Will Software Modules for Performing Arts be Sustainable ?

A. Bonardi, J. Barthélemy, R. Ciavarella, and G. Boutard, *Ircam*

Abstract-- **When moving from hardware device to computers on stage at the end of the 1970's, artists did not realize they would sometimes be unable to reperform their works twenty years later. Many of them now use software modules that face important sustainability problems. In the framework of Caspar European project, the On Line Service Team at Ircam proposes a series of new tools to accompany the maintenance of software modules required by performers on stage.**

Index Terms--**preservation, softwares for electronic music, sustainability.**

I. Introduction

A. Historical introduction to the issue

Since the 1970's, the field of performance arts has quickly evolved thanks to the appearance of computers, softwares and computerized devices that have transformed stage practices. Whereas performers used hardware device for all signal processing required on stage, they progressively moved to software environments enabling to develop personal interactive modules. This concerned first of all music, but quickly expanded to dance, theatre, installations, etc, as we will show below.

More than thirty years later, performers are now developing their own software modules (they often recruit developers in that purpose). But individual artists as well as creation centres or institutional studios face the same problem: the sustainability of these software modules. They now use such graphical languages as Max/MSP[1] or PureData[2] which are submitted to software upgrades and hardware evolutions.

Many researchers, users and composers, especially in electronic studios, have become aware of the fragility of pieces using electronics [11]. As Nicola Bernardini and Alvise Vidolin have noticed, the situation is quite paradoxical [2]: "real-time/performed electro-acoustic music (also known as live electro-acoustic music) is currently facing a serious sustainability problem: while its production is indeed considered very recent from the music history point of view, several technological generations and revolutions have gone by in the meantime".

This sentence may be applied to all artistic works using realtime electronics. Being able to reperform correctly the most important pieces previously created in the studios of institutions becomes important for them, since they all try to find a balance between the constitution of a repertoire and the promotion of creation [12], [10]. At Ircam for instance, a repertoire of nearly 60 works using software modules is now identified as a core of music pieces considered as being interesting for future reperformance.

Whereas preservation of music has been studied and practiced for many years on a wide range from manuscripts to instruments, improving the sustainability of live electronics pieces has recently become a growing issue of late. Several recent publications (in 2004 and 2005, [3] [6]) show that many institutions feel concerned. A European project named Caspar[3] (Cultural, Artistic, and Scientific Knowledge for Preservation, Access and Retrieval) has been launched in 2006, bringing together 17 partners, including IRCAM, on the general topic of preservation of digital data. This project intends to address three different communities, by developing three different testbeds: one for scientific knowledge, one for cultural heritage, and one for performing arts.

B. Scope of uses of software modules on stage

Our study concerns software modules for performance implied in all arts on stage, used for such various activities as signal processing or symbolic calculation. Performing arts include of course music, but also dance, theatre, video, interactive installations, etc. They may require human performers or not.

We give below a few examples, that are of course not exhaustive.

TABLE I
Examples of uses of software modules on stage

Configuration	**Example of work**
Solo instrument and live electronics	*Anthèmes II*, by Pierre Boulez, for violin and live electronics (1997)
Soloists, ensemble and live electronics	*Répons*, by Pierre Boulez, for six soloists, chamber ensemble and live electronics (1981-1984)
Dance and Music Performance	*L'écarlate*, dance performance designed by Myriam Gourfink, choreographer, music by Kasper Toeplitz (2001)
Opera with live sound transformations	*K,* music and text by Philippe Manoury (2001)
Theatre with live sound transformations	*Le Privilège des Chemins*, by Fernando Pessoa, stage direction by Eric Génovèse, sound transformations by Romain Kronenberg, 2004
Theatre and image generation	*La traversée de la nuit,* by Geneviève de Gaulle-Anthonioz, stage direction by Christine Zeppenfeld, realtime neural networks and multi-

Work partially supported by European Community under the Information Society Technologies (IST) programme of the 6th FP for RTD - project CASPAR contract IST-033572. The authors are solely responsible for the content of this paper. It does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of data appearing therein.

A. Bonardi, J. Barthélemy, R. Ciavarella and G. Boutard belong the Online Service Team at IRCAM, 1 place Igor-Stravinsky, 75004 Paris, France. {alain.bonardi, jerome.barthelemy, raffaele.ciavarella, guillaume.boutard}@ircam.fr

[1] Proposed by Cycling'74 Company, at http://www.cycling74.com

[2] Developed by Miller Puckette, and downloadable at http://crca.ucsd.edu/~msp/software.html

[3] One may refer to the official Website of the project: http://www.casparpreserves.eu

Configuration	Example of work
	agent systems by Alain Bonardi, 2003
Musical and video performance	*Sensors Sonic Sights (S.S.S.)*, music/gestures/images with Atau Tanaka, Laurent Dailleau and Cécile Babiole (performed since 2004)
Installation	*Elle et la voix*, virtual reality installation by Catherine Ikam and Louis-François Fléri, music by Pierre Charvet (2000)

C. Performance works curators and designers look for sustainability

The worse situation for all these artistic works is the impossibility of reperformance (for various reasons) after the creation. Therefore, institutions concerned and composers are interested in sustainability. It paradoxically means preserving authenticity and at the same time enabling possibilities of evolution.

On the one hand, each time a work is played again, performers face the issue of authenticity, trying to set a kind of loyalty, in reference to the materials and to previous reference performances (perhaps belonging to opposite interpretation traditions), famous or not. This requires the storage of results as sound samples to keep a mimimal memory of previous performances, but may be insufficient to evaluate the reperformance. Indeed authenticity criteria seem quite difficult to define, since composers have different conceptions of it: musical assistant Serge Lemouton reports how the same processes of migration of Max/MSP patches (from Next environment to Macintosh) applied to several works at IRCAM led to various reactions: whereas composer Philippe Manoury considered the new result as too close to the original, composer Michael Jarrell found it too far from the original.

On the other hand, composers often want to modify their works, or performers intend to adapt a work to other configurations. Maintenance of software patches is a very difficult task, since they are not structured as programs for instance; therefore slight modifications of patches a few months after completing them may become quite laborious. During an interview in September 2006, Andrew Gerzso, who is Pierre Boulez' musical assistant, indicated us that important patch revisions for technical reasons are opportunities for musical transformations. This was for instance the case for *Anthèmes II*, for violin and live electronics (1997), by Boulez, before its recording.

II. GLOBAL PRESENTATION OF SOFTWARES FOR PERFORMERS

Either hardware (analogical device) or software (patches), active modules provide control and signal processing functions. Musicians nowadays intensively use such graphical languages as Max/MSP or PureData software to implement the required processes. In these frameworks, patches are windows where processes are represented by virtual boxes connected by wires. They are metaphors of hardware devices used in physics labs. These tools enable realtime processing and prototyping, to set up or dismantle a process.

One of the keypoints of electronic music is the deep influence of signal processing. In the past, composers would use analogical device (most often, synthetizers) and physically connect modules together, having the output of one of them linked to the input of another one. Today the situation is the same, except that nearly all artists use, for signal processing, graphical languages that are based on the same paradigm: boxes that provide signal transformations (in a broad sense) are connected together. This construction can recursively be repeated on various levels: a patch can be encapsulated as a sub-patcher of another patch, and also saved independently; in this case it is named an abstraction.

Examples: let us examine two patches achieving the same result, the first one in Max/MSP, the second one in PureData. The outputs of two oscillators, one at 100 Hz frequency, the other one at 200, are added, and the result is played. Oscillators (cycle~ or osc~) and the addition (+~) or product (*~) functions are presented as objects, linked by connections.

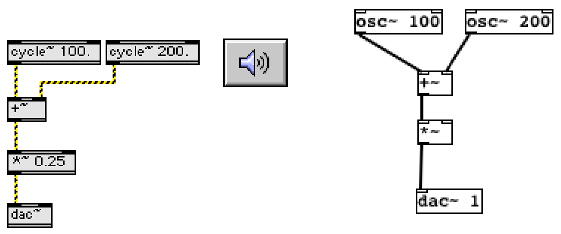


Fig. 1. Adding two signals, with a Max/MSP patch (left) or a PD patch (right).

End-user configuration patches using proprietary software and/or hardware technologies, and binary proprietary file formats, are especially concerned by obsolescence. For instance, Max/MSP uses by default a proprietary binary file format; there is also a text file format, which is quite difficult to interpret since it works like a script language, not as a structured description of the process. Moreover, it has evolved for nearly twenty years by adding blocks of new functions and managing exceptions and aliases. PureData only proposes text file format very similar to the one in Max/MSP. Here is for instance the code of the Max/MSP patch on the left that adds two sinusoids.

Example of a Max/MSP code to add two sinusoids

```
max v2;
#N vpatcher 428 395 883 739;
#P window setfont "Sans Serif" 9.;
#P newex 130 259 31 196617 dac~;
#P newex 130 212 27 196617 *~;
#P newex 130 172 27 196617 +~;
#P user ezdac~ 283 129 327 162 0;
#P newex 199 129 64 196617 cycle~ 200.;
#P newex 130 129 64 196617 cycle~ 100.;
#P connect 0 0 3 0;
#P connect 3 0 4 0;
#P connect 4 0 5 0;
#P fasten 1 0 3 1 204 159 152 159;
#P pop;
```

Another difficulty to achieve sustainability is the division between the graphical representation of signal processing functions and the particular mode of running of these software modules. Whereas the graphical scheme on screen or even the description in the text file format may suggest parallel running, the effective running of a patch is never parallel but always sequential. A Max/MSP patch in fact executes the different processes from the right of the screen to the left, and from the top to the bottom. But what can happen when maintaining such a complex patch as this one just below?

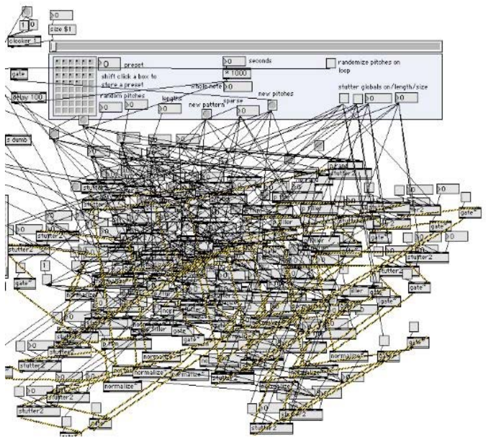


Fig. 2. A complex patch by Olivier Pasquet, musical assistant at IRCAM.

In such a situation, manual move of objects on the screen may cause dysfunctions: since the sequencing of processes depends on the position of objects in the patch, switching two objects is not neutral. It is important to notice that no information about the sequencing appears neither on the screen nor in the text file. The default running mode of the software is not described anywhere according to an external formalism.

III. Strategies for Sustainability

In the framework of Caspar project, the On Line Service Team at IRCAM has evaluated several strategies to face sustainability and re-performance requirements.

From one hand, the simplest way to make an artistic work re-performable is to keep safe all what used in the first performance, including the knowledge to do that. This strategy leads to the institution of a sort of "museum of artistic works" and needs all the related cares (including maintenance, periodic test of devices, periodic retraining of technical personnel, etc.).

On the other hand, the best solution would be to save only the meaning that is inside an artistic work, so that it could be re-performed at any time with the technology of the moment. This approach introduces to the use of new languages needed to describe artistic works [13] and doesn't solve the problem of the already produced works. In the same perspective, some researchers as Yann Orlarey have explored mathematical formalization of signal processing [5] [7]. Often, especially for the early works, people were specifically trained for the performance and all the informations where transmitted orally. Nowadays some authors, performers and technicians are no longer available to reconstruct the original environment. This lack of contents makes it impossible to "translate" intentions into a virtual language. So the strategy is, maybe, the best one for future productions, but only a partial solution for the whole problem. On this, here at IRCAM, we are spending a lot of research, in fact, the instruments to obtain the goal are still to be invented.

Between the first strategy, conservative, and the second, innovative, some other ways can be followed.

One way may be to virtualize hardware, using emulators, and leaving the original software untouched. This is good because the re-performance will use the most recent available hardware but need to develop an emulator for every used device of the past and not solve the problem of knowledge about works.

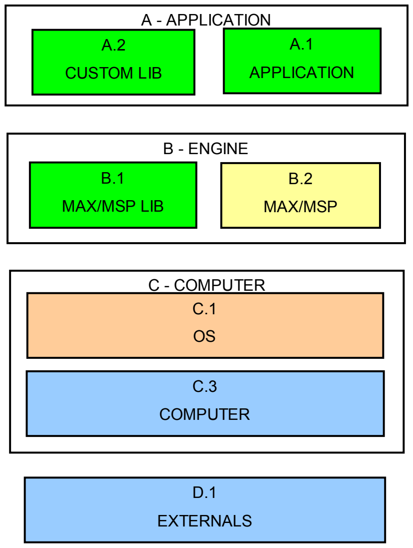


Fig. 3. The simplified schema of the digital implementation for an artistic work

Another way is to "port" the artistic work on more recent environments every time the last used is near to become obsolete. This solution only delays the problem but does not solve it, in fact, the author must be part of every new porting process. Generally he introduces changes into the new work. So, a new artistic work is generated and it has to be conserved too.

Finally, is to be considered the so called "problem of authenticity". It means that every time we modify something into an existing artistic work, we have to prove that the new product is "equivalent" to the original. This can only be achieved using a standard methodology, with absolute physical measures and throw the intervention of reference "validators" (authors, performers, musical assistants, etc.).

IV. Developing Tools to Reinforce Sustainability

A. *Parsing and repository tools*

We have first developed a set of tools named *Patcher Tools* based on a parser of Max/MSP modules enabling to give the whole structure of a patch and its sub-patchers.

For instance, let us consider the Max/MSP patch of *Jupiter*, a work for flute and live electronics composed by Philippe Manoury in 1987, and regularly performed since its creation[4]. Here is a screenshot of the top patch:

[4] This is particularly due to the existence of the MUSTICA data base, where many elements to perform the piece may be downloaded [1].

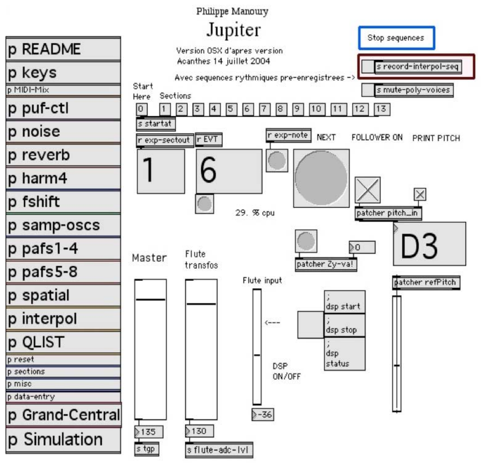
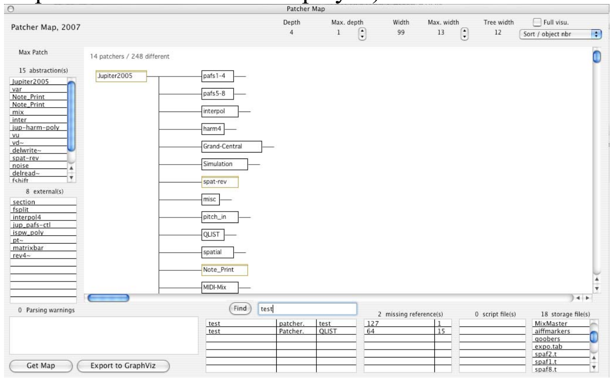


Fig. 4. A screenshot of the main patch of *Jupiter* by Philippe Manoury (musical assistant: Serge Lemouton)

The boxes in the left column are sub-patches that make together a kind of electronic orchestra: 'reverb' is the module of reverberation, 'fshift' is the frequency shifter (a signal process that adds and subtracts a certain frequency to the fundamental of the input signal), and so on. The horizontal series of figures (from 0 to 13) at the top of the patch are buttons enabling to trigger the beginning of the corresponding sections in *Jupiter*. Other boxes are for instance faders to adjust sound levels (input and output) or start/stop buttons.

To have an overview of the whole hierarchy of patches is not possible within Max/MSP. This is our first motivation to develop the *PatcherMap* tool, which can provide it. Another important aspect is to check the set of resources required by the patch. Figure 4 below shows various lists: on the left, the list of abstractions used (they are the files required) ; the list of externals that are third-party objects to complement Max/MSP objects. At the bottom of the screen, the list of missing references (it means sub-patches referenced but not found on the computer where the patch is run), the list of script files used, and the list of storage files saved or read by the patch. In the center of the window, the hierarchy is displayed as a tree, with the possibility of showing only a part of it (by limiting the depth or the width to be displayed).


Patcher Map

Patcher Map, 2007

Depth 4 · Max. depth 1 · Width 99 · Max. width 13 · Tree width 12 · Full visu. · Sort / object nbr

Max Patch

15 abstraction(s): Jupiter2005, var, Note_Print, Note_Print, mix, inter, jup-harm-poly, vu, vd~, delwrite~, spat-rev, noise, delread~, fshift

8 external(s): section, fsplit, interpol4, jup_pafs-ctl, ispw_poly, pt~, matrixbar, rev4~

14 patchers / 248 different

Jupiter2005 → pafs1-4, pafs5-8, interpol, harm4, Grand-Central, Simulation, spat-rev, misc, pitch_in, QLIST, spatial, Note_Print, MIDI-Mix

0 Parsing warnings

Find: test

test	patcher.	test
test	Patcher.	QLIST

2 missing reference(s)

127	1
64	15

0 script file(s)

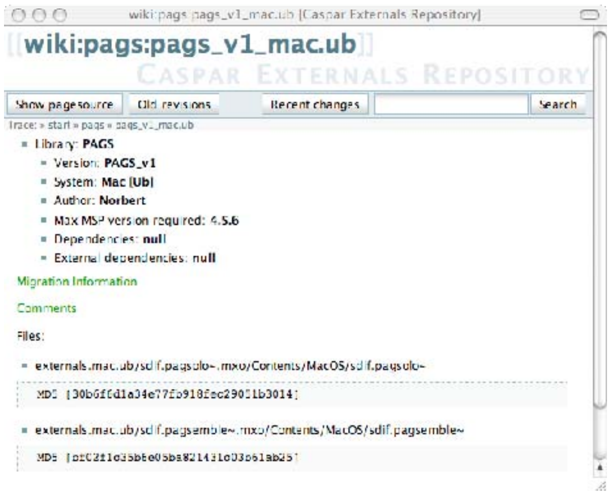
18 storage file(s): MixMaster, aiffmarkers, goobers, expo.tab, spaf2.t, spaf1.t, spaf8.t

Fig. 5. A screenshot of the *PatcherMap* application showing the hierarchy of modules of *Jupiter* by Philippe Manoury, at level one.

B. Components

For externals components such as those generally known in the community as Externals and Abstractions, it is needed to document their features (version, date of last update, etc.), as well as their behaviour, and to develop a specific repository.

For this purpose a portal architecture containing several portlets is considered. The repository would be a mean to store objective informations, as the features listed before, aggregate external sources data and, moreover, grab subjective informations. The last ones ideally being handled by musical assistants, through one or several wiki portlets, that will define informations such as migration information (heuristics...), comparisons, comments...


wiki:pags:pags_v1_mac.ub [Caspar Externals Repository]

[[wiki:pags:pags_v1_mac.ub]]

Caspar Externals Repository

Trace: » start » pags » pags_v1_mac.ub

- Library: **PAGS**
  - Version: **PAGS_v1**
  - System: **Mac (Ub)**
  - Author: **Norbert**
  - Max MSP version required: **4.5.6**
  - Dependencies: **null**
  - External dependencies: **null**

Migration Information

Comments

Files:

- externals.mac.ub/sdif.pagsolo~.mxo/Contents/MacOS/sdif.pagsolo~

```
MD5 [30b6f6d1a34e77fb918fec29051b3014]
```

- externals.mac.ub/sdif.pagsemble~.mxo/Contents/MacOS/sdif.pagsemble~

```
MD5 [[illegible]]
```

Fig. 6. A first prototype of the portal for components

In addition to the information provided, queries will be possible on the content for the purpose of localization of, for instance, a newer version of a component, or a version of the same component on a different operating system… Hash values will be provided in order to enable precise identification of the required components.

Moreover, it will become possible to analyze the text provided as comments in order to build thesauruses, dictionary of terms, and relationship between elements. These different elements will enable the building of a specialized ontology of digital instruments.

C. Display of information

The hierarchy can then be displayed thanks to GraphViz, an open source and standard application for the visualization of hierarchies represented by .dot files. It shows dependencies between different modules (figure 7).

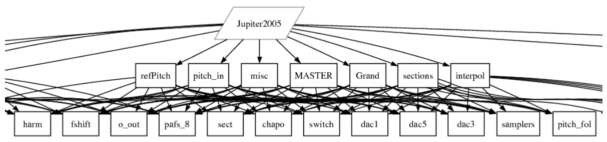


Fig. 7. An extract of the screenshot of the export of Jupiter parsing to GraphViz open application.

These tools may be used to have an overview of patches during their reimplementation but also for the evaluation of scenarios.

D. Software engineering tools

Among the tools developed or planned to give right structuring to the problem we can distinguish two different classes: the ones for static code analysis and the others for run-time performance diagnosis.

The static code analysis class of tools is intended to provide, in structured schemes, further informations about the inside of an artistic work, including environment description, project structure, file structure and catalogs for libraries. We developed an “intelligent” code coverage analysis technique, we think to use it to identify code holes, lacks of consistency, complexity and styles of writing. All the collected informations, as mentioned, will be fit into the repository and will be available as meta-data of the work.

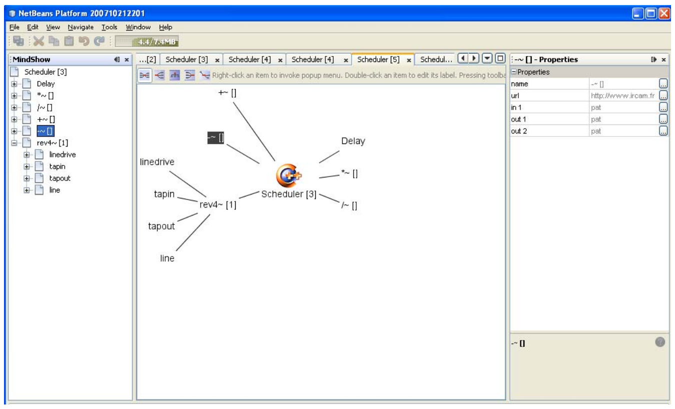
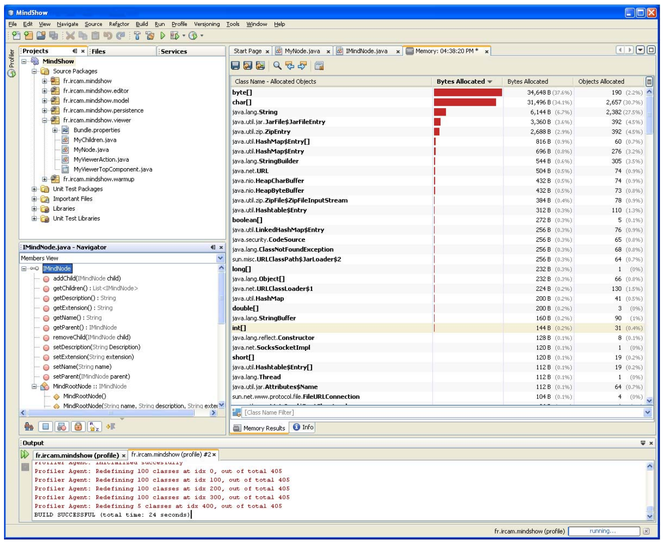


Fig. 8. Static analysis IDE.

The run-time tools will provide the way for both right dimensioning of hardware and monitoring all aspects of performances. Based on professional profilers technology, they monitor global parameters like memory and cpu, but can furnish detailed values on every function running inside the project (number of instances, parameters passing overhead, memory usage, cpu usage, etc.) The tools are specially oriented to musical processing, so directly provide complex structures like buses, pipe-lines and special format meaning.



Class Name - Allocated Objects	Bytes Allocated	Bytes Allocated	Objects Allocated
byte[]		34,648 B (37.6%)	190 (2.2%)
char[]		31,496 B (34.1%)	2,657 (30.7%)
java.lang.String		6,144 B (6.7%)	2,382 (27.5%)
java.util.jar.JarFile$JarFileEntry		3,360 B (3.6%)	392 (4.5%)
java.util.zip.ZipEntry		2,688 B (2.9%)	392 (4.5%)
java.util.HashMap$Entry[]		816 B (0.9%)	60 (0.7%)
java.util.HashMap$Entry		696 B (0.8%)	276 (3.2%)
java.lang.StringBuilder		544 B (0.6%)	305 (3.5%)
java.net.URL		504 B (0.5%)	74 (0.9%)
java.nio.HeapCharBuffer		432 B (0.5%)	74 (0.9%)
java.nio.HeapByteBuffer		432 B (0.5%)	73 (0.8%)
java.util.zip.ZipFile$ZipFileInputStream		384 B (0.4%)	78 (0.9%)
java.util.Hashtable$Entry		312 B (0.3%)	110 (1.3%)
boolean[]		272 B (0.3%)	5 (0.1%)
java.util.LinkedHashMap$Entry		256 B (0.3%)	76 (0.9%)
java.security.CodeSource		256 B (0.3%)	65 (0.8%)
java.lang.ClassNotFoundException		256 B (0.3%)	68 (0.8%)
sun.misc.URLClassPath$JarLoader$2		256 B (0.3%)	64 (0.7%)
long[]		232 B (0.3%)	1 (0%)
java.lang.Object[]		232 B (0.3%)	66 (0.8%)
java.net.URLClassLoader$1		224 B (0.2%)	130 (1.5%)
java.util.HashMap		200 B (0.2%)	41 (0.5%)
double[]		200 B (0.2%)	3 (0%)
java.lang.StringBuffer		160 B (0.2%)	90 (1%)
int[]		144 B (0.2%)	31 (0.4%)
java.lang.reflect.Constructor		128 B (0.1%)	8 (0.1%)
java.net.SocksSocketImpl		120 B (0.1%)	1 (0%)
short[]		120 B (0.1%)	19 (0.2%)
java.util.Hashtable$Entry[]		112 B (0.1%)	19 (0.2%)
java.lang.Thread		112 B (0.1%)	1 (0%)
java.util.jar.Attributes$Name		112 B (0.1%)	64 (0.7%)
sun.net.www.protocol.file.FileURLConnection		104 B (0.1%)	4 (0%)

```
Profiler Agent: Redefining 100 classes at idx 0, out of total 405
Profiler Agent: Redefining 100 classes at idx 100, out of total 405
Profiler Agent: Redefining 100 classes at idx 200, out of total 405
Profiler Agent: Redefining 100 classes at idx 300, out of total 405
Profiler Agent: Redefining 5 classes at idx 400, out of total 405
BUILD SUCCESSFUL (total time: 24 seconds)
```

Fig. 9. Profiler

V. CONCLUSION

We have shown how artists now experiment the issue of sustainability of the software modules they use in their performances on stage. The main challenge is the possibility of reperformance several years after the creation, though computers and softwares evolve very quickly.

In the framework of Caspar european project, Ircam has started developing tools to provide some help for sustainability: parsing tools, software engineering ones (static and dynamic analysis).
But the field of research is wide, and a lot remains to be done, on both aspects: sustainability of previous works, and sustainability of undergoing creations. They both require the implementation of human-to-machine processes to achieve a minimal sustainability with a minimal authenticity.

VI. REFERENCES

[1] Bachimont, B., Blanchette, J.-F., Gerszo, A., Swetland, A., Lescurieux, O., Morizet-Mahoudeaux, P., Donin, N., Teasley, J. 2003. Preserving Interactive Digital Music: A Report on the MUSTICA Research Initiative. In Proceedings of the Third International Conference on WEB Delivering of Music (WEB’03), Leeds, England, 2003.

[2] Bernardini, N., and Vidolin, A. 2005. Sustainable Live Electro-acoustic Music. In *Proceedings of the International Sound and Music Computing Conference*, Salerno, Italy, 2005.

[3] Bosma, H. 2005. Documentation and Publication of Electroacoustic Compositions at NEAR. In *Proceedings of the Electroacoustic Music Studies Network International Conference* (EMS 05), Montreal, Canada, 2005.

[4] Bullock, J., and Coccioli, L. 2005. Modernising Live Electronics Technology in the Works of Jonathan Harvey. In *Proceedings of the International Computer Music Conference*, Barcelona, Spain, 2005.

[5] Graef, A., Kersten, S., Orlarey, Y. 2006. DSP Programming with Faust, Q and SuperCollider. In *Proceedings of Linux Audio Conference 2006*, Karlsruhe, Allemagne, 2006.

[7] Orlarey, Y., Fober, D., Letz, S. 2002. An Algebra for Block Diagram Languages. In *Proceedings of International Computer Music Conference ICMA 2002*, Göteborg, Sweden, 2002.

[8] Puckette, M. 2004. *New Public-Domain Realizations of Standard Pieces for Instruments and Live Electronics*. In *Proceedings of the International Computer Music Conference*, Miami, 2004.

[9] Risset, J.-C., Arfib, D., De Sousa Dias, A., Lorrain, D., Pottier, L. 2002. De Inharmonique à Resonant Sound Spaces. In *Proceedings of the Journées d'Informatique Musicale*, AFIM, Marseille, Gmem, 2002.

[10] Roeder, J. 2006. Preserving Authentic Electroacoustic Music: the InterPARES Project. In *Proceedings of the IAML-IASA Congress 2006*, Oslo, Norway, 2006.

[11] Teruggi, D. 2001. Preserving and Diffusing. *Journal of New Music Research*, vol. 30, n. 4, 2001.

[12] Tiffon, V. 2005. Les musique mixtes: entre pérennité et obsolescence. *Revue Musurgia*, XII/3, Paris, 2005.

[13] Music Notation Journal, Fall, 1986, 4(2) pp. 47-48

VII. BIOGRAPHIES

Alain Bonardi , PhD, was born in 1966. He is a researcher at Ircam, mainly dealing with computer art preservation and music representation, member of the Online Services team. **Jérôme Barthélemy**, was born in 1955. He is responsible of the Online Services team since 2005. Since 2000, he participated to several national or EU projects for IRCAM. **Raffaele Ciavarella**, PhD, was born in 1960. He works as an engineer at the Online Services team. **Guillaume Boutard**, was born in 1972. He works as an engineer at the Online Services team.