FTM — COMPLEX DATA STRUCTURES FOR MAX

Norbert Schnell *Riccardo Borghesi* *Diemo Schwarz* *Frederic Bevilacqua* *Remy Müller*
IRCAM — Centre Pompidou
Paris, Europe
Real-Time Applications Team & Performing Arts Technology Research Team

ABSTRACT

This article presents FTM, a shared library and a set of modules extending the Max/MSP environment. It also gives a brief description of additional sets of modules based on FTM. The article particularly addresses the community of researchers and musicians familiar with *Max* or Max-like programming environments such as *Pure Data*.

FTM extends the signal and message data flow paradigm of Max permitting the representation and processing of complex data structures such as matrices, sequences or dictionaries as well as tuples, MIDI events or score elements (notes, silences, trills etc.).

1. INTRODUCTION

The integration of references to complex data structures in the Max/MSP data flow opens new possibilities to the user for powerful and efficient data representations and modularization of applications. FTM is the basis of several sets of modules for Max/MSP specialized on score following, sound analysis/re-synthesis, statistical modeling and database access. Designed for particular applications in automatic accompaniment, advanced signal processing and gestural analysis, the libraries use a common set of basic FTM data structures. They are perfectly interoperable while smoothly integrating into the modular programming paradigm of the host environment Max/MSP [5].

Inheriting most of its functionalities and implementation from the former jMax project [2], FTM concentrates on providing a set of optimized services for the handling and processing of data structures related to sound, gesture and music representations in real-time. FTM includes a small and simple C-written object system and graphical Java editors embedded into Max/MSP. FTM is distributed in form of a shared library and a set of external modules under the LGPL open source host [1] .

The acronym FTM is a reference to FTS *Faster Than Sound* [6], the real-time monitor underlying the Max software on the ISPW platform, which became later the sound server of other real-time platform projects at IRCAM. One can imagine FTM standing for *Faster Than Music* or *Funner Than Messages*.

The original motivation for the development of FTM was the need for a flexible score representation related to score following and an efficient representation of matrices and vectors allowing for modular implementation of various analysis/re-synthesis algorithms in a unified framework. Today, external modules for both applications have been implemented in packages: *Suivi* and *Gabor*. Further packages are following addressing gesture analysis and database access (see section 3).

FTM is available for Max/MSP on Mac OS X and Windows. The porting of FTM to *Pure Data* [7] on Linux is in progress.

[1] http://www.gnu.org/licenses/

2. FTM FEATURES AND SERVICES

The features of FTM can be summarized as follows:

- static and dynamic creation of data structures (*FTM objects*) of predefined classes
- editors and visualization tools
- expression evaluation including functions, method calls, and arithmetic operators
- import/export of text, standard MIDI, SDIF [9] and the usual sound file formats
- object serialization and persistence

2.1. Data Structures and Operators

FTM allows for static and dynamic instantiation of FTM classes creating FTM objects. Static FTM objects are created in a patcher using a dedicated Max/MSP external module. Dynamic object creation is provided by a new-function within the FTM message box and by other external modules. The objects are represented by references, which can be sent within the data-flow between the Max modules as arguments of lists and messages.

FTM strictly separates data objects and operators. Only basic operations on FTM objects are implemented as methods of the FTM classes which can be invoked within the FTM message box or by sending a message to a statically created object. More complex calculations and interactions with objects are implemented as Max/MSP external modules receiving references to FTM objects into their inlets or referencing objects by name as their arguments.

2.1.1. Classes and Objects

The following FTM classes are currently provided with documentation:

mat ... matrix of arbitrary values or objects
dict ... dictionary of arbitrary key/value pairs
track ... sequence of time-tagged items
fmat ... two-dimensional matrix of floats
fvec ... reference to a col, row or diag of an *fmat*
expr ... expression
bpf ... break point function
tuple ... immutable array of arbitrary items
scoob ... score object (note, trill, rest, etc.)
midi ... midi event

FTM classes are predefined. They are implemented in C and optimized for real-time performance. The classes themselves are kept as generic as possible providing a maximum of interoperability.

In the current packages based on FTM, the matrix and dictionary class are mainly used to organize data. Since they can contain references to other objects they easily allow for building up recursive structures such as matrices of matrices or dictionaries of sequences. The *tuple* class gives the possibility of creating lists of lists (i.e. tuples of tuples). The classes *fmat* and *bpf* are well adapted to real-time processing of sound and movement capture data.

The generic two-dimensional float matrix *fmat* represents various data such as vectors of sounds samples, spectral data, coefficients and movement capture data. Complex calculations are implemented within specific methods, functions or processing modules requiring two-column matrices as input.

2.1.2. Modules, Messages, Names and Expressions

FTM is released with a set of external Max/MSP modules providing basic functionalities for the creation and handling of objects and operations related to the provided classes. All operations on FTM objects requiring or providing additional memory, timing or visualization are implemented in form of external Max/MSP modules rather than methods of the FTM classes.

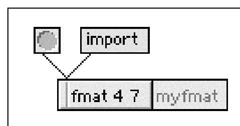

Figure 1. Example of a static FTM object named *myfmat* in a Max/MSP patcher

Figure 1 shows an FTM object statically created with the `ftm.object` module in a Max/MSP patcher. The module defines a *fmat* matrix of floats with 4 rows and 7 columns. The module redirects all incoming messages to the defined FTM object. A bang message causes the module to output a reference to the object from the left outlet. The object can be given a name within global or local scope. Local scope is limited to a patcher file such as a loaded top level patcher or an instance of an abstraction. This way local names can be defined and used within a patcher file and all its sub-patches, while different patcher files and abstractions can have each their private name definitions. FTM names are used with a leading '$' character in all FTM modules including the FTM message box.

The persistence of the content of a static FTM object as well as its name and scope can be set by graphical interactions with the module or using an associated Max/MSP inspector. A persistent object saves its content within the Max/MSP patcher file and restores its content when it is copied and pasted to a patcher using the serialization mechanism described at the end of 2.1.3. More over, the content of any statically defined object can be saved to and restored from a text file.

Static object definitions are invalid (i.e. `ftm.object` appear opaque) when they include a reference to an undefined name and turn automatically into valid objects as soon as the name is defined by another `ftm.object` module.

The module `ftm.mess` provides the possibility to compose and output messages in a way which is similar to the usual message box built into Max/MSP. As an extension the FTM message box allows the dynamic evaluation of expressions. Figure 2 shows several examples of expressions in the `ftm.mess` module. Function calls require parenthesis around the function name followed by the arguments. Method calls are similar within parenthesis starting with an object followed by the method name and arguments. Methods can be invoked on objects referenced by name or by references received into the inputs using numbered references (i.e. $1, $2, etc).

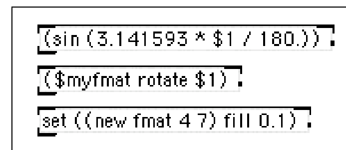

Figure 2. Three examples of messages using expressions in the FTM message box

The return values of FTM class methods can be used in an arithmetic expression or as argument of another method or function call.

2.1.3. References, Data-flow and Persistence

The introduction of references to complex data structures into the Max data-flow creates new possibilities as well as unusual Max programming paradigms. While Max messages are immutable and copied in order to perform successive calculations module by module following the patches connections, the FTM objects floating in a Max patch are often modified by the modules they traverse.

Figure 3 shows a simplified example of a patch calculating the logarithmic magnitude of an FFT spectrum and a smoothed spectral envelope of a frame of 512 samples

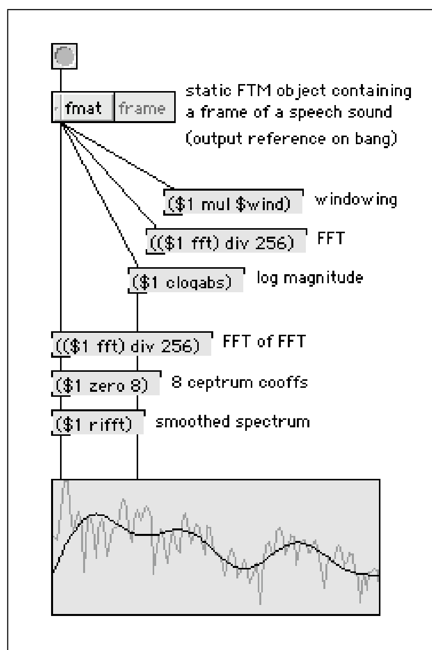

Figure 3. Max data flow with in-place calculations

of a speech sound. Each of the messages boxes invoke one or two methods performing an in-place calculation which destructively transforms the content of the matrix. Some methods even change its dimensions.

The Max control flow executes the message boxes in right-to-left and top-to-bottom order. All of the *fmat* methods used in the example (`mul`, `div`, `clogabs`, `zero` and `rifft`) return a reference to the *fmat* named *frame*. The method calls can be chained as in the expression `‘(($1 fft) div 256)‘` and the message boxes can be connected in series or in parallel respecting the right-to-left output order. Since the execution order of two Max modules connected to the same outlet depends on the graphical position of the modules, moving FTM modules in the patcher window can change drastically the result of the calculation. Like the Max message box, `ftm.mess` allows for separating expressions by comma successively evaluating and outputting the resulting values or lists in left-to-right order. A semicolon instead of the comma suppresses the output while keeping the expression evaluated.

FTM objects such as a *mat*, *dict* or *track* can contain references to other objects. More over many FTM modules, such as the message box or the module `ftm.play` interpreting a *track* sequence, store references to objects. The destruction of statically or dynamically created objects is handled by a simple reference count garbage collector. Objects which have been referenced by other objects or FTM modules are immediately destroyed when the last reference to the object has been released.

For persistency, FTM provides a serialization mechanism recursively saving the content of objects and the objects contained as references. Using this mechanism, FTM objects containing FTM objects can be saved and restored within a Max/MSP patcher file and copied and pasted between patchers.

2.2. FTM Interfaces, Interchange and Integration

2.2.1. Graphical Editors

FTM provides editors for most of the complex classes. Similar to other Max/MSP modules, an editor can be opened by double-clicking on the an `ftm.object` module. While some editors such as those for the *track* and *bpf* classes allow for a graphical representation of the objects content, others consist of a simple textual table view (e.g. *mat* or *dict*). All editors use Java [2] and integrate into Max/MSP using the *mxj* Java interface.

The currently most developed FTM editor is available for the *track* class when representing a sequence of *scoob* objects. The editor provides a chronometric representation for musical scores as shown in figure 4. The representation integrates score events such as notes, rests and trills, with a temporal structure of bars and additional markers. The same content can be edited in parallel in a table view describing the displayed score objects and markers as a textual list.

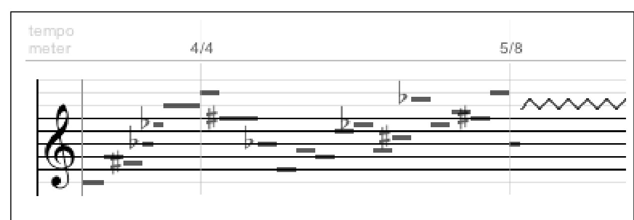

Figure 4. Detail of a screenshot of the score editor

2.2.2. SDIF

The Sound Description Interchange Format (SDIF) [9] is a file format of increasing popularity for the storage and exchange of sound data in various representations including frequency domain descriptions such as partials, spectral envelopes, STFT frames, FOF parameters or LPC coefficients but also PCM samples or PSOLA markers. SDIF can also be used for motion capture data and other data sets with a temporal development.

The import and export methods of the FTM *track* class support SDIF. A *track* object represents an SDIF file as a sequence of *fmat* objects, each *fmat* object representing a matrix of the SDIF file.

2.2.3. Max/MSP Integration

FTM can be seen as partly independent from Max/MSP and can easily be integrated into other environments. However special care is taken to assure the seamless integration of FTM into Max/MSP.

It has been chosen to represent references to FTM objects in the Max data-flow on the level of elementary types such as int, float and symbols. Single FTM objects are sent as a single argument of a special message “ftm.obj“, which is only understood by the FTM external modules.

[2] http://java.sun.com/

As a consequence some Max/MSP modules such as `pack` don't apply to FTM objects. In this case an FTM specific replacement is provided.

2.2.4. *Platform Independent API for External Modules*

FTM provides an API for the development of FTM modules such as Max/MSP externals independently from a specific real-time environment. The API is currently implemented for Max/MSP and *Pure Data*. It assures the possibility of easily porting the available FTM modules to any Max-like environment. The API supports the declaration of Max/MSP-style attributes and transparently includes a redefinition mechanism for named references to FTM objects in instantiation arguments as explained in section 2.1.2

3. FTM PACKAGES

Several packages dedicated to different domains of application are available for FTM.

The package *Suivi* contains modules performing score following based on *Hidden Markov Models* (HMM) [4].

The package consists mainly of two objects performing score following on MIDI and audio input. They reference *track* objects containing the score information.

The *Gabor* package is a toolbox for analysis/re-synthesis applications. The modules of the *Gabor* package are built around the notion of generalized granular synthesis processing atomic units of short sounds (frames, grains, wave periods, etc.) [8]. *Gabor* provides a unified framework for granular synthesis, PSOLA [3], phase vocoder and other overlap-add techniques.

The package *MnM* ("Music is not Mapping") [1] is a set of modules providing basic linear algebra, mapping and statistical modeling algorithms such as *Principal Component Analysis* (PCA), *Gaussian Mixture Models* (GMM) and *Hidden Markov Models* (HMM). The close integration of motion capture with complex statistical models and sound analysis/re-synthesis is a promising platform encouraging the development and composition of new artistic applications going far beyond simple mappings.

The most recent package *FDM* (*FTM Data Management*), introduces relational SQL database access (SQLite) using FTM classes (*mat, dict*). Other FTM objects can be stored as BLOBs (*fmat*) or using the serialization mechanism. The package prepares the real-time implementation of concatenative data-driven synthesis [10].

4. CONCLUSIONS

FTM provides a consistent set of features integrated to Max/MSP opening new possibilities for the development of interactive music and multi-media applications. FTM successfully absolved a phase of proof-of-concept and is today freely distributed with a set of packages oriented towards different domains forming a coherent ensemble around a kernel of basic FTM modules.

FTM and its libraries have been successfully employed in various concert, dance and theatre performances for score following, voice and sound processing, mapping and gesture recognition.

FTM is released under the *Lesser GNU Public License* (LGPL). Recent releases are available from the web page of the IRCAM Real-Time Applications Team [3] . The packages *Gabor*, *MnM* and *FDM* are released with the FTM distribution for Max/MSP. *Suivi* as well as advanced examples and additional phase vocoder components are available within the IRCAM Forum [4] .

5. ACKNOWLEDGMENTS

The development of FTM wouldn't have been possible without the contribution of musical assistants and com-