Modalys in jMax: real-time modal-synthesis

Francisco Iovino
Norbert Schnell
Olivier Lartillot

Institut de Recherche et Coordination Acoustique Musique
1, place Stravinsky, 75004 Paris
iovino@ircam.fr, schnell@ircam.fr, Olivier.Lartillot@supelec.fr

Abstract

A modal synthesis package has been developed for jMax. The whole set of bodies and interactions of the Modalys synthesizer have been rewritten. The package allows a new approach to the synthesis techniques known from Modalys providing real-time interaction. It extends the existing set of synthesis classes in jMax with the possibilities of physical modelling. A virtual instrument is built from individual jMax objects representing bodies and interactions linked by named variables. An additional object implements the modal synthesis loop as a single jMax signal processing routine respecting the sample by sample constraint compelled by the physical system.

1 Introduction

Modalys is an environment for the conception and performance of virtual instruments which has been satisfactorily employed in many fields of computer music [1, 2, 3]. The virtual instrument is defined by an assembly of physical bodies, coupled by physical interactions. The instrument is performed by activating the interactions with dynamic parameters that describe a gesture.

The original version of Modalys has been implemented as a set of primitives extending the Scheme programming language to control a synthesis motor written in C++. The Modalys interface provides primitives to instantiate and parametrize bodies and their interactions as well as a rich set of controllers and utilities to describe the performance of an instrument.

Although Modalys provides a solid basis for musical creation by means of a powerful and expressive programming language, it does not satisfy the desire to interactively control virtual instruments in real-time, which has been constantly expressed by the users. In fact, one of the motivations to use physical modelling sound synthesis lies in the direct casual relationship between the performer's gesture and the instantaneous acoustical response of the instrument. Moreover, a real-time implementation would allow a more efficient cycle of editing, performance and refinement.

jMax is a cross-platform graphical programming environment for real-time music applications currently running on SGI workstations and Linux PC [4, 5]. It uses an object oriented approach to create programs (patches) from predefined modules (objects) organized in packages. As IRCAM's most recent real-time system, jMax was the natural target platform for a real-time implementation of Modalys.

A very first prototype of Modalys in real-time was implemented in 1997 by Francisco Iovino and consisted in packaging the original non real-time C++ modal synthesis routines into a library controlled in real-time by Max/FTS [6]. In 1998 Richard Dudas developed a set of optimized modal synthesis external objects for Max/MSP based on the Modalys algorithms, he called NVM [7]. With these objects the user can construct and control a simple virtual instrument consisting of an excitator and a resonator as a Max/MSP patch. In 1998 Olivier Lartillot implemented a set of modal synthesis classes for jMax, which has been a starting point for the work presented in this paper.

2 Porting Modalys to jMax

Since jMax already provides a rich set of facilities that generate dynamic control data, an implementation of Modalys in jMax can concentrate on data structures and modal synthesis issues. However, one should be conscious that this way some interesting features of the Modalys control environment, which can be difficulty reimplemented using the Max language, will be left out.\(^1\)

---

\(^1\) The "scheme controller" is an example of a Modalys controller which cannot easily be ported to jMax. With this controller, the user can map information only available at synthesis runtime into synthesis control decisions.
2.1 Constructing a virtual instrument

Modalys uses an object oriented approach allowing an incremental construction of virtual instruments from predefined modules, which matches very well the graphical programming paradigm of jMax.

In Max, the usual way to define a relationship between two objects is to connect an outlet of one object and an inlet of another object by a graphical patchchord. Since these connections represent an unidirectional flow of messages or audio rate signals, they cannot adequately represent the mutual physical dependencies established by the internal couplings of a virtual instrument.

A more convenient way to express the relationship between bodies and interactions was found in the concept of named variables, recently introduced to jMax. This way, two different representations are used to define the relationships inside a patch: variables, for the construction of a virtual instrument, and patchcords, for its integration with the jMax environment².

A virtual instrument is built from the following elements, represented as jMax objects:

- bodies (physical structures)
- interactions (physical couplings)
- inputs (force injection)
- outputs (vibration pickup)

2.2 Controlling a virtual instrument

jMax integrates control messages and audio rate signals. Different sets of objects handle the two meta data types using two different computation models. For each relevant parameter of the modal synthesis classes, it had to be decided whether it would be implemented as a control message or a signal, whereby efficiency issues as well as the necessary temporal resolution of a parameter had to be considered.

In general, the forces and movements driving the bodies and interactions are represented by audio signals, while the parametrization of the physical bodies and the positioning of their interaction points is implemented as control methods.

A simple example in section 4 will show how a virtual instrument can be controlled using standard jMax primitives.

2.3 Computing a virtual instrument

The signal computation subsystem in jMax allows an object to declare a DSP subroutine, which is computed for a whole vector of samples respecting the constraints given by the signal flow connections to the object. The vector length is typically 64 samples or bigger and should not be less than 32 or 16 for efficiency reasons.

Since the computation cycle of Modalys simultaneously solves the current interaction forces and updates the dynamic state of each body sample by sample, it has to be included in a single jMax signal class, which holds references to all involved objects. The modal synthesis objects introduced in the section 2.1 can be understood as an interface to this synthesis motor, which is implemented as a separate object.

While all calculations depending on a particular body geometry or interaction type are done inside the interface objects, the modal synthesis motor operates only on generic modal representations. Since most of the computation is concentrated in the generic computation in the synthesis motor, any optimization of the involved routines will speed up the execution for all configurations.

3 The jMax modal synthesis package

3.1 Bodies and interactions

A body is implemented as a MODbody object taking it's geometry (like string, plate, etc.) as an argument. The object has a control inlet for each of the geometry dependent physical properties (like length, tension, etc. for the string). An interaction is realized as a MODinteraction object which similarly requires an argument specifying it's type (like strike, bow, etc.).

The link between bodies and interactions is established using named variables. Each MODbody object defines a variable. The MODinteraction object is instantiated with two variables which refer to the two bodies it couples³.

The initial locations of the access points of the interaction in respect to the body's geometry are given as arguments and can be controlled via inlets of the MODinteraction object. Additional inlets refer to the dynamic parameters which depend on the interaction type (e.g. bow velocity, bow pressure and rosin coefficient for the bow interaction).

³ Since the jMax message system keeps track of references to variables, an interaction object is automatically disabled when one of the referenced body objects is deleted.
3.2 Signal inputs and outputs

The MODin~ object allows to apply a force at a certain point of a body, while the body's movement can be obtained using MODout~. Both objects take the reference to a body and the initial location of the access point as arguments. The access location can be changed via an inlet.*

Instead of controlling the amount of force injected to a body's access point, a MODin~ can be configured to determine its speed or position. As well, the MODout~ object can output either speed or position.

3.3 Modal (sub)patchers

The modal synthesis computation loop is implemented inside an additional jMax class, which is called MODpatcher since it defines a context for the other classes. The MODpatcher is the only class of the library declaring a subroutine to jMax's DSP subsystem. It defines a named variable, which is referenced by the MODbody objects in order to inscribe themselves into a modal (sub)patcher.

Multiple MODpatcher objects can be instantiated in a jMax patcher. An interaction can only be defined between two bodies of the same modal patcher.

4 An Example

The figure at the end of this article illustrates how a virtual instrument is constructed as a jMax patcher. The instrument consists of a mass striking a string, which at the same time resonates from a microphone signal.

The MODpatcher object in the bottom defines the variable "mama" as context of the modal synthesis. The string and the mass are represented by two MODbody objects. Both objects refer to the variable "mama" and define each of themselves a variable, "string" and "ball". Since no initialization arguments are given for the physical properties, default values are assumed. The properties can be controlled one by one via the inlets or as a list into the first inlet, as shown for the string.

The object MODinteraction establishes an interaction enabling the ball to strike the string. It references the variables "string" and "ball" and gives the initialization parameters for the access points. These are, for each body, a physical coordinate in respect to the body's dimension, normalized between 0 and 1, and the initial position of the body's access point, relative to a virtual root point in meters. The ball is initially placed one meter away from the string.

The upper left subgraph in the patcher defines a gravitation force applied to the ball using the MODin~ object. The amount of force (in newtons) can be determined via a slider and, because of its negative sign, drives the ball towards the string. The ball will strike the string and repeatedly bounce until the initial energy is dissipated.

The second MODin~ object in the upper right of the patcher uses the string as a resonator, taking the system's audio signal input as a force which excites the string. The access point of the force can be moved with the slider connected to the right inlet of the MODin~ object.

The MODout~ object in the bottom of the patch picks up the vibration of the string (its instantaneous speed) at an access point initialized to three quarters of the strings length, which can be moved by the slider in the middle. The resulting sound is sent to the system's monitor output.

5 Conclusions and Perspectives

The paper presents a real-time implementation of the modal synthesis algorithms of Modalys as a jMax package. The considerations made during the porting are explained starting from a historical overview. A detailed description of the included classes is given.

Future versions of the synthesis package will include further types of bodies and interactions. The real-time performance of the synthesis engine will be improved and the dynamic change of physical properties will be optimized.

A compatibility with existing descriptions of Modalys instruments as Scheme files would be desirable. A collaboration with Richard Polfremant, from the University of Hertsfordshire, UK, will target a multiplatform graphical 3D interface (based on the ideas of the Modalys [8] program) which hopefully will result in an ergonomic and intuitive front edge for the synthesizer.

We expect to start very soon to collaborate with composers interested in exploring the package in order to validate it in different musical contexts.

---

* Even if these inputs and outputs of the virtual instrument could be thought of as inlets and outlets of a body object other then those they can be added one by one dynamically during the construction and have each a controllable parameter.
References


![Modal synthesis example](rebounce.max)

exciting string with rebouncing ball

---

**Figure:** A mass rebouncing on a resonating string.

ICMC Proceedings 1999 – 107 –