MfOM: a Library for Building and Controlling Physical Model Instruments in OpenMusic

Richard Polfreman

Music Department, University of Hertfordshire
email: r.p.polfreman@herts.ac.uk

Abstract

IRCAM’s OpenMusic is a visual-programming environment based on Digitool’s Macintosh Common Lisp and using graphical patch representations. While OpenMusic can be used purely for Object Oriented Lisp programming (using CLOS), a rich set of music related classes and functions are provided for use in the context of algorithmic composition. We present a first version of a new library and project for OpenMusic allowing the graphical construction and control of physical model synthesis using the modal modelling engine of Modalys and Modalys-ER. This allows complete physical model synthesis specification within OpenMusic’s graphical environment while retaining compatibility with previous systems.

1 Introduction

While OpenMusic (Assayag, et al 1999) has been much used in the area of algorithmic music composition, the more recent addition of the OMChroma library (Agon, Stroppa and Assayag 2000) has allowed control of a variety of synthesis engines within the OpenMusic environment.

OMChroma proposes a generic solution to the problem of synthesis control, in particular providing abstractions for handling potentially very large amounts of control data. Previously OMChroma has been used for the control of Csound, Chant (FOF synthesis) and Modalys - with the OM-Modalys library by Mauro Lanza. Typically the synthesis engine exists as a separate application with aspects of the synthesis having to be defined externally to OpenMusic in a format native to the engine, e.g. a Csound .orc file or a Modalys Scheme file. Functions in the library then access these files to extract elements that will be specified from OpenMusic in some way. A patch is then created to generate these values which are then written out to a file or files in the native format and can be executed in order to produce the synthesized sound. In the case of Csound, the execution is automated through Mac OS inter-application communication (AppleEvents), while the Modalys execution is manual. Control of Chant is slightly different in that there is a choice of Chant patches (synthesis configurations) and the synthesis engine is in the form of a shared or dynamic library.

The MfOM library has been developed in order to provide control of Modalys synthesis without requiring any knowledge of the native format (i.e. the Scheme programming system of Modalys) and to allow the definition of the synthesis configuration as well as its control within OpenMusic. In addition to lowering the burden of knowledge on the composer and allowing complete synthesis specification within OpenMusic, there is much potential for algorithmic construction of instruments according to music requirements. Given that control of physical modelling synthesis can be difficult, it is hoped that OpenMusic with the MfOM library will also provide a useful test-bed for the development of intelligent synthesis control systems.

2 Modalys and Modalys-ER

Modalys (Morrison and Adrien 1993) is a physical modelling synthesis system using modal modelling techniques. This allows the user to construct their own instruments by combining simple objects such as masses, strings, plates with interactions such a striking, plucking, bowing and kinetic modifiers such as force and speed. The combinations can be arbitrarily complex, allowing sophisticated simulations to be assembled. Objects and interactions are specified using physical parameters (e.g. length, radius, force, relative distance) and so users are not required to have a detailed knowledge of DSP concepts and techniques. Many parameters can be made to change over synthesis time, in addition to the obvious ones required to create dynamic interactions. These include pickup positions/amplitudes and locations and radii of holes in a tube. Unusual behaviors can be defined such as hybrid objects that morph from one object into another dynamically. Interesting applications of Modalys include the processing of input sounds by injecting them into resonant objects/systems. While the modal approach has a number of advantages for users in terms of familiar acoustic analogy, flexibility of instrument design and
use of physical terminology, the issue of control remains difficult. Which precise values of force, speed, or breath pressure (for example) should be used in a given context is not obvious, and this is exacerbated by the non-real time nature of the synthesis, making exploration of alternative parameter values a drawn out process.

While Modalys uses the Scheme (a Lisp dialect) language as the basis for synthesis specification, Modalys-ER (Polfreman 1999) provides a graphical user-interface to the modal synthesis engine. In Modalys-ER a synthesis is divided, along familiar lines, into an instrument and a score. The instrument defines a construction – an interconnected set of objects and their interactions using a graphical patch notation – and a set of techniques. The techniques define mappings of a standardized set of score parameters (classified into pitch, excitation and timbre types) onto dynamic controls within the instrument, with one-to-many mappings possible. A score in Modalys-ER is simply a set of envelopes of different types for specifying parameter changes over time.

3 The MfOM Library

MfOM (Modalys-ER for OpenMusic) is a new experimental library for OpenMusic that ports elements of Modalys-ER into OpenMusic and uses control systems in keeping with the OMChroma library. While developed independently of the OMModalys library, there should be scope for using some elements of that library within MfOM or developing components in common.

Similar to Modalys-ER, a synthesis specification with MfOM begins with an instrument definition. This role is fulfilled by the ModInst class. Opening an instance of this class (or more usually a factory for this class – a patch object that generates instances) provides an editor window similar to an instrument editor in Modalys-ER, but with only a construction area since we wish to use the more sophisticated control mappings possible in OpenMusic.

Figure 1. A simple MfOM struck plate instrument. From top left, a force affecting a mass that will strike a circular plate attached to pickups for sound capture.

Figure 1 shows a simple instrument construction using a mass to strike a tuned circular plate. The mass (actually a two-mass object) is connected to the plate via a strike connection that defines the initial object separation and the locations of the strike points on the two objects. A force connection is made to the mass so that it can be moved to strike the plate. Two pickup connections are made to the plate at different points in order to take the vibrations of the plate into a stereo sound file. Each of the items in the instrument can be opened in order to specify its values. A modal object can be tested by running a short synthesis from inside its editor, giving an idea of its timbre and mechanical characteristics. This test synthesis simply strikes the object at a user-defined point and with variable force.

In some circumstances it is not necessary to have any external control of an instrument. For example, if we just wanted to drop a mass onto a plate, we could set the mass force value to a constant, negative number of Newtons and ensure a reasonable positive initial separation of the mass and plate. In such a case we can directly connect the ModInst instance or factory to the function simplemodsynth (figure 2). This function will construct the instrument as specified and run a given amount of synthesis time (at a given sample rate) using the integrated synthesis engine in the form of a shared library. Evaluating a sound factory connected to the output of the function will cause the synthesis to run and the resulting output file will be loaded into OpenMusic.

Figure 2. A simple synthesis patch with no external control. Numeric inputs (left to right) are duration, number of channels and sample rate.

Usually we will want to use dynamic control over the synthesis. In order to do this we first specify one or more parameters in the instrument construction to be dynamic. We then connect the ModInst to the function get-modalyser-instrument. Evaluating this function defines a new subclass of MO-evt, which in turn is a subclass of the OMChroma defined SynthesisEvt class. This process is a parallel to the use of the OMChroma get-instrument function, which parses a Csound orchestra file in order to define a new Csound event class. The new event class can be thought of as containing a copy of the instrument as defined when the function was evaluated and providing a set of keyword inputs, one for each dynamic control in that instrument. This does mean that if you edit the instrument you must redefine its event class by re-evaluating get-modalyser-instrument.
Opening an event factory or instance allows you to edit a breakpoint function for each dynamic parameter in the instrument. However, it is more useful to generate control information using a variety of OpenMusic objects and functions and pass this information on to the event factory for synthesis. To actually run a synthesis using an event it is connected to the generic function `Synthesize`. The behavior of this function is dependent upon the synthesis engine selected in its first input; in this case we select `mfm`. The event is connected to the second input and other inputs are used to specify the duration, sample rate, number of audio channels and output file name. Evaluating this function invokes the synthesis and as before a sound factory can be used to import the sound produced back into OpenMusic.

Figure 3: A bowed string instrument. The top mass is used as a bow, the bottom mass as a finger with a fingerboard connection.

Figure 4: High-level control of bowed string pitched using a fingerboard. The top envelope in the event is the finger position, the bottom the bow position.

Figure 5. ‘Parametization’. Pitch and material of the plate are parametized, pitches identified from a MIDI file and material randomly selected.

translates a chord-seq object into controls for these dynamics. A set of sub-patches do the work of calculating a set of bow movements on and off the string for each note, and also calculating the finger position on the string in order to achieve a set of well tempered pitches.

4 Handling Complexity

There are two main ways in which construction of complex instruments is supported in MfOM. The first of these is through the use of combi objects within an instrument. These objects are not part of Modalys, but are also used in Modalys-ER. A combi is simply a type of sub-patch object that allows arbitrary nesting of groups of objects and connections within an instrument. Connections can be made from higher levels down into lower levels within a construction. This is a familiar concept that has been used in many systems before (e.g. Max and OpenMusic itself) and while a useful aid, particularly for organization and layout, it does not always simplify the task of instrument design greatly.

The second approach begins to take advantage of the OpenMusic environment in terms of algorithmic generation of values. In what we have termed ‘instrument parametization’, the get-modalys-instrument function can be passed lists of parameter names and associated values. When evaluated, the function then makes a set of duplicates of the instrument within the event class, each duplicate using a different value from the parameter list. A typical application of this is to build a polyphonic instrument from a monophonic template.
Figure 5 shows an OpenMusic patch that reads in a short MIDI file, extracts the first track and displays this as a chord-seq. The number and values of discrete pitches are determined and these used to parametrize a struck plate instrument in order to set the plate pitch in each duplicate. In addition, a material for each plate is randomly selected from the set of default materials that accompany the library. The resulting event class then has (in this case) five inputs, one for controlling the hammer force in each duplicate. The sub-patch polyhitevent generates a control envelope for each hammer force from the sequence of events in the MIDI file. In addition to lowering the workload on the user by generating the duplicates for each pitch, this synthesis is relatively efficient in that only the duplicates necessary to perform the given MIDI file are created. Clearly if another MIDI file is used, it is unlikely to involve an identical set of pitches and so get-modalyser-instrument must be re-evaluated. In order to simplify organization in this patch we order the found pitches from low to high, so that the event’s inputs will be ordered similarly.

5 Conclusion and Further Work

The MfOM library, while still experimental, we believe provides a valuable extension to the synthesis capabilities of OpenMusic. While the OM-Modalys library provides a useful framework for Modalys synthesis, MfOM provides a more integrated solution and knowledge of the Modalys Scheme language is not required. Since the MfOM library does not allow complete access to all Modalys functionality (although most useful areas are covered) we see MfOM as complimentary to OM-Modalys, with expert Modalys users perhaps finding OM-Modalys more useful, and Modalys-ER or novice Modalys users preferring MfOM. In any case there is scope for development of utilities that could be used by both libraries.

The use of parametization is a valuable algorithmic tool and we are looking to extend this approach, with for example, partial duplication that could be used to generate a set of tuned holes in a tube or set of hammers striking a single object. Instruments can be exported from an upcoming release of Modalys-ER in a format that can be imported into MfOM (construction part only), allowing re-use of existing Modalys-ER instruments within MfOM. Also, a built in debug mode allows the user to produce a Scheme code representation of a synthesis which can be run in Modalys if desired. While aimed at aiding the debugging of a synthesis, this option may have other useful applications for Modalys users.

One of the key gains achieved by MfOM is that high-level representations such as MIDI files or Common Music Notation can be mapped effectively onto control of instruments, rather than Modalys-ER’s more limited use of envelopes. However, the design of a mapping system usually has to be at best tailored, or at worst built from scratch, for each individual instrument. Some useful mappings have been built as part of this library and also the OM-Modalys library. We hope to develop further a set of mapping systems for a variety of interaction controls and also explore the potential for intelligent systems that given an instrument and some form of CMN score will attempt to derive an appropriate mapping automatically – a kind of ‘virtual musician’. Given the various AI applications of Lisp and the ability to build arbitrary instruments and scores with MfOM and OpenMusic, we believe we have a suitable platform for such exploration.

6 Acknowledgments

Thanks to Carlos Agon and Marco Stroppa for their assistance in the early development stages of MfOM.

References