A new CHANT Synthesizer in C
and its Control Environment in Patchwork

Jean-Baptiste Barrière, Francisco Iorio, Mikael Lauson
IRCAM, 31 rue Saint-Merré, 75004-Paris, France
Tel: 33-1-42.77.12.33 Fax: 33-1-42.77.92.64
Net address: barrier@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 present stage. The design of the new synthesizer allows one to combine different kinds of objects such as gate handled by the original CHANT program: PFGs (formant wave functions) banks, filter banks, white noise generators, and oscillators 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 synthesizer parameters, a simple interfacing with the following environments has been provided: Formas (under Unix only), Elk-Schets (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 parameters or models of resources. Some new Patchwork graphical modules designed to control CHANT, permit one to access data bases of formants values and trajectories, and to manipulate data by rules (e.g. algorithms describing corrections between the various parameters), that either have been either transported from the CHANT and FORMES original libraries, or have been developed with these new tools.

The resulting environment is a unique workbench for control of synthesis and processing. The system will be demonstrated on 1 Macintosh. Sound examples and extracts of musical works now in progress will be played during the talk.

1) Introduction: aims and design concepts
A new CHANT (Roden 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 away-proposer FPS-100 version (disconnected this writing), which was used for musical productions at IRCAM since 1983 as a peripheral of the Vax-780 running FORMES (Roden 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 compositional environments already available or yet to come (namely 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 functionalities of the older version which allowed a mixture of foFs (formant wave functions; see 'Roden 84b') and filters, main design improvements of the synthesizer involved: a 'plug-and-play' design (for further compatibility with the new IRCAM Musical Workstation), polyphony, and therefore patching capabilities between the different voices as well as resources available in the program (i.e. forms 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 Mid and a can be controlled with either Common Lisp and/or Patchwork, or Elk-Schets. Originally, the basic interfacing was written and tested in the Elk-Schets 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 compositional functions, music notation editors, breakpoint functions editors, etc.

2) Objects of the synthesizer and their lower level control in Common Lisp
Objects of the new CHANT synthesizer are entries which generate or transform a signal. All of the objects must be created before being used and sold when and for how long to trigger themselves. All the objects

ICMC 11
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 '(0 0 0) (1 1) ... (n n))

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 formant) which excites the formant configuration. Each formant consists of 7 parameters: exo (excitation time of the f0f), debat (beginning time of attenuation of the f0f), aten (duration of attenuation of the formant), amp (amplitude of the f0f, freq (frequency of the formant), bw (bandwidth of the formant), phase (initial phase of the f0f). For each formant an amplitude description must be provided for each channel.

- (create-fofbank n) allocates a f0f bank of n formants maximum. returns a pointer to the new allocated object.
- (init-fofbank f0fbank start-time end-time init-orf synth) The f0fbank is active from start-time up to end-time. The initial number of formants is specified by init-orf. f0f synth tells which object is actually synthesizing the f0fs triggered by the configuration.
- (set-fofbank f0fbank k) assigns k to the number of current formants in the configuration.
- (set-fofbank-freq f0fbank '(0 0 0) (1 1) ... (n n)) assigns the specified BPF to the parameter j (from 0 to 6 : exo, debat, aten, amp, freq, bw, and phase).
- (set-fofbank-chan f0fbank j '(0 0 0) (1 1) ... (n n)) assigns the BPF to the amplitude descriptor of channel j of formant j.

2.2) F0F synthesizer

The f0f bank describes the evolution in time of the parameters of the f0f configuration and the impulse generator. For the actual synthesis of the samples, a f0f synthesizer must be associated to the f0f bank; the same f0f synthesizer can be connected to several f0f banks. To patch a f0f synthesizer into a filter, one must define an amplitude control to the output of the f0f synthesizer. This amplitude is a scaler on the non-multiplexed output of the f0f banks connected to the f0f synthesizer.

- (create-f0fsynth allocates a f0f synth; returns a pointer to the new allocated object).
- (init-f0fsynth f0fsynth start-time end-time) assigns f0fsynth to start at start-time up to end-time.
- (set-f0fsynth-temp f0fsynth '(0 0 0) (1 1) ... (n n)) assigns the specified BPF to the amplitude control of the f0f synthesizer, i.e., the sum of the non-multiplexed outputs of all the connected f0f 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 (bandwidth of the resonator), and freq (frequency of the resonator). In addition, for each filter an amplitude description can be provided for each channel.

- (create-filterbank n) allocates a filter bank of n filters; returns a pointer to the new allocated object.
- (init-filterbank filterbank start-time end-time init-of) same as for the f0f bank object.
- (set-nfli filterbank k) assigns k to the number of current filters in the configuration.
- (set-filterbank-par filterbank j '(0 0 0) (1 1) ... (n n)) assigns the specified BPF to the parameter j (from 0 to 3 : gain, bw, freq).
- (set-filterbank-chan filterbank j '(0 0 0) (1 1) ... (n n)) assigns the BPF to the amplitude control of channel j of filter i.
- (comm-f0fsynth-filterbank f0fsynth filterbank) assigns filter bank to filter the f0f synthesizer's output.
- (comm-sound-fit sound fit) assigns filter bank to filter the white noise generator's output.

2.4) White noise generator (sound)

Generates a random signal which can be shaped by an amplitude control.

- (create-sound allocates a 'sound'; returns a pointer to the new allocated object).
- (init-sound sound start-time end-time)
- (set-sound-amp synth '(0 0 0) (1 1) ... (n n))

2.5) External sound source (external-sound)

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 '(0 0 0) (1 1) ... (n n))

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.
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-filt-tables)
(Init-gen-lists)
(set-mem-channels 1)
(set-sensor-rate 44100)
(setf f0-fonyn (create-f0-fonyn 512))
(setf f0-fonyn 0 44100)
(setf f0-fonyn 1 10)
(setf f0-fonyn-freq f0-fonyn 0 0.0 0.0 0 0 0 0 0 0 0 0)
(setf f0-fonyn-freq f0-fonyn 1 0 1 0 0 0 0 0 0 0 0)
(setf f0-fonyn-freq f0-fonyn 2 0 0 0 0 0 0 0 0 0 0)
(setf f0-fonyn-freq f0-fonyn 3 0 0 0 0 0 0 0 0 0 0)
(setf f0-fonyn-freq f0-fonyn 4 0 0 0 0 0 0 0 0 0 0)

; 1st formant parameters (1st element of list is time, 2nd is value)
(setf f0-fonyn-par f0-fonyn 0 0 0 0 0 0 0 0 0 0 0)
(setf f0-fonyn-par f0-fonyn 1 0 0 0 0 0 0 0 0 0 0)
(setf f0-fonyn-par f0-fonyn 2 0 0 0 0 0 0 0 0 0 0)
(setf f0-fonyn-par f0-fonyn 3 0 0 0 0 0 0 0 0 0 0)
(setf f0-fonyn-par f0-fonyn 4 0 0 0 0 0 0 0 0 0 0)

; set the 1st formant frequency breakpoint function to 110 Hz
(setf f0-fonyn-par f0-fonyn 0 (1 0 1 0 0 0 0 0 0 0 0))
(setf f0-fonyn-par f0-fonyn 1 (1 0 1 0 0 0 0 0 0 0 0))
(setf f0-fonyn-par f0-fonyn 2 (1 0 1 0 0 0 0 0 0 0 0))
(setf f0-fonyn-par f0-fonyn 3 (1 0 1 0 0 0 0 0 0 0 0))
(setf f0-fonyn-par f0-fonyn 4 (1 0 1 0 0 0 0 0 0 0 0))

; set atten
(setf f0-fonyn-par f0-fonyn 0 (0 0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))
(setf f0-fonyn-par f0-fonyn 1 (0 0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))
(setf f0-fonyn-par f0-fonyn 2 (0 0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))
(setf f0-fonyn-par f0-fonyn 3 (0 0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))
(setf f0-fonyn-par f0-fonyn 4 (0 0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))

; set bandwidth
(setf f0-fonyn-par f0-fonyn 0 (0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))
(setf f0-fonyn-par f0-fonyn 1 (0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))
(setf f0-fonyn-par f0-fonyn 2 (0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))
(setf f0-fonyn-par f0-fonyn 3 (0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))
(setf f0-fonyn-par f0-fonyn 4 (0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))

; set phase
(setf f0-fonyn-par f0-fonyn 0 (0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))
(setf f0-fonyn-par f0-fonyn 1 (0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))
(setf f0-fonyn-par f0-fonyn 2 (0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))
(setf f0-fonyn-par f0-fonyn 3 (0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))
(setf f0-fonyn-par f0-fonyn 4 (0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0))

; set channel amplitude
(setf f0-fonyn-par f0-fonyn 0 (0 0 0 0 0 0 0 0 0 0 0))
(setf f0-fonyn-par f0-fonyn 1 (0 0 0 0 0 0 0 0 0 0 0))
(setf f0-fonyn-par f0-fonyn 2 (0 0 0 0 0 0 0 0 0 0 0))
(setf f0-fonyn-par f0-fonyn 3 (0 0 0 0 0 0 0 0 0 0 0))
(setf f0-fonyn-par f0-fonyn 4 (0 0 0 0 0 0 0 0 0 0 0))

(open-port "sound-dir/fool.hnm"
"open-soundfile"
"coms-f0-fonyn-port-f0-fonyn"
"synthesis 0 1000"
"close-port"
(coer-fm "sound-dir/fool.hnm" "afff"
"play-out-file "sound-dir/fool.hnm" 44100 0 0 0 0 0 0 0 0 0 0)

3.1 Control of the synthesizer by Patchwork

Patchwork is a graphical compositional environment [Lauron 89] which inherited the experience developed at IRCAM around computer aid to composition through the FORMES, Preform (Boyson 86), and Esquisse (Baisnée 88) Projects. Since the last paper at ICCM 90, Patchwork has been ported to Common Lisp, and is running and being developed now only on Macintosh Allegro Common Lisp 2.0 (after theExtend), taking therefore full advantage of the system. All references to the Perform object system have vanished in favor of using CL一世S, as to allow smaller specific code and easier portability.

Control of the CHANT synthesizer in Patchwork is achieved in a simple way: each filter 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 editors, functions editors, chords and rhythm boxes, traditional music notation editors, etc.), describing musical parameters. Special uses of the filters 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, octavation, 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 patch ed 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 compositional needs of a particular rule.

The following patch example shows a pseudo-instrument built around a for-synthesis to which are added rules for automatic calculation of amplitudes and bandwidths according to the frequencies of formants, as well as rules for spectral enrichment according to the fundamental, jitter, scatter, and LFO on formants frequencies, and rule for octavation (control of the formants amplitudes one period over two) (Redif 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 'M6N' 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 'SFP2' window). For instance, the function notation editor module, when clicked on, opens as a traditional music notation editor with extended features: notes/ events can be edited in all dimensions; to
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 developed 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 compositional environment, including a variety of applications: the Esquisses compositional functions (for manipulation on pitch, durations, timbral parameters, etc.), the control of Sound, as well as a growing variety of editors and functionalities. Control of the Modalic physical modelling package [Morrison 91], as well as the new SVP analysis/synthesis-processing package [Depalle 91], are now 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 this year. Connection with the new IRCAM Musical Workstation should allow for instance real-time control and synthesis of CHANT.

References:
[Rodet 84c] Rodet (X.), Cointé (P.) — "Formes: Composition and Scheduling of Processors" (Cambridge, Massachusetts, Computer Music Journal, Vol 8, n°3, 1994.)