Freer Than Max - porting FTM to Pure data

IOhannes m ZMÖLNIG, Thomas MUSIL and **Winfried RITSCH**
Institute of Electronic Music and Acoustics
University of Music and Dramatic Arts
Graz, Austria,
{zmoelnig, musil, ritsch}@iem.at

Norbert SCHNELL
Real-Time Applications Team & Performing Arts Technology Research Team
IRCAM – Centre Pompidou
Paris, France
norbert.schnell@ircam.fr

Abstract

FTM is an environment that allows the processing of complex data structures such as sequences, matrices, dictionaries, break point functions, tuples and whatever might seem helpful for the processing of music, sound and motion capture data within graphical computer music systems. While FTM itself is published under a free license (LGPL), until recently the only supported host system has been Max/MSP, a commercial data-flow environment for Mac OS-X and W32.

In this paper we present a port of FTM to Pure-data, Max's free sibling that is available for Mac OS-X, W32, FreeBSD and (among others) of course linux.

Keywords

Pure-data, complex data, ...(max. 5)

1 Introduction

Graphical computer-music languages such as Pure-data provide possibilities to handle musical signal-processing in an intuitive way. Due to the used data-flow metaphor, there is usually a focus on the "signal-processing" side rather than the handling of more "musical" complex data-structures. On the contrary, most graphical data-flow languages provide almost no possibilities to handle data-types that are more complex than simple lists of values (like numbers, or symbols/strings; but excluding other lists), which makes them a less valuable tool for algorithmic composition.

The library **FTM**, developed at *IRCAM*[1], tackles these problems as a framework that provides efficient low-level access to data-types related to the representation of sound, gesture and music, such as matrices, dictionaries, (time-tagged)sequences or score-objects[1].

While providing a set of modules that allow direct access to these data-types from within a data-flow language, FTM also provides a framework to write other modules that have a more direct (and efficient) access to the data, and which can then provide higher-level access for the user[2; 3]

Though published under an Open Source license, until recently FTM was only available as a library for Max/MSP[4], a commercial data-flow language for Mac OS-X and W32. However, because the framework is offspring (and remains) of another data-flow language jMax[5], it has been designed not exclusively for the proprietary Max/MSP-platform but for an abstract computer-music data-flow language.

1.1 Some alternatives for handling complex data

FTM is not the first framework with a graphical computer-music language that provides access to and manipulation of complex data-structures. One obvious alternative is already built into Pure-data: graphical data-structures [6]. These provide both low and high level access to complex data sets on the patch level. Their main drawback is, that they are currently implemented in a not very optimized way, which makes them less than ideal for real-time processing of large amounts of complex data. Additionally there are caveats in the persistency of data-structures: while persistency is built into Pd, it is currently limited to a rather low (something around 400) entries per field, which is rather limiting.

A more optimized approach similar to FTM is the `PDContainer`-library by Georg Holzmann [7]. This library provides a set of objects that expose the C++-Standard Template Library, thus allowing to work with the usual complex data-types in an efficient way.

2 The Framework

FTM itself consists of a dynamic library and a small set of objects that expose this library to a "Realtime Environment" (Pure-data, Max/MSP or whatever).

[1] http://ircam.fr

As can be seen in fig.1, only a small portion of the FTM-framework depends on the realtime environment.

This portion is again split into two parts:

- ftmext (FTM External Interface): provides a unified access to the "C" plugin-API of the various realtime environments. This is mainly implemented as a set of preprocessor macros.
- ftmrte (FTM Real-Time Environment): provides a unified access to the infrastructure of the realtime environment that is not directly related to the instantiation and housekeeping of objectclasses.

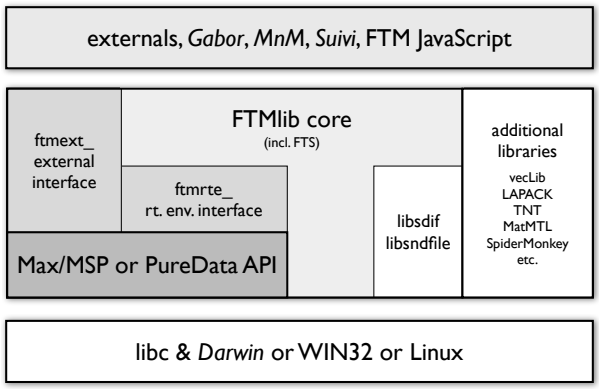

Figure 1: Block diagram of the structure of the audio engine [IRCAM]

The rest of FTM is implemented in a generic, platform-independent way. Leaving a small, rather well defined set of functionality to port to any new platform.

2.1 Towards binary compatibility

The "ftmext" interface provides an API to write plugins that can be compiled for various realtime environments out of the box. Because most of "ftmext" is implemented via preprocessor macros, this compatibility is only superficial, as it does not provide a binary compatibility layer: plugins compiled for a certain realtime-environment will need to be recompiled for every other realtime-environment (even if both environments were running on the same operating system).

Furthermore, the macro approach forces all externals to be recompiled whenever the "ftmext"-implementation changes.

These two issues strongly suggest to wrap the ftmext interface into a dynamic library and minimize the use of macros. In theory this could provide binary-compatible externals for various platforms (compile once, run on Pd and Max).

Whether a library-version of the compatibility wrappers is markedly less efficient than a macro-based implementation remains to be evaluated.

3 The core objects

Since ftm provides new atom types (representing the data structures) within the realtime environment, it also needs to provide a set of objects that can handle these atom types.

3.1 Generic core objects

Most of these objects are rather generic in terms of demands towards the realtime environment. They can therefore for be implemented as ftmext externals. However, most of these generic objects have actually been implemented in a platform dependent way as Max/MSP externals.

Re-implementing these objects as ftmext externals, serves two purposes: For once, these ojbects are now written in a platform-independent way, which will make the eventual port of FTM to whatever future data-flow language even simpler than it was with Pure-data. On the other side, these implementation makes an optimal test-bed for the ftmext framework. Since the core objects cover a wide range of functionality, they use most of the ftmext API.

3.2 Platform dependent core objects

A few objects cannot be implemented in a platform-independent way via ftmext, as they rely heavily on features of the hosting realtime environment.

The most noteable of these are the "ftm.object" object and the "ftm.mess" pseudo-messagebox object. These mimick enhanced version of the usual "object" and "messagebox" by exploiting the host's graphical capabilities.

4 Some more objects: going GUI

FTM does not only provide programmatic access to it's complex data-types, but also a set of objects that allow to visually interact with the data.

The visual interaction objects range from simple spreadsheet like editors for two-dimensional numeric matrices to complex score editors that handle several sets of (in)dependent data sequenced in time.

Originally these graphical objects had been implemented in java using the mxj java-API for

Max/MSP. While there is also a port of mxj to Pd, called pdj[8], this is unfortunately of now big help, as pdj does implement all mxj functionality but the graphics.

However, FTM is currently moving away from using java as a GUI-toolkit in favour of the cross-platform C++-toolkit "juce"[2].

The separation of pd-core and it's tcl/tk-based GUI keeps the integration of 3rd party GUI toolkits from being straight forward.

Editors can require huge amounts of data which will block bottlenecks such as the connection between pd and pd-gui, when moving the data from one instance to the other. One solution to this problem is to use shared-memory for moving the data.

A more naive approach that is used here, is to move part of the graphic-rendering from the pd-gui into the main application. For obvious reasons, the editor-interfaces are therefore not integrated in the patch-windows but reside in their own windows (which are technically windows of the pd-core process rather than the pd-gui process)

5 Conclusions

The port of FTM to Pure-data brings yet another framework for handling complex data-types to Pd. While this my seem needless at first glance, FTM brings the bonus of interoperability with a widely used commercial system that otherwise lacks such complex data-types.

Due to the level of abstraction of the hosting realtime environment within the FTM-framework, porting of the basic functionality was rather painless.

The FTMext-API provides a cross-platform API to write plugins for graphical computer-music languages in a simple language like "C".

Having FTM available on a free platform will hopefully trigger the development and porting of higher-level data-processing libraries like *Gabor* or *MnM*.

6 Acknowledgements

Our thanks go to the IRCAM Real-Time Application Group, especially Diemo Schwarz and Norbert Schnell.

Extra thanks to Tommaso Bianco, who did the first steps in porting FTM to Pure-data.

[2] http://www.rawmaterialsoftware.com/juce/

References

[1] Norbert Schnell, Riccardo Borghesi, Diemo Schwarz, Frederic Bevilacqua, and Remy Müller. FTM — complex data structures for max. In *Proc. of the ICMC*, Barcelona, 2005.

[2] Frederic Bevilacqua, Remy Müller, and Norbert Schnell. MnM: a max/msp mapping toolbox. In *Proc. of New Interfaces for Musical Expression*, Vancouver, 2005.

[3] Norbert Schnell and Diemo Schwarz. Gabor, multi-representation real-time analysis/synthesis. In *COST-G6 Conference on Digital Audio Effects (DAFx)*, pages 122–126, Madrid, 2005.

[4] Miller Smith Puckette. Combining event and signal processing in the max graphical programming environment. *Computer Music Journal*, 15(3):68–77, 1991.

[5] Francois Dechelle, Riccardo Borghesi, Maurizio De Cecco, Enzo Maggi, Butch Rovan, and Norbert Schnell. jmax: a new java-based editing and control system for real-time musical applications. In *Proceedings of the 1998 International Computer Music Conference*, San Francisco, 1998.

[6] Miller Smith Puckette. Using pd as a score language. In *Proceedings of the ICMC*, pages 184–187, Gothenburg, 2002.

[7] Georg Holzmann. Pdcontainer library for pd. `cvs://pure-data.cvs.sourceforge.net/cvsroot/pure-data/externals/grh/PDcontainer/`, 2004.

[8] Pascal Gauthier. pdj - java external plugin for pure-data. `http://www.le-son666.com/software/pdj/`, 2004.