Short Paper

IMPLEMENTING THE SPATDIF LIBRARY

Chikashi Miyama
College of Music Cologne
Studio for Elektronic Music
Unter Krahnenb?umen 87 50668 K?ln, Germany
me@chikashi.net

Jan C. Schacher
Zurich University of the Arts
Institute for Computer Music and Sound Technolog
Baslerstrasse 30, Zurich, Switzerland
jan.schacher@zhdk.ch

ABSTRACT

Here we have an abstract

1. INTRODUCTION

In this article we present the development of a tool for easy 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 approach for working with spatial sound information, addressing the different tasks involved in creating and performing spatial sound.

The goal of SpatDIF is to simplify and enhance the methods of working with 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 [**peters˙caa07**] when Peters stated the necessity for a format to describe spatial sound scenes in a structured way, since at that time the available spatial rendering systems all used self-contained syntax and data-formats. Through a panel discussion [**2008ICMCpanel, Peters:2008spatdif**] and other meetings and workshops, the concept of SpatDIF has since 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 2011 in Huddersfield in August 2011, at a workshop at the TU-Berlin in September 2011 and in its current form in a Computer Music Journal article in 2013 [] 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 specification [**scheirer1999audiobifs**]. In addition, many features necessary to make this lightweight standard functional were put forward, for example the capability of dealing with temporal interpolation of scene descriptors.

3. CONCEPTUAL EXPLANATION / STRUCTURE

One of the guiding principles for SpatDIF is the idea that authoring and rendering of spatial sound might occur at completely separate times and places, and be executed with tools whose capabilities cannot be known in advance. It formulates a concise semantic structure that is capable of carrying the necessary 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 the aspects required for the storage and transmission of *spatial information*. A complete work typically contains additional dimensions that lie outside the scope of SpatDIF. These are only addressed to the extent necessary for linking the elements to the descriptions of the spatial dimension (i.e. the Media extension).

score-example in XML / JSON

format of SpatDIF ‘bundle’

4. CURRENT ACTIVITIES

5. LIBRARY DETAILS/STRUCTURE

5.1. Overview

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 scenes from/to XML, JSON or YMAL formatted string - addition, deletion, and modification of entities in SpatDIF scenes - addition, deletion, and modification of events and associate them to the existing entities in a scene - activation and deactiovation of extensions - answering queries about the data stored in a scene - realtime data handling from a remote host via OSC protocol

In order to maintain platform independency and provide client applications with maximum flexibility, the library does not handle XML, JSON or YAML files directly.

Instead, it interprets provideded XML, JSON or YAML formatted string. SpatDIFLib does not has scheduler or timer, all these functionalities should be implemented in the client application. Though SpatDIFLib is controllable employing OSC formatted string, it does not handle network sockets directly for OSC communication; the library simply interprets provided OSC string as commands to the library. The client application should prepare the appropriate sockets for OSC communication.

5.2. C++ Class Structure

Figure XXX shows simplified class hierarchy of the library. The following is the description of the most essential classes in the structure.

5.2.1. sdScene

sdScene maintains all data associated to a certain SpatDIF scene. This class offers clients the following three functionalities

1. adding, removing and modifying entities in its scene 2. activation and deactivation of the extensions 3. editing meta data associated to the scene

Once the client activate an extension in a scene, sdScene automatically add extended functionality and storage to all existing and newly created instances of sdEntityCore. By the deactivation of an extension, sdScene removes all extended functionality and previously allocated storage of all existing sdEntities. Subsequently, all the extended data will be discarded. The only way to instantiate sdEntityCore for the clients is to invoke sdScene::addEntity member function. The instances created through this function will be registerd in an internal vector of sdScene and can be reffered by name of the entity, employing sdEntity::getEntity member function. An instance of sdScene can be converted to XML, JSON, or YAML storable string, using sdSaver and vice versa using sdLoader.

5.2.2. sdLoader/sdSaver

These two classes provide several utility functions and enable clients to easily convert a XML, JSON, or YAML string to a sdScene and vice versa,

5.2.3. sdEntityCore

An instance of sdEntityCore maintains events with SpatDIF core descriptors and keep a vector to hold instances of SpatDIF extensions. This class is also responsible for answering query from the client about events. For example, if a client asks an entity a value of a certain descriptor at a specific time, the sdEntity returns value to the client. The client is able to request multiple events within a certain time frame and filter events by descriptor. If the client request values of extended desciriptors, the sdEntityCore forwards the query to the attached extension.

5.2.4. sdEntityExtension

This is a pure abstract class of extensions. The subclasses of this class. e.g. sdEntityExtensionMedia handle the events with extended descriptors. If the cleint activates an extension in a scene, all existing sdEntityCore create an instance of designated subclass of sdEntityExtension and register it to the internal vector.

5.3. Simple Example

The following example code shows how to load a XML formatted string to a sdScene and query a event stored in it.

How to query.cpp

5.4. Future plan

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 currently examined against SpatDIFRenderer, in order to furhter improve flexibility and practicality. All extensions defined in the SpatDIF specification 0.3 will be implemented in early 2014 and C interface will be added in the late 2014. The library will be completed by the end of 2014 and released under MIT or similar license. definition/features/tasks of the lib

interfaces in/out → (native C/C++ / OSC)
class structure
fileparsing XML/JSON
OSC 1.0
how to query in 3 lines (C++ code snippet)

6. REFERENCE RENDERER

(rough sketch....)

app scope / what does it do

app interfaces to lib → task distribution between app and lib

i.e. scheduler

or sockets / file-IO

7. AUTHOR'S PROFILE

Author's Name