Structuring Music through Markup Language:

Designs and Architectures

Jacques Steyn
Monash University, South Africa

Information Science
REFERENCE

Managing Director: Lindsay Johnston
Editorial Director: Joel Gamon
Book Production Manager: Jennifer Romanchak
Publishing Systems Analyst: Adrienne Freeland
Assistant Acquisitions Editor: Kayla Wolfe
Typesetter: Lisandro Gonzalez
Cover Design: Nick Newcomer

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Structuring music through markup language : designs and architectures / Jacques Steyn, editor.
p. cm.
Includes bibliographical references and index.
Summary: “This book offers a different approach to music by focusing on the information organization and the development of XML-based language, presenting a new set of tools for practical implementations, and a new investigation into the theory of music”--Provided by publisher.
ISBN 978-1-4666-2497-9 (hardcover) -- ISBN 978-1-4666-2498-6 (ebook) -- ISBN 978-1-4666-2499-3 (print & perpetual access) 1. Music--Data processing. 2. XML (Document markup language) I. Steyn, Jacques.
ML74.S77 2013
780.285’674--dc23

2012023347

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 7
Structuring Music-Related Movements

Alexander Refsum Jensenius
University of Oslo, Norway

ABSTRACT

The chapter starts by discussing the importance of body movement in both music performance and perception, and argues that for future research in the field it is important to develop solutions for being able to stream and store music-related movement data alongside other types of musical information. This is followed by a suggestion for a multilayered approach to structuring movement data, where each of the layers represents a separate and consistent subset of information. Finally, examples of two prototype implementations are presented: a setup for storing GDIF-data into SDIF-files, and an example of how GDIF-based OSC streams can allow for more flexible and meaningful mapping from controller to sound engine.

INTRODUCTION

Looking at the last century of music research, particularly what has been done within musicological departments, we find an overwhelming focus on analyses of symbolic music notation, particularly in the form of scores. The focus on scores as the centre that much of western music theory is circulating around is obviously because their symbolic nature allows for an easily quantifiable representation that can be used for many different types of analyses. This is probably also one of the reasons why much of computer based music analysis have focused on scores as the basic material, which again has led to an interest in developing various types of markup languages for representing the scores.

While scores can undoubtedly reveal many interesting aspects of the powers of music, we should we should not forget that they are mainly a representation of the composer's intention of musical structures to be performed. This also means that although many parameters can be derived from the score, others can only be found when listening to the musical sound. The advent of faster computers have made it possible to carry out analysis on the musical sound itself, and a large community of researchers have developed various

DOI: 10.4018/978-1-4666-2497-9.ch007

methods and tools for extracting features ranging from low (signal), middle (perceptual) and high (cognitive/emotional) levels. This has again led to standardisation needs for such audio based descriptors, e.g. the Sound Description Interchange Format (SDIF) and MPEG-7 descriptors.

The last couple of decades have seen yet another shift in attention among music researchers, mainly that of studying music not only as notes in a score or the physical and perceptual properties of musical sound, but rather to investigate body movement involved in both music performance and perception. While there are few that dispute the importance of body movement in performance and perception of music, comparably little attention has been given to movement when music has been analysed and discussed. While thinking about music as "organised notes," or "organised sound" seems mainstream, it is still radical to suggest that music could be thought of as "organised movement." Recent research goes even further in arguing that not only is body movement crucial for the performance of music, but that we also think about music through mental imagery of sensations originating in the body (see e.g. Godøy, 2004). This is the background for our interest in studying music-related movements in both performers and perceivers, and some theoretical discussions, observation studies and experiments on this can be found in e.g. Godøy and Leman (2009).

This chapter is concerned with how we can create representations of music-related body movement, covering topics such as synchronisation of multidimensional data streams, classification of movements, and issues related to streaming and storing of such data.

BACKGROUND

A major obstacle to music-related movement research is the lack of appropriate formats and standards. Working with hardware and software tools that cannot efficiently interchange information is a limitation not only for the single researcher, but it also effectively blocks collaboration between researchers and institutions. Rather than analysing data, this makes researchers spend a lot of time and effort just getting their data transferred from one system to another. Developing formats that can be used in music-related movement research therefore seems of great importance, and is something that can also benefit other fields, e.g. music technology and human-computer interaction.

Data streaming and storage has been a major challenge in our research group for a long time. Only storing data from a single sensor interface, and synchronise these data with simultaneously recorded audio and video files is not a trivial task. In the beginning, we stored raw sensor data in text files, with references to the corresponding audio and video files. However, since we did not store any information about the setup, scaling and filtering used, etc., these data files quickly became useless as we continued to change our experimental setups. In later studies, we were more careful about taking notes, and including these with the data sets, but we still ended up with lots of problems relating to synchronisation between the data files and the audio and video files.

After talking to other researchers in the field, and starting formal collaboration with some other groups, we realised that nobody seemed to have a general solution to store their experimental data coherently and consistently. This was the reason I started sketching out a suggestion for the Gesture Description Interchange Format (GDIF), of which a first proposal was presented at the Conference on New Interfaces for Musical Expression (NIME) in 2006 (Jensenius, et al., 2006). Over the last years several prototype setups have been developed to test out GDIF ideas in practice: using hand movements to control sound spatialisation in realtime (Marshall, et al., 2006), analysing violin performance (Maestre, et al., 2007), and using multilayered GDIF streams in a networked setup (Jensenius, 2007). GDIF was also discussed in a workshop of the EU COST Action 287 ConGAS4

in Oslo in December 2006, and was discussed in a panel session on music-related movement formats at the International Computer Music Conference (ICMC) in Copenhagen in August 2007 (Jensenius, et al., 2007).

A number of formats exist that are related to, and cover parts of, our needs to stream and store music-related movement data. Some of these formats are directly linked to the motion capture systems they are accompanying, such as the AOA format used with the optical tracker systems from Adaptive Optics, the BRD format used with the Flock of Birds electromagnetic trackers, and C3D used with the Vicon infrared motion capture systems. Of these three, C3D seems to have emerged as the de facto standard format for many motion capture applications. There are also several formats that have been developed for using motion capture data in animation tools and software, such as the BVA and BVH formats from Biovision, the ASF and AMC formats from game developer Acclaim (Lander, 1998), and the CSM format used by Autodesk 3ds Max.

Some of these motion capture formats are used in our community of researchers on music-related movement, but often these formats create more problems than they solve. One problem is that they are proprietary and often closely connected to a specific hardware system and/or software solution, which makes them inflexible and impractical for our needs. Another problem is that they often focus on full-body motion descriptors, based on a fully articulated skeleton. This may not always be convenient in setups where we are only interested in a specific part of the body, or if unspecified objects (e.g. an instrument) is also being recorded. Yet another problem is that most of these standards are intended to store only raw data and/or basic movement descriptors, which leaves little room to store analytical results and annotations, as well as various types of music-related data (audio, video, MIDI, OSC, notation, etc.).

Another type of format that could be interesting for our needs is the large number of XML-based formats used in motion capture and animation, for example the Motion Capture Markup Language (MCML—Chung & Lee, 2004), Avatar Markup Language (AML—Kshirsagar, et al., 2002), Sign Language Markup Language (Elliott, et al., 2000), Multimodal Presentation Markup Language (MPML—Prendinger, et al., 2004), Affective Presentation Markup language (APML—De Carolis, et al., 2004), Multimodal Utterance Representation Markup Language (MURML—Kranstedt, et al., 2002), Virtual Human Markup Language (VHML—Beard & Reid, 2002), etc. Again, none of these languages are directly useful for our needs, but there are certainly interesting parts that could be used and which should be evaluated more closely.

The MPEG series of formats and standards are also relevant. Developed by the Motion Picture Expert Group (MPEG), these standards have gained a strong commercial position and wide distribution, and allow for audio and video compression (MPEG-1, MPEG-2, and MPEG-4—Hartmann, et al., 2002), as well as for storing metadata and annotations (MPEG-7—Martinez, et al., 2002; Manjunath, et al., 2002). While these standards could potentially solve many of our storage problems, including synchronisation with audio and video, MPEG-7 lack a number of features that are vital for our needs.

When it comes to streaming and storing low-level movement data, the Gesture Motion Signal (GMS) format is particularly interesting. GMS is a binary format based on the Interchange File Format (IFF) standard (Morrison, 1985), and was developed by the ACROE group in Grenoble and partners in the EU Enactive Network of Excellence. GMS provides a solution for structuring, storing, and streaming basic movement data (Evrard, et al., 2006; Luciani, et al., 2006), but unfortunately such files do not make it possible to include mid- and higher level features. This, on the other hand, has been the focus in the development of the Performance Markup Language (PML—McGilvray, 2007), and the Music and Gesture File

(MGF—Pullinger, et al., 2008) formats developed at the University of Glasgow. PML is an extension to the Music Encoding Initiative (MEI—Roland, 2002), and the main idea is to facilitate the analysis and representation of performance in the context of the musical score. PML allows the description of performance events and analytical structures with reference to the score and external sources such as audio files. The current version uses MusicXML as a basis for score representation, though PML can be used with any XML based score representation.

Finally, we should not forget the Sound Description Interchange Format (SDIF), the inspiration for GDIF. SDIF was developed jointly by CNMAT at University of California, Berkeley, IRCAM in Paris and the Music Technology Group at Pompeu Fabra University in the late 1990s, as a solution to standardise the way audio analysis is stored (Wright, et al., 1998). Although SDIF may not be the most well known format, it is supported and used in several audio programs and programming environments (Schwarz & Wright, 2000). Most importantly, the SDIF specification and software implementations have tackled a number of challenges relating to the synchronisation of multiple streams of high-speed data, and, by necessity, it ensures synchronisation between audio and other types of data. That is also the reason why GDIF development has focused on being an add-on to SDIF. That said, GDIF development is mainly focusing on laying the ground for creating structured representations of music-related movement at various levels, and an XML-based implementation is certainly something that is feasible (and also partially implemented, see Maestre, et al., 2007).

NEEDS

We have a number of different needs when creating solutions to stream and store music-related movement data, some of which will be presented in this section. First, we need a structured and coherent approach to handling a large number of different types of raw movement data and associated analytical results, including:

- **Raw Data:** Unprocessed data from sensing devices, for example motion capture systems, game controllers, MIDI devices, and custom built controllers.
- **Pre-Processed Data:** Filtered, scaled and normalised data that are the basis for further analysis.
- **Analytical Results:** The data resulting from both quantitative and qualitative analysis of the above mentioned data.

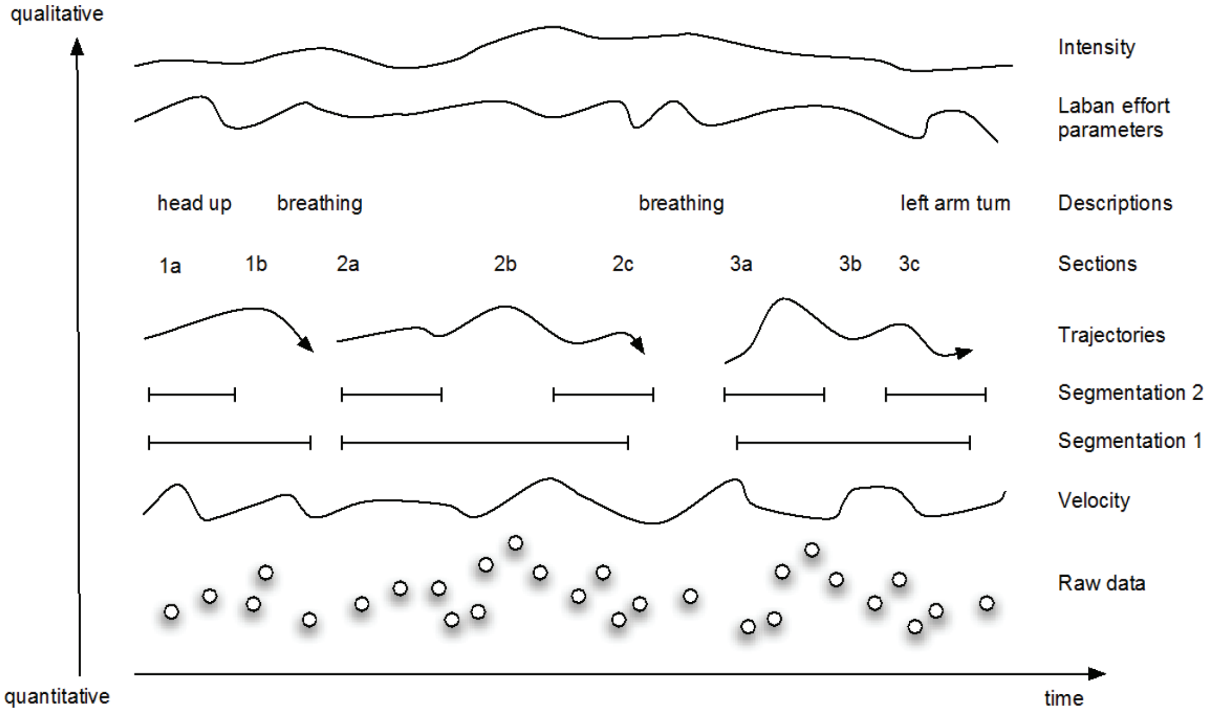
The idea is that all these data may be streamed and stored next to each other, as illustrated in Figure 1. A GDIF file may therefore contain a number of different time-synchronised streams so that it is possible to retrieve all the data from any point in time. It should also be possible to add new data streams resulting from various types of analyses, something that will allow different researchers to work on the same material.

Different Types of Raw Data

Let us start by looking at some of the different types of raw data typically being used in music-related movement research, all having different features and properties:

- **Motion Capture Systems:** Such systems usually output data at high speeds (up to 4000 Hz) for a number of multidimensional markers, anything from 2 to 50, sometimes even more. Each of the markers can be tracked with 3 or 6 degrees of freedom; position in XYZ coordinates and sometimes also orientation (azimuth, elevation, roll). Many commercial motion capture systems use their own proprietary software and formats to retrieve and store the data, although several seem to be able to export data in the C3D format.

Figure 1. A sketch of the different types of raw data and analytical results that could be streamed and stored in GDIF



- **MIDI Devices:** Most commercial music controllers and instruments output MIDI. Since the MIDI standard is an event-based protocol, there is no information about the movements associated with the control messages. As such, MIDI is not particularly suited for streaming or storing continuous movement related data, but due to its widespread use, we need to find solutions to include data from such devices. This should not be too problematic since the MIDI specification is clear, and the messages are well defined. MIDI streams are typically output at fairly low sampling rates (up to a 100 Hz), and low bit-rates (7-bit).
- **Generic Controllers:** Game controllers, graphical tablets, and other commercial devices often use well-known protocols (e.g. Human Interface Device—HID) and use more or less well-defined ranges and resolutions. As is the case for MIDI devices, such general input devices mainly focus on control aspects, and not the body movements associated with these control messages, although the continuous control of a joystick may provide valuable information about the movement of the hand that is used to control the device.
- **Custom-Made Controllers and Instruments:** Such devices are more problematic to make general specifications for, since they often only exist as a single prototype. That said, custom-made controllers are often built with standard sensors, and use sensor interfaces based on a well-known protocols for data transfer (e.g. MIDI or Open Sound Control—OSC), which may be used as a starting point for a general specification.

Most of these devices have some sort of native format and messaging structure that should be preserved in later markup for backwards compatibility. This is also important in experiments,

where it may be necessary to check the raw data in case there are problems or uncertainties with the formatting or analysis resulting from the markup in a new format.

Time and Synchronisation

One of the major challenges when it comes to finding a solution to structuring music-related movement, is that of *time coding* and *synchronisation*. Only structuring the raw data as mentioned above, poses a number of challenges when it comes to different sampling rates, number of streams, etc., and this is complicated further by the necessity to maintain time coding within streams, and synchronisation between streams of data.

First, time coding data that come from commercial motion capture systems that claim to operate with a "fixed" sampling rate should be an easy task. However, we have experienced that many of these systems are not as accurate as they claim to be, and there may be a substantial drift over time. This is particularly evident when recording data from high-speed systems that demand a lot of computer memory, disk access, and CPU processing. To keep up with the idea of being able to get back to the original raw data, our solution is to store both the time code from the device itself, along with a global time code for our own research setup.

Another challenge related to time coding inside streams is that several devices (for example MIDI) operate with delta time, based on specifying events relative to the preceding and/or succeeding events. The messages from such devices are typically sent at an uneven rate, transmitting only when they are active. Several challenges arise when synchronising data streams from devices using relative time coding with data sets using a fixed sample rate, and it is necessary to find a solution to handling such a time coding consistently. Here an important question is whether all streams should be time coded and sampled with the same time code. This would allow for easier synchronisation, but it would probably also lead to redundant information being stored. For example, if data from a MIDI device is to be synchronised with data from a high speed motion capture system, the MIDI data would have to be recorded at a much faster rate than they are output from the MIDI device. Again, our solution is to allow each stream to operate with its own time coding, and create a system to secure synchronisation between the streams.

A third challenge is related to synchronising the various data streams with simultaneously recorded audio and video. Fortunately, there are several industry standards for both audio and video time coding and synchronisation, one of the most commonly used being the SMPTE time code. Using a video time code as the basis for time coding data in GDIF may not be the best option, though, since several of these time codes are based on a combination of time and video frames. In addition, the question of time code is related to the media formats and codecs being used. A research format that is intended to live for a while should not rely on specific audio and video formats/codecs (e.g. QuickTime, Windows Media Audio/Video, etc.), so we need to find a more generic solution to synchronise such media with the movement data.

Finally, we also have to tackle challenges related to synchronising data and media that are not based on a specific time code. Music is largely based on relative timing, often defined by a musical score, and different performances of the same piece often vary considerably in duration and timing. It is therefore important to develop solutions to synchronise the absolute or delta time codes of movement data, with the relative time codes of musical scores. This will probably have to be based on solutions to "time warp" data sets, or set musical "keyframes" that define synchronisation points.

REQUIREMENTS

When it comes to the format itself, there are several criteria that are important to make it versatile and useful in music-related movement research. The format should be:

- **Open:** The specification and the implementations should be free and open (as in open source) so that everyone can use them and expand them as they see fit.
- **Human-friendly:** It should be possible to easily understand what the data means by looking at the data stream or file, or there should be converters readily available if storing in a binary format.
- **Multiplatform:** Cross-platform and cross-software support is essential to ensure compatibility with systems past, present, and future.
- **Compatible:** It is important to ensure that the format will work with other relevant formats and standards. As far as possible, it should also be based on other well-known standards to allow for easy adaptation.
- **Simple:** The basic set of descriptors should be as small and simple as possible so that it is easy to get started using the format.
- **Extendable:** The format should allow for a great deal of flexibility when extending it to cover new areas that were not thought of in the original specification.
- **Efficient:** Since the main focus here is that of offline research, the above mentioned elements are the most important, and many of these contradict efficiency. That said, efficiency will also be a goal as long as it does not come in conflict with the other aims.

These requirements have been the basis for our choice of solutions and implementations that will be described in the following sections.

SOLUTION: A MULTILAYERED APPROACH

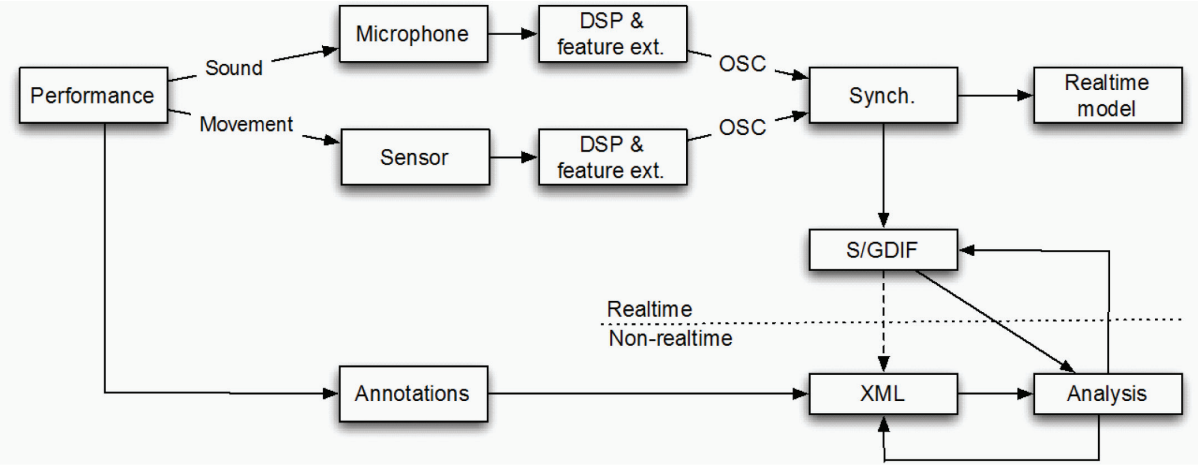
While many data formats often focus only on storage, we believe it is important to create a solution that allows for both streaming and storage. This allows for using the format not only for experimental setups, but also as a tool for realtime control in installation or performance. In fact, even though streaming and storage may seem as two widely different domains, they are often closely connected when working with music, since both sound and movement unfold in time. Streaming data from a controller and mapping these data to features in a music engine, involves many of the same types of processing and analyses as recording data to a file. In addition, since we see GDIF development as much about conceptualisation as it is about computer formats, both streaming and storage should be included in the process.

Streaming and Storage

GDIF is currently being developed in multiple directions: an implementation for realtime control of sound synthesis, and two different storage solutions, which is sketched out in Figure 2. The idea is that features from the recorded sound and sensor data will be converted to OSC streams and passed on using network communication (typically UDP/IP or TCP/IP). These OSC streams can then be mapped directly for use in a realtime model, or they can be passed on for storage.

For storing data from the OSC streams we are currently using the IRCAM's FTM library for the graphical programming environment Max/MSP/Jitter (Schnell, et al., 2005) to record files using the SDIF specification. This allows for synchronisation with audio, audio analysis and MIDI, and makes it possible to use the analytical tools available to SDIF to analyse GDIF data. For a more structured approach, and to be compatible with other software solutions, we have also developed

Figure 2. Sketch of a setup using a GDIF OSC namespace for streaming, and storing these GDIF streams in SDIF files or structured XML files



solutions for storing GDIF data in XML files. An example of this, in the context of analysing the movements in violin performance is discussed in Maestre et al. (2007).

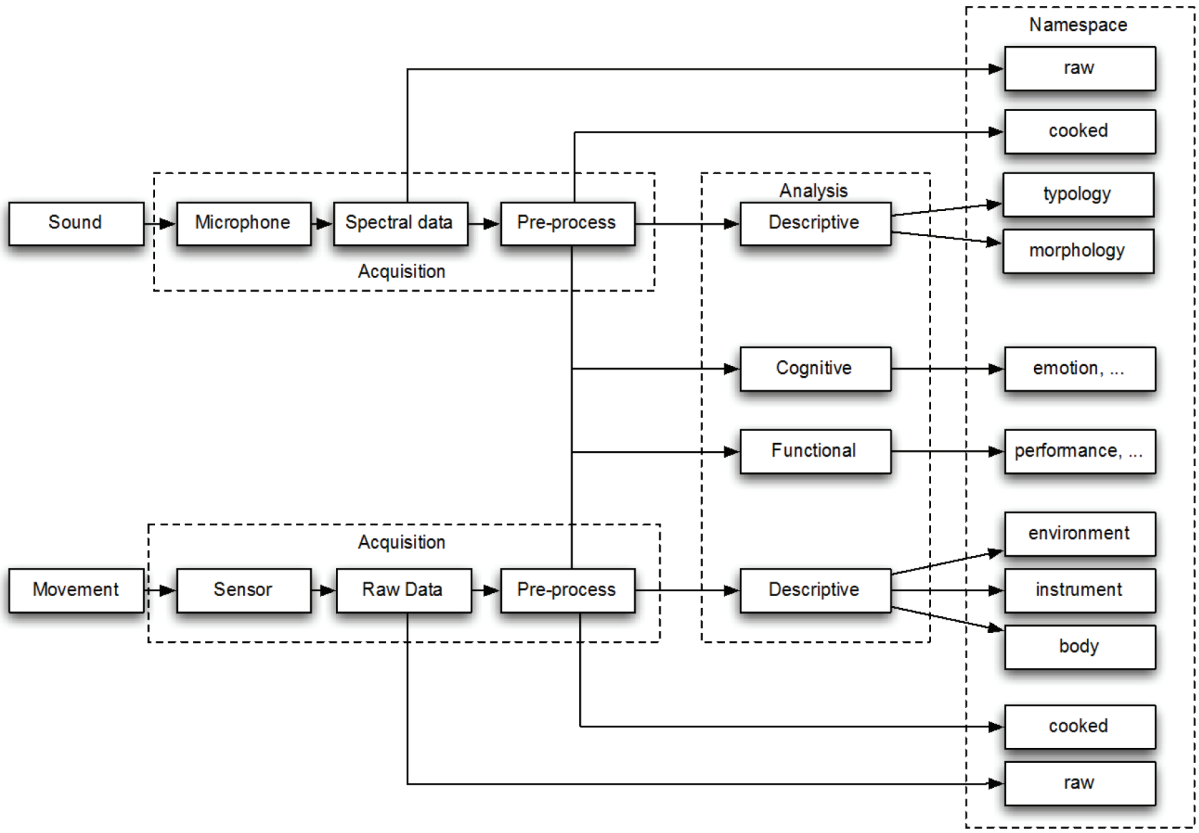
The streaming/storage solution suggested in Figure 2 is but one of many ways to stream and store the data. In fact, we believe that the biggest challenge in future research is to work on the descriptors and specification of the content to be stored. For this reason, we have been focusing on understanding how different types of data are related, and how we can create solutions to structure them. An important starting point here is to differentiate between the different levels of data. Both practically and conceptually, it helps if we manage to separate between raw data coming from sensors, pre-processed data and the data resulting from various types of qualitative or quantitative analyses. A sketch showing one approach to this and a suggested namespace to use in OSC is shown in Figure 3. In this example, raw and pre-processed data of both sound and movement are separated into different streams called raw and cooked. Then a descriptive layer is suggested for handling analysed data that are directly referring to features in the sound and movement that can be described using techniques such as Laban Movement Analysis (LMA) (Laban, 1980), or features from Schaeffer's typomorphological structures (Schaeffer, 1966; Godøy, 2006). The structure and properties of the different layers will be discussed in the following sections. The organisation of sound features will be left out in this discussion, and have just been included in the sketch to show that mid- and high-level features usually have to be considered from a multimodal perspective.

Acquisition Layers

The acquisition layers in Figure 3 refer to pre-analysed data: raw data coming from the device (raw) and pre-processed data (cooked). In the following discussion OSC-style namespaces are used as examples since this provides for a more compact notation, but the structures may also easily be formatted using XML tags. An important thing to remember is that all layers suggested here are optional and may be skipped if there is no need for them. The idea of having multiple layers is not to create large sets of redundant messaging, but rather suggest a structure for the way different types and levels of data could be communicated.

The raw layer is used to pass on raw data coming from the device. Here the idea is that the namespace should be constructed so that it reflects

Figure 3. Sketch of the multilayered approach suggested for the GDIF OSC namespace



the way the data was received from the device. Game controllers, MIDI devices and other types of commercial controllers all have more or less well defined messages that can easily be turned into raw messages. For example, the namespace for a simple controller could be:

```
/raw/name/<controller number> <value>
```

Similarly, data from a MIDI keyboard could be coded as:

```
/raw/keyboard/<note> <velocity>
<channel>
```

It is more difficult when it comes to data from custom-built controllers, in which the values often depend on the sensor interface being used. Here it might be relevant to code the strings from either the device, the sensor interface or the sensor:

```
/raw
        /device/<sensor number>
<value>
        /sensor-interface/<sensor
number> <value>
        /sensor/<sensor name> <value>
```

This could result in messages like:

```
/raw
      /cheapstick/2 32
      /teabox/4 255
      /sensor/accelerometer 24 54 40
```

Such messages may not be particularly useful without some extra information about the device, but, as will be described later, the idea is to allow metadata to be available in the system to provide information about what the data mean.

The structure of the cooked layer is similar to the raw layer, except that the data may have been filtered (for example using a low-pass filter to remove noise in the signal), and/or scaled to a useful range. For example, data from a joystick may be coded as:

```
/cooked/joystick/<controller number>
<0. - 1.>
```

Descriptive Layers

The descriptive layers are used for structuring data resulting from the analysis of the raw and cooked data layers. A first step here is often to clearly define which analytical perspective is used when describing data, and in the following, we will look at this from three separate layers: device, body, and environment.

Device Layer

The device layer represents data analysed with respect to the sensor, instrument, or controller used, totally independent of the performer and the performer's movements. The idea is that the device layer should be fairly generic, in the sense that devices with similar properties should be coded in the same way. This could be accomplished by creating a "device library," a collection of standardised namespaces for devices that share similar properties. For example, even though most commercial joysticks have a varying number of buttons and look differently, they still behave more or less similarly. Thus, it is possible to create generic device specifications for various popular devices, for examples joysticks, gamepads, mice, graphical tablets, etc. These specifications can then be used as the basis for all devices that have a similar (but not necessarily identical) functionality. This will allow for more flexibility and the possibility to interchange devices without having to worry about whether the raw data they output are identical.

One question in the device layer is how the values should be grouped. For example, the raw data from a joystick is usually transmitted as separate strings containing a control number and the associated value. But sometimes several such control numbers may be linked and dependent on each other, such as the two control numbers used to describe the tilt (x,y) of a joystick, and the rotation of the stick. Should all these messages be passed together in one message, or be split into multiple messages? Wright et al. (2001) argued for the former, and suggested that data from a joystick could be represented as:

```
/joystick/b xtilt ytilt rotation ...
```

when button "b" is pressed. Here they even included information about the buttons in a long string containing a number of different messages. This is a computationally efficient way of passing the data, but the problem is that the end user will not necessarily know what the different values mean. A more verbose and human-friendly, although less computationally efficient, way of structuring the message could be:

```
/device
      /joystick/tilt <x, y>
      /joystick/rotation <-1. - 1.>
```

An even more structured approach would be to pass on all messages as separate strings:

```
/device
      /joystick/button/b <1/0>
      /joystick/tilt/x <-1. - 1.>
      /joystick/tilt/y <-1. - 1.>
      /joystick/rotation <-1. - 1.>
```

This makes for more readable messages, and makes it very easy to parse the messages, but it will increase the traffic and computational load in the system, since the number of communicated bits is much higher than for the combined messages.

Ranges and Units

Notice that the range used in the above example is -1.–1. This is because a joystick will typically have a centre position around which it can be tilted and rotated in either direction. Using a -1.–1. range will secure that the centre position is at (0, 0), which seems more intuitive than having a centre position at (0.5, 0.5). Notice also that the above example is hierarchically organised around the functional aspect (tilt) of the device, rather than the dimensional (x,y). In some cases, however, it may be better to do it the other way around, for example:

```
/device
        /joystick/x/tilt <-1. - 1.>
        /joystick/y/tilt <-1. - 1.>
```

Such structural differences should be allowed, as long as they are clearly documented and implemented. However, we should be careful about allowing for too many different solutions, since one of the main goals of GDIF is to create a specification that will standardise the way messages are communicated.

An important issue in handling music-related movement data is the way units are used and defined. As far as possible it should be advised to use meaningful units, if they are known. However, when this is not possible, it is probably better to use a generic normalised range like 0.–1. or -1.–1 than sticking with some arbitrary range defined by the bit rate. For example, in the case of a joystick it may be better to convert the 8-bit (0-255) range returned from the HID protocol to a degrees when describing the rotation of a joystick:

```
/device/joystick/rotation <0 - 360>
```

The next question is how the unit should be described. Should it be part of the message:

```
/device/joystick/rotation <0 - 360>
degrees


/device/joystick/rotation/unit de-
grees
/device/joystick/rotation <0 - 360>
```

Again, this may depend on the usage. In a system where the units are fixed and well defined, the latter solution could be useful. However, other times it may well be that it could be useful to switch between Cartesian and polar coordinates, something which would make the former example a better choice.

Body Layer

The body layer describes the body and the body's movement in relation to the device, controller or instrument, as seen "through the eyes" of the performer. While the device layer describes activity in terms of the different elements of the device (e.g. joystick and buttons), the body layer focuses on movement of parts of the body (e.g. hands and fingers). Such body movement may sometimes be directly inferred from the raw sensor data (e.g. using a joystick), while other times they will have to be more loosely described. An example namespace of the movements associated with a joystick could be:

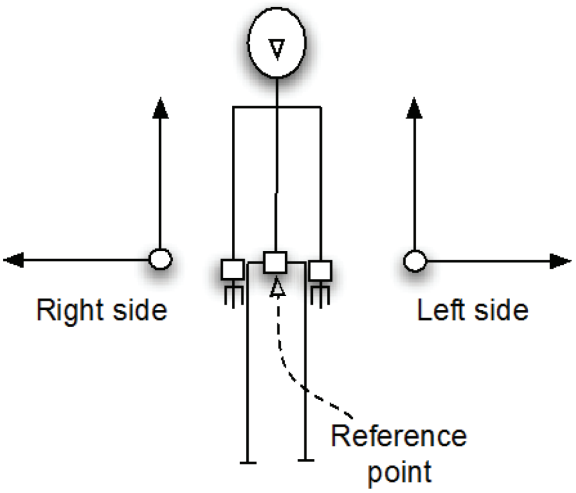
```
/body
     /hand/right/finger/2/press <1/0>
     /hand/right/location/horizontal
<-1. - 1.>
```

Notice that the namespace only refers to the body, and no information is available about the device. This opens for a more intuitive mapping process, and may allow for devices with similar movement properties to be exchanged unobtrusively.

There are a number of issues that will need to be addressed when developing the specification of the body layer. First, what type of coordinate system(s) should be used to describe the body? What type of naming convention(s) should be used? Should values be defined absolute or relative to a specific point in the body? If so, where should the reference point be? For a prototype setup of control of spatialisation with hand movements (as described in Marshall et al., 2006), we used a mirrored body model as shown in Figure 4. Here a positive movement space was defined for each hand in the forward, upward, outward quadrant of the body, which is typically where most of the movement takes place. The result was positive values when subjects moved their arms outwards, to the front, and upwards, something which simplified the mapping process in the setup.

One reason why the mirrored body model turned out to be so easy to work with is probably because it reflects the way we think and talk about our body. For example, we often think about an arm movement as "outwards," and not "left" or "right." Thus, creating such perceptually relevant namespaces may allow for a more intuitive mapping process. That said, further development of the body layer should probably also look at various types of biomechanical models used in motion capture systems, and see how they can fit into such a perceptual and descriptive paradigm.

Figure 4. Sketch of the mirrored and human-focused body model used in a prototype setup to control sound spatialisation with hand movements (Marshall, et al., 2006). Notice that the body is split in two, and two positive quadrants are defined on either side.



Environment Layer

The *environment* layer is intended to code information about the relationship between various bodies and devices in a space. For example, in the prototype spatialisation setup described above, the orientation of the person was used to control the location of sound in the space. Further development of this layer can be done in collaboration with the Spatialisation Description Interchange Format (SpatDIF) developed at McGill University (Peters, 2008).

Functional Layers

The *functional* layers will be independent of both the technical and body-related descriptors presented above, and focus on describing the functional aspects of the movement. This can be based on the taxonomy of music-related movements presented as: *sound-producing*, *sound-facilitating*, *sound-accompanying* and *communicative* (Jensenius, et al., 2009). It can also be based on traditional music performance vocabulary, such as describing the dynamics of the performance (e.g. *ppp – fff*) or the playing style (e.g. *staccato* or *legato*). There is still a lot of conceptual work to be done before these layers can be specified, and it could be interesting to investigate how the structure from the *Performance Markup Language* (PML) could be used to create suitable namespaces.

Cognitive Layers

The *cognitive* layers are intended to store data relating to higher-level features, for example abstract representations, metaphors, and emotional states. In many ways, this level is the most interesting when it comes to understanding and representing our experience of music, but it is also one of the most challenging to formalise and create a consistent structure of. For example, a metaphor like "intensity" may be used to describe music verbally, but it is not a trivial task to develop taxonomy for such a descriptor, and relate it to the low-level features of musical sound and movement. There is a growing body of research on these topics that could form the basis for creating relevant descriptors, for example by Bresin and Friberg (2000) and Camurri et al. (2003). They have been working on systems to analyse and model "expressive gestures" based on emotion parameters such as anger, fear, grief, and joy. So further development need to be done in close collaboration with researchers working on these topics.

PROTOTYPE IMPLEMENTATIONS

Over the last years we have developed a number of prototype setups using GDIF, and we shall here look at two of these, one for the recording of music-related movements in lab experiments, and another for a simple mapping example. These examples should also help to illustrate some of the benefits, but also challenges, that we are facing in future development of formats for handling music-related body movement.

A Multilayered Setup for Motion Capture Studies

We are currently working on explorations of coarticulation and goal-points in music performance (Godøy, et al., 2008), and have created a laboratory setup using a multilayered GDIF-based recording system (Jensenius, et al., 2008). This modular setup, developed using components and modules in the Jamoma open source project for Max/MSP/Jitter (Place & Lossius, 2006), allows for quickly setting up observation studies with various combinations of motion capture equipment, video cameras and audio devices.

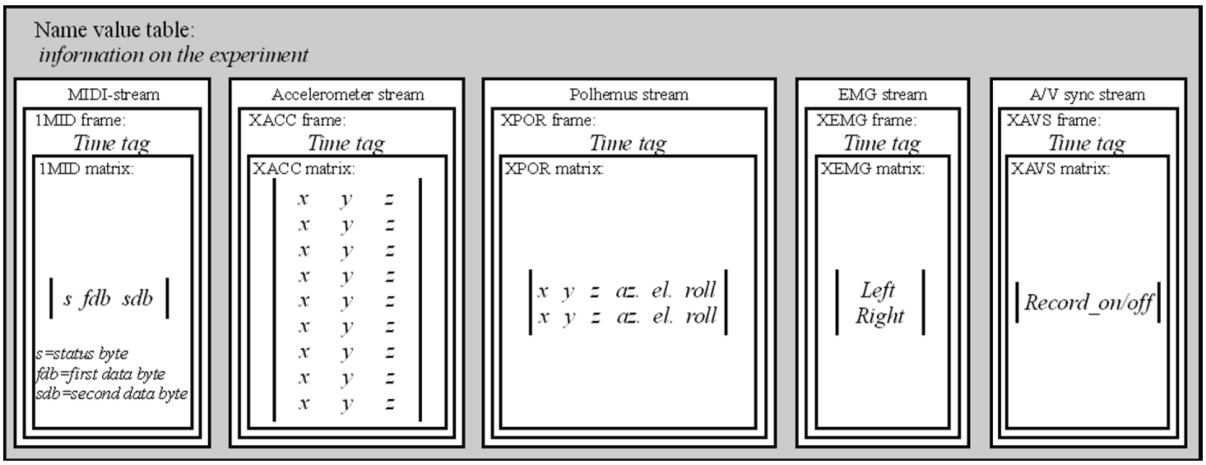
Table 1 shows an overview of some of the input devices we typically work within our setups, the number of values recorded, and the sampling and bit rates used. All of these are very different in nature, yet we still need to be able to record everything in a synchronized and structured manner.

Data from each of the devices (except for audio and video) are being converted to OSC streams at the first possible stage, and passed over the network to the master computer writing one combined SDIF file. As sketched in Figure 5, the

Table 1. List of data used in our setup, columns from left: input device, sampling rate in Hz, number of channels, resolution in bits, approximate bitrate in kbps

Input	# values	SR (Hz)	Bit
Optitrack IR motion capture	3-150	100	12
Phidgets accelerometers	3-30	60	32
Polhemus electromagnetic tracker	12	60	32
Biosensors	1-10	100	7
Video	1-3	86	8
Audio	2-8	44100	16
MIDI	3-20	100−1000	7

Figure 5. Sketch of GDIF data written to a SDIF file. The various streams consist of different types of frames and matrices, each of which have different resolution and sampling rates.



SDIF file contains separate *streams* recorded along a common timeline, where each stream consists of time-tagged *frames* within which the actual data are stored as *matrices* (Wright, et al., 1998). The frames and matrices are type-specific, meaning that different frame types consist of a different number of matrices, and different matrix types consist of a different number of rows and columns. This allows for handling data at various resolutions and speeds while still being synchronised.

The data are referred to with a set of custom defined frame and matrix types. For storing position and orientation data, we have defined three different matrix types:

```
1MTD XPOS { X, Y, Z }
 1FTD XPOS { XPOS position; }
1MTD XORI { azimuth, elevation, roll
}
 1FTD XORI { XORI orientation; }
1MTD XPOR { X, Y, Z, azimuth, eleva-
tion, roll}
 1FTD XPOR { XPOR position_and_orien-
tation; }
```

In the future, it may also be desirable to open for position descriptions using polar coordinates, but for the moment, we are using Cartesian coordinates for the descriptions of points in space. We have also found the need for storing both velocity and acceleration data, and have defined two matrix types for this:

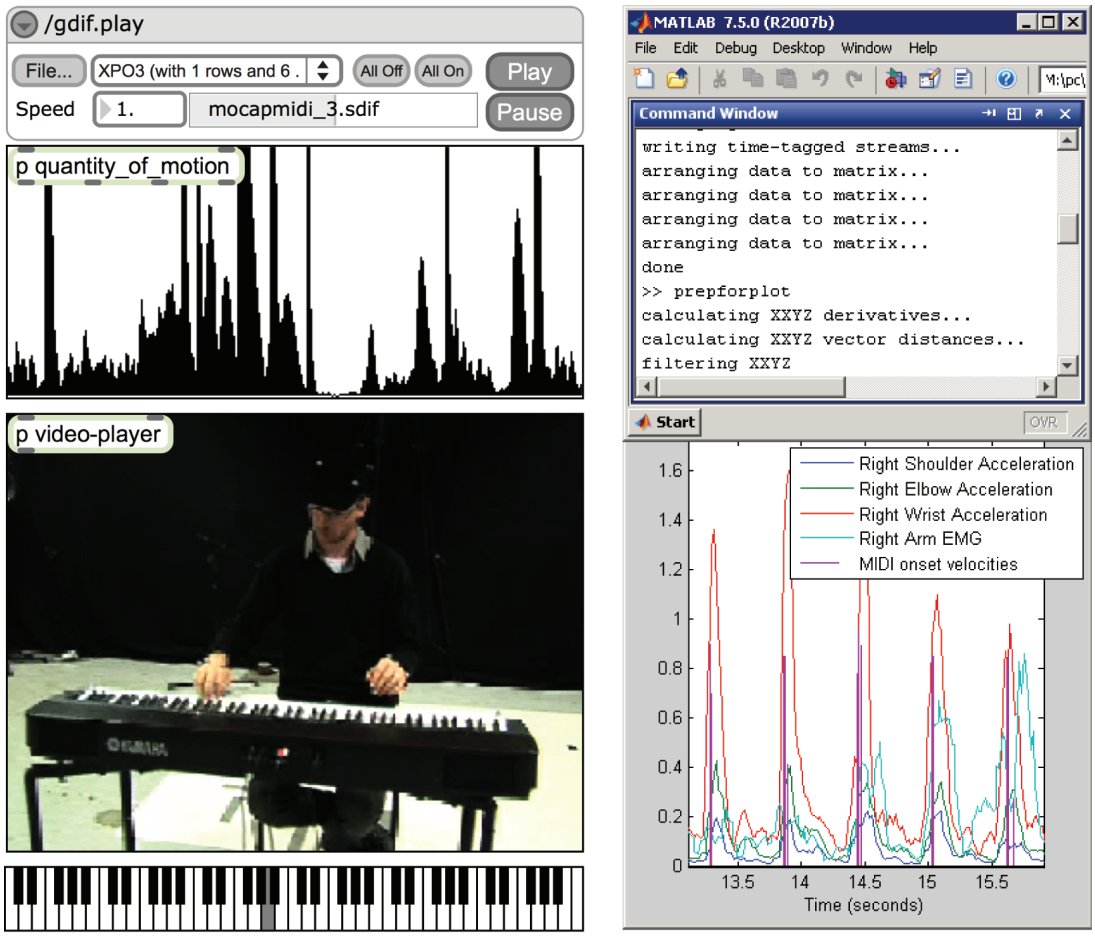
```
1MTD XVEL { X, Y, Z }
1FTD XVEL { XVEL velocity; }
1MTD XACC { X, Y, Z }
1FTD XACC { XACC acceleration; }
```

Since audio and video files are recorded on a separate computer, we use a synchronisation stream containing a binary on/off message recorded into a stream called XAVS:

```
1MTD XAVS { record_on }
 1FTD XAVS { XAVS audio_video_sync;
}
```

The end result is a structured SDIF/GDIF file where all data is tightly synchronized, and where the A/V sync stream also makes it possible to play back the data together with the recorded audio and video files. With custom built modules for Max/MSP and a toolbox for Matlab, it is possible to both record and playback data and media in Max/MSP and carry out non-realtime analyses in Matlab (Figure 6).

Figure 6. Example of SDIF/GDIF module in Max/MSP playing back recorded accelerometer data synchronised with video (left). Matlab script for reading SDIF/GDIF files and plotting the data.



Mouse Mapping

Mapping between controller inputs to sound engine parameters is one of the most important aspects when creating new electronic instruments. While this may be hard-coded in either hardware or software, taking a more structured approach could greatly help in improving the time spent when creating mappings, but also the perceptual quality of the mappings being made. This section will briefly present an example of how a GDIF-based OSC namespace can help in setting up mappings from a computer mouse to a sound engine.

Rather than using the raw data coming from a computer mouse for setting up a mapping, we can define a set of descriptive layers that are more perceptually relevant from a user's point of view. As mentioned above, the device layer is focused around the functionality and elements of the device. The device layer for a mouse can be implemented with the following namespace:

```
/gdif/device
/mouse/button/1/press <1/0>
/mouse/location/horizontal <-1. - 1.>
/mouse/location/vertical <-1. - 1.>
```

Except for the different naming, this layer could be seen as fairly similar to how a raw and cooked layer could look like. One important difference is that here the range is normalised to a range -1.–1., where 0,0 is in the middle of the screen. This could be seen as more useful than for example having screen pixel values originating at the top left corner, which is the values being sent from the operating system.

The body descriptive layer, which describes the values from the perspective of the performer using the device, could be implemented with the following namespace:

```
/gdif/body
/hand/right/finger/2/press <1/0>
/hand/right/location/horizontal <-1.
- 1.>
/hand/right/location/vertical <-1. -
1.>
/hand/right/motion/quantity <-1. -
1.>
/hand/right/motion/direction <0. -
360.>
```

Notice how the namespace is built progressively from larger to smaller features (body, hand, finger), and that the specifications of the hand (right) and finger (2) are parts of the namespace. Here numbers are used to indicate the fingers, but we should work towards a better and more uniform terminology of the various parts of the body. As the mapping example in the next section will show, this allows for wildcards when parsing the values. This body layer will necessarily have to be subjective and context-dependent, so here the hand and finger that I use with the mouse have been chosen. Regarding the device layer, I also find it practical to use a -1.–1. range with the home position in the middle (0,0) of the movement space. In addition to the location, the body layer also displays the direction and quantity of motion, since it is often more interesting to use such movement data for control purposes. This is yet another way of making the returned values less dependent on the specific device being used, and rather focus on properties of the movement.

Header Information

All of the above mentioned data types and streams refer to dynamic data. However, it could also be useful to have access to "static" information that can be stored as header information at the beginning of a file, or be queried for in realtime. First, general information about the location and the setup should be defined, for example:

```
/general
/date 2009.01.01
/time 12:44:00 CET
/location "FourMs lab, University of
Oslo"
/author "Alexander Refsum Jensenius"
/experiment "Mouse-Joystick"
/session 1
/devices "Trust GM-4200 and Saitek
ST290"
/description         "Testing if us-
ers prefer a mouse over a
joystick in the control of a VST soft
synth running in Max."
```

It may also be necessary to include more specific information about the devices listed in the general information. They could be defined based on the generic device type they represent, such as:

```
/device
/type mouse
/name GM-4200
/manufacturer Trust
/id S12116847
/dof 3
/buttons 5
/description "A right handed computer
mouse"
```

The idea is that all relevant information should be available at any time. In realtime situations, this may allow for networked setups in which a new user could browse the network for available devices, find their properties, and set them up for control. These data could also be written in the headers of the stored files, so that it is possible to identify what the data mean, how they were recorded, and which devices they were recorded from.

Flexible and Meaningful Mapping

One of the main ideas behind the realtime implementation of GDIF is to allow for the quicker and easier creation of mappings between controllers and sound engines. The mapping system suggested by Malloch et al. (2007) is a good example of how such a structured approach could simplify the mapping process. Here the idea is to use a mapping patch that will automatically list all the available devices and their namespaces, and allow for mapping between controllers and sound engines by dragging lines between the various parameters.

The possibility to create flexible mappings is important, but even more important is to work towards solutions for creating meaningful mappings. This is something, which was explored in ESCHER, a modular system where the descriptions of the movements carried out with a controller are separate from the parameters of the synthesis engine, and where mappings are created with an intermediate abstract parameter level between the movement and the sound parameter levels (Wanderley, et al., 1998). These ideas have later been developed into models focusing on using several perceptual mapping layers: one layer for movements, one for sound, and one between movement and sound (Hunt & Wanderley, 2002; Arfib, et al., 2002). This makes it possible to create mappings between actions and sounds based on perceptually relevant features, rather than technical parameters.

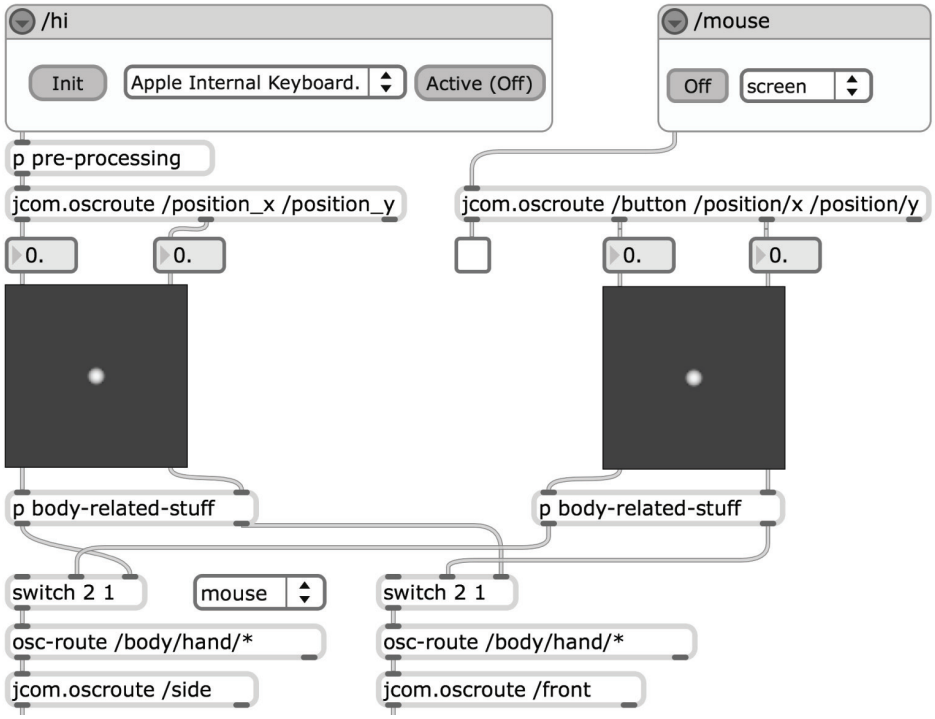
Creating mappings based on meaningful parameters would be closer to how composers and performers usually think about movement and sound, for example in relation to the body: "I want a forward movement of my right hand to control the brightness of the sound" or "I want to control granularity in the sound with this gentle finger tapping." Rather than forcing everyone into the constraints of the technical system, we should try to create setups that make it possible for mappings to be created at such a meaningful level. If the idea is to have a forward movement in the right hand to control the brightness of the sound, it might not be important whether that movement is done with a bow, a joystick or a custom built controller, since the quality of the movement will be the same. If the mappings were created at this level, it would also be possible to retain the mappings even though either the controller or the sound system (or both) were changed.

A very simple example of such a meaningful mapping process is shown in the Max/MSP patch shown in Figure 7. This example patch retrieves data from two devices, a computer mouse and a joystick, and codes the data into body layer GDIF streams. The point of the patch is to show that it is possible to use either device to control a sound synthesis model, and to switch between them without having to create any new mappings between the control values and the parameters in the sound synthesis model. This is because the mappings are created at a body level, and only refer to the movement of the hand used in the interaction. Here both the mouse and joystick are controlled by the right hand, and they both allow for horizontal and vertical movement of the hand. It is therefore possible to use these body movement features to control the sound model. This is but a simple example, but similar meaningful mappings can also be developed between multi-dimensional controllers and sound engines.

CONCLUSION

The chapter has presented some general thoughts on challenges related to structuring music-related

Figure 7. An example of mapping movements from either mouse or joystick to a simple sound synthesis using a GDIF namespace. Since the values from both devices are coded using the same body-related descriptors, it is possible to switch between the devices without having to create new mappings. Note that a star (*) is used as a wildcard for choosing either left or right hand movements.



movement data, and also shown examples of how some of these thoughts have been tested in various prototype GDIF setups using both SDIF-files for storage and OSC streams for realtime control. GDIF development started out as an attempt to create a practical solution to store data from our observation studies, but we soon realised that the real challenge is not on the technical side but rather to create taxonomies of the different types and layers of movements that we are studying. This development will, obviously, have to mature with the continuous efforts of a number of research groups currently engaged in exploring music-related body movement.

Developing formats and tools can therefore be seen as an attempt to define a vocabulary for music-related movement in general. Fortunately, considering the many researchers that are currently involved in the field makes it possible that we may eventually succeed in creating a common format to stream and store music-related movement data.

ACKNOWLEDGMENT

Thanks to Rolf Inge Godøy and Marcelo M. Wanderley for supervising the PhD dissertation on which this chapter is based. Thanks to the GDIF community for further development of the thoughts presented in the chapter, and to Kristian Nymoen for the illustrations in Figures 5 and 6.

REFERENCES

Arfib, D., Couturier, J.-M., Kessours, L., & Verfaille, V. (2002). Strategies of mapping between gesture data and synthesis model parameters using perceptual spaces. *Organised Sound*, *7*(2), 135–152. doi:10.1017/S1355771802002054

Beard, S., & Reid, D. (2002). MetaFace and VHML: A first implementation of the virtual human markup language. In *Proceedings of the AAMAS Workshop on Embodied Conversational Agents-Let's Specify and Evaluate Them*. AAMAS.

Bresin, R., & Friberg, A. (2000). Emotional coloring of computer-controlled music performances. *Computer Music Journal*, *24*(4), 44–63. doi:10.1162/014892600559515

Camurri, A., Lagerlof, I., & Volpe, G. (2003). Recognizing emotion from dance movement: Comparison of spectator recognition and automated techniques. *International Journal of Human-Computer Studies*, *59*(1-2), 213–225. doi:10.1016/S1071-5819(03)00050-8

Chung, H., & Lee, Y. (2004). MCML: Motion capture markup language for integration of heterogeneous motion capture data. *Computer Standards & Interfaces*, *26*(2), 113–130. doi:10.1016/S0920-5489(03)00071-0

De Carolis, B., Pelachaud, C., Poggi, I., & Steedman, M. (2004). APML: A mark-up language for believable behavior generation . In Prendinger, H., & Ishizuka, M. (Eds.), *Life-Like Characters: Tools, Affective Functions and Applications* (pp. 65–85). Berlin, Germany: Springer-Verlag.

Elliott, R., Glauert, J. R. W., Kennaway, J. R., & Marshall, I. (2000). The development of language processing support for the visicast project. In *Proceedings of the Fourth International ACM Conference on Assistive Technologies*, (pp. 101–108). New York, NY: ACM Press.

Evrard, M., Couroussé, D., Castagné, N., Cadoz, C., Florens, J.-L., & Luciani, A. (2006). *The GMS file format: Specifications of the version 0.1 of the format*. Technical report. Grenoble, France: INPG, ACROE/ICA.

Godøy, R. I. (2004). Gestural imagery in the service of musical imagery. *Lecture Notes in Artificial Intelligence*, *2915*, 55–62.

Godøy, R. I. (2006). Gestural-sonorous objects: Embodied extensions of schaeffer's conceptual apparatus. *Organised Sound*, *11*(2), 149–157. doi:10.1017/S1355771806001439

Godøy, R. I., Jensenius, A. R., & Nymoen, K. (2008). *Production and perception of goal-points and coarticulations in music*. Paper presented at the ASA-EAA Conference. Paris, France.

Godøy, R. I., & Leman, M. (2009). *Musical gestures - Sound, movement, and meaning*. New York, NY: Routledge.

Hartmann, B., Mancini, M., & Pelachaud, C. (2002). Formational parameters and adaptive prototype instantiation for MPEG-4 compliant gesture synthesis. [Computer Animation.]. *Proceedings of Computer Animation*, *2002*, 111–119.

Hunt, A., & Wanderley, M. M. (2002). Mapping performer parameters to synthesis engines. *Organised Sound*, *7*(2), 97–108. doi:10.1017/S1355771802002030

Jazzmutant. (2006). *Extension and enchancement of the OSC protocol*. Paper presented at the OSC-meeting at NIME 2006, IRCAM. Paris, France.

Jensenius, A. R., Camurri, A., Castagne, N., Maestre, E., Malloch, J., & McGilvray, D. … Wright, M. (2007). Panel: The need of formats for streaming and storing music-related movement and gesture data. In *Proceedings of the 2007 International Computer Music Conference*, (pp. 13–16). Copenhagen, Denmark: International Music Conference.

Jensenius, A. R., Kvifte, T., & Godøy, R. I. (2006). Towards a gesture description interchange format. In N. Schnell, F. Bevilacqua, M. Lyons, & A. Tanaka (Eds.), In *Proceedings of the 2006 International Conference on New Interfaces for Musical Expression*, (pp. 176–179). Paris, France: IRCAM – Centre Pompidou.

Jensenius, A. R., Nymoen, K., & Godøy, R. I. (2008). A multilayered GDIF-based setup for studying coarticulation in the movements of musicians. In *Proceedings of the 2008 International Computer Music Conference*, (pp. 743–746). Belfast, Ireland: ACM.

Kranstedt, A., Kopp, S., & Wachsmuth, I. (2002). MURML: A multimodal utterance representation markup language for conversational agents. In *Proceedings of the AAMAS Workshop on Embodied Conversational Agents – Let's Specify and Evaluate Them*. AAMAS.

Kshirsagar, S., Magnenat-Thalmann, N., Guye-Vuillome, A., Thalmann, D., Kamyab, K., & Mamdani, E. (2002). Avatar markup language. In *Proceedings of the Workshop on Virtual Environments 2002*, (pp. 169–177). Aire-la-Ville, Switzerland: Eurographics Association.

Laban, R. V. (1980). *Mastery of movement* (4th ed.). Plymouth, MA: MacDonald & Evans Ltd.

Lander, J. (1998, January). Working with motion capture file formats. *Game Developer*, 30–37.

Luciani, A., Evrard, M., Castagné, N., Couroussé, D., Florens, J.-L., & Cadoz, C. (2006a). A basic gesture and motion format for virtual reality multisensory applications. In *Proceedings of the 1st International Conference on Computer Graphics Theory and Applications.* Setubal, Portugal: ACM.

Luciani, A., Evrard, M., Courousse, D., Castagne, N., Summers, I., Brady, A., … Pirro, D. (2006b). *Report on gesture format: State of the art*. Partners' propositions. Deliverable 1 D.RD3.3.1, IST-2004-002114-ENACTIVE Network of Excellence.

Maestre, E., Janer, J., Blaauw, M., Pérez, A., & Guaus, E. (2007). Acquisition of violin instrumental gestures using a commercial EMF tracking device. In *Proceedings of the 2007 International Computer Music Conference*. Copenhagen, Denmark: ICMA.

Malloch, J., Sinclair, S., & Wanderley, M. M. (2007). From controller to sound: Tools for collaborative development of digital musical instruments. In *Proceedings of the 2007 International Computer Music Conference*. Copenhagen, Denmark: ICMA.

Manjunath, B., Salembier, P., & Sikora, T. (2002). *Introduction to MPEG-7: Multimedia content description interface*. New York, NY: John Wiley and Sons.

Marshall, M. T., Peters, N., Jensenius, A. R., Boissinot, J., Wanderley, M. M., & Braasch, J. (2006). On the development of a system for gesture control of spatialization. In *Proceedings of the International Computer Music Conference*, (pp. 360–366). New Orleans, LA: ICMA.

Martinez, J., Koenen, R., & Pereira, F. (2002). MPEG-7: The generic multimedia content description standard, part. *IEEE MultiMedia*, *9*(2), 78–87. doi:10.1109/93.998074

McGilvray, D. (2007). *On the analysis of musical performance by computer*. (PhD Thesis). University of Glasgow. Glasgow, UK.

Peters, N. (2008). Proposing spatdif - The spatial sound description interchange format. In *Proceedings of the 2008 International Computer Music Conference*. Belfast, Ireland: ACM.

Place, T., & Lossius, T. (2006). Jamoma: A modular standard for structuring patches in max. In *Proceedings of the 2006 International Computer Music Conference*, (pp. 143–146). New Orleans, LA: ICMA.

Prendinger, H., Descamps, S., & Ishizuka, M. (2004). MPML: A markup language for controlling the behavior of life-like characters. *Journal of Visual Languages and Computing*, *15*(2), 183–203. doi:10.1016/j.jvlc.2004.01.001

Pullinger, S., McGilvray, D., & Bailey, N. (2008). Music and gesture file: Performance visualisation, analysis, storage and exchange. In *Proceedings of the 2008 International Computer Music Conference*. Belfast, Ireland: ACM.

Roland, P. (2002). The music encoding initiative (MEI). In *Proceedings of the First International Conference on Musical Applications Using XML*, (pp. 55–59). ACM.

Schaeffer, P. (1966). *Traité des objets musicaux*. Paris, France: Editions du Seuil.

Schnell, N., Borghesi, R., Schwarz, D., Bevilacqua, F., & Muller, R. (2005). FTM – Complex data structures for Max. In *Proceedings of the 2005 International Computer Music Conference*, (pp. 9–12). Barcelona, Spain: ICMA.

Schwarz, D., & Wright, M. (2000). Extensions and applications of the SDIF sound description interchange format. In *Proceedings of the 2000 International Computer Music Conference*, (pp. 481–484). Berlin, Germany: ICMA.

Sinclair, S., & Wanderley, M. M. (2007). Defining a control standard for easily integrating haptic virtual environments with existing audio/visual systems. In *Proceedings of the 2007 Conference on New Interfaces for Musical Expression*. New York, NY: NIME.

Wanderley, M. M., Schnell, N., & Rovan, J. B. (1998). Escher-modeling and performing composed instruments in real-time. In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, (vol. 2, pp. 1080–1084). San Diego, CA: IEEE Press.

Wright, M., Chaudhary, A., Freed, A., Wessel, D., Rodet, X., & Virolle, D. … Serra, X. (1998). New applications of the sound description interchange format. In *Proceedings of the 1998 International Computer Music Conference*, (pp. 276–279). Ann Arbor, MI: ICMA.

Wright, M., Freed, A., Lee, A., Madden, T., & Momeni, A. (2001). Managing complexity with explicit mapping of gestures to sound control with OSC. In *Proceedings of the 2001 International Computer Music Conference*, (pp. 314–317). La Habana, Cuba: ICMA.