Suggestions for the Integra Environment.

Henrik Frisk
Malmö Academy of Music - Lund University
henrik.frisk@mhm.lu.se
http://www.henrikfrisk.com/

March 27, 2006

1 Introduction

Following the presentations by Jamie Bullock, Birmingham and Henrik Sundt, NOTAM at our meeting in Birmingham last month and the discussions at the Integralive web site I have a suggestion for an XML file structure that would allow for pieces developed in or transferred to the Integra environment to be easily stored, upgraded, and transferred to different technologies. Further, developing a graphical editor for the XML files that would act as a hub between the synthesis engine(s), the database and the GUI would allow for using multiple pieces of synthesis and analysis software in the same piece of music. Finally, this document also approaches the timeline issue.

The Integra goals as they were defined in the presentation by Henrik Sundt and NOTAM in Birmingham have been the guideline for the suggestions presented in this document:

- User friendly tool for live electronics.
- Methods for preservation of works.
- Encourage inter-program communication, and contribute to a modular environment for live electronics.

Apart from the already introduced technologies (OSC and SQL of some flavor) this suggestion also relies on:

- XML (see `http://www.w3schools.com/xml/xml_whatis.asp`)
- Jack (see `http://jackit.sourceforge.net/`)
- SMIL (see `http://www.w3schools.com/smil/smil_intro.asp`)
- VST and LADSPA

An overview of the structure of the connections between all the pieces referred to in this document can be seen in figure 1.

The assets of the suggestions in this document may be summarized as:

- Completely open as to choice of synthesis/analysis engine.
- Different synthesis/analysis engines can easily be compared and exchanged.

- An integrated format in wich a complete description of the electronics in a piece of music can be saved independently of the technology used to implement it.
- Each file can be validated generally and specifically against an external Integra DTD (Document Type Definition).

The drawbacks may be summarized as:

- Uses several standards.
- Possibly requires a larger programming effort from the Integra team.
- Jack does not work on Windows (yet).

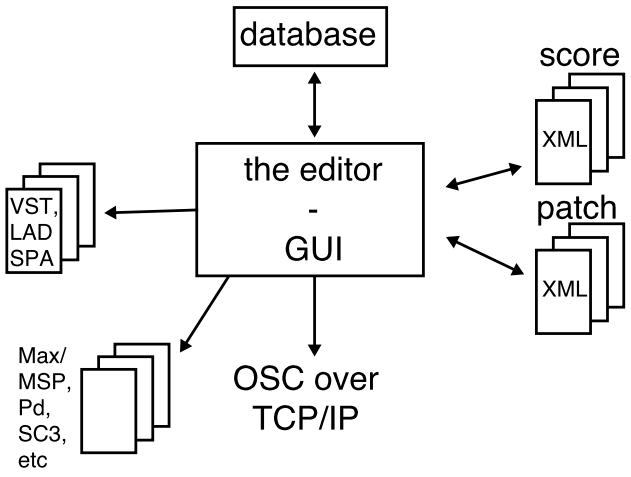

Figure 1: Structure.

2 File structure for storing Integra 'patches'.

Following is a very generalized example of how the Integra xml file format could look like. This particular example describes the single oscilliator 'patch' in figure 2:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE root-element SYSTEM "http://www.somewhere.org/integraDTD-0.1.dtd">
<integra>
  <head>
    <inputs physical="2" virtual="2" />
    <outputs physical="2" />
  </head>
  <body>
    <generalAudioModule>
      <generalSynth>
```

```
<simpleSynth name="smplsnt" frq="200" unit="hz"
                     controlSource="generalController/physicalSlider/midiFader/fader1"
                     audioIn1="0" audioOut1="3">
  <documentation title="Simple oscilliator"
 content="Documentation can be included in the elements."/>
</simpleSynth>
      </generalSynth>
      <util>
<gain name="gain1"
              range="0, 127"
              default="64"
              map="0., 1."
              controlSource="null"
              audioIn1="3"
              audioOut="1, 2" />
      </util>
    </generalAudioModule>
    <generalController>
      <generalFader>
<physicalFader>
  <midiFader name="fader1"
                     range="0, 127"
                     map="100, 1000"
                     receiveCh="1"
                     controller="7" />
</physicalFader>
      </generalFader>
    </generalController>
  </body>
</integra>
```

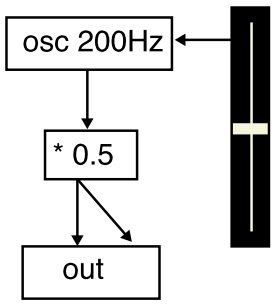

Figure 2: Simple patch example.

The file has a `<head>` tag and a `<body>` tag. The head tag would hold information on

the number of physical and virtual audio inputs and outputs etc. The body tag contains the actual modules and their attributes.

The XML structure follows the OSC class hierarchy for each module contained in it and for each module it holds the attributes for default values, control sources, information on mapping of values etc.

3 The editor

The file above can be generated in the editor simply by adding and connecting objects in a Max-Pd fashion. However, *only objects of the abstract base class* can be added in the GUI. In the example above the only two objects that are added are one object of class `Integra/GeneralAudioModule/GeneralSynth` and one of class `Integra/General-Controller/GeneralFader` (the gain module would be implied at any sound generating or processing output[1]). After the abstract class of the object has been chosen and is displayed an instance of this class can be created. The list of available modules is dynamically fetched from the database and in the case of a `GeneralSynth` it could for example be:

- FM
  - Pd
    - hexter
    - dssi
  - MIDI
    - DX7
- Additative
  - ...
- JAMOMA

For the chosen module the documentation is included in the XML file directly. Using XLS transforms this allows for exporting the documentation to any one of a number of formats (PDF, HTML, etc) as well as displayed within the editor. The 'patch building GUI' would obviously be significantly simpler to learn, understand and implement than that of Max or Pd.

After the Integra XML file is loaded or created a set of on screen faders (or whatever graphical component is the most suitable for the particular parameter) corresponding to the objects created are automatically generated.

4 The processing.

Once the structure of the synthesis/processing/analysis patch is created it needs to be exported to the actual software packages handling the audio processing. One way this could be done is to support a plug-in structure in which support for a specific piece of software can be implemented. With such plug-ins, or parsers, the Integra XML file would be used to render a fully functional and Integra OSC namespace compatible patch in the desired program, be it PD, Max/MSP, SuperCollider or any combination of software.

[1] Exactly how, and what modules should be combined to accomplish a functional meta-module such as a synthesizer is an issue that, no matter how the final design will look like, needs to be discussed in greater detail.

The other way to work would be to use modules from different software packages, i.e. using Max/MSP and Jitter for realtime analysis of video input, using that data to control the synthesis implemented in SuperCollider and playing back samples on a hardware sampler. By using Jack as the audio server audio streams can be sent between applications with low latency. Presumably a fully functional and Integra OSC compliant module should be contained in the database and easily downloaded to the client. With a class structure of OSC-to-MIDI transformation modules hardware systems could easily be integrated into such a system. Even without the use of Jack this model is possible with external mixing - there is no need for all the modules to reside on the same computer.

5 The timeline

The issue of a timeline - basically the possibility to automate control signals and to trigger chains of events at certain points in time - was discussed at our meeting in Birmingham. In this suggestion I argue that we should separate the time critical information, the cue list, from the 'patch' mainly for clarity.

Some aspects to consider:

- How is the cue list edited?
- How is the state of all modules updated when rehearsing?
- How are triggers handled? Tempo changes?

One suggestion that effortlessly would deal with all of the above and at the same time liberate us from having to implement our own timeline editor would be to let the editor described above export a light-weight VST or LADSPA plug-in that holds all the parameters contained in the Integra XML file. The only thing the plug-in does is transformation of VST/LADSPA control signals to OSC. Loading the plug-in in a sequencer of choice allows the user to edit all the parameters using all the tools available in the sequencer and at the same time sending the relevant data to the 'patches'. A special cue plug-in could be used to handle cue points in the 'score'. Once the editing is done, the entire cue list is exported to an Integra score file-format which can be loaded in the editor. Once it is loaded in the editor the score playback is controlled by play, stop, pause and step buttons in the GUI of the editor. The option of patching in a user input module (sensor, foot pedal, etc.) should also be present.

5.1 Assets

- For those used to working in a sequencer, the environment is well known.
- Preparing a cue list for a work in which the instrumental part is already recorded becomes very convenient - load the track into the sequencer and load the plug-ins and draw the automation.
- Relieves the Integra team of having to develop their own timeline.
- Since exporting the data would write every value at some control rate (rather than the ususal line segments) allows for complete parameter update at every point along the timeline.
- Making use of existing technology.

5.2 Drawbacks

- Relies on external protocols (VST, LADSPA) and technology (sequencer).
- The sequencer timeline may not be the most appropriate representation for recording data in some styles of music.
- The extra steps of exporting of a plug-in, loading it in a sequencer etc. is not very intuitive.

5.3 The score file format

An XML format for synchronizing streaming media, SMIL (see `http://www.w3schools.com/smil/smil_intro.asp`) already exists and is now at version 2.0. It is my belief that a lot would be gained by using the SMIL format as a starting point and use our own namespace that extends the basic functionality of SMIL. A simple SMIL file that plays back a soundfile from a server once and then exits can be seen below. This particular file uses the Real Networks (RealAudio) namespace extensions.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN"
  "http://www.w3.org/2001/SMIL20/WD/SMIL20.dtd">
<smil xmlns="http://www.w3.org/2000/SMIL20/CR/Language"
      xmlns:rn="http://features.real.com/2001/SMIL20/Extensions">

  <head>
    <meta name="title" content="A great song"/>
    <meta name="author" content="Henrik Frisk"/>
    <meta name="abstract" content="This item may be used for comments"/>
    <layout>
      <root-layout width="0" height="0" />
      <region id="t" top="0" left="0" width="300"
      height="200" z-index="1"
      background-color="black" />
    </layout>
  </head>

  <body>
    <par repeatCount="1">
      <audio src="http://www.somewhere.com/aSoundFile.au"
     clip-begin="0s" />
    </par>
  </body>
</smil>
```

An advantage with using the SMIL format is that images and video may easily be incorporated and synchronized with the audio for artistic as well as documentation purposes.

6 Conclusion

I have discussed the issues brought up in this document with my collegues in Malmö, and to some extent with Henrik Sundt after the Birmingham meeting. This document should be regarded as a rough sketch and I have quite possibly overlooked some things. However,

if the ideas, or some of the ideas, put forward in this document appear to be interesting to the rest of the Scientific Group, a more detailed and well structured presentation could be given at our meeting in Krakow in May.

Using XML for storing the structure of the patches and the control data needed to 'play' a piece of music containing elements of live electronics has many advantages, but especially the possibility for including documentation in the files combined with publicly available tools for transforming this data to any number of output formats makes it an attractive possibility for the Integra project.