**JMax: A New JAVA-based Editing and Control System for Real-Time Musical Applications.**

Déchelle, François, dechelle@ircam.fr
Borghesi, Riccardo, borghesi@ircam.fr
De Cecco, Maurizio, dececco@ircam.fr
Maggi, Enzo, maggi@ircam.fr
Rovan, Butch, rovan@ircam.fr
Schnell, Norbert, schnell@ircam.fr

IRCAM - 1 place Stravinsky - 75004 PARIS - France

**Abstract**

In this paper we present jMax, the new editing and control system for Ircam's real-time musical applications. More than a simple evolution of MAX, jMax is a completely new and extensible environment. The components of the architecture will be presented, as well as its application framework.

**Introduction**

jMax is the latest generation of the programming and runtime environment developed and used at Ircam for real-time audio and signal processing applications. Its new architecture has been presented at ICMC'96. Still organized around a real-time DSP server (Déchelle et al., 1998a), the graphical user interface is now fully JAVA-based.

1 - Architecture

The client/server architecture of jMax has previously been described in (Déchelle et al., 1994; Déchelle & De Cecco, 1995). Although the implementation language of the graphical user interface has changed, the global structure is unchanged, with a communication protocol that can utilize various transport layers: pipes, TCP/IP and UDP sockets, or shared memory in specialized hardware architectures.

As described in (Déchelle & De Cecco, 1995), previous client applications of FTS were build on top of a "client library" implemented in C, which handled low level access to the protocol. The new JAVA version of this library offers the same kind of services, and extends them by providing a JAVA abstraction of the FTS objects. This library, named the application layer, is described in more detail in (Déchelle et al., 1998a).

2 - JAVA choice

The new jMax graphical user interface is fully JAVA-written (Maggi & Déchelle, 1996). The choice of JAVA was initially guided by portability considerations: without Java, porting of the graphical user interface would be extremely costly and would require at least three different implementations.

Apart from its multi-platform capabilities, however, JAVA also provides a rich programming environment that can be used to extend the system in many ways, opening a wide range of extensions, including:
- editors
- control panels
- general purpose extensions, such as an integrated HTML browser for on-line documentation (IRCAM, 1993; IRCAM, 1998).

3 - Scriptability

In addition to the role of jMax as a fast prototyping and interactive audio environment, the number and complexity of the musical applications written with jMax have shown the need to also use it as a coherent and flexible programming environment. Consequently, user configurability, automation of repetitive tasks, and the notion of scriptability have become of paramount importance.

The issue was not solved satisfactorily in the preceding environment (MAX). One of the major drawbacks of the graphical MAX user interface paradigm is, in fact, its unsuitability for complex actions, i.e., actions that depend on parameters, or that are inherently repetitive. In addition, the multiplatform implementation of the new
system, together with the proliferation of possible hardware and audio configurations, imposes the presence of a flexible, platform-independent configuration system.

In order to face these needs, the design of jMax includes a scripting subsystem, fully integrated into the environment, currently based on a JAVA implementation of TCL (Ousterhout, 1994; Ousterhout, 1997; Stanton, 1998). From a user's point of view, the TCL interpreter is an easy way to access and control the various functionalities of the system; examples include controlling the graphic layer via a set of specialized commands to handle parameters of the graphic objects, such as position or size, or controlling the system's setup parameters via specialized ucs commands (Déchelle & De Cecco, 1995). From the developer's point of view, scripting is an easy way to extend the functionalities of jMax with the creation of new commands and scripts.

Almost every functionality of the current system is scriptable: the user can write scripts to open a document, load a patch into FTS, send messages to specific FTS objects, start and stop DSP execution, and to change parameters of a running patch.

In addition, it must be pointed out that having a scriptable system means having clear APIs and well-defined entry points into the architecture. The presence of a scripting system influenced the jMax design, made it clearer, and provided the development team with a powerful tool for tests and benchmarks.

4 - jMax components

The jMax user environment is historically centered on the patcher editor. In the old system (MAX), every functionality was accessible only through the patcher, and every graphic entity had meaning only in relation to an object in the patch. In jMax, however, the patcher editor is just one of the possible clients of the architecture. In fact, data is no longer handled in the graphic layer: it resides in FTS, while a Java API, the application layer, mirrors the data in the user environment and provides flexible access to it (Déchelle et al., 1998a).

The graphic paradigm of the jMax patcher editor is the natural evolution of the well-known MAX graphic language. As such it offers the user a high degree of compatibility with the old environment, while at the same time offering a completely new set of editing possibilities, together with an improved user interaction. Among other enhancements, the jMax editor's look and feel offers a new graphic appearance, with the introduction of colors to express special situations (i.e., errors) or to clearly separate "user interaction" objects from "language" objects in a patch.

In addition to the patcher editor, the jMax user environment also offers a set of specific editors associated with complex data structures, each with their own editing paradigm. Examples of these are:
- a graphic table editor, which also includes a formula evaluator based on TCL
- a qlist editor, i.e., an editor of "events" internally used by a patcher
- a sequence editor, allowing the representation of "scores" with different visualization options, and which can be used in conjunction with score-following algorithms.

The system offers a programming interface that allows third-party development of specific editors, either based on predefined data types, or on user-defined ones. The creation of external editors is greatly simplified by the presence of a growing library of tools and graphic APIs to be used in user sources. This, together with the existing public API for the development of FTS objects (Déchelle & De Cecco, 1995), and with the planned extensions of the application layer toward a collection of dynamic, extendible collection of modules, forms the basis of a new paradigm for the extension of the system by the developer on different levels of the architecture.

The presence of an abstract layer representing the data in FTS allows the presence of multiple client applications at the same time, each sharing the same data in the server. A first application of this capability is Espresso, a tool for the construction of generic, reusable control panels composed of simple controllers that can be associated to specific objects in a running FTS process. The logic of Espresso is simple: starting from a script description of a panel, it dynamically creates the graphic components, and associates them to the requested parameters in FTS. The application layer takes care of returning the right reference to the desired FTS object, as well as communicating to FTS the values coming from the user interaction with the controllers.

The user can easily create Espresso panels to show only the relevant controllable part of a patcher, while hiding the underlying computational structure. The Espresso prototyping tool is the first effort in the direction of a clearer separation between program and user interaction in a patch. Furthermore, since Espresso is based on JAVA-Beans, it can benefit from the growing base of components created with this mature technology.

5 - jMax user environment
The configuration of data and I/O resources, as well as the setup and appearance of graphical components, is handled by several mechanisms provided by the user environment of jMax. While most parts of the environment are dynamic--dealing with the runtime of jMax--a configuration can be considered static for a particular running real-time algorithm, though multiple setups may be applied sequentially within an application such as a musical piece.

The following issues can be considered as part of an environment setup:
- I/O devices configuration: audio, MIDI, files, network connections, ...
- bindings of dynamically-loaded components: classes, patches, GUI, ...
- general graphic appearance
- setup of windows and panels
- key bindings, menus and palettes
- data initialization and global presets

The configuration can be divided into installation defaults, user preferences and application settings. The installation defaults take into account the resources of a particular machine and provide a complete environment for an inexperienced user. If so desired, a user can specify his own preferences in the defaults, which might be completed or overwritten by settings associated to a particular application stored in the "project environment".

A project can be understood as the generalization of the application term covering single patches and experimental programming stages as well as finalized complex applications, while a session constitutes the top level for the configuration of a concrete jMax application. In the session settings the active components (loaded patches and panels) as well as their initial state and appearance on screen are stored. The session mechanism can be used to "remind" a certain state of an application under development as well as to determine the complete setup or even multiple stages of the changing configuration of an application.

Two basic mechanisms are provided to handle the use of dynamically-loaded components: packages and pools. A package is an entity referenced by a configuration (or another package) as a whole by a require-provide mechanism. While packages contain different kinds of components together with their configuration, a pool is declared for a particular (or multiple) type(s) of data items (class libraries, patches, soundfiles, ...) without the explicit declaration of its contents and is looked up when a component is called by name.

6 - Applications

Typically a jMax application refers to multiple packages providing components related to the user interface and the computation engine of jMax, such as:
- configuration primitives and panels
- classes, templates and patches
- graphical data editors and controllers
- data resources

A number of atomic jMax applications can be identified, supported by object libraries, example patches and tutorials:
- synthesis (additive, granular, formants, FM, wave guides, modal, ...)
- transformation (filtering, pitch shifting, delay effects, dynamics, ...)
- analysis (pitch, amplitude, LPC, partials, STFT, ...)
- data storage (sampling, hard disk recording, sequencing, ...)

Several other components of jMax are centered around score following and score recognition. Since most of the musical pieces written for the Max environment over the years make use of an automatic synchronization of electronic score to live soloist, score following and score recognition are applications of jMax which are vital to the IRCAM repertoire. This traditional score following method based on pitch analysis of the soloist and a subsequent matching with stored score is the topic of ongoing research at IRCAM.

So far there are two larger jMax object libraries, each dedicated to a single application: the "Spatialisateur" (Jot & Warusfel, 1995), used to build multi-channel spatialization and room-effect applications, and a library to create instruments based on additive synthesis (Rovan et al., 1997).

Although the current field of jMax applications owes much to the legacy of Max as an environment for musical applications, the generalization and revision of some aspects of the programming environment will hopefully lead to an extension of this field, targeting a more general prototyping of control and sound processing algorithms as well as the building of complex finalized applications.
7 - Evolutions

Among the planned extensions to the jMax user environment will be the introduction of sets of tools for patch and object development. These "helpers" will include generic inspectors for patches and objects, control and signal probes, breakpoints, and step-by-step execution.

Summary

In this paper we presented the latest generation of Ircam's environment for real-time musical applications. The different components of the system and the scripting language that binds them were detailed. Finally, the application framework was presented, as well as points for future development.

References


