Journal of the Japanese Society for Sonic Arts, Vol.5 No.3 pp.1-5

Research Report

SPATDIF LIBRARY – IMPLEMENTING THE SPATIAL SOUND DESCRIPTOR INTERCHANGE FORMAT

Chikashi Miyama
College of Music Cologne
Studio for Electronic Music
me@chikashi.net

Jan C. Schacher
Zurich University of the Arts
ICST
jan.schacher@zhdk.ch

Nils Peters
Centre for Interdisciplinary Research
in Music Media and Technology
nils.peters@acm.org

ABSTRACT

The development and specification of SpatDIF, the spatial sound descriptor interchange format, is complemented with an actual software implementation in order to become usable in various environments. In this report, the current state in the development of a software library called 'SpatDIFlib' is discussed. The design principles derived from the concepts and specifications of SpatDIF, the class structure of the library, and code demonstrating its usage is presented. Furthermore, an application that utilizes the library is introduced as an exemplary use case.

1. INTRODUCTION

In this article we present the development of a software tool aimed at simple integration of SpatDIF into existing software. The concepts and guidelines are implemented in a C-Library and applied in an example surround-playback application.

SpatDIF, the Spatial Sound Description Interchange Format, presents a structured syntax for describing spatial sound information, addressing the different tasks involved in creating and performing spatial sound.

The goal of the SpatDIF approach is to simplify and enhance the methods of creating and exchanging spatial sound content. SpatDIF proposes a simple, minimal, and extensible format as well as best-practice implementations for storing and transmitting spatial sound scene descriptions. It encourages portability and the exchange of compositions between venues with different surround sound infrastructures. SpatDIF also fosters collaboration between artists such as composers, musicians, sound installation artists, and sound designers, as well as researchers in the fields of acoustics, musicology, sound engineering and virtual reality.

SpatDIF is developed as a collaborative effort and has evolved over a number of years. The community and all related information can be found at *www.spatdif.org*.

2. HISTORY OF THE PROJECT

SpatDIF was coined in 2007 [4] when Peters stated the necessity for a format to describe spatial sound scenes in a structured way, since all of the available spatial rendering systems used self-contained syntax and data-formats at that time. Through a panel discussion [1, 3] and other meetings and workshops, the concept of SpatDIF has been extended, refined, and consolidated.

After a long and thoughtful process, the SpatDIF specification was informally presented to the spatial sound community at the ICMC in Huddersfield in August 2011 and at a workshop at the TU-Berlin in September 2011. The responses in these meetings suggested the urgent need for a lightweight and easy to implement spatial sound scene standard, which could contrast the complex MPEG-4 scene description specification [7]. In addition, several functions necessary to make this lightweight standard become functional, such as the capability of dealing with temporal interpolation of scene descriptors as described, were introduced in [5].

3. CONCEPTS

One of the guiding principles for SpatDIF is the idea that authoring and rendering of spatial sound may occur at completely separate times and places, and be executed with tools whose capabilities cannot be known in advance. SpatDIF formulates a concise semantic structure that is capable of carrying all the relevant information, without being tied to a specific implementation, thought-model or technical method. SpatDIF is a syntax rather than a programming interface or file-format and may be represented in any of the structured mark-up languages or message systems that are in use today or in the future. It describes only those aspects required for the storage and transmission of *spatial sound information*. Because a complete work typically contains aspects that are outside the realm of such spatial sound description, SpatDIF judiciously provides interfaces to link these aspects to the spatial dimensions. For example, to be able to render the audio objects with the correct audio content in the spatial scene, the storage location of the unrendered audio content needs to be defined via SpatDIF's media resources.
After establishing a coherent specification with example use cases in textual form only, the next development step is the implementation of software that embodies the specified concepts. For this purpose a platform-independent software library was designed and is being implemented. Additionally, to demonstrate the usage of SpatDIF, an example application which utilizes this library was created. This application features SpatDIF file-handling and rendering of SpatDIF sound scenes into multichannel sound files.

By providing a software library rather than just a com-

plete software application, implementations in many different software environments are facilitated, which is one of the strategic goals of the project.

3.1. Example Scene

Referring back to the canonical example 'Turenas' by John Chowning (see also [5]), the beginning of a SpatDIF example scene, including only the 'insect' trajectory at second 0:44, contains the following elements in an XML format:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?
    >
<spatdif version="0.3">
  <meta>
    <info>
      <annotation>turenas insect trajectory</annotation>
      <date>2013-10-23</date>
      <author>jasch</author>
    </info>
    <extensions>media</extensions>
    <ordering>time</ordering>
  </meta>
  <time>44.0</time>
    <source>
      <name>insect</name>
      <position>0.0 8.0 0.0</position>
      <media>
        <id>sound_0001</id>
        <type>file</type>
        <location>/sound/insect.wav</location>
        <channel>1</channel>
      </media>
    </source>
  <time>44.078</time>
    <source>
      <name>insect</name>
      <position>1.359056 7.757522 0.0</position>
    </source>
```

Apart from the SpatDIF-compliant XML file, or other structured markup formatted file, the corresponding sound files have to be stored and transported alongside, in order to permit recreation of the scene. It is therefore important to think in terms of 'SpatDIF bundles' or projects rather than single files. A deliberate choice was made not to propose a format that combines sound files and scene descriptors in a binary format, since the readability without additional software tools would be lost.

4. THE LIBRARY

In this and the following sections, the concepts and structures implemented in the library and the example application are described. Although this information is mainly relevant to software developers, implementing SpatDIF in audio software, we believe that showing these technical details also provides an additional perspective on the possibilities that SpatDIF offers as a syntax.

4.1. Features

SpatDIFLib is an open source C/C++ multi-platform library, that offers the following functionalities to the developers of SpatDIF compatible softwares (clients).

- Loading and storing SpatDIF scenes from/to a XML, JSON, and YAML formatted strings
- Addition, deletion, and modification of entities (e.g., sources) in SpatDIF scenes
- Addition, deletion, and modification of events
- Associating events with entities
- Activation and deactivation of extensions
- Answering queries in regard to entities and events in SpatDIF scenes
- Controlling data in SpatDIF scenes via OSC Messages

Though SpatDIFLib can be controlled by OSC [9, 8] formatted strings, the library does not handle network sockets directly for the OSC communication; the clients should prepare sockets and threads for OSC communication. Likewise no scheduler or timer is implemented in the library.

4.2. C++ Class Structure

Figure 1 shows the simplified class hierarchy of the library.

An instance of *sdScene* class represents a SpatDIF scene and maintains instances of *sdEntityCore*. The functionalities of *sdEntityCore* may be extended by the descendants of *sdEntityExtension*. The activation and deactivation of the extension are global within a scene. Thus, *sdScene* is also responsible for the extension handling. Each instance of *sdEntityCore* maintains instances of *sdEvent*, that represent events of the entity they are attached to. The followings are brief descriptions of the most important classes.

4.2.1. sdScene

An instance of *sdScene* maintains all data associated to a SpatDIF scene. This class offers clients the following three functionalities.

- Addition, deletion and modification of entities in the scene
- Addition and modification of the meta data associated to the scene
- Activation and deactivation of the extensions in the scene

Once the client activates an extension in a scene, a *sdScene* automatically adds extended functionalities and allocates extra buffers to all existing and newly created instances of *sdEntityCore*. By deactivating an extension, *sdScene* removes all extended functionalities and previously allocated buffers from all existing *sdEntitieCores*. Subsequently, all data stored in the extension buffers are discarded.

4.2.2. sdLoader/sdSaver

These two classes provide several utility functions and enable clients to create an instance of *sdScene* from a XML, JSON, or YAML string and vice versa. In order to maintain platform independence and to achieve maximum flexibility, the library does not handle files directly; the client software is responsible for the file management.

These functions utilise two external libraries for parsing markup formatted strings. TinyXML-2 [1] and libjson [2].

[1] http://www.grinninglizard.com/tinyxml, accessed Oct. 9. 2013
[2] http://sourceforge.net/projects/libjson, accessed Oct. 9. 2013

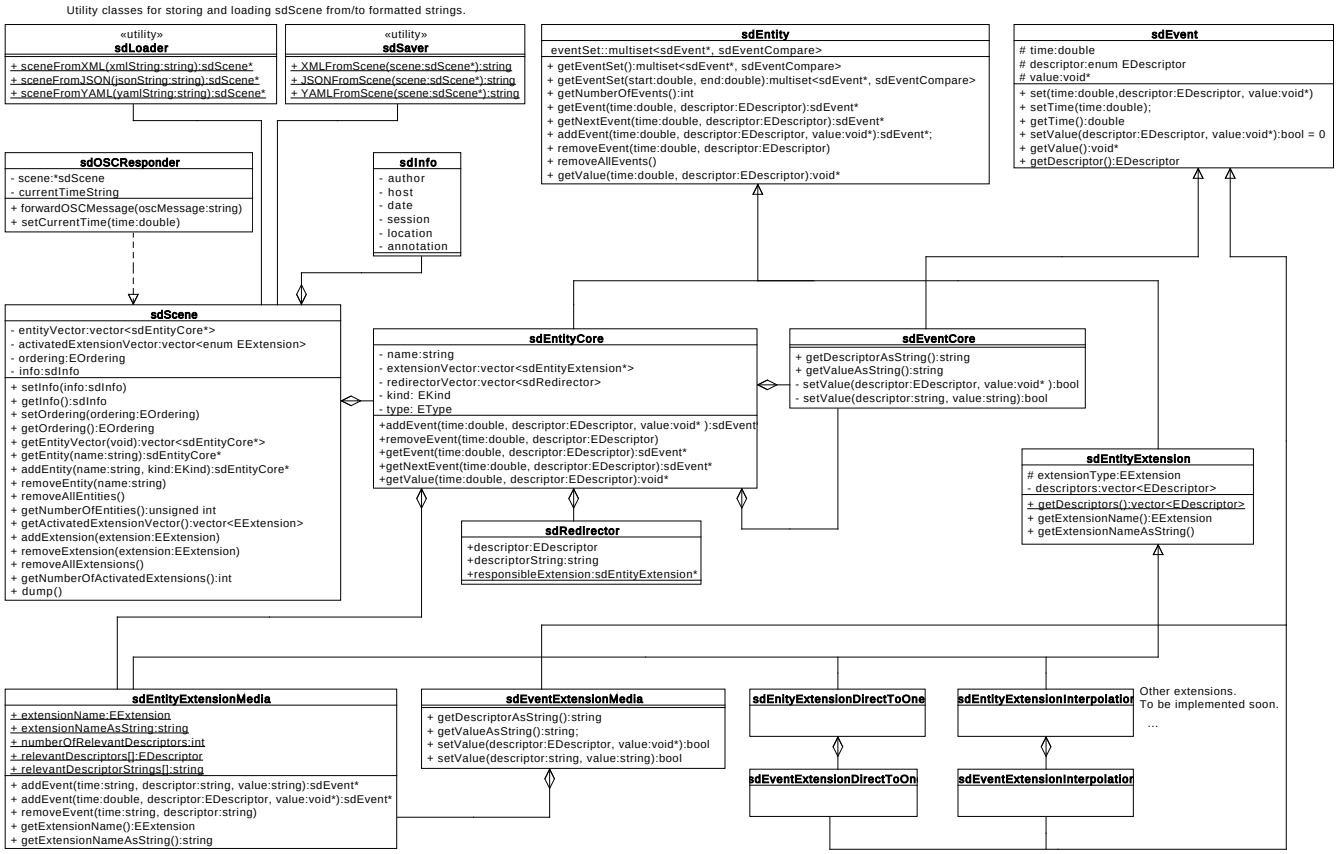

Figure 1. Class structure of the library

4.2.3. sdEvent

This is a pure abstract class of event, that maintains following three data items.

– time - absolute time of the event

– descriptor - type of event

– value - actual data

4.2.4. sdEntity

This is a pure abstract class of entity in SpatDIF scenes. Basic functionalities, such as addition, deletion, and modification of events are implemented.

4.2.5. sdEntityCore

An instance of *sdEntityCore* maintains events with SpatDIF core descriptors and has a vector to store instances of SpatDIF extensions. This class is also responsible for answering queries from the client conceding its events. For example, if a client asks a *sdEntityCore* a value of a certain descriptor at a specific time, the *sdEntityCore* returns value to the client. The client is able to raise an query about multiple events within a certain time frame and filter events by descriptors. If the client requests values of extended descriptors, a *sdEntityCore* forwards the query to the attached extensions stored in the internal vector.

4.2.6. sdEntityExtension

This is a pure abstract class of extensions. The descendants of this class. e.g. *sdEntityExtensionMedia* handles the events with extended descriptors. If a client activates an extension in a scene, each existing instance of *sdEntityCore* instantiates the designated subclass of *sdEntityExtension* and register it in its internal vector.

4.3. Simple Code Example

The following code listing shows how to load an XML-formatted string obtained from a SpatDIF file into an *sdScene* and query the entity called ‘insect’ for the first occurrence of an event which contains a position and a media descriptor.

```
1  sdScene scene = sdLoader::sceneFromXML(xmlString);
2  sdEntityCore* insect = scene.getEntity("insect");
3  sdEvent* insectPositionEvent = insect->
       getFirstEvent(SD_POSITION);
4  sdEvent* insectMediaLocationEvent = insect->
       getFirstEvent(SD_MEDIA_LOCATION);
5
6  cout << "Entity Name: " << insect->getName() <<
       endl;
7  cout << "First event is at: " <<
       insectPositionEvent->getValueAsString() <<
       endl;
8  cout << "Attached Sound File: " <<
       insectMediaLocationEvent->getValueAsString()
       << endl;
9  cout << "Attached at: " << insectMediaLocationEvent
       ->getTimeAsString() << endl;
```

The code produces this console-output:

```
Entity Name: insect
First event is at: 0.0 8.0 0.0
Attached Sound File: insect.wav
Attached at: 44.0
```

The following processes are executed in the example:

– Line 1: *sdLoader::sceneFromXML* static function loads a SpatDIF scene from a XML formatted string.

– Line 2: A pointer to an entity, named 'insect' is obtained by *scene.getEntity* member function.

– Lines 3-4: The entity 'insect' is requested to return a pointer to the first event with the position and media location descriptor.

– Line 6: Querying the 'insect' entity for its name

– Lines 7-9: Posting the values and time of events to the console

The above-mentioned basic class hierarchy is already implemented in the library. The library is also able to interpret simple XML, JSON and OSC messages and is currently examined against the example renderer application, in order to further improve its performance. In the next phase of the project, the complete set of extensions defined in the SpatDIF specification 0.3 [6] will be implemented and finally a C interface will be added, which will make this tool applicable to various types of software.

5. THE EXAMPLE APPLICATION

An example application that implements an entire workflow for the playback of SpatDIF files is being developed. The application is called a 'renderer' in analogy to visual tools, because it renders audible, in a surround setup, the information contained in a SpatDIF 'bundle'. It also serves to validate the development of the library, in the sense of a complex test case that reflects real-life usage. The implementation has to solve all the question relating to file-handling, OSC-streams, instantiating the voices of the playback engine, panning, distance cues, and the other descriptors present in the specifications 0.3.

More importantly however, it demonstrates the power and simplicity of SpatDIF by showing how one of the most common use cases is tackled.

5.1. Scope

In order to provide a relevant example for the application of the 'SpatDIFlib', the scope of the application has been limited deliberately. The panning algorithm is a simple spatial windowing algorithms named "ambipanning" [2] that is highly flexible, easy to implement, not tied to a specific number of speakers and usable without modification both in two and three dimensional spatialization situations. The application provides a stand-alone implementation, with a basic visualization of the scene, and the possibility to play the scene on a stereo or multichannel speaker setup. It allows to load a SpatDIF file with associated sound files and play it through a few predefined multichannel speaker layouts. Currently two mark-up languages are supported, XML and JSON, with more to come in the future.

5.2. Implementation Details

This application is implemented in the creative-coding environment openFrameworks [3] , which provides a powerful C++ toolset and a thriving community. Since it is not particularly oriented towards sound programming, the provided classes are somewhat rudimentary. However – and that is its strength – many extensions exist and it is relatively easy to add new functionalities and libraries. One of these extension-libraries that is being used for this application is libsndfile [4] , which provides a powerful audio file handling toolset. The SpatDIF-library is linked in as a dynamic library, and provides the entire data-structure and methods for handling the scene. The renderer runs both a sonic and visual playback of the scene. The visual representation is a bare-bones wireframe drawing of the scene in OpenGL. Figure 2 shows the scene of the example application. The sound playback runs through the sound-stream interface provided by openFrameworks. After retrieving samples from the sound files via libsndfile, panning and distance correction is applied, before the blocks of audio-samples are output. The signal processing chain is deliberately kept simple, to provide a clear example of the implementation of such a process.

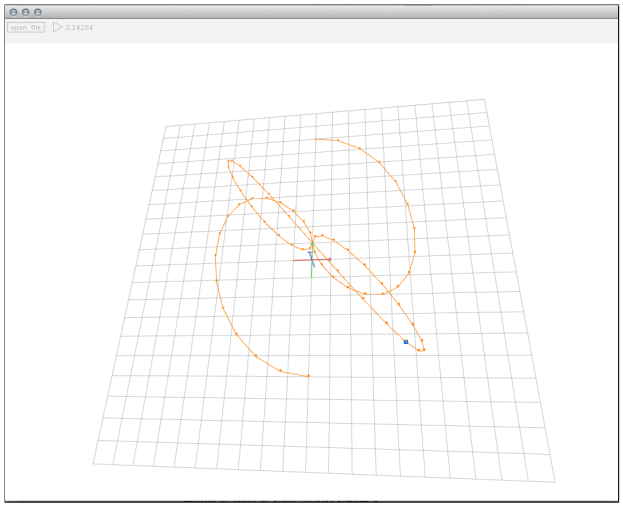

Figure 2. Graphical User Interface of the example renderer application, displaying the Lissajous trajectory from Chowning's 'Turenas'.

5.3. Task Separation between SpatDIFLib and the Client Application

The separation of labour between the library and a client application such as the example renderer presented here is very deliberate.

SpatDIFLib builds and maintains in memory the SpatDIF scene, either obtained from an already existing description stored in a file, or on-the-fly in real time from elements received via OSC-formatted network packets. It also provides an application programming interface (API) that hides most of the complexity of handling the scene data. Two reading and two writing interfaces are planned

[3] http://www.openframeworks.cc, accessed Oct. 23. 2013
[4] http://www.mega-nerd.com/libsndfile, accessed Oct. 23. 2013

for the library, one interfacing the file-system and the other providing the network-interface via OSC-packets.

The client application is in charge of loading the text files containing the SpatDIF-scene from the file system and handing them to the library in a text buffer. It deals with all the audio related processes, such as loading audio files, playing them back, and configuring the audio-system. Some of the information necessary to do this is provided by the SpatDIF scene. In addition, the client application can open network sockets and forward OSC-formatted data to the library. It queries the library for scene information at initialization as well as at runtime, based on its own scheduler.

6. AVAILABILITY

Both the library and the example application will be made freely and publicly available, as soon as the full feature-set of the 0.3 specifications are implemented and thoroughly tested. Of course, access to the code-base can be granted on demand.

7. ACKNOWLEDGEMENTS

This software development would not have been possible without the generous support of the Institute for Computer Music and Sound Technology ICST of the Zurich University of the Arts.

References

[1] Gary S. Kendall, Nils Peters, and Matthias Geier. “Towards an Interchange Format for Spatial Audio Scenes”. In: *Proc. of the International Computer Music Conference*. Belfast, UK, 2008, pp. 295–296.

[2] Martin Neukom and Jan Schacher. “Ambisonics equivalent panning”. In: *Proc. of the International Computer Music Conference*. Belfast, UK, 2008, pp. 592–595.

[3] Nils Peters. “Proposing SpatDIF - The Spatial Sound Description Interchange Format”. In: *Proc. of the International Computer Music Conference*. Belfast, UK, 2008, pp. 299–300.

[4] Nils Peters, Sean Ferguson, and Stephen McAdams. “Towards a Spatial Sound Description Interchange Format (SpatDIF)”. In: *Canadian Acoustics* 35.3 (2007), pp. 64–65.

[5] Nils Peters, Trond Lossius, and Jan C Schacher. “The Spatial Sound Description Interchange Format: Principles, Specification, and Examples”. In: *Computer Music Journal* 37.1 (2013), pp. 11–22. DOI: 10.1162/COMJ_a_00167.

[6] Nils Peters, Jan C. Schacher, and Trond Lossius. *Specification of the Spatial Sound Description Interchange Format (SpatDIF) V. 0.3*. http://redmine.spatdif.org/projects/spatdif/files. 2010-2012.

[7] E.D. Scheirer, R. Vaananen, and J. Huopaniemi. “AudioBIFS: Describing audio scenes with the MPEG-4 multimedia standard”. In: *IEEE Transactions on Multimedia* 1.3 (1999), pp. 237–250.

[8] Andy Schmeder. *Everything you ever wanted to know about Open Sound Control*. Tech. rep. Mar. 2008.

[9] David Wessel, Matthew Wright, and John Schott. “Intimate musical control of computers with a variety of controllers and gesture mapping metaphors”. In: *Proceedings of the Conference on New Interfaces for Musical Expression*. Dublin, Ireland, 2002, pp. 192–194.

8. AUTHOR’S PROFILES

Chikashi Miyama

Chikashi Miyama is a composer, video artist, interface designer, performer. He received a MA (Sonology/2004) from Kunitachi College of Music, Tokyo, Japan, a Nachdiplom (Komposition im Elektronischen Studio/2007) from Music academy of Basel, Switzerland, and a Ph.D (Composition/2011) from University at Buffalo, New york, USA. His compositions have received an ICMA award (2011/UK) from the International Computer Music Association, a second prize in SEAMUS commission competition (2010/St. Cloud, USA), a special prize in Destellos Competition (2009/Argentina), and a honorable mention in the Bourges Electroacoustic Music Competition (2002/France). Several works of him are included on the DVD of the Computer Music Journal Vol.28 by MIT press, and ICMC official CD/DVD(2005/2011). In 2011, he received a research grant from DAAD (German Academic Exchange Service) and worked as a visiting researcher at ZKM, Karlsruhe, Germany. He has taught computer music at University at Buffalo, USA and College of Arts Bern, Switzerland. He is currently teaching at College of Music and Dance Cologne, College of Music Basel, and Zurich University of the Arts. http://chikashi.net

Jan C. Schacher

A doublebass-player, composer and digital artist, Jan Schacher is active in electronic and exploratory music. His main focus is on works combining digital sound and images, abstract graphics and experimental video in the field of electro-acoustic music and in mixed-media projects for the stage and in installations. Jan Schacher has been invited as artist and lecturer to numerous cultural and academic institutions and has presented installations in galleries and performances in clubs and at festivals such as the Sonar Festival (Barcelona), Edinburgh Festival, the Singapore Arts Festival, the Holland Festival (Amsterdam), the Sonic Circuits Festival, Washington DC and numerous venues throughout Europe, North America, Australia and Asia. He is an associate researcher at the Zurich University of the Arts and doctoral candidate at the Royal Conservatory Antwerp. He has published peer-reviewed articles in the context of computer music, new interfaces for musical expression as well as on the topic of artistic research.