The jMax environment: an overview of new features

François Déchelle, Norbert Schnell, Riccardo Borghesi, Nicola Orio
{
dechelle, schnell, borghesi, orio}\@ircam.fr
IRCAM - CENTRE GEORGES-POMPIDOU
1, pl. Igor Stravinsky, F-75004 Paris, France
http://www ircam fr

ABSTRACT

This paper presents the recent developments of jMax, IRCAM’s control and signal real-time processing system. After highlighting the contribution of the free software community, it will make an overview of the recent developments in jMax: kernel architecture, configuration and user environment, network audio streaming. The sequencer and the score following frameworks will be detailed. A comparison of the performance of the different platforms running jMax will be given. An experimental pure JAVA implementation of the MAX visual programming paradigm will finally be described.

1 FREE SOFTWARE

jMax is distributed under GNU’s General Public License since July 1999 [IRCAM, 1998]. This decision has been warmly welcomed by the computer music community and has since proved its effectiveness.

The first contribution of the jMax free software community has been an improvement both of the compilation and installation procedure and of the documentation. Another contribution includes development of objects and patches and is still to be completed; its importance is essential for the success of projects based on jMax.

A set of porting initiatives was started in the first half of the year 2000: porting to MacOS has been undertaken by Aadaljan van der Helm at V2 Lab and porting to Windows is under development by Peter Hanappe.

2 SERVER KERNEL

The jMax architecture is a “client-server” architecture where the client is the JAVA graphical user interface and the server is the FTS real-time computation engine [Déchelle and Cecco, 1995, Déchelle et al., 1998a, Déchelle et al., 1998b, Déchelle et al., 1999a].

Differences in dynamic linking characteristics between Unix one on hand and Windows and MacOS on the other hand have motivated a re-organization of the servers architecture.

The kernel of the jMax server has been split into a library and a set of executables. The library contains the message interpreter, the DSP interpreter and the scheduler. The final jMax server FTS executable is built from this library.

However, the improved modularity of the FTS kernel permits the creation of stand-alone applications as well as its integration into plug-in environments such as VST and LADSPA.1

The communication between client and server has been merged with the FTS message system using the same kind of messages sent between an FTS object and its client proxy as between two FTS objects (such as in a patch). This significantly simplifies the creation and handling of objects with graphical user interfaces.

Moreover, a simplified FTS object class has been implemented which is used for all objects which are not visible in a patcher. These simple FTS objects are used for instance in the sequence data structure described below (see 5): each track as well as each event is a simple FTS object with an individual client proxy.

3 PROJECTS, PACKAGES, CONFIGURATION AND SCRIPTING

The management of projects and packages, including dynamic loading of components and devices configuration, was formerly handled using Tcl scripts interpreted by the jMax client [Déchelle et al., 1998a, Déchelle et al., 1999a]. Apart from its lack of support and code size, Tcl offers limited Java integration. It has therefore been decided [Déchelle et al., 1999b] to drop Tcl and to use Scheme as the jMax scripting language.

In order to be able first to run the JAVA graphical user interface and the server on machines that do not share the same file system, and second to configure the server without the JAVA GUI, a very simple configuration language is under development. This language is used for both the configuration of the system (client and server) and for description of packages and projects [Déchelle et al., 1998a]. The configuration, project and package files are loaded on both client and server side; this allows the server to run without the client, for instance in a plug-in application or as a stand-alone non real-time audio processor.

The Scheme scripting language is integrated into the graphical user interface, giving access to edition and control functionalities, as an complement and alternative to graphical programming. Scripts can be easily integrated in the menus of the GUI or be associated with key bindings.

4 REAL-TIME NETWORKED AUDIO STREAMING

A network audio device has been developed in jMax for sending and receiving a continuous audio stream over a standard IP network. Its main purpose is to exchange audio signals in real-time between several FTS processes running on different machines, thus allowing the development of distributed audio processing applications. It can also be viewed as an efficient way to exchange audio data between jMax and other applications independently of the platform.

Existing implementations of real-time media streaming mostly use RTP (Real-Time Transport Protocol) [Schulzrinne et al., 1996]. RTP is a transport protocol designed for the delivery of real-time data, including audio and video. Even if RTP is independent of the underlying transport layer, it is mainly associated to the UDP transport protocol which is much better suited for real-time applications than TCP because of its low-latency characteristics.

However, the standard formats defined by RTP do not fit well to multi-channel audio streams, and implementing a new RTP type...
would have been another project altogether. The choice has there-
fore been to define a simple packet format and to use UDP as the
transport protocol. In the current implementation, each packet
header contains the following fields: sample rate, sample format
(size and endianism), number of channels, number of samples in
packet and packet counter.

The JMax network audio device is integrated in the stan-
dard JMax devices framework [Déchelle and Cecco, 1995,
Déchelle et al., 1999a, Déchelle, 2000], thus allowing it to be used
for default audio input and output accessed though the adc and
dac objects.

An extension of the network audio streaming, currently un-
der development, is the connection of JMax with multicast MP3
streaming servers such as kecast. A planned development is the
use of the existing device for streaming SDIF (Sound Description
Interchange Format) parameters [Wright et al., 1998].

5 SEQUENCING IN JMAX

A new object, called sequence, has been developed in JMax to
deal with sequences of events. In this context, an event may be any
kind of time-tagged data characterized by its onset time2 inside
the sequence and its value. The value can be of any kind, from integer
to compound object.

The sequence object represents a generic data structure. It can
be used to perform or record events through external player and
recorder objects. Files can be imported from and exported to in-
terchange file formats like Standard MIDI Files and SDIF.

Events are collected in multiple tracks. Each track contains
events of the same type providing a clear and efficient interface
for graphical editing and addressing of events from the client and
by different JMax objects embedded in a patcher.

The sequence object is also the principal data structure used
in the score following framework, which will be described in
Section 6. The data of the performer’s score as well as the
electronic score of real-time control parameters will be handled by
the sequence object.

5.1 The Sequence Editor

A complete sequence editor, loosely based on the object
“explode” [Puckette, 1990], has been developed for the sequence
object. The editor can deal with different kinds of events in a
unified way adapted to its data type.

The different tracks are horizontally aligned to the same
time-scale providing visual cues to help the user in aligning
the events of different tracks temporally. A ruler with units,
alternatively in seconds, milliseconds, or samples, is displayed on
the top of the window.

A number of tools for editing tracks and events has been
developed. The same tools are used for all kinds of tracks and all
kinds of events. Hence, together with classical functionalities in
the “Edit” menu, the GUI provides tools for selecting, inserting,
deleting, and changing the value of any event as well as its
position in the sequence (i.e. timestamp).

Each track has a contextual pop-up menu. Common function-
alties are insertion, deletion, selection, change in the top-down
ordering of tracks and iconification of a track. Moreover, it is pos-
sible to have a textual representation of a track as an ordered list of
events, displayed as a separate window. All editing operations are
synchronized between the graphical and textual view. Changing
the onset time or a value of an event in the textual view will also
move or transform its graphical representation.

The editor provides vertical and horizontal scrolling. plus
zooming that affects all tracks. A screen-dump of the editor, with
five tracks, is depicted in Figure 1.

So far, tracks of three different event types are supported: notes,
integers, and messages.

- The note track

The note track contains events whose data-types are note val-
ues. At the moment, these events are represented by a pair of
values corresponding to pitch and duration in addition to its times-
tamp.

There are two alternative graphical representations of a note
track: the classical piano-roll view and a simplified music nota-
tion view. The latter is halfway between a piano-roll notation and
classical score notation, where up to six staves are displayed in
order to cope with all the possible pitch values. Each event is rep-
resented by a horizontal bar (see figure 1).

- The integer track

There are three alternative views of an integer track: peaks, steps,
and breakpoints. In all of the views, the integer value is repre-
sented by the height of the graphical object, which is left adjusted
to the event timestamp. In steps view, each event is represented by
separate vertical bars; in step view, each event is represented by a
rectangle, lasting until the next integer event; in breakpoint view,
events are represented by dots connected by line segments. The
user may choose the most suitable view depending on the mean-
ing of the stored integer events3.

- The message track

The message track contains events that are JMax generic mes-
gages. Graphically, each event is represented by a box which is
left adjusted to the event timestamp. The message can be edited
either directly in the graphical view or through the associated tex-
tual view. In order to simplify the overall view of the track, each
message box can be iconified.

The vertical range of a track (i.e. the displayed values) can be
adjusted from the contextual menu both in note and in integer
tracks.

In the future, the sequence editor will be enhanced, in particular
to cope with new requirements of the score follower framework
which defines more complex events. An overview of these events
is given in Section 6.1.

6 THE SCORE FOLLOWER

The problem of automatically following a performance, played
either on a MIDI or on an acoustic instrument, has been of interest
to the computer music community since 1984. Dannenberg
[Dannenberg and Thompson, 1984] and Vercoe [Vercoe, 1984]
dependently proposed two different solutions to the problem.
Different approaches to score following problems have been
proposed, among them the Max explode object, developed by
Puckette [Puckette, 1990], which is already implemented in JMax.

The goal of these new developments is to satisfy the needs of
composers inside and outside Ircam. A first target is to create a
more open definition of followed sound events and performed ac-
tions which could be more complex than simple MIDI-like notes.
A second target is to provide a representation of complex scores
which more closely resemble the actual performance state during
the synchronized performance (i.e., events recognized, events followed, and action performed).

Here follows a brief overview of the problems that are ad-
dressed in the score follower framework.

3Similar to the diversity of its graphical representation, the same data can be dif-
ferently interpreted by different Max objects referring to the same sequence track. It
is evident that two different algorithms must be used in order to output simple integer
values or generate breakpoint functions. For now, the choice of the graphical repre-
sentation and the algorithm processing the events of a certain track are completely
independent.
Automatic recognition of sound events. The common approach for recognizing sound events is the development of a pitch tracker, usually in the form of a pitch-to-midi converter. While this approach is practical for older tonal western music, it does not suffice for contemporary music. In fact, the relevant feature of some compositions, hence the one that should be recognized and followed, may be any sound parameter: absolute or relative amplitude, attack time, timbre or spectral information, absolute or relative duration. A number of different analyses should be used to process the incoming sound, each one extracting a feature of interest. The kind of performed analysis depends on the indications in the score which are known in advance. The possibility of a mismatch between the performance and the score, due to musician or system errors, suggests the use of parallel analyses when different features have to be followed.

Segmentation of incoming audio flow in sound events is required whenever the relevant parameters of a performance are of a different nature. A score follower based on pitch tracking performs an implicit segmentation of pitch contour, usually depending on quantization of pitches. With a more general definition of events, it is required to perform a segmentation based on a combination of different strategies, like onset and offset detection [Smith, 1994], amplitude envelope analysis [Bajda, 1996], or spectral flow [Rossignol et al., 1998]. Segmentation can be used to identify sound events, which may be defined or recognized just in terms of their length. Moreover, it is also useful for extracting timing information about the time elapsed between two subsequent segments and about segment length.

Synchronization. In real-time systems, the problem of synchronization has two key aspects: latency and the notion of local time. Apart from classical real-time issues, the problem of latency arises in particular cases where the processing algorithm requires periodic accumulation of data as in the classical example of pitch tracking for low frequencies. In the case of score following, latency becomes an issue when the automatic performance is event-driven by the musician’s performance. A solution, which will be investigated, is the use of different cues for following and for synchronizing. That is, given the knowledge of the previous position in the score, the system should “trust” that the musician will not make any error and synchronize just with the onset of the next event. It can be noted that this situation is typical in synchronization among musicians: a piano player does not wait to recognize the pitch sung by the soprano to prepare and play the chord on the keyboard. On the other hand, the notion of local time is necessary whenever the system is requested to perform events between the musician’s events or during silence. Local time computation
for beat tracking is a well-known problem including the case of audio input [Goto and Muraoka, 1998]. A prototype tool, based on Large’s model [Large, 1996], has already been developed at Ircam.

● Displaying system state. The development of a user-friendly editor, able to store and represent the complex information of a contemporary music score, has been addressed in Section 5.1. Usually, there is no conceptual difference between the representation at editing time and the one at performing time. On the other hand, in the framework of the JMax score follower, it has to be considered that the system will need a supervisor during rehearsal and concert. Operator tasks include controlling the correctness of the following, resynchronizing in case of mismatch, and eventually interacting via the normal user interface. It is likely that the operator will be familiar with the score and with JMax patches, and so most of the detailed information that is needed by the score follower is not needed by the operator. This then justifies a separation between the editing interface and the real-time display.

The first step in the development of the score follower in JMax has been to define the classes of sound events that can be recognized and followed. The term “class”, derived from object-oriented programming, is used in this context because it can highlight a hierarchical structure among different events, with rules of inheritance. In Section 6.1, sound events are described using this object-oriented approach that will also be helpful in Section 6.2, where the graphical interface is discussed.

6.1 Classification of Sound Events

At a first level, two categories of events are defined. It is possible to compare the two categories as two superclasses. The first superclass groups Atomic Events, which are sound events where the spectrum has only slight changes in time (e.g., due to a vibrato). Atomic Events are presented according to a degrading level of generality, which in turn may correspond to a hierarchical ordering among event objects. For each event highlighted the default properties, but it should be noted that the final choice of the properties of each event is left to the user, that is to the composer.

○ Quasi-stationary Events
spectral envelope almost constant over time, no assumption about harmonicity; time-varying amplitude; eventual onset and offset times; example: a non-percussive noise from the voice or wind instruments.

○ Tone Events
constant pitch, possibly with vibrato; time-varying amplitude; eventual onset and offset times; example: a note from any instrument without glissando or strong legato.

○ Midi-like Events
pitch and amplitude completely defined at the beginning of the event; onset timestamp; not necessarily coming from a Midi interface or a pitch to midi converter; example: a tone from any instrument without vibrato or changes in amplitude.

○ Pulse Events
no relevant, or perceptible, pitch; amplitude, and possibly spectrum, totally defined at onset time; short duration; example: a percussive noise from any instrument or an occlusive consonant.

The second superclass groups Non-Atomic Events, which have a time-varying characteristic of spectrum. It may include either the presence of a pitch contour or changes in the spectrum. Non-Atomic Events are presented according to a decreasing level of generality.

○ Noisy Events
time-varying spectrum, not necessarily harmonic; time-varying amplitude; starting and ending times; example: a noise with a varying spectral envelope.

○ Phrase Events
continuous pitch trend, not likely to be segmented in a sequence events; time-varying amplitude; starting and ending times; example: a sequence of tones with a glissando or a strong legato.

There exist other sound events which are not considered in this classification, for instance noise that gradually becomes a pitched sound and, most of all, chords. In any case, it is impossible to make a complete classification of all the possible events, because this would mean drawing boundaries on musicians’ research on acoustic sound production.

As mentioned in the previous section, the proposed classification of sound events is conceptual, or representative, and does not correspond to the actual implementation. In JMax these events are objects with a number of different properties, like pitch, amplitude envelope, duration, spectral centroid, and so on. The features highlighted for Atomic and Non-Atomic events can be seen as the default properties. Another property of sound events is the feature, or the list of features, which the system has to consider for recognizing, matching, and following incoming sound events performed by the musician. This feature allows the composer to choose and, eventually, to modify the strategy used by the system to synchronize with the performer. Also in this case, each kind of event has a default feature for the following.

6.2 The Score Follower GUI

To speak as the Human-Computer Interaction community, a GUI is used to display what is relevant, from the user’s point of view, of the internal state of the system. Considering that the score follower will be used in a “on stage” context, and with a final user who is familiar with the score, (and familiar with JMax environment), only a restricted number of the system parameters has to be displayed. This implies that more resources will be available for JMax server, and that the user will not be overwhelmed by redundant information.

Because of this approach, the interface for editing is different from the interface for displaying. The interface for editing, which has already been developed, is general purpose within JMax and can be thought as a tool for the score follower. The interface for displaying, which is under development, has to deal with the possible needs of the user, who probably would like to have:

● a scrolling window with a representation of the score performed by the musician and the actual position.

● an eventual simplified representation of the complete score, which will be helpful during rehearsals, with widgets for moving along the score and starting/stopping the follower.

● the cues used to recognize and to follow, especially when they are not the default cues; these cues can be represented by icons connected to their corresponding event; the user will be able to resynchronize the system on a given event just by clicking on its icon.

● the actions which the system will take when following the performance; they may be represented by icons connected with their triggering events, which in general may differ from followed events; the user will be able to force the system to make a given action just by clicking on its icon.

Since western humans typically process information left-to-right and top-down, it is likely that the score will roll to the left, with the icons for following and taking actions respectively over and below it.

The most important aspect is a compact and clear representation of the score. To this aim, the proposed classification of sound events becomes useful. A number of different visual cues, which should be easily recognizable by the user, have been designed, one for each different sound event. In the implementation, the score is stored as an array of events. Each event defines, through its representation property, the way it has to be displayed. The display property, together with properties regarding features to be
recognized/followed and features to be used for controlling, has a default value which can be changed by the user. Here follows a list of representation of the sound events previously classified:

- **Quasi-stationary Events** are represented by a colored, continuous line, which shape displays, by analogy, the cue feature of the event, like loudness or spectrum; a different shape (or color) for the line is associated to the particular feature;
- **Tone and Midi-like Events** are represented by the classical score or piano-roll displaying;
- **Pulse Events** can be represented both by a score-like display, by a continuous line which trend represents the melody shape, or both depending on user’s choices;
- **Noisy Events** can be represented in a variety of ways, from the totally analog (a sonogram) to the totally symbolic (semantic labels).

### 7 PLATFORMS COMPARISONS

In order to compare the performance of the different platforms running jMax, a useful measurement can be the maximal number of samples that can be computed by a patch in a fixed time interval. This maximum theoretical sampling rate is obtained by running the server without synchronizing it with an external sampling clock.

Table 1 details this performance comparison, the chosen patches being one example patch from the Spatialisateur library [Jot and Warusfel, 1995] and the patches of En écho, a piece by Philippe Manoury for voice and electronics which has been created on the Ircam Musical Workstation. The results are the maximum theoretical sampling rate in kilohertz.

A obvious remark that follows from this comparison is the excellent performance of the Linux version that provides, on low-cost hardwares, a computing power superior to Silicon Graphics R10000 CPU.

### 8 PJMAX, A PURE JAVA MAX

The fact that, in jMax, the implementation language of the graphical user interface is JAVA [Déchelle et al., 1998a, Déchelle et al., 2000] led naturally to the idea of implementing in JAVA also the computational part, i.e. the MAX language interpreter itself. This implementation, as yet purely experimental, is named pjMax (for Pure Java MAX) and offers an implementing in JAVA also the computational part, i.e. the MAX component framework and the event design pattern. According to Sun’s definition, a JavaBeans component is a reusable software component that can be manipulated visually in a builder tool. The JavaBeans API introduces the concept of events, where an event source contains a dynamic list of listener objects and fires an event by calling the listener’s methods. In this framework, the definition that seems the best suited for an inlet is a graphical representation of a listener’s methods. Similarly, an outlet is a graphical representation of an event source (it must be noted that a JavaBeans component can be a source of different events). A MAX message is then nothing other than a JavaBeans event.

It follows from the previous considerations that pjMax uses all the JAVA mechanisms: the pjMax dispatch mechanism is the JAVA dispatch mechanism, a pjMax class is a JAVA class, a pjMax object is a JAVA object, etc.

Figure 2 gives as an example the code of a PJMax object, here the inlet object that converts a MIDI pitch to a frequency.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Patch</th>
<th>CPU</th>
<th>MIPS</th>
<th>IS</th>
<th>Pentium-III</th>
<th>Linux</th>
<th>PPC</th>
<th>En Echo</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Patch</td>
<td>Patches</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LR10000</td>
<td>225</td>
<td>136</td>
<td>61</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pentium-III</td>
<td>600</td>
<td>256</td>
<td>73</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G3</td>
<td>400</td>
<td>195</td>
<td>71</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: An example of a PJMax object code
REFERENCES


