A new CHANT Synthesizer in C and Its Control Environment in *Patchwork*

Jean-Baptiste Barrière, Francisco Iovino, Mikael Laurson
IRCAM, 31 rue Saint Merri, 75004-Paris, France
Tel : 33-1-42 77 12 33 Fax : 33-1-42 77 92 64
Net address : barriere@nadia.ircam.fr

Abstract

A new CHANT synthesizer has been written in C in order to be transportable onto a variety of machines. Unix and Macintosh implementations are available at the present stage. The design of the new synthesizer allows one to combine different kinds of objects such as those handled by the original CHANT program: FOFs (formant wave functions) banks, filter banks, white noise generators, and soundfiles as inputs to the filter banks. In the new implementation it is possible to have several of these objects instantiated and patched arbitrarily together at a given time, also permitting polyphonic voices. For the control of the synthesis parameters, a simple interfacing with the following environments has been provided : Formes (under Unix only), Elk-Scheme (under Unix and on the Macintosh), and Common Lisp (on the Macintosh).

Applications have also been designed to control this synthesizer in the *Patchwork* compositional environment (in its latest Common Lisp version on the Macintosh), allowing the generation and manipulation of complex data, e.g. timbral interpolations with phonems or models of resonances. These new *Patchwork* graphical modules designed to control CHANT, permit one to access data bases of formants values and/or trajectories, and to manipulate data by rules (e.g. algorithms describing correlations between the various parameters), that either have been either transposed from the CHANT and FORMES original libraries, or have been developped with these new tools.

The resulting environment is a unique workbench for control of synthesis and processing. The system will be demonstrated on a Macintosh. Sound examples and extracts of musical works now in progress will be played during the talk.

1) Introduction : aims and design concerns

A new CHANT [Rodet 84b] software synthesizer was written in the C language and has been running since the beginning of the year on Unix machines (DecStations) at IRCAM. This new version was first meant to replace the array-processor FPS-100 version (disconnected this spring), which was used for musical productions at IRCAM since 1983 as a peripheral of the Vax-780 running FORMES [Rodet 84c] to control it. The original idea was to make a portable version running on any C platform, connected: on the one hand to FORMES, in order to preserve compatibility with older libraries and applications, on the other hand to various control and/or compositionnal environments already available or yet to come (mainly *Patchwork* , Max, etc.). Considering the amount of control platforms available today, it seemed indeed interesting to develop a synthesis engine rather that a complete program, so as to let users choose their own way to control it. The decision to keep at the level of a synthesis engine rather than a complete program was also directed by the will to keep it compact and easily transportable, avoiding in this way problems such as having to choose a graphical standard.

Beyond the functionnalities of the older version which allowed a mixture of fofs (formant wave functions ; see [Rodet 84b]) and filters, main design improvements for the synthesizer involved : 'real-time ready' design (for further compatibility with the new IRCAM Musical Workstation), polyphony, and therefore patching capabilities between the different voices as well as ressources available in the program (i.e. fofs synthesis and filtering of the previous and/or an external sound source and/or a noise source). Another concern, beyond Unix transportability, was to make CHANT available on the Macintosh, in order to satisfy the demands of many composers working at home, rather than (only) in institutions.

The primary aim of keeping FORMES with a CHANT synthesis engine was successfully achieved : continuity has been preserved for the ten year old community of IRCAM users. The other aims have also been realized : the new synthesizer achieves polyphony and patching inside and between voices, and it runs now on the Macintosh using MPW and it can be controlled with either Common Lisp and/or *Patchwork*, or Elk-Scheme.

Originaly, the basic interfacing was written and tested in the Elk-Scheme interpreter running in the Unix environment of IRCAM. On the Macintosh, Common Lisp was chosen as the supporting language for the synthesis control at the low level. This decision was taken in order to take full advantage of direct communication with the *Patchwork* compositional environment, which offers a wide range of algorithmic and graphical control tools : libraries of compositionnal functions, music notation editors, breakpoint functions editors, etc.

2) Objects of the synthesizer and their low level control in Common Lisp

Objects of the of the new CHANT synthesizer are entities which generate or transform a signal. All of the objects must be created before being used and told when and for how long to trigger themselves. All the objects

have some parameters which describe their behavior. In this version the evolution in time of each parameter must be described as a break point function (BPF) i.e. a list of time-value pairs.
- **(create-object)**
- **(init-object object** start-time end-time)
- **(set-object-par** object par '((t0 v0) (t1 v1) ... (tn vn)))

2.1) FOF bank
This object consists of a set of formants coupled to the impulse generator (described as fundamental frequency control, in fact the rate of excitation of the fof) which excitates the formant configuration. Each formant consists of 7 parameters : tex (excitation time of the fof), debatt (beginning time of attenuation of the fof), atten (duration of attenuation of the fof), amp (amplitude of the fof, freq (central frequency of the formant), bw (bandwith of the formant), phase (initial phase of the fof). For each formant an amplitude description must be provided for each channel.
- **(create-fofbank** n) allocates a fof bank of n formants maximun. ; returns a pointer to the new allocated object.
- **(init-fofbank** fofbank start-time end-time init-nfor fofsynth). The fofbank is active from start-time up to end-time. The initial number of formants is specified by init-nfor. fofsynth tells which object is actually synthesizing the fofs triggered by the configuration .
- **(set-nform** fof-bank n) assigns n to the number of current formants in the configuration.
- **(set-fofbank-freq** fofbank '((t0 v0) (t1 v1) ... (tn vn))) assigns the specified BPF to the impulse generator's fundamental frequency of the fof bank.
- **(set-fofbank-par** fofbank i j '((t0 v0) (t1 v1) ... (tn vn))) assigns the specified BPF to the parameter j (from 0 to 6 : tex, debat, atten, amp, freq, bw, and phase)
- **(set-fofbank-chan** fofbank i j '((t0 v0) (t1 v1) ... (tn vn))) assigns the BPF to the amplitude descriptor of channel j of formant i.

2.2) FOF synthesizer
The fof bank describes the evolution in time of the parameters of the fof configuration and the impulse generator. For the actual synthesis of the samples, a fof synthesizer must be associated to the fofbank ; the same fof synthesizer can be connected to several fof banks. To patch a fof synthesizer into a filter, one must define an amplitude control for the output of the fof synthesizer. This amplitude is a scaler on the non-multichannel output of the fof banks connected to the fof synthesizer.
- **(create-fofsynth)** allocates a fof synth ; returns a pointer to the new allocated object.
- **(init-fofsynth** fofsynth start-time end-time) assigns fofsynth to start to be active at start-time up to end-time.
- **(set-fofsynth-amp** fofsynth '((t0 v0) (t1 v1) ... (tn vn))) assigns the specified BPF to the amplitude control of the fof synthesizer, i.e, the sum of the non-multichannel output of al the connected fof banks.

2.3) Filter bank
This object consists of a set of second order filters. Each filter consists of 3 parameters : gain (scaler on the filter's input), bw (bandwith of the resonator), and freq (central frequency of the resonator). In addition, for each filter an amplitude description can be provided for each channel.
- **(create-filbank** n) allocates a filter bank of n filters ; returns a pointer to the new allocated object.
- **(init-filbank** filbank start-time end-time init-nfil), same as for the fof bank object.
- **(set-nfil** filbank n) assigns n to the number of current filters in the configuration.
- **(set-filbank-par** filbank i j '((t0 v0) (t1 v1) ... (tn vn))) assigns the specified BPF to the parameter j (0 to 3 : gain, bw, freq)
- **(set-filbank-chan** filbank i j '((t0 v0) (t1 v1) ... (tn vn))) assigns the BPF to the amplitude of channel j of filter i.
-**(conn-fofsynth-filbank** fofsynth filbank) assigns filter bank to filter the fof synthesizer's output.
- **(conn-souf-fil** souffle fil) assigns filter bank to filter the white noise generator's output.

2.4) White noise generator (souffle)
Generates a random signal which can be shaped by an amplitude control.
- **(create-souffle)** allocates a 'souffle' ; returns a pointer to the new allocated object.
- **(init-souffle** souffle start-time end-time)
- **(set-souffle-amp** synth '((t0 v0) (t1 v1) ... (tn vn)))

2.5) External sound source (external-source)
Generates a random signal which can be shaped by an amplitude control.
- **(create-external-source)** allocates a external source ; returns a pointer to the new allocated object.
- **(init-external-source** external-source start-time end-time)
- **(set-external-source-amp** synth '((t0 v0) (t1 v1) ... (tn vn)))

2.6) General purpose functions
Besides initializations and file handling that will not be described here, the user can control the number of channels to which the sound will be synthesized, and actually start the synthesis and playing.
- **(synthesis** start-time end-time) calculates the output signal from start-time up to end-time and writes it to the port file.

- (**play-rsrc-file** file-name nsamples srate) generates a snd resource and plays it with the sound driver.

2.7 A simple example

Here is the Common Lisp code to generate a one second sound at 110hz with one formant of central frequency gliding from 500hz to 1000hz.

```
(init-all-tables)
(init-gen-lists)
(set-numchannels 1)
(set-samrate 44100.0)
(setq fofsynth (create-fofsynth 512))
(init-fofsynth fofsynth 0 44100)
(setq fofbank (create-fofbank 1))
(init-fofbank fofbank 0 16000 1 fofsynth)
(set-fofbank-freq fofbank '((0.0 110.0) (1.0 110.0)))        ; set the 1st formant frequency breakpoint function to 110 Hz
; 1st formant paramaters  (1st element of lists is time, 2nd is value)
(set-fofbank-par  fofbank 0 0 '((0.0 0.002) (1.0 0.002)))    ; set atten
(set-fofbank-par  fofbank 0 1 '((0.0 0.01) (1.0 0.01)))      ; set debatt
(set-fofbank-par  fofbank 0 2 '((0.0 0.002) (1.0 0.002)))    ; set atten
(set-fofbank-par  fofbank 0 3 '((0.0 1.0) (1.0 1.0)))        ; set amplitude
(set-fofbank-par  fofbank 0 4 '((0.0 500.0) (1.0 1000.0)))   ; set central frequency
(set-fofbank-par  fofbank 0 5 '((0.0 66.0) (1.0 66.0)))      ; set bandwidth
(set-fofbank-par  fofbank 0 6 '((0.0 0.0) (1.0 0.0)))        ; set phase
(set-fofbank-chan fofbank 0 0 '((0.0 1.0) (1.0 1.0)))        ; set channel amplitude
(open-port "sound-dir;foo.bin")                              ; open soundfile
(conn-fofsynth-port fofsynth)
(synthesis 0 44100)
(close-port)                                                 ; close soundfile
(convsf "sound-dir;foo.bin" 'aiff)                           ; convert soundfile to aiff (on Macintosh)
(play-rsrc-file "sound-dir;foo.bin" 44100 441000.0)          ; play soundfile
```

3) Control of the synthesizer by *Patchwork*

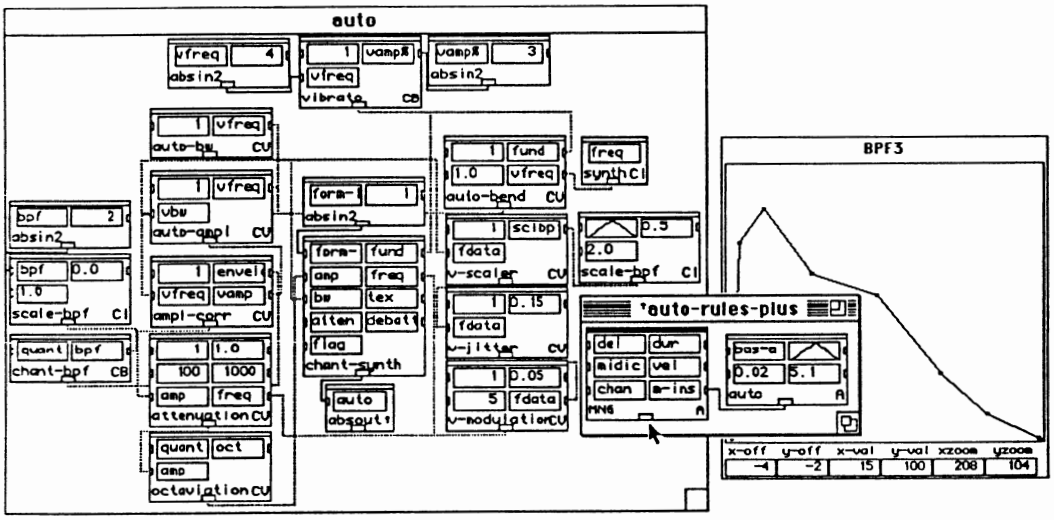
Patchwork is a graphical compositionnal environment [Laurson 89] which inherited the experience developped at IRCAM around computer aid to composition through the FORMES, Preform [Boynton 86], and Esquisse [Baisnée 88] Projects. Since the last paper at ICMC 90, *Patchwork* has been ported to Common Lisp, and is running and being developped now only on Macintosh Allegro Common Lisp 2.0 (after LeLisp), taking therefore full advantage of CLOS (Common Lisp Object System). All references to the Preform object system have vanished in favor of using CLOS, so as to allow smaller specific code and easier portability.

Control of the CHANT synthesizer in *Patchwork* is achieved in a simple way : each fof or filter synthesizer becomes a module or pseudo-instrument. These pseudo-instruments can be connected together to make more complex ones than can be abstracted, i.e. become a higher level module. Inputs of these modules can be controlled by patches (previously existing or created on purpose) and/or various kinds of editors (breakpoint functions editors, chords and rhythm boxes, traditionnal music notation editors, etc.) describing musical parameters. Special uses of the fofs or filters banks to achieve additive synthesis or models of resonance synthesis [Barrière 85] have also been set as modules, thus completing the resources palette.

Most of the rules (i.e. correlations algorithms between parameters) from the older SAIL and FORTRAN versions, as well as FORMES code, have been reimplemented : spectral slope enrichment, octaviation, periodic or aperiodic perturbations on formant parameters, envelopes, etc.. Data bases have been also ported : vowels and formants trajectories, instrumental databases, models of resonance, etc. The rules, described as Common Lisp functions, can become themselves modules, and therefore patched according to the needs. An important feature of the rules is that each of them has its own quantum, i.e. rate of calculation ; the system allows indeed any number of calculation loops, according to the specific computational needs of a particular rule.

The following patch example shows a pseudo-instrument built around a fof-synthesizer to which are added rules for automatic calculation of amplitudes and bandwidths according to the frequencies of formants, as well as rules for spectal enrichment according to the fundamental, jitter, scaler, and LFO on formants frequencies, and rule for octaviation (control of the formants amplitudes one period over two) [Rodet 84b]. The patch consisting of all these modules is then abstracted in a higher level module (here the 'auto' window) than can be instantiated automatically a number of times, depending on the score, and edited in a music notation editor module (here 'MN6' in the 'auto-rules-plus' window). Values for control parameters than have been selected as inputs to the abstracted module, can then be calculated as a network of modules and through editors (like the breakpoint function editor in the 'BPF3' window). For instance, the music notation editor module, when clicked on, opens as a traditionnal music notation editor with extended features : notes/events can be edited in all dimensions ; to

click on them opens their patch/instrument definition, and allows to change its arrangement or connections of modules, and/or to specify the specific values of their inputs. In this way, *Patchwork* achieves an optimal dynamic relation between the definition of a possibly complex instrument model and its more or less modified instantiations in a specific score.



4) Conclusion

The new CHANT synthesizer is now more complete and versatile than previous versions. It is also more easily accessible : various implementations and controls can be easily developped from these foundations.

Control of CHANT with *Patchwork* is progressing rapidly for musical production and research purposes [see e.g. Lerdahl 91]. *Patchwork* itself on the Macintosh, is becoming an integrated compositionnal environment, including a variety of applications : the Esquisse compositionnal functions (for manipulations on pitch, durations, timbral parameters, etc.), the control of Csound, as well as a growing variety of editors and functionnalities. Control of the Mosaïc physical modelling package [Morrison 91], as well as the new SVP analysis/resynthesis/processing package [Depalle 91], are for instance, under plan.

The new CHANT synthesizer is already available on the Next machine with Common Lisp and Elk-Scheme interfaces, and transport of *Patchwork* will be made during next year. Connection with the new IRCAM Musical Workstation should allow for instance real-time control and synthesis of CHANT.

References :

[Baisnée 88] Baisnée (P.F.), Barrière (J.B.), Dalbavie (M.A.), Duthen (J.), Lindberg (M.), Potard (Y.), Saariaho (K.) — "Esquisse: a Compositional Environment" (*Proceedings of 1988 International Computer Music Conference, Köln*, Berkeley, Computer Music Association, 1988.)

[Barrière 85] Barrière (J.B.), Potard (P.), Baisnée (P.F.) — "Models of Continuity between Synthesis and Processing for the Elaboration and Control of Timbre Structures" (*Proceedings of the 1985 International Computer Music Conference, Vancouver*, Berkeley, Computer Music Association, pp.193-198, 1985).

[Boynton 86] Boynton (L.), Duthen (J.), Potard (Y.), Rodet (X.) — "Adding a Graphical Interface to FORMES" (*Proceedings of the 1986 International Computer Music Conference, La Haye*, Berkeley, Computer Music Association,1986).

[Depalle 91] Depalle (P.), Poirot (G.) — "SVP : a Modular System for Analysis, Processing and Synthesis of Sound Signals" (*Proceedings of the 1991 International Computer Music Conference, Montreal*, Berkeley, 1991)

[Laurson 89] Laurson (M.), Duthen (J.) — "*Patchwork* : a Graphic Language in Preform" (*Proceedings of the 1989 International Computer Music Conference, Columbus*, Berkeley, 1989)

[Lerdahl 91] Lerdah l (F.), Chabot (X.) — "A Theory of Poetry as Music and its Exploration through a Computer Aid to Composition" (*Proceedings of the 1991 International Computer Music Conference, Montreal*, Berkeley, 1991)

[Morrison 91] Morrison (J.) — "Mosaïc" (*Proceedings of the 1991 International Computer Music Conference, Montreal*, Berkeley, 1991)

[Rodet 84a] Rodet (X.) — "Time Domain Formant-Wave-Function Synthesis" (Cambridge, Massachusets, *Computer Music Journal*, Vol 8, n°3, 1984).

[Rodet 84b] Rodet (X.), Potard (Y.), Barrière (J.B.) — "The CHANT Project: from synthesis of the Singing Voice to Synthesis in general" (Cambridge, Massachusets, *Computer Music Journal*, Vol 8, n°3, 1984).

[Rodet 84c] Rodet (X.), Cointe (P.) — "Formes: Composition and Scheduling of Processes" (Cambridge, Massachusets, *Computer Music Journal*, Vol 8, n°3, 1984).