
Peter Lundén
Dept. of Speech Communication and Musical Acoustics
Royal Institute of Technology (KTH)
Box 700 14
S-100 44 Stockholm
Sweden
email: lisd@akcor.kth.se

Abstract. CPL is a computer programming environment aiming at composers and researchers in the field of electroacoustic music (Lundén 89). It is an interactive, object-oriented system implemented in Lisp (LeLisp 15.2) for both real-time and non-real-time applications such as the controlling of MIDI-synthesizers, Digital Signal Processing and Algorithmic Composition. The concept of Sound-Models is used in CPL to represent knowledge about sounds and sonic-structures. This paper will mainly discuss the use of Sound-Models to control a software Formant-Wave-Function synthesizer (FOF) (Roth 85), a combination that have proven itself to be a rich and powerful tool for both producing and controlling sounds. Dynamic objects can be used to build complex controlling structures to control Sound-Model objects. This kind of objects are well suited to handle gestural control of arbitrary sounds and sonic-structures.

Introduction

A Sound-Model is a representation of the knowledge needed by the system to produce a particular type of sound. The knowledge is distributed in a hierarchy of classes. An instance of a Sound-Model class can be thought of as an "instrument". It describes a sound with a set of parameters which are appropriate for that type of sound, not with arbitrary parameters enforced by the synthesis method. i.e. a Sound-Models that behaves like a drum could have parameters like drumstick hardness, point of impact and hitting force.

The Structure of Sound-Models

Sound-Models are based on the object oriented system in CPL and are implemented as a hierarchy of classes. In Sound-Models more general aspects of sounds are taken care of in the lower levels of the hierarchy while more specific aspects are handled in the top levels. The most fundamental class is the TimeQuantaTion class. It takes care of the scheduling of the Sound-Model instances. Next level in the hierarchy is the Process class which handles all time and timing information. This class is the engine of a Sound-Model and is responsible for sending the appropriate messages at the right time.

At the next level we find the Sound class which is an abstract class. Other basic characteristics than time are handled by this class such as fundamental-frequency and amplitude. This means that all subclasses of the Sound class have a start-time, a duration, a frequency and an amplitude parameter. This guarantees the uniformity at the very basic level and makes it easy to use sound-models in higher level abstractions.

The next class in the hierarchy is the Foc class. It takes care of all communication with the synthesizer-module. Interfaces to other synthesizer-modules could be implemented at this level. The Foc class adds knowledge about the FOF-synthesis-method and parameters.
like raise-time of the local envelop. cut-off level and bandwidth are added to the class. Instances of this class can not reproduce any sounds of particular interest, only single impulses. To get more interesting results subclasses must be added to the \texttt{F0} class. This is discussed later in this paper.

The \texttt{eachTime} cycle

When an instance of a Sound-Model class becomes active messages has to be sent to it to initialize it and to \texttt{keep} it alive. The basic mechanism to handle this is implemented in the \texttt{Process} class. When an instance becomes active and before any other message, the \texttt{firstTime} message is sent to the instance. The purpose of this message is to initialize the instance and perform the necessary setups. The next thing that happens is the invocation of a single \texttt{eachTime} cycle. The cycle is divided into three phases the \texttt{preEachTime}, the \texttt{eachTime} and the \texttt{postEachTime} phase. At each phase the corresponding message is sent to the active instance. Before each cycle begins the calculation of some variables with dynamic scope are performed: the global \texttt{time} variable which keeps the current global time in the system, the local \texttt{time} variable which contains the time since the current Sound-Model instance started and the relative \texttt{time} variable which is the local \texttt{time} divided by the duration of the instance and always has a value between 0 and 1.

In the \texttt{preEachTime} phase the input parameters are evaluated and the results are stored in reserved instance slots. This provides a partial-lazy-evaluation mechanism which guarantees that the input parameters are not evaluated more than once during each cycle. The main calculations are done in the \texttt{eachTime} phase. If the instance has to gain control in the future, it has to send a \texttt{delay} message to itself. This message schedules a new cycle but only if the end-time of the instance is not reached before the beginning of that cycle. Each cycle ends with the \texttt{postEachTime} phase in which the data is collected and sent to the synthesizer-module. When the end-time of an instance is reached the \texttt{lastTime} message is sent to it and takes care of the necessary housekeeping.

Dynamic Control of Sound-Models.

In CPL it is easy to control the parameters of a Sound-Model instance dynamically. The evaluation mechanism in the \texttt{eachTime} cycle and the redefinition of the \texttt{eval} function facilitates this. In CPL the \texttt{eval} function is redefined such as: if the argument is a CPL object then an \texttt{eval} message is sent to that object otherwise the old \texttt{eval} function is called. This means that a class can define the behaviour of the evaluation of its instances. The default behaviour of an \texttt{eval} message is to return the receiver of the message. The concept of evaluable objects blurs the difference between object and closure. An object can act like a closure where the slots behaves like the closure variables. The most important consequence of evaluable objects is that not only numerical values, but also objects which has a dynamic behaviour can be used as parameters in instances of Sound-Model classes.

In CPL there is a particular class structure, designed with the dynamic control of Sound-Models in mind. The root of this hierarchy is the \texttt{Controller} class which is an abstract class. Every \texttt{Controller} is a mapping from a reference to the value. The \texttt{Controller} class implements a slot, called reference, which is keeping the independent variable of the mapping. It would be far to discuss all the subclasses to the \texttt{Controller} class, therefore only a few examples are discussed to show the most important use of dynamic objects.

The most commonly used classes are the \texttt{Log} class and the \texttt{Lin} class. They are subclasses of \texttt{Lin} which is a subclass of \texttt{Controller}. Instances of this classes can describe...
time dependent values as a piece-wise linear or logarithmic function which is defined by pairs of time and value. To make an instance of the Log class or the Lin class time dependent one of the globalTime, localTime or relativeTime variables is given as the reference.

Another interesting branch of the Controller tree is the Random class and its descendents. Instances of this classes acts like random-number-generators and are used to model stochastic processes.

The Noise class

Noise is a subclass of Fof and it is a model of noisebands (not to wide). The noiseband is achieved by adding identical fof-impulses randomly spread in time. The definition of a Noise object named noise-1 is shown in fig 1. The center-frequency of noise-1 is assigned to a BrownCut object which is a model of a one dimensional Brownian motion in a limited space. In the beginning of the noise-1 Hz's frequency will fluctuate between 880 and 1760 Hz. After 3.2 seconds the interval start to grow until the frequency finally fluctuates between 880 and 3520 Hz. The amplitude of noise-1 is controlled by a Log object and grows from -30 dB to 0 dB.

```
(defObject (Noise) noise-1
  (name 'noise-1')
  (duration 8.0)
  (freq
    (defObject (BrownCut) {})
    (sigma 80.0)
    (min 880.0)
    (max
      #x(Log relativeTime
        0.4 1760.0
        1.0 3520.0))
  )
  (ampl
    #x(Log relativeTime
      0.0 (db -30)
      0.4 (db -20)
      0.8 (db 0)))
)
```

Fig 1. Dynamic control of an instance of the Noise class.

The Spectrum class

The Spectrum class implements a type of granular synthesis with Fof impulses used as grains. The behaviour of Spectrum objects is now described. The frequency domain is quantified into a ladder structure, where the first step is defined by the freq parameter and the distance between the steps are ((df -1) * freq) and the upper limit is defined by fmax. The grains are randomly distributed both in time and frequency domain constrained by the ladder structure. The amplitude of each grain is frequency-dependent and is defined by the ampl and the aspSlope parameters. The durSlope is similar but it defines, together with grainDur, the duration of a single grain. The density parameter determines the number of simultaneous grains.

The definition of a Spectrum object named spec-1 is shown in fig 2. In spec-1 the fmax parameter will change from 100 to 600 Hz and the density from 200 to 60 during the duration of the sound.
Related work
The FORMES system has an interface to a FOF-synthesizer [Coite et al. 87], but this interface is connected through the Chant program. This makes it very difficult to control the sounds in detail like in CPL.

Conclusion and Future Plans
Sound-models is a powerful tool for representing knowledge about sound synthesis in a CAC system. However sounds has to be classified to fit in to this concept and there exists no classification system known to the author which is consistent enough the serve this purpose. Such classification system could be developed.

The dynamic objects can be used to build complex controlling structures. This kind of objects are well suited to handle gestural control of arbitrary sound-structures.

CPL is completely reimplemented in Common LISP. The object system in CPL is replaced by CLOS in the new version. This will add a lot of power to the Sound-Model concept and to CPL in general. The new version is called Common CPL (CCPL).

References


ICMC 105