JUPITER: FOUR DECADES OF EVOLUTION OF A LIVE COMPUTER MUSIC PIECE

Miller PUCKETTE

IRCAM / UCSD
Reprinted from SMC 2025, Graz, Austria

ABSTRACT

Philippe Manoury's piece *Jupiter* (1987), for flute and live electronics, initially took two years and many lines of bespoke real-time code in C and microcode to realize. The source files from that era, as well as many stages of subsequent development, are available and can be used to track the technological evolution of the piece and of various software tools that have played roles in its realization over the years. It now exists as realizations in both Max and Pure Data, both of which are based on the original text-based sequencing and score-following source files from 1987, with occasional updates by the composer. The Pure Data implementation has been at least partly future-proofed using a novel continuous integration ("CI") framework named Reality Check.

1. INTRODUCTION

The first two decades of IRCAM's existence, roughly from 1977 to 1995, were an intense period of tool-building. It was not even clear at IRCAM's outset that computers would play the dominant role in audio processing there, and in the early days the computer department was only one of several parallel efforts. Since computers of that era were not able to compute high-quality audio in real time, a parallel effort, led in succession by Guiseppe Di Giugno and Eric Lindemann, centered on the development of real-time audio synthesis and processing hardware, at first highly specialized but increasingly relying on off-the-shelf components, until around 1997 general-purpose RISC architectures began to appear that could take over from the specialized systems.

Over the same time period, real-time software environments emerged in a variety of languages and programming styles. An early non-graphical version of "Max" became available in 1985 [1], but only picked up its now-familiar graphical patching language appearance in 1988 [2], and was extended to provide signal-processing "tilde" objects, using Eric Lindemann's ISPW hardware, in 1990 [3]. A first-person recounting of some of this history [4] provides more details.

Pierre Boulez was always eager to use the real-time tools under development at IRCAM. (While he actively encouraged research in other directions as well, including the acoustics of musical instruments, the development of Lisp-based compositional software, and non-real-time synthesis systems including CHANT, he was not personally interested in using those developments in his own compositions. It is to his credit that he encouraged a wide variety of different research projects and musical productions that were not useful in his own work but sometimes proved of great interest to other composers.)

It was Boulez's desire to make a new version of his piece, *Explosante-Fixe*, on which he had started work in 1971, that provided the impetus for his invitation to Barry Vercoe to come to IRCAM and develop a real-time score following system. The result was Vercoe's Synthetic Performer [5], a system that ran on a PDP11 control processor and the 4X, of which there was one wire-wrapped prototype at the time. The system was presented live at the 1984 ICMC which, by chance, was held at IRCAM. An independently developed score following system by Roger Dannenberg was also presented at the same conference [6].

Score following algorithms of the time relied on real-time pitch detection. Vercoe's 4X implementation relied on a system designed by the flutist Laurence Beauregard, in which he equipped 15 of the keys of a standard flute with mechanical switches to aid the PDP11 in following the pitches played by the flute. Since Boulez's plans for *Explosante-Fixe* were to use the flute as the solo instrument, Boulez naturally followed Vercoe's and Beauregard's developments with deep interest. His next step was to invite two young composers, Philippe Manoury and Thierry Lancino, to write pieces that would use the new score following systems, featuring a flute in a solo role. Both productions led to premieres in April 1987, for the tenth anniversary concerts of IRCAM. Of the two, Manoury's piece, *Jupiter*, has proved the longer-lasting.

A detailed analysis of *Jupiter* is available elsewhere [7]. Here we will focus on the history and current state of its realization(s).

Over its 38-year lifespan, *Jupiter* has gone through five major iterations, using four different software platforms. The initial implementation of *Jupiter* used the 1995 non-graphical version of Max to schedule and parametrize a purpose-built microcode program that ran on the 4X processor. This implementation contained much *ad hoc* code and took about two years to realize. The following two implementations used the Max/FTS system, first requiring three ISPW boards hosted by a NeXT computer and then, once unix workstations were powerful enough to run

date	software	hardware
1987	text-based "Max" + 4X microcode	4X, control CPU, Beauregard flute
1994	Max/FTS	ISPW
1997	Max/FTS	SGI workstation
2002	Max/MSP	Macintosh
2003	Pure Data	PC+Linux

Table 1. Five successive implementations of *Jupiter*. Each one is derived from the preceding one except that the 2 final ones derive from the Max/FTS version.

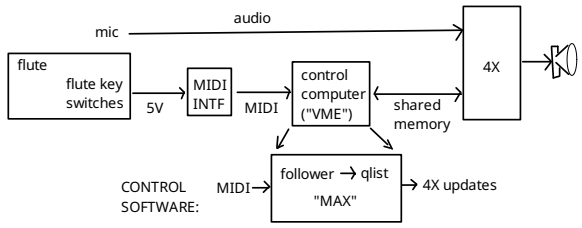

Figure 1. Block diagram for original realization of *Jupiter*

the patch, on off-the-shelf Silicon Graphics (SGI) computers. Subsequently the Max/FTS version was "ported" into two distinct software environments, Max/MSP (by Serge Lemouton, 2002) and Pure Data (Puckette, 2003). The evolution is summarized in Table 1.

2. THE 4X IMPLEMENTATION, 1987

The original hardware and software for *Jupiter* are as shown in Figure 1. The 4X and its controlling computer (referred to as the "VME" after its bus and backplane) occupied two 19-inch racks about four feet tall each. One such pair, along with the original Beauregard flute, is now on display in the Cité de la Musique in Paris as a "historical musical instrument". (Fun fact: the widely-available photograph of that system shows the flute incorrectly plugged directly into a MIDI port; in fact it required an interface, designed by Michel Starkier, to encode the switches into MIDI).

The software used, already referred to as "Max", hosted a patch specified in an ASCII file entered using a text editor. Various "control processes" corresponding to today's object boxes in Max or Pd managed individual MIDI controls, caught keyboard events, printed output to a text-based CRT terminal, or, most importantly, took care of score following and parameter sequencing. Sequencing was managed by two objects, a "qlist" that generated messages for other objects in the Max patch, and an "upd4x" object that updated memory locations in the 4X given as pairs of binary addresses and values. A short extract of the patch for the opening of section 4 of the piece is as shown here:

```
qlist4  -w p4.x --
upd4x4 -w part4.b --
notegroup1  109 ; trigger1 ok --
trigger1 ok ; sin go --
```

Here an object of class "qlist" named "qlist4" is created and fed the input file "p4.x"; similarly, a "upd4x" object (which sends low-level parameter updates to the 4x) is created named "upd4x4". A "notegroup" object is created that waits for either a single note or a combination of notes within a time limit; its output is the object "trigger1" to which it sends a message "ok". The "trigger" object in turn waits for an "ok" message and sends a message "go" to the object "sin" (although this is overridden below).

The qlist file, p4.x, contains entries such as this one for the first event of section 4:

```
10 10000 upd4x4 start 10 11 ;
10 10020 4x reset ;
10 10020 trigger1 set qlist4 start 20 21 ;
10 10020 notegroup1 set 65 ;
10 10050 print (10) ;
10 10070 foo (10) ;
```

These lines are expanded from the input string "b 10 65 0", which declares that the event labeled 10 will be set off by the flute playing a low F (MIDI pitch 65). (The "0" terminates a group of notes of which there is usually only one). The notegroup object is set to wait for the pitch 65, at which time it sends ""ok" to the trigger object (roughly equivalent to a modern message box), whose outgoing message is set to go back to "qlist4" and tell it to go on to event 20. Meanwhile the upd4x object is started at event 10.

The upd4x object reads a file that is generated from the following shorthand entered by the composer:

```
b 10 0 .dr T .rf T
+ T 71
R 400000 r I
b 10 200 R 0
```

but which is converted to a low-level sequence of 4X addresses and values as follows:

```
10 10000 R2_dtor 7fffff R2_rtof 7fffff
ctof 0 R2_fpos 7fffff R2_fshift 3f387
revgate 400000 revgn 3fff00 revlp 1000

10 10200 revgate 0
```

The effect is to send the flute signal to a reverberator and to set the reverb time to infinity; then, after 200 milliseconds, the reverb input is closed by the "10 10200 revgate 0" line. The corresponding 4X microcode is generated by the following lines, written in the 4x patch language:

```
(realrev) = lp(revgn);
revts = revsnd * revgate;
(rrevsnd) = lp(revts);
(hrsend) = gain(hrmout, htor);
(frsend) = gain(fsh, ftor);
(drsend) = gain(input, dtor);
(xrsend) = gain(wawa, stor);
```

```
rs1 = rrevsnd * OVP(hrsend + frsend);
rs2 = rrevsnd * OVP(drsend + xrsend);
(a1,a2) = alp8(rs1, rs2);
(rev[0], rev[1], rev[2], rev[3]) =
    rev4(a1, a2, realrev, revlp);
```

(these lines appeared in a different order and interspersed with other operations in order to optimize the 4x microcode; they are straightened out here for clarity). The calls to "lp", "gain", "alp8", and "rev4" are macros defined elsewhere in the overall patch description. The effect of these lines is to make a mix of four input signals, adjust its overall gain using the "revgate" parameter, and send the result to a reverberator consisting of "alp8" and "rev4" in series.

In addition to the above capabilities that relied on standard ("vanilla") features of Max and the 4X patch language, several purpose-built Max objects were used. These were adapted from standalone C programs, mostly by Olivier Koechlin, that controlled specific 4X microcode patches. For example, an object named "interpol" was able to record two short sequences of incoming flute notes, after which it played a new sequences whose rhythms were calculated in real time as interpolations of the two recorded rhythms. This and a few other such programs were adapted into bespoke Max objects that were added to the standard ensemble of Max objects, somewhat like today's external objects in Max or Pd.

3. THE ISPW IMPLEMENTATION, 1994

The ISPW came online roughly in 1991. It boasted six Intel i860 processors, each running at 40 MHz and having a floating point unit that could perform a multiply-add operation in one cycle. There was a C compiler but optimized DSP code had to be hand-microcoded. So instead of compiling directly from a text-based DSP "patch" into microcode, about 25 primitives, each operating on vector inputs and outputs, were hand-coded, networks of which were assembled by the user in a graphical patch language called Max/FTS. Unlike the 4X, the control code (in C) and the DSP code (in microcode) ran on the same processors. A multi-processor scheduler maintained deterministic timing between the control and DSP operations on all six processors. The host Next machine was not used for any real-time operations but only as a graphical front end and for its filesystem.

The DSP code (which previously had run on the 4x) was rewritten by hand in Max/FTS. An *ad hoc* C program was written to translate the 4x update codes to Max's qlist format. The code for event 10 of section 4 (our earlier example) became this:

```
0 1 ---------------------- 10;
dtor 127;
rtof 127;
fpos 127;
fsfre 71;
rgate 117;
revfb 127;
200 rgate 0;
```

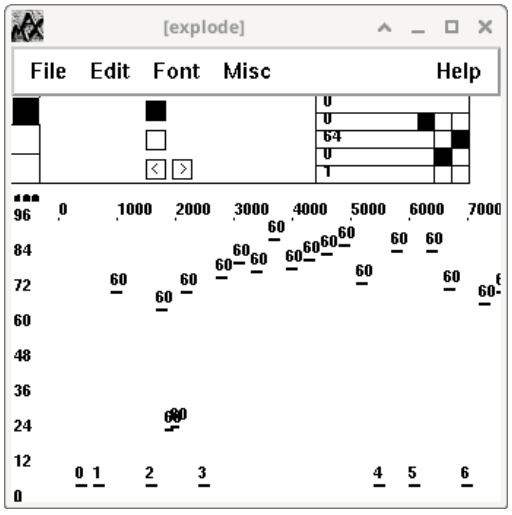

Figure 2. Score following setup using the "explode" object in Max/FTS (figure generated using Max/FTS on linux). The flute part is on MIDI channel 1 and the low-pitched "notes", on channel 2, encode the associated event number as MIDI velocities.

The *ad hoc* objects such as the rhythmic interpolator were replaced, again by hand, by Max/FTS patches.

Score following was completely reworked using an acoustic-only pitch detector and the built-in score following capability of Max/FTS's "explode" object [8] (Figure 2). Because the acoustic pitch detection isn't as reliable as Beauregard's mechanical solution, the score follower had to be replaced with the more robust version available in Explode. The events (points of synchronization between the flute and Max/FTS) were renumbered so that event 1 in the explode window corresponds to event 10 in the original part.

Manoury took advantage of the Max/FTS reimplementation of *Jupiter* to make some major updates in the synthetic accompaniment to the flute, introducing sample playback and the PAF (Phase-Aligned Formant) synthesis technique to replace or improve many of the relatively crude 1987-vintage sounds. The sample rate was increased from the 4X's 32 KHz to the "CD standard" rate, 44.1 KHz.

In 1993-4 Paul Foley made an X/Motif port of the graphical interface (the "GUI") of Max/FTS, allowing us to run the old Max/FTS patches on unix-family machines. Starting in the late 1990s there were workstation-class machines capable of running ISPW patches, and several performances were made of pieces by Manoury and Boulez using SGI hardware. (A partly functioning version of Max/FTS can still be compiled and run on linux. This was used, for example, to generate Figure 2.)

4. THE PURE DATA IMPLEMENTATION, 2003

A little-known feature of Pure Data is its ability to import Max/FTS patches, which is done by opening a file with extension ".pat" instead of ".pd". The patches do not work out of the box. Max/FTS patches are normally

distributed over up to six separate processors (or, in the X/Motif implementation, separate processes linked with pipes). When the patch is imported to Pd it is normally run in a single process and there are inevitably name clashes between parts of the patch that had previously lived in separate address spaces. In addition, Pd's data types, particularly arrays, do not line up with the ones from Max/FTS. In all, a fair amount of adaptation is needed to run the Max/FTS patches in Pd. There are currently Pd "ports" of the Max/FTS patches for several old IRCAM pieces (including a reprise of the long-dormant Manoury piece, *Partition du Ciel et de l'Enfer*, performed by David Pirro (electronics) and the Ensemble Moderne in 2023. There are archived copies of a few other such pieces waiting to be ported if they are to be performed again.

The Pd implementation of *Jupiter* relies almost exclusively on "Pd vanilla", that is, the bare Pd program without the help of dynamically loaded "externs" or of third-part libraries. The one exception is the score folllowing object, "scofo", which is compiled separately from Pd and is dynamicaly linked into Pd at run time. The C language source code for scofo is included as part of the distributed Pd realization of *Jupiter*, so that a full realization of the piece can be built from source using only Pd's own source package plus the patch and its supporting files.

5. CONTINUOUS INTEGRATION, 2025

As the reader might have observed, it can require a huge investment in time to maintain old pieces of live electronic music [9]. In response, a new approach to continuous integration of electronic realizations has been proposed, named Reality Check [10]. To use Reality Check, a Pd patch is tested by running it virtually inside Reality Check, which is itself another Pd patch. The Reality Check patch presents the testing patch with a set of known audio and control inputs, collects its output audio stream, and checks it against a reference output file. If the output of the testing patch still matches the reference output (to within a very small tolerance to allow for numerical differences that can arise on different architectures or operating systems), the patch is considered to have been verified. If so, then in principle the patch can be run with real inputs during a live performance of the piece, in the reasonable expectation that the music will come out properly.

To set up a testing regime, one can start with a planned performance of an existing, working realization of a piece, which may be a public performance or a studio recreation. (In the latter case it usually will suffice to perform only an extract of the piece). During the performance all the inputs to the performance patch, both audio and control inputs, are recorded, along with the audio outputs of the patch. The audio outputs become the reference output used by Reality Check, which can then run the performance patch while reproducing all the inputs. (In practice, it is usually necessary to regenerate the reference outputs since it will probably be hard to exactly reproduce the control inputs from the performance. The composer or musicians should be present during this process so that they can verify that the output audio stream is still correct.)

If any components of the realization subsequently become unavailable or incompatible—either the patches and supporting files for the piece, the sources for any external objects used, the operating system or hardware, or Pd itself—the problem can be detected by testing the realization against Reality Check. The testing process can be set up to run automatically on a regular schedule, or alternatively by hand whenever a new machine or operating system or version of Pd becomes available. Among other uses, Reality Check is invoked to verify that new releases of Pd are backwards compatible in the sense that a locally maintained repertory of realizations passes Reality Check when run against the new version of Pd.

6. CONCLUSION AND GUIDE TO THE ARCHIVE

The realizations described here are archived on msp.ucsd.edu/ideas/2025.03.02.jupiter . The 4x realization from 1987 is not functional since there is as yet no virtual implementation of the 4x, but the source files for the piece are intact and they offer a guide as to how the piece was later adapted. The 1987 version of "Max" is unfortunately lost along with the 4x microcode compiler. However, the source files for the Max "externals" are intact and show fairly clearly how that early version of Max worked.

The Max/FTS source code as adapted for X/Motif comes with x86_64 executables for Max and FTS (the GUI and the real-time program) and with plenty of example patches. There is a separate archive with the documentation as it was shipped with the ISPW version of Max/FTS.

Finally, the latest version of the Pd realization is provided with an early draft of a continuous integration framework using Reality Check. The continuous integration check only covers the extract of section 4 described above, and thus it only exercises a small portion of the overall realization. A more complete test is under development using a recording of the solo flute part by Elizabeth McNutt.

The score following is still in a crude state and should be improved in the future (for example, for historical reasons the patch still relies on the obsolete "fiddle" pitch tracker). But this raises an important question. If we were to improve the piece, either in the numerical accuracy of the audio processing, or perhaps making one or another algorithm more robust, this should be considered an improvement, not a degradation, of the realization. Unfortunately the continuous integration system will report such improvements as errors. The most practical way to maintain the piece might be to keep both the unimproved and any future "better" versions separately, each checked against its own reference outputs.

This situation might someday come to be considered as part of the practice of "period" performances of classical music: sometime in the future, one might have a choice between performing a modernized version of *Jupiter* or its reference realization as preserved using Reality Check. We can even envision someday restoring the original 1987 version of *Jupiter* by emulating the 4X system to be able to use the original materials from the archive described above. The same back-and-forth discussion about the virtues of

modernized versus period performances will doubtless be transposed someday to apply to electronic music performance practice.

Acknowledgments

Thanks to Laurence Beauregard, Barry Vercoe, Cort Lippe, and Olivier Koechlin, who all contributed to the original implementation of *Jupiter*. Max/FTS documentation was edited by Curtis Roads. Additional objects and features in Max/FTS were contributed by Zack Settel, and many design ideas are due to David Zicarelli and were first adopted for the commercial version of Max. Thanks to Paul Foley who made the X/Motif version of Max/FTS that is still runnable, and to Elizabeth McNutt for providing the flute part used to set up Reality Check. Section 2 of this document is partly based on an oral presentation at AREM 2022 in Dresden. And as always, thanks to the reviewers for their helpful comments.

This work is supported by IRCAM and by the DAFNE+ project under Horizon Europe Grant Agreement number 101061548.

7. REFERENCES

[1] E. Favreau *et al.*, "Software developments for the 4x real-time system," in *Proceedings of the International Computer Music Conference*. Ann Arbor: International Computer Music Association, 1986, pp. 369–373. [Online]. Available: quod.lib.umich.edu/i/icmc/

[2] M. S. Puckette, "The patcher," in *Proceedings of the International Computer Music Conference*. Ann Arbor: International Computer Music Association, 1988, pp. 420–429. [Online]. Available: quod.lib.umich.edu/i/icmc/

[3] E. Lindemann *et al.*, "The architecture of the IRCAM music workstation," *Computer Music Journal*, vol. 15, no. 3, pp. 41–49, 1991.

[4] M. S. Puckette, "Max at 17," *Computer Music Journal*, vol. 26, no. 4, pp. 31–43, 2002. [Online]. Available: msp.ucsd.edu/Publications/dartmouth-reprint.pdf

[5] B. Vercoe, "The synthetic performer in the context of live musical performance," in *Proceedings of the International Computer Music Conference*. Ann Arbor: International Computer Music Association, 1984, p. 185. [Online]. Available: quod.lib.umich.edu/i/icmc/

[6] R. Dannenberg, "An on-line algorithm for real-time accompaniment," in *Proceedings of the International Computer Music Conference*. Ann Arbor: International Computer Music Association, 1984, pp. 193–198. [Online]. Available: quod.lib.umich.edu/i/icmc/

[7] A. May, "Philippe Manoury's Jupiter," in *Analytical methods of electroacoustic music*. Routledge, 2005, pp. 145–185.

[8] M. S. Puckette, "Explode: A user interface for sequencing and score following," in *Proceedings of the International Computer Music Conference*. Ann Arbor: International Computer Music Association, 1990, pp. 259–261. [Online]. Available: quod.lib.umich.edu/i/icmc/

[9] E. Lindemann, M. Puckette, and P. Manoury, "Keeping real-time electronic music alive," 2020. [Online]. Available: msp.ucsd.edu/tools/reality/elind-pma-msp-prop.pdf

[10] M. Puckette, "The null piece and reality check," *Revista Vórtex*, vol. 9, pp. 1–15, 12 2021. [Online]. Available: msp.ucsd.edu/Publications/vortex-reprint.pdf