Software Developments for the 4X Real-time System

**E. Favreau , M. Fingerhut,
O. Koechlin, P. Potacsek
M. Puckette, R. Rowe**

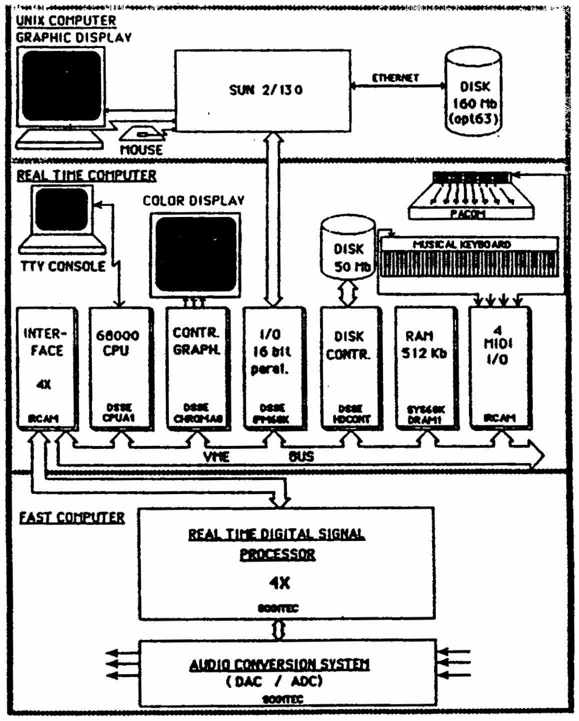
IRCAM 31, rue St. Merri 75004 Paris France
Tel : (1) 42 77 12 33 ext. 48 15
uucp net address : seismo!ircam!ef,mf,ok,pp,msp,rowe

Abstract
The 4X system, realized at IRCAM by G. Di Giugno, is a powerful tool for the synthesis and treatment of sound signals in real-time. Recent developments in software for the 4X include : ORPHE, a monitor for its 68000 host computer; 4xy, a compiler of real-time control programs; MAX, a set of real-time process; and graphic interfaces for the 4X patch language and PACOM console. This paper will describe these new developments in building a real-time application environment using the 4X.

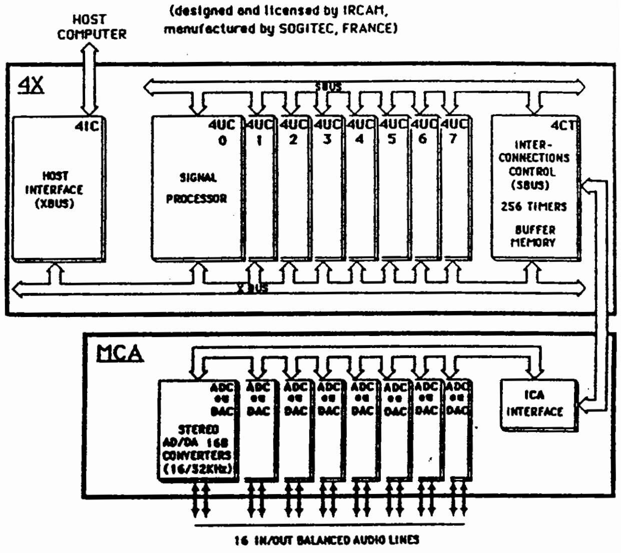
1-INTRODUCTION

The 4X workstation consists of : a Sun host computer running UNIX, connected to a real-time 68000 VME computer, the 4X itself, and a DAC/ADC conversion module.

4X MUSICAL WORKSTATION
GENERAL DIAGRAM
IRCAM
O.KOECHLIN



4X REAL TIME DIGITAL SIGNAL PROCESSOR
AND AUDIO CONVERSION SYSTEM
GENERAL DIAGRAM
IRCAM
O KOECHLIN
(designed and licensed by IRCAM,
manufactured by SOGITEC, FRANCE)



The *real-time monitor ORPHEE*, developed by M. Fingerhut, is distributed between the VME and the host. The VME resident part implements a Unix-like command language mainly used to manipulate files on local and remote storage devices and to run programs; provides such system services as input-output (to local and remote files), memory allocation and remote program invocation through Unix-like system calls; and services real-time interrupt requests. The host's resident part services such remote requests as file manipulations, navigation through its file system and communications with host-resident processes.

4xy is a compiler of control programs which run on the VME 68000 and pilot the 4X in real-time during the execution of a given application. 4xy was designed by R. Rowe and O. Koechlin, and implemented by R. Rowe. Applications have to date included programs for sampling live sounds, real-time FM models, test programs for the 4X and its environment, and musical compositions.

4xy consists of a set of language extensions to the C programming language and a run-time environment. The compiler is designed to support the description of functions, called Actions, which will be executed in real-time and regarded as running in parallel, and to facilitate the interface of these Actions with the 4X patch language and hardware peripheral environment. Many of the constructs in 4xy are meant to simplify the coding of compositional algorithms, and to enhance the ability of such algorithms to be controlled in a real-time environment.

MAX, realised by M. Puckette from MIT, is an *implementation of a set of real-time process scheduling and communication ideas* aimed at making it possible to design elements of a system which can be combined quickly and without changing code.MAX is partly written in 4xy language. MAX greatly speeds the development of new 4X applications, making it possible to use a 4X patch without writing a dedicated control program for it.

The *4X patch language* of P. Potacsek, described in earlier papers, has been equipped with a *graphic interface* on the Sun and Macintosh workstations. The power of the 4X and the multiplication of real-time controls has led to a system of increasing complexity. The development of interactive graphic tools addresses this complexity by providing an elegant and effecient mechanism for generating both binary code to be used directly by the system, and program text which can be incorporated in user applications. In the case of the patch language, an interpreter is loaded into the 4X and executed. This interpreter reads commands sent it through a link with the graphic stations. On the screen, the user can manipulate symbols representing various 4X and higher-level synthesis modules, connecting them to produce and control sound in real-time. Ensembles of such symbols can be grouped and given a name, to be used in other configurations. Once a patch has been developed, the source code for its realization can be output to a file.

A similar tool for the PACOM control console has been developed by Emmanuel Favreau. The same communication link is used between the 4X and graphic host, which this time represents the devices available on the console. The user can assign the output of different devices to registers in the 4X controlling the patch, and immediately test the control on the console itself. The range of values output by each device can be defined, as well as various types of scaling which may be applied before sending a new control value to the 4X. The combination of this tool with the graphic patch tool described above engenders a powerful means of designing and developing patch programs for the 4X and real-time environments to control them.

2-ORPHEE A REAL TIME, UNIX-LIKE MONITOR FOR A 68000 VME-BASED SYSTEM

(M. Fingerhut)

This chapter describes the programming environment of the 4X system. This system is composed of three main parts: the 4X, the VME (a 68000 VME-based system), and the host (a Sun station).It provides a running environment for programs developed under UNIX and used to control the 4X and MIDI devices.

ORPHEE, the real-time monitor, is distributed between the VME and the host. The VME resident part (henceforth referred to as the monitor) implements a Unix-like command language mainly used to manipulate files on the storage devices stored in a hierarchical directory structure, and to run programs; provides such system services as input-output (to local and remote files), memory allocation and remote program invocation through Unix-like system calls; and services real-time interrupt requests from the 4X and other peripheral devices.

The host's resident part (henceforth called the server) services such remote requests as file manipulations, navigation through its file system and communications with host resident processes.

2.1-Boot

ORPHEE can be directly booted from any of the local disks or down-loaded from the host.

2.2-Development Tools

This section lists the specific tools used to prepare a program to be run on the system.

User programs are typically distributed between the 4X and the VME, and can be developed on the host or another remote computer. The VME resident programs can be written either in C or in an extension, called the 4xy language, while those to be run on the 4X are written in a language called patch. In adddition to such standard tools as editors, the following are available:

ccvme	compiler for VME programs written in C
cpat	compiler for 4X programs written in the patch language
csco	compiler for VME programs written in the 4xy language

2.3-Monitor Commands

The monitor commands available at the keyboard are mainly used to manipulate local or remote files, and to navigate in the local or remote file system.

Commands issued at the terminal are of one of two kinds: local (i.e., executed by the monitor) or remote (shipped to the host and executed by the server). Remote command names are preceded by a period.

For example,ls is the list command executed locally, while .ls is the shell (remote) list command. This also applies to command files: one whose name is not preceded by a period will be executed locally (even if it is found on the host), while if it is preceded by a period it will be executed by the shell on the host.

Remote commands, being executed by the remote shell, can manipulate only remote files. Local commands can manipulate local files, and in most cases, also remote files.

2.4-File Names and Directories

User files can reside either on the local or remote file system. The local one is currently composed of 2 drives, 0 and 1, drive 0 being a fixed disk dedicated to the system, and drive 1 a removable cartridge whose contents are left to the discretion of its owner.

The remote file system is the one supported on the host.

The local file system supports hierarchical directories, the root directory being denoted by '/'. There is always a current (local) working directory, displayed by the pwd command and altered by means of the cd command.

A file name is a null-terminated string of ascii characters. Those built-in commands and monitor system calls which manipulate files may need the name(s) of one or more input or output files. These files may, in turn, be indifferently located on the local devices or the host.

2.5-Redirection

Local commands support a limited form of redirection of standard input and output.

2.6-Local Commands

In much the same way as the Unix shell, these commands fall into the following categories:

- built-in, executed by code wired in the monitor,
- command files containing one or more monitor commands,
- executable code files containing code produced by ccvme or csco.

Command files and executable code, though executed locally, can reside on the host.

The built-in commands implemented to date are:

TM	transparent mode
cat	print a file on the standard output
cd	change local working directory
chfn	change label of mounted disk
compress	compactify a fragmented local disk
cp	copy a file onto a new file
creat	create a new file
df	display file system map
dm	display memory map
echo	echo the arguments of the command
ed	edit a text file
exit	exit from the monitor to the VME DSSEBUG monitor.
fsck	check local file system consistency
help	list all commands or help for a specific command
ls	list one or more files on currently mounted local device
mkdir	create a subdirectory in the current working directory
mkfs	make a new local file system
mount	mount a local device or show which is currently mounted
mv	move a file
passwd	change the password of a local disk
pwd	display absolute path to current working director
rm	remove one or more files or directories
setsys	install a bootable system
type	enter the contents of standard input into the file

2.7-Remote Commands

Remote commands are a subset of Unix commands, that are shipped by the monitor to the server, which forks a shell to execute them. They currently include commands such as :

cd, date, echo, ls, mv, printenv, ps, pwd, rm, size, vi.

Note that the local versions of cp and mv can also act on remote files (but since they execute locally, they will be slower than the corresponding remote commands).

2.8-System Calls and User Librairies

Most of the system calls available in a user program running on the VME are Unix-like, which allows for using (most of) the C library functions (e.g., stream-oriented input-output) in a user program. They allow for manipulation of local and remote files, as well as other local monitor services, and currently include:

close	close a file
free	return a block of memory to free storage
lseek	move read/write pointer (also:to find size of a file)
malloc	allocate a contiguous block of free memory
open	open (and create) a file for reading and/or writing.
popen	initiate a remote process on the host
read	read from an open file
unlink	remove a file
write	write into an open file

The open call has been extended to allow for the specification of file search strategies on the local disks and on the host.

3-4xy
A COMPILER OF CONTROL PROGRAMS FOR THE 4X SYSTEM

(R. Rowe, O. Koechlin)

4xy consists of a set of language extensions to the C programming language and a run-time environment. The compiler is designed to support the description of functions, called Actions, which will be executed in real-time and regarded as running in parallel, and to facilitate the interface of these Actions with the 4X patch language and hardware peripheral environment. The source code written by the user is treated successively by the C pre-processor; the 4xy compiler (written with yacc), which recognizes and treats elements of the language and passes C constructs untouched; the C pre-processor for a second time; and finally the C cross-compiler for the MC68000.

4xy is a compiler of control programs for the 4X real-time system, developed at IRCAM by G. di Giugno and his team. The term "control program" is meant to lay emphasis on the fact that 4xy programs run in real-time, in fact only in real-time, and pilot the 4X during the execution of a given application. Such applications have to date included programs for sampling live sounds, real-time FM models, test programs for the 4X and its environment, and musical compositions. This last area is one of special interest for the compiler, since it functions as the "score language" counterpart to the 4X "patch language". Many of the constructs in 4xy are meant to simplify the coding of compositional algorithms, and to enhance the ability of such algorithms to be controlled in a real-time environment.

Actions in 4xy are C functions which are executed following the occurence of one of a set of events. These events include :
- timer interrupts from the 4X
- peripheral interrupts
- execution calls from other Actions or the main program
- the positive evaluation of a boolean expression

Actions cannot be passed arguments and do not return a value. They are, however, organized in a tree structure which permits children of a given node to access certain information about their parent. The user determines the construction of the tree by selecting the point of creation of each Action. Actions created from within other Actions are children of the latter. Actions created in main are children of the tree's root. A single Action can occupy several different positions in the tree : for instance, as a child of the root, child of another Action, and grandchild of still another.

Any Action can be created with several distinct instances. When this happens, the copies of that Action can be regarded as running in parallel. Several language facilities are available to the user to control separate instances of an Action and the communication between a parent and children, which may be multiple instances of a different Action.

To obtain control over an individual instance, the user has access to the keyword variable Instance. The scope of this variable is the entire Action. Instance represents the number of the instance currently being executed : instances are numbered from 0 to the total number of instances - 1. Further, one can access the instance numbers of nodes higher up in the tree structure with the syntax Instance ^^ n, where n indicates how far back up the tree one wants to go. Therefore Instance ^^ 1 refers to the parent's instance number, Instance ^^ 2 to the grandparent's, and so on.

It is also possible to reference the number of executions an Action has undergone since its creation through the keyword variable Count. Count is incremented each execution at the beginning of the Action, and will equal 0 the first time the Action is run, 1 the second time, etc. Count values for higher tree nodes can be accessed through the same convention as that described for Instance : Count ^^ 1, Count ^^ 2, and so on. Counts are distinct for each instance of any Action, and can be assigned a new value by the user. This is useful when one is using Count to index an array, for instance. Count is reset to zero whenever an Action is created, which means that if an Action is created when it is already active, the only effect will be to reset its Count to zero.

Actions are activated, killed, and rescheduled using a set of conditions recognized by the language. For instance, the define an Action one makes the declaration

```
Action <name> (<begin condition> <,end condition>)
```

and to reschedule an Action one includes in its body a statement

```
Next (<condition>)
```

These conditions refer to various devices and expressions in the 4X environment. The available keywords at present include : Boolean, Force, Keyon, Keyoff, Switchkx, Switchon, Switchon, Time, Tty, and Forever. As an example, Keyon refers to pressing a key on a MIDI keyboard. Whenever an Action has been scheduled or rescheduled with the Keyon keyword, that Action will be executed whenever any key on the MIDI keyboard is pressed. Similarly, Actions conditioned by Time will be called when a 4X timer, set by the user in the condition specification to a number of milliseconds, fires.

The instruction Go generates a background loop which receives interrupts, polls virtual devices, and calls their associated Actions. The instruction Stop can be included in any Action. When this statement is encountered, execution of the background loop is terminated and control returned to the instruction following Go.

The interface with the patch language is accomplished through the mediation of a header file, generated by the patch language, which bears the name of the patch followed by the extension ".h". This file contains a set of #define statements which permit the programmer to use the same symbolic names for 4X data memory locations as those which were used in the patch language.

A wide range of scalers and reverse scalers are provided, which allow the manipulation of values expressed in terms of decibels, hertz, or other scales. Such values, modified in a 4xy Scale statement, can be passed to and from the 4X such that both the human and the machine understand them.

Other facilities support the description of breakpoint envelopes, pitch, and duration sets which can include real-time calculations. A full range of library routines and utilities complete the 4xy system.

4xy, which was begun in March of 1985, is now used for most of the applications developed for the 4X machine, ranging from test programs to musical compositions. The greatest virtue of the compiler has proven to be its ease of adaptation : since it is an extension of C, features which do not exist in the language can be coded directly and added to the language itself once the generality and function of a given technique have been thoroughly tested. This method insures both that 4xy itself will continue to grow, and that efforts of development will not be needlessly duplicated.

4-THE MAX REAL-TIME CONTROL SYSTEM
(M. Puckette)

MAX (named in gratitude to Max Mathews) is an implementation of a set of real-time process scheduling and communication ideas aimed at making it possible to design elements of a system which can be combined quickly and without changing code. Many such elements are standardly available, such as processes which read potentiometer values and update 4X synthesis parameters, or which wait for a specified event and carry out an associated action. MAX greatly speeds the development of new 4X applications, making it possible to use a 4X patch without writing a dedicated control program for it. When actions are desired which are outside the score of the standard MAX objects, it is easy to add new objects and interface them to the existing ones.

Communications between the objects, which in MAX are called control processes (CPs), is done by "letters", which are messages with associated times. The letter is sent to the MAX scheduler, which sends it to the destination CP at the letter's "start time." If more than one letter's start times have arrived, the scheduler delivers the letter with the earliest "deadline."
Letters are formatted as lists of symbols, so that it is easy to type in a letter to send to a control process, or to print the contents of any letter.
The lack of data structure implies the greatest possible flexibility in configuring CPs.

To use MAX, the user specifies the set of CPs he desires in
a configuration file which MAX reads at run time. A simple such file might contain:

```
4x --
tty ttyx --
```

which allocates one CPs named "4x" which writes, reads, and graphs 4x registers and function tables and another named "tty" which prompts for messages typed on the teletype and sends them as letters to other CPs. This configuration as it stands is enough to do most patch debugging. For more immediate control, MAX makes pots and switches available; thus, for example, the configuration

```
4x --
tty ttyx --
switch ks0 4x w amp 0 --
pot kp0 4x freq 0 0x1000 --
```

adds to th first configuration a momentary switch whose action is to send"4x" a letter, "w amp 0", which writes a value to a 4X register, and a potentiometer which controls the value of another register. Some twenty standard CPs perfoorm functions such as I/O, basic event detection, timed sequencing of events (which may themselves be letters and hence may do a wide range of things.) Since all instructions to a CP are in the form of letters, any event can be given any desired action. This may be applied self-referentially; the "action" specified might in turn associate another event with another action.

Although MAX is written under 4xy, it is not currently easy to combine CPs in MAX with Actions in 4xy; this is the subject of current work.
We are planning to add a "4xy" CP to MAX and a "Max" keyword to 4xy allowing a user to write his own 4xy application with MAX as a subset.
In addition, a reimplementation of MAX in lisp is planned for 1987; the same scheduler (in c) can be used but the letters will be replaced by lists containing the arguments of a message; the "delivery" of the letter will be done by a "send" of the message at the appropriate time.

5-DEVELOPMENT OF A GRAPHIC INTERFACE FOR THE 4X PATCH LANGUAGE

(P. Potacsek)

The increasing power of digital signal processors and the multiplication of real-time controls (keyboards, sequencers, customized controls, etc.) has led to the realization of workstations which are more and more complex. Usually, different elements have specific programming demands even though an efficient man-machine interface should use a more unified approach to different components of the system. In the case of the 4X musical workstation, programming the system can be separated into two parts : first, the definition of the instrument or signal processor algorithm with the "patch" language, and the second being the control of execution in real-time, using peripheral inputs, through a "control" language.

One of the most delicate points of such a system, for a novice user, is programming and debugging an algorithm composed of signal processing techniques (scaling, sampling, filtering, and the like) which a machine such as the 4X requires.

To assist with this problem, we have designed and implemented a mouse-driven, bitmap graphic interface for use in programming the 4X. The interface consists of an iconic command language, a graphic editor, and a graphic debugger.

The iconic command language makes a link between the graphic world and the ASCII, text world. It makes conversions between bitmap files and "patch" language files in both directions. It also allows the user to command the 4X and load binary files. Figure 1 shows the different commands available in this language.

The graphic editor permits the description, creation, and symbolic manipulation of signal processing modules, which we will henceforth refer to as "functions", in an iconic form. The icons represent language primitives (addition, subtraction, multiplication, etc.), or other functions already defined. The user arranges icons on the screen and makes connections by drawing lines between them. These connections symbolize the flow of data between functions. The editor supports the hierarchical management of its function library. Figure 2a shows the description of an oscillator by means of language primitives, figure 2b shows the creation of an oscillator function, and figure 2c shows the description of an oscillator bank using the defined function.

The graphic debugger is primarily a tool for interactive testing and debugging with the 4X. When a patch has been compiled and loaded into the 4X, we can examine the patch graphically, traverse its hierarchy, (i.e. obtain the description of each icon in the patch), and at any point listen to, visualize, or reinitialize any of its data using graphic sliders (figure 3).

In conclusion, we have developed a set of graphic tools which provide an elegant solution to the problems outlined in the introduction. The realization in real-time was made possible by the development of software distributed between the VME-68000 and a host computer.

Two versions of the interface described have been developed. first, implemented on a Macintosh, permits programming and tes a patch in real-time. The Mac handles lexical, syntactic, and sema analysis, and sends commands to an interpreter running on VME-68000 which is composed of a macro-language of comma a micro-code optimizer, a loader, and several utilities. The sec version is in development and uses a SUN 2/130 and VME-68000. This version will have two options : the first will al programming and testing in real-time; the second will only al testing of compiled patches, but these patches will have undergo better optimization.

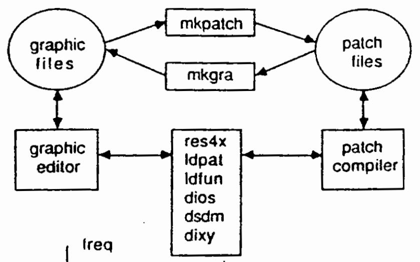


figure 1

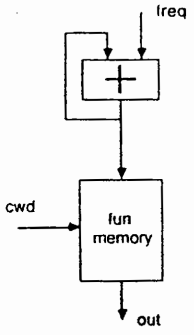


figure 2a

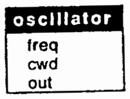


figure 2b

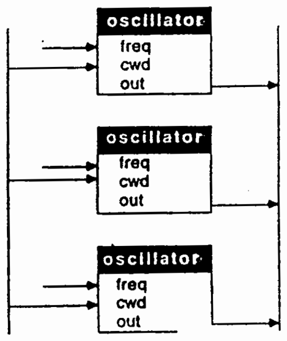


figure 2c

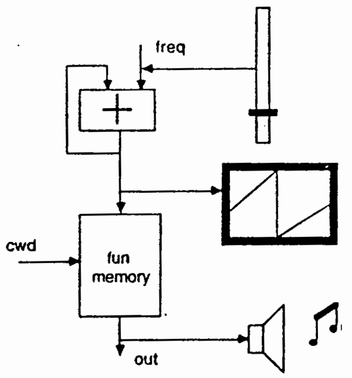


figure 3