STSM Report - GDIF Development at McGill

Alexander Refsum Jensenius
Musical Gestures Group
University of Oslo

February 2007

This report summarises the results of my COST Action 287 ConGAS Short Term Scientific Mission (STSM) to the the Input Devices and Music Interaction Laboratory (IDMIL) at McGill University in February 2007.

1 Prelude

The aim of the STSM was to continue development of the Gesture Description Interchange Format (GDIF). GDIF was suggested in [Jensenius et al., 2006] as a tool for standardising the way we describe, stream and store music-related movement data. This initiative grew out of the need to share data between some of the partners in the ConGAS project. Since most of our equipment use different standards for sending and storing information, we encountered numerous obstacles when it came to synchronising and storing the movement data. Similar problems were found when trying to store and share analysis results. Thus the need for a common standard for streaming and storing music-related movement data.

So far, two prototype setups have been developed and presented: using multi-layered GDIF streams for controlling sound spatialisation in realtime using hand movements [Marshall et al., 2006]; and analysis of instrumental gestures in violin performance [Maestre et al., 2007]. This report will present and discuss the current state of development as of February 2007.

Throughout the report, I will use *movement* to denote the act of changing physical position of one or more objects; *action* will be used for goal-directed movement or manipulation; and *gesture* will be used to denote the meaning aspect of the actions, i.e. the perceived qualities of actions.

1.1 Streaming and Storage

GDIF is currently being developed along several lines: an implementation for realtime control of sound synthesis, and two different storage alternatives. As Figure 1 shows, movement can be captured by sensor devices which output a stream of *raw data*. The data are then processed in various ways (see next section) and sent out as GDIF streams on the network[1] to control audio/video synthesis models in realtime.

[1] Typically using UDP/IP for communication but other protocols could also be possible

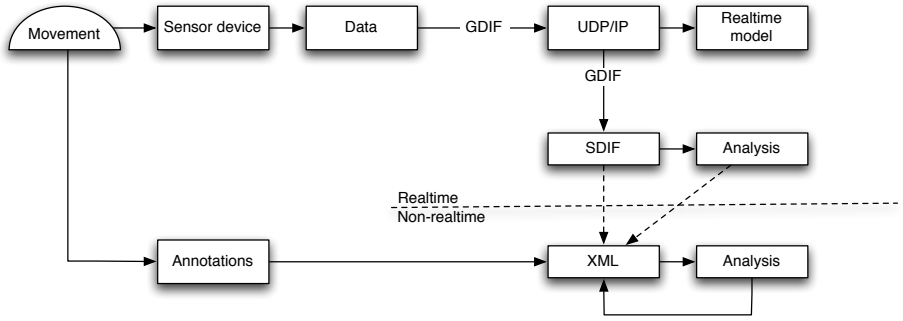

Figure 1: Sketch of data flow for GDIF streaming and storage.

The GDIF streams can also be sent to a data logger, and in our current setup we have been testing IRCAM's FTM tools[2] for storing the data as streams in SDIF-files[3] in Max/MSP. As discussed in [Maestre et al., 2007], the data can later be analysed and stored in structured XML files which will simplify further analysis.

One thing is to be able to store data from various devices, and synchronise these, but we are also interested in storing and synchronising with audio and video recordings, as well as manual annotations and other qualitative data. In an ideal system, it should be possible to easily and quickly retrieve all these different forms of data and media at any point in time. This would provide an efficient tool for further analysis and sharing. A sketch of some of the content of such a multilayered GDIF file is shown in Figure 2.

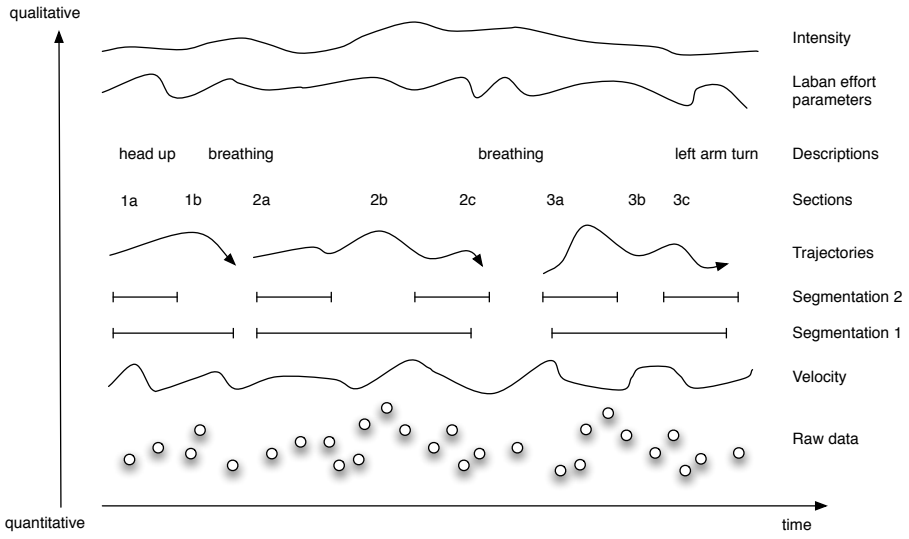

Figure 2: Sketch of movement data that could be stored in a GDIF file.

Notice that we are also open for including multiple versions of data (e.g. segmentation),

[2] http://freesoftware.ircam.fr/wiki/index.php/FTM

[3] The Sound Description Interchange Format (SDIF) was developed in the late 1990s as a tool for storing results of audio analysis, e.g. F0 and spectral data [Wright et al., 1998]. GDIF can be considered a sibling of SDIF.

which will allow several researchers to work on the same material and store analysis results next to each other.

The rest of this presentation will focus on the realtime implementation of GDIF, which was the primary aim of my STSM to McGill.

2 Multilayered Namespaces

The realtime implementation of GDIF is based on OSC[4]. The open and flexible structure of OSC, with no namespaces defined by the standard, has caught on well in the research community, but has yet to be adopted at a larger scale. Thus, while being one of its strengths, this flexible structure is probably also the greatest weakness. Throughout the years there have been some attempts to create common namespaces (see for example [Wright et al., 2001]), but so far nothing has been agreed upon.

While we do not intend to create namespaces that will satisfy everyone, we hope that the approach we have chosen in GDIF development can also be useful for the community at large. We are currently using a namespace structure which is modelled after ideas of splitting *descriptive* and *functional* analysis presented in [Ramstein, 1991]. We also believe it is important to separate data resulting from analysis from the *raw* and *pre-processed* movement data. The layout that we are currently working from is sketched in Figure 3.

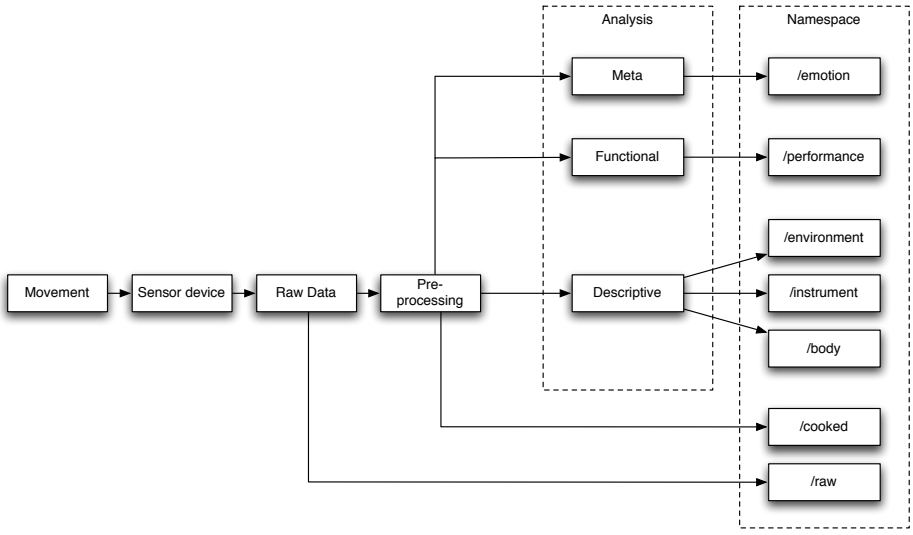

Figure 3: A sketch of how we envisage a multilayered OSC namespace in GDIF.

The rest of this section will describe the general ideas for each namespace, while Section 3 will show an example of an implementation which might help to clarify our thoughts. Before delving into the discussion, I should mention a disclaimer: this is work in progress. Nothing has yet been fixed, and everything is still open for discussion and reconsideration!

[4] Open Sound Control

2.1 Acquisition

The *acquisition* layer is handling what could be considered pre-analysed data, what we have chosen to call the *raw* and *cooked* layers.

2.1.1 Raw

The namespace for the *raw* layer is supposed to pass on the data in the same form as it is received. It is very seldom that we actually use raw data for anything, since it is typically scaled and filtered before passed on, but we find important to be able to access the original data to for example check the validity of some experiments. As far as possible, the subnamespace for the raw data should reflect the structure that the raw data is transmitted in (if possible).

2.1.2 Cooked

The *cooked* layer would typically pass on data that is low-pass filtered (to remove noise in the signal), normalised and/or scaled to some useful range. This is the data that we would probably use as the basis for further processing and analysis.

2.2 Descriptive

The *descriptive* layer is focusing on streaming and storing the *kinematics* of the movements. At this stage, we are still only considering the continuous flux of movement, as the *actions* will be described in the functional layer.

Considering the descriptive aspect of movement, there often seems to be some confusion in terms of the *analytical perspective* taken (see [Kvifte and Jensenius, 2006] for more on this). The descriptors often mix body-centred and instrument-centred data, and often fail to take into account how both the body and the instrument exists in space. We therefore suggest to split the descriptive layer into categories representing three analytical perspectives: *body*, *device* and *environment*.

2.2.1 Body

The *body* layer is looking at actions "through the eyes" of the performer, and is coding the data with respect to this person. As the example in the next section will show, we use a body-centred namespace, naming the hand, finger, etc. The idea is that these descriptions should be independent of the instrument, so that it is also possible to create mappings on a body-level which would remain constant even though the device is interchanged.

2.2.2 Device

The *device* layer codes data with respect to the sensor, instrument or controller used. Here, the performer and performer's movements are not relevant at all, as it is only the button, slider, etc. which is in focus.

2.2.3 Environment

The *environment* layer is intended for coding information about the relationship between various bodies and devices in a space, and/or the relationship between the bodies/devices and the space. This is typically useful if the performer's position in space

is important (e.g. for controlling spatialisation), or the relationship between several performers. For a more traditional setup with one performer and one instrument this namespace is probably not applicable.

2.3 Functional

The *functional* layer is where we start analysing the *actions* of the performer, and the function of the action. We are particularly interested in trying to code the actions so that they can be easily mapped to corresponding sound synthesis parameters. The namespace is inspired by the taxonomy of *instrumental gesture* (which I would call *instrumental action*) as summarised in [Cadoz and Wanderley, 2000].

2.3.1 Excitation

We are looking for whether the action could be classified as:

- *Impulsive* An instantaneous energy transfer with no control of the tone after the attack, e.g. percussion and piano instruments.
- *Continuous* A continuous energy transfer allowing for continuous control of the tone, e.g. wind and string instruments.
- *Iterative* A combination of impulsive and continuous, with rapid repeated impulsive attacks, e.g. shakers and other percussion instruments.

This is not an easy task, and we believe that coding this is an important part of the design of a new instrument and/or composition for an instrument.

2.3.2 Modulation

Actions that result in modulation of the sound can be classified as:

- *Parametric* Continuously changing a parameter, e.g. bow pressure in violin performance.
- *Structural* Modifying or changing the structure of the object, e.g. key in a wind instrument.

2.3.3 Selection

Selection would imply choosing among different structures, but without any sound-production or object modification, e.g. choosing which string to play on, or the bowing direction in string instruments.

2.3.4 Performance

It would also be relevant to code data using vocabulary from music performance, such as:

- The dynamics of the performance, i.e. *ppp* - *FFF*
- The playing style, i.e. *staccato* or *legato*
- ...

2.4 Meta

The *meta* layer is focusing on the meaning of the actions (i.e. the *gesture*), e.g. imagined emotional quality, or what is often referred to as *expressive gesture* qualities [Camurri et al., 2003]. We have not worked on this so far, but believe it is important to keep this aspect in mind while developing GDIF.

3 A Max/MSP Implementation

To test the theoretical ideas in practice, I have transformed **jmod.mouse**, a simple Jamoma module returning data about the computer mouse connected to the system, into a GDIF-compliant module called **jmod.mouse.gdif**. The computer mouse is not the most exciting device for musical control purposes, but it was chosen as an easy and versatile case study. The module can be downloaded from Jamoma's repository at SourceForge (currently at revision 1631).

3.1 Jamoma

Jamoma is an open-source project[5] based on the engine of Jade[6] by Tim Place [Place and Lossius, 2006]. It is currently being developed by a small team including several researchers affiliated with ConGAS. The idea is to develop a flexible framework for creating modules within Max/MSP, as well as developing a number of modules that can be easily combined by the end user. The aim is to help Max/MSP developers to work more efficiently, and share more of their work, as well as allow end users the power of Max/MSP while not having to learn it from the ground up.

Jamoma is based around **jcom.hub** providing a centralised router for all information passed between modules. By using OSC internally, Jamoma is also flexible when it comes to control, and being controlled by, other systems.

A Jamoma module typically consists of several files, splitting the GUI from the algorithm. For example, the **jmod.mouse.gdif** module consists of the following files:

- **jmod.mouse.gdif.mxt** the graphical interface of the module.
- **jalg.mouse.gdif.mxt** the algorithm doing all the processing.
- **jmod.mouse.gdif.help** the help file showing how the module can be used.
- **jmod.mouse.gdif.xml** the preset file including the initial values of the module.

The rest of this section will present the **jmod.mouse.gdif** module. I will not go into details about the implementation, but rather focus on the functionality that is offered when using GDIF.

3.2 jmod.mouse.gdif

The help file of **jmod.mouse.gdif** can be seen in Figure 4. It consists of the module itself with rather self-explanatory parameters: choosing whether to poll or sample data, sampling rate and choice of reference point. Data are output through the left outlet and

[5] Jamoma is licensed under GNU LGPL
[6] http://www.electrotap.com/jade/

is formatted as OSC streams that can be easily parsed by an OSC router[7]. I have chosen to leave the data output from the original module untouched to not break compatibility with the previous version of the module, but will focus on the GDIF data in the rest of this section.

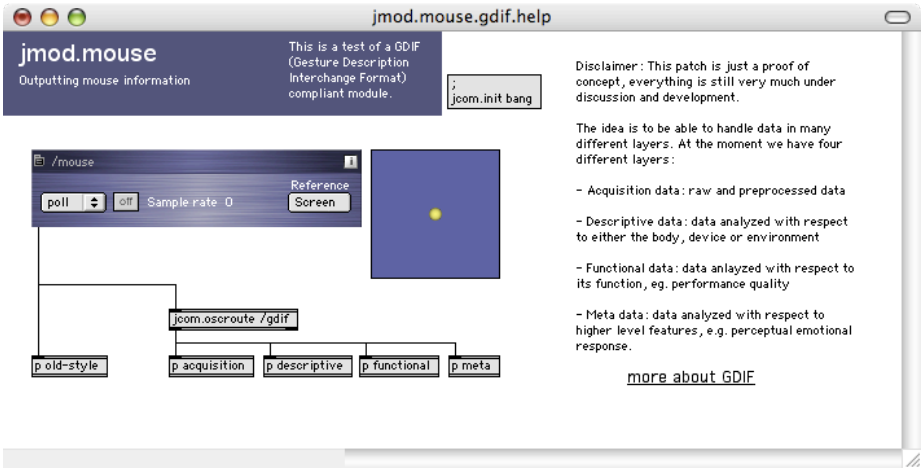

Figure 4: Help file of the Jamoma module **jmod.mouse.gdif** for outputting data from computer mouse.

An inspector button is available in the upper right corner of the module, which opens the inspector window where it is possible to choose which GDIF streams to output (Figure 5). Since most of the GDIF data will probably be redundant most of the time, we decided to leave it up to the end user to choose whether data should actually be sent or not, to prevent flooding the system/network with too much data. The rest of this section will focus on the namespace used for each of the layers described above.

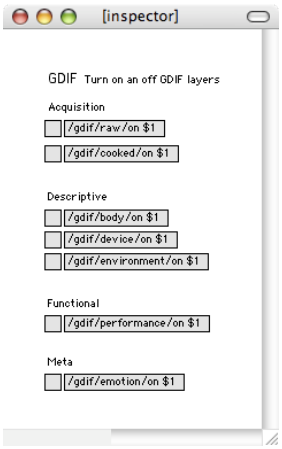

Figure 5: The inspector window of the module.

[7] I use Jamoma's **jcom.oscroute** but CNMAT's **osc-route** would also work.

3.2.1 Acquisition

Figure 6 shows the subpatch parsing the *acquisition* layer. The *raw* data layer contains the data as it is output from the **mousestate** Max object which is at the heart of the module. It is formatted using the following namespace:

```
/gdif/raw/button <1/0>
/gdif/raw/location/horizontal <pixels>
/gdif/raw/location/vertical <pixels>
```

The use of location/horizontal and /vertical in the namespace is because these are the names of the outlets of the Max object (and are probably more meaningful than most other raw data names that we will come across). The most important is to keep the information about raw data so that it is possible to go back and check if necessary.

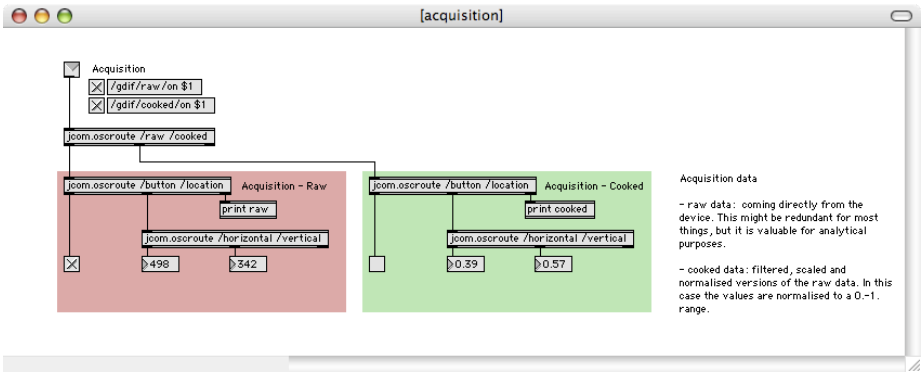

Figure 6: Subpatch parsing the *acquisition* layer of the **jmod.mouse.gdif**, consiting of the raw and cooked data layers.

The other half of the the *acquisition* layer shown in Figure 6 shows the pre-processed data. In this example these data have been normalised to a 0.–1. range based on dividing the location of the mouse (in pixels) by the dimensions of the computer screen (in pixels). The values have also been changed so that origo is now in the bottom left corner of the screen as opposed to upper left corner for the raw data. This seems to be more in line with how we would think about the position of the mouse.

```
/gdif/cooked/button <1/0>
/gdif/cooked/location/horizontal <0. - 1.>
/gdif/cooked/location/vertical <0. - 1.>
```

3.2.2 Descriptive

Figure 7 shows the *descriptive* data layers. These layers could be considered as transformations of the cooked layer with respect to either the body of the performer, the device or the environment.

First, the *body* layer consists of the following namespace:

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

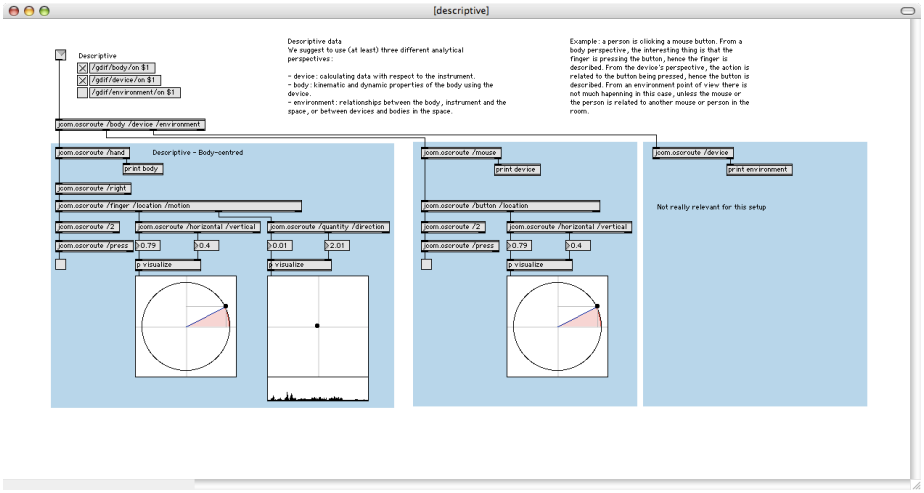

Figure 7: Subpatch parsing the *descriptive* layer.

Notice how we use values in the range between -1. and 1. This is because we imagine that the user will have a *home position* which is surrounded by an *action space* on all sides. This might be different in other action modes.

Similarly, the *device* layer consists of the following namespace:

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

For the case of the mouse, where the location of the mouse and the location of the body fall together, this layer outputs the same values as the body layer, but with a different namespace.

We have started developing namespaces for a number of different devices, and imagine having a library of generic device types to choose from when encoding the data. For example, most mice, game controllers and MIDI keyboards look and work more or less similarly, even though there might be manufactural differences. Thus, they might output somewhat different raw data, but the descriptive data would be coded according to some generic norm for the device.

3.2.3 Functional and Meta layers

I have only briefly started to implement functional and meta layers for the mouse module, but these are not yet finished so I will leave that discussion for later. An important question to consider in relation to this is whether the processing of such layers should be handled in the module itself, or whether it would be better to have a separate module that could be connected. I tend to think that the latter would be better. The functional and meta layers deal with data that can be calculated from the other layers, and can often be based on processing that can be reused for other modules, e.g. calculating velocity and acceleration, segmentation, etc.

What is most important is that it should be fairly simple to make changes in the namespace and/or scaling to reflect the setup used. Even though it would be interesting to strive for some kind of "universal" coding of both the functional and meta layers,

I believe this would be very difficult. The interpretation of the data is very much dependent on the use, and it could also be seen as part of the mapping process to do the analysis and set up a namespace that works well.

4 Mapping

One of the main ideas behind the realtime implementation of GDIF is to allow for quickly creating mappings between movement/action/gesture and sound. At first sight it might seem cumbersome to create such a large namespace for a simple device, as probably a lot of the information will be redundant. However, we believe it could also be the basis for a very flexible system that could feel liberating to work with.

In the IDMIL, as in many other music technology labs, there are a large number of controllers, synthesisers and computers. We envision a setup where everything can be connected in any way anytime. Then it is possible to really test out various *action-sound couplings* by just creating mappings from one device to another.

There are already some mapping modules available in Jamoma, that allow for easily creating mappings within Max/MSP. Joe Malloch is also working on a more general mapper, built in Max/MSP but with the scope of working as a standalone, which is similar to the way Jack and Soundflower routes audio information. Even though this is a one-to-one mapper, we believe that the added flexibility of the many layers in GDIF, many of which are based on multidimensional transformations, will allow for a much more flexible mapping process than is currently used in many systems.

4.1 ZeroConf

An important part of creating a mapper that works in a multi-device/computer setup, is to figure out how to easily and efficiently send data between devices. After coming up with a number of different solutions, we have tested using ZeroConf[8] for the discovery of addresses and namespaces. This protocol is currently implemented as Bonjour in Mac OS X and is also available for other platforms. The good thing about ZeroConf is that it handles port allocation automatically and makes it is easy to distribute information about clients that are available on the network.

Our current idea is to use ZeroConf for announcement and discovery of devices and modules on the network, and develop a system for doing handshaking and set up communication between modules. Some ideas about this was presented at the OSC meeting during NIME 2006 [Jazzmutant, 2006], and we will be actively involved in discussing this further within the OSC community.

During my STSM, I worked with Steve Sinclair testing ZeroConf in OS X and Linux. After some fiddling, we managed to get the OSCBonjour externals[9] by Remy Mueller to work between different platforms. Our testing looks promising and we will continue to follow this path in further development.

5 Postlude

Based on my frustration of working with equipment and software that was using totally different standards (if any standard at all...), I started thinking about GDIF two years

[8] http://www.zeroconf.org/
[9] OSCBonjour external

ago. We have come a long way when it comes to clarifying needs and sketch paths for further development since then, but there is a much longer road ahead of us. A number of challenges will have to be discussed and solved in the future:

- Namespace. The ideas presented in this report are in continuous revision. As we have a bottom-up approach to the namespace, implementing whatever we see that we need for our research, it will hopefully slowly develop into a more solid and general structure.
- Units. Right now we are using all sorts of units for storing data, and we want to be able to have a flexible system that can handle any type of unit we might want to use. To be able to structure this, we are thinking about creating a *unit library* that can store information about all the units we are using. This should also include information about unit conversion to allow for easy mapping between modules using different modules.
- Handshaking. We need to develop a robust system for creating handshaking between devices, modules etc. This seems to be on the agenda for OSC 2.0, so we will follow this development closely.
- Efficiency. We are currently using a verbose namespace, and will have to look at how to more efficiently handle data flow. We have had numerous discussions on how to allocate different ports for heavy traffic as well as using OSC aliasing, but this will have to discussed and tested more in the future.
- Storage. So far, the focus has been on making a prototype for realtime processing, but we also need to get back to handle storage and synchronisation of data.

We plan the following short term actions/activities:

- Submit a joint publication to ICMC 2007.
- Propose a panel session on movement/action/gesture data formats for ICMC 2007.
- Discuss GDIF development during the Jamoma developer's workshop in Albi in March 2007.
- Organise a GDIF workshop in Oslo in 2007 to continue development.

6 Acknowledgments

Thanks to ConGAS for granting this STSM and to Marcelo M. Wanderley for hosting me. The work was carried out in collaboration with Steve Sinclair and Joe Malloch. Thanks also to Nils Peters, Mark Marshall and Carmine Casciato for good discussions during my STSM, and to the virtual presence of the Jamoma developers: Tim Place, Trond Lossius, Pascal Baltazar and Dave Watson.

References

Cadoz, C. and M. M. Wanderley (2000). Gesture-music. In M. M. Wanderley and M. Battier (Eds.), *Trends in Gestural Control of Music [CD-ROM]*, pp. 71–94. Paris: IRCAM - Centre Pompidou. 5

Camurri, A., B. Mazzarino, and G. Volpe (2003). Analysis of expressive gestures in human movement: the eyesweb expressive gesture processing library. In *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003), Firenze, Italy, May 8-9-10, 2003*. 6

Jazzmutant (2006). Extension and enchancement of the osc protocol. Draft Presented at the OSC-meeting at NIME 2006, IRCAM, Paris. 10

Jensenius, A. R., T. Kvifte, and R. I. Godøy (2006). Towards a gesture description interchange format. In *Proceedings of New Interfaces for Musical Expression, NIME 06, IRCAM - Centre Pompidou, Paris, France, June 4-8*, pp. 176–179. 1

Kvifte, T. and A. R. Jensenius (2006). Towards a coherent terminology and model of instrument description and design. In N. Schnell, F. Bevilacqua, M. Lyons, and A. Tanaka (Eds.), *Proceedings of New Interfaces for Musical Expression, NIME 06, IRCAM - Centre Pompidou, Paris, France, June 4-8*, pp. 220–225. Paris: IRCAM - Centre Pompidou. 4

Maestre, E. G., J. Janer, A. R. Jensenius, and J. Malloch (2007). Extending GDIF for instrumental gestures: the case of violin performance. In *Submitted to NIME 2007*, New York. 1, 2

Marshall, M. T., N. Peters, A. R. Jensenius, J. Boissinot, M. M. Wanderley, and J. Braasch (2006). On the development of a system for gesture control of spatialization. In *Proceedings of the International Computer Music Conference, 6-11 November, New Orleans*, pp. 360–366. 1

Place, T. and T. Lossius (2006). Jamoma: A modular standard for structuring patches in max. In *Proceedings of the 2006 International Computer Music Conference, 6-11 November, New Orleans*. 6

Ramstein, C. (1991). *Analyse, représentation et traitement du geste instrumental*. Thèse de docteur ingénieur spécialité informatique, Institut National Polytechnique de Grenoble. 3

Wright, M., A. Chaudhary, A. Freed, D. Wessel, X. Rodet, D. Virolle, R. Woehrmann, and X. Serra (1998). New applications of the sound description interchange format. In *Proceedings of the International Computer Music Conference, Ann Arbor, Michigan*, pp. 276–279. 2

Wright, M., A. Freed, A. Lee, T. Madden, and A. Momeni (2001). Managing complexity with explicit mapping of gestures to sound control with OSC. In *Proceedings of the 2001 International Computer Music Conference, Habana, Cuba*, pp. 314–317. 3