES)*, Berlin, Germany, May 2017.

[20] F. Zotter and M. Frank, "All-round ambisonic panning and decoding," *Journal of the Audio Engineering Society*, vol. 60, no. 10, pp. 807 – 820, 2012.

[21] N. Epain, C. Jin, and F. Zotter, "Ambisonic Decoding With Constant Angular Spread," *Acta Acustica united with Acustica*, vol. 100, pp. 928 — 936, 2014.

[22] F. Zotter, H. Pomberger, and M. Noisternig, "Energy-preserving ambisonic decoding," *Acta Acustica united with Acustica*, vol. 98, pp. 37 – 47, 2012.

[23] T. Carpentier, "Ambisonic spatial blur," in *Proc of the 142nd Convention of the Audio Engineering Society (AES)*, Berlin, Germany, May 2017.

[24] M. Kronlachner and F. Zotter, "Spatial transformations for the enhancement of Ambisonic recordings," in *2nd International Conference on Spatial Audio (ICSA)*, Erlangen, Germany, February 2014.

[25] ITU, "ITU-R BS.2076 (ADM Audio Definition Model)," www.itu.int/rec/R-REC-BS.2076, Tech. Rep., 2015. [Online]. Available: https://www.itu.int/rec/R-REC-BS.2076/

[26] M. Geier, T. Carpentier, M. Noisternig, and O. Warusfel, "Software tools for object-based audio production using the Audio Definition Model," in *Proc. of the 4th International Conference on Spatial Audio (ICSA)*, Graz, Austria, Sept 2017.

[27] T. Carpentier, "Panoramix: 3D mixing and post-production workstation," in *Proc. 42nd International Computer Music Conference (ICMC)*, Utrecht, Netherlands, Sept 2016, pp. 122 – 127.

[28] ——, "A versatile workstation for the diffusion, mixing, and post-production of spatial audio," in *Proc. of the Linux Audio Conference (LAC)*, Saint-Etienne, France, May 2017.

[29] J. Ratcliff, *Timecode: A user's guide, 3rd Edition*. Focal Press, 1999.

[30] J. Bresson, C. Agon, and G. Assayag, "OpenMusic – Visual Programming Environment for Music Composition, Analysis and Research," in *Proc. of the 19th ACM International Conference on Multimedia (OpenSource Software Competition)*, Scottsdale, USA, 2011.

[31] S. Agger, J. Bresson, and T. Carpentier, "*Landschaften* – Visualization, Control and Processing of Sounds in 3D Spaces," in *Proc. of the International Computer Music Conference (ICMC)*, Shanghai, China, Oct 2017.

[32] A. W. Schmeder and M. Wright, “A Query System for Open Sound Control (Draft Proposal),” Center for New Music and Audio Technology (CNMAT), UC Berkeley, Tech. Rep., 2004.