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 Musical Interaction Group
IRCAM – Centre Pompidou
Paris, France
norbert.schnell@ircam.fr

Abstract

FTM is an environment that allows the processing of complex data structures such as matrices, sequences, 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 graphical programming environment for Mac OS X and Win32.

In this paper we present a port of FTM to Pure Data, Max's free sibling that is among others available for Mac OS X, Win32, FreeBSD and Linux.

Keywords

Pure Data, complex data, Max/MSP, porting

1 Introduction

Graphical computer music languages such as Pure Data (Pd) 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" data structures. As a matter of fact, few graphical data-flow languages provide possibilities to handle data types that are more complex than simple lists of values (like numbers, or symbols/strings) supporting applications such as content based audio processing and algorithmic composition.

The library *FTM*[1], developed by the Real-Time Musical Interaction Group at *IRCAM*[2], tackles these problems as a framework that provides efficient access to data types for the representation of sound, gesture and music, such as matrices, time-tagged sequences, break point functions or dictionaries [1].

[1]http://ftm.ircam.fr
[2]http://imtr.ircam.fr

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 implement operator modules that have a direct and efficient access to the data and can provide high-level access to the FTM data structures [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 Win32. However, FTM has been initially developed as an offspring of *jMax* [5], another Open Source project implementing a Max-like data-flow language.

1.1 Some Alternatives for Handling Complex Data

FTM is not the first framework that provides access to and manipulation of complex data structures within a graphical computer music language. Pd already provides inbuilt graphical data structures [6] including both low and high level access to complex data sets on the patch level. Their main drawback is their current inefficient implementation that prevents from its application for real-time processing of larger amounts of complex data. Additionally there are caveats in the persistency of these data structures: while persistency is built into Pd, it is currently limited to a rather low (something around 400) entries per field, which makes it suboptimal for handling larger data-sets such as musical scores or audio analysis data sets.

A more optimised approach similar to FTM is the *PDContainer* library by Georg Holzmann [7]. This library provides a set of external modules 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 *FTMlib* and a small set of external modules that expose this library to a real-time host environment such as Pd or Max/MSP.

As can be seen in fig.1, only a small portion of the FTM implementation actually depends on the real-time host environment.

The platform-dependent parts of the FTM architecture are isolated into two APIs:

- **FTMEXT** (FTM external interface): Unified C-API for the implementation of external modules hiding the native API of the host environment.
- **FTMRTE** (FTM real-time environment): Interface to native elementary data structures *(atoms)* and services of the host environment such as error messages, dialogues and data persistence mechanisms.

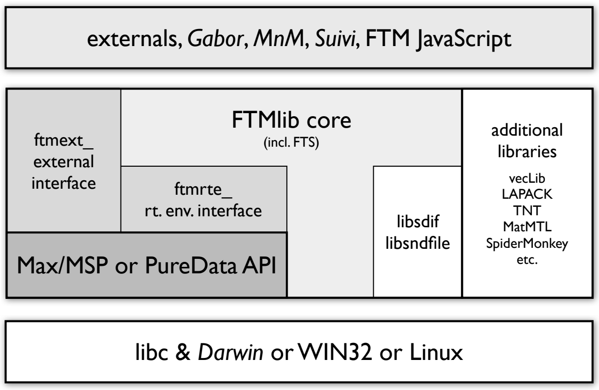
externals, Gabor, MnM, Suivi, FTM JavaScript
ftmext_ external interface
FTMlib core
(incl. FTS)
additional libraries
vecLib
LAPACK
TNT
MatMTL
SpiderMonkey
etc.
ftmrte_ rt. env. interface
libsdif
libsndfile
Max/MSP or PureData API
libc & Darwin or WIN32 or Linux

Figure 1: Overview of the FTM architecture

While the rest of FTM is implemented in a generic, platform-independent way, the FTMEXT and FTMRTE APIs had to be re-implemented for Pd based on the Max/MSP version.

2.1 The FTMEXT API

The FTMEXT API has been finalised in parallel to the FTM Pd porting. The API represents a simplified version of the external APIs of of Max/MSP and Pd functionalities and can be seen as their "greatest common divider". It has been designed also regarding future implementations for JavaScript, VST or similar host environments.

Since most of FTMEXT has been implemented using preprocessor macros, it does not provide binary compatibility. In consequence, plugins compiled for a certain real-time environment will need to be recompiled for every other real-time environment (even if both environments were running on the same operating system).

Furthermore, the macro approach forces all externals to be recompiled whenever the implementation of the FTMEXT API changes.

These two issues strongly suggest to wrap the FTMEXT interface into a dynamic library and minimise 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 Modules

Since FTM extends the elementary data types handled by its host environment, it has to provide a set of modules that can handle newly introduced values and objects. The standard distribution FTM includes a set of modules giving access to the libraries data structures and basic functionalities.

3.1 Generic Core Modules

Most of the FTM external modules use a restricted set of functionalities provided by the host environment and can be implemented as FTMEXT externals. However, some of these generic modules have actually been implemented directly using functions of the Max/MSP externals API. These modules have been adapted or re-implemented to use the FTMEXT API in the framework of the FTM Pd porting. As a consequence FTM now has a complete set of platform-independent external modules that also constitute an excellent test-bed for the FTMEXT framework since it covers most of the functionalities of the FTMEXT API.

Moreover this work has contributed to the portability of FTM and simplifies the eventual port of FTM to further host environments.

3.2 Platform Dependent Core Modules

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

The most notable of these are `ftm.object` the `ftm.mess` that rely on the host environments graphical capabilities. The `ftm.object` module provides the possibility to statically instantiate FTM objects and to associate them with names. The module `ftm.mess` provides an extended message box that also allows for accessing FTM data structures and the evaluation of functions and infix expressions. In order to provide a complete set of modules for the first version of FTM for Pd, two platform-independent alternative modules, `ftm.o` and `ftm.m`, have been developed that are entirely based on the FTMEXT API.

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. However, FTM is currently moving away from using Java as a GUI-toolkit in favour of the cross-platform C++-toolkit "juce"[3].

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 Pd 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 real-time environment within the FTM-framework, porting of the basic functionality was rather painless.

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

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 Musical Interaction Group. Extra thanks to Tommaso Bianco, who did the first steps in porting FTM to Pd and to Pierre Duquesne who laid the first stones.

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. ICMA.

[2] 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.

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

[4] Miller S 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. International Computer Music Association.

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

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