



HAL
open science

First steps toward embedding real-time audio computing in Antescofo

Nicolas Schmidt Gubbins, Arshia Cont, Jean-Louis Giavitto

► **To cite this version:**

Nicolas Schmidt Gubbins, Arshia Cont, Jean-Louis Giavitto. First steps toward embedding real-time audio computing in Antescofo. *Journal de Investigacion de Pregado (Investigacion, Interdisciplina, Innovacion)*, Pontificia Universidad Católica de Chile, 2016, 6. hal-01257524

HAL Id: hal-01257524

<https://hal.archives-ouvertes.fr/hal-01257524>

Submitted on 17 Jan 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



1 FIRST STEPS TOWARDS EMBEDDING REAL-TIME AUDIO
2 COMPUTING IN ANTESCOFO

3 Nicolás Schmidt Gubbins¹, Arshia Cont², Jean-Louis Giavitto³

4
5 ¹ Pontificia Universidad Católica de Chile. Escuela de Ingeniería, Departamento de Ciencia de la Computación. 6to año.
6 nschmid1@uc.cl

7 ² IRCAM UMR STMS. INRIA - MUTANT TEam-project. Arshia.Cont@ircam.fr

8 ³ IRCAM UMR STMS. CNRS. INRIA - MUTANT TEam-project. Jean-Louis.Giavitto@ircam.fr
9

10
11 **Abstract**

12
13 The develop of Antescofo software has allowed contemporary musicians to create
14 interactive music pieces in a more precise way in terms of the synchronization between
15 human and machine. INRIA's MUTANT team has been developing a version of Antescofo
16 that changes the DSP computation paradigm and translates the responsibility to the
17 software and no longer to the host environment. Thus, the composer gets more freedom to
18 create his own effects. Plus, it allows the composer to change the sound link network in
19 execution time. Also, the computational power required for the DSP is optimized. Lastly,
20 this new version creates the possibility of generating a self-contained score for UDOO
21 platform, creating this way the first steps toward the preservation of interactive musical
22 pieces through time.

23 In this article the context of the new version of Antescofo is presented, as well as the
24 benefits of it. The methodology of compilation of the Faust DSP tool in Antescofo is
25 described and a comparison of the profiling tests between the old and new versions is
26 detailed. The results shows an improvement of about 46% in terms of the computational
27 power needed to process the signals and they represent clear indicators of optimization that
28 can be extended to the compilation of the Antescofo software for the UDOO platform.

29
30
31 *Key words:* Antescofo, UDOO, Faust, Real-time Audio Interaction, Interactive Music.

32
33
34 **1. Introduction**

35
36 **a) Antescofo and Interactive Music.**

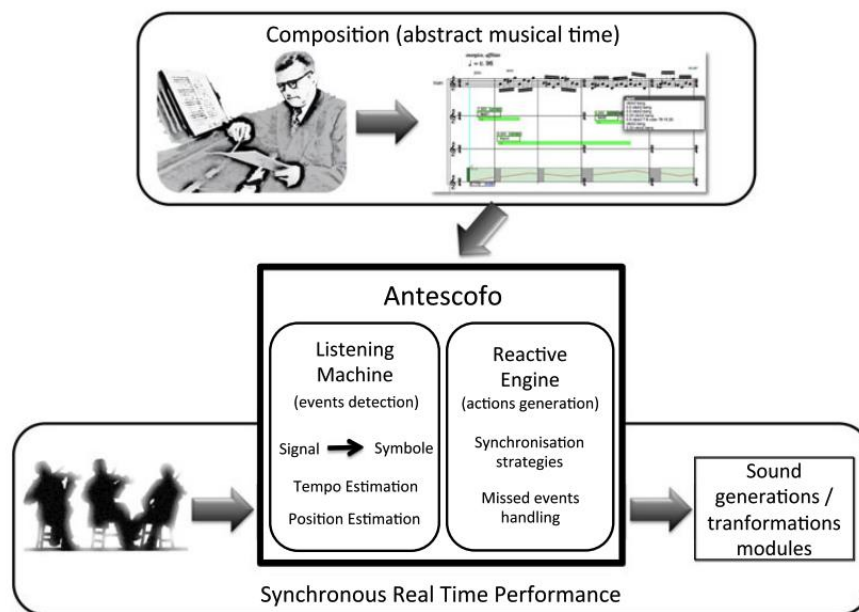
37 There is a field in contemporary music called interactive music. Interactive music is about
38 the execution of a musical piece that has been written for both human and also electronics
39 parts. In this kind of compositions, the electronic system responds to the music interpreted

40 by the musicians and a mutual feedback grows between machine and human. Since the
41 beginning of this field, the problem of synchronization between this two agents has always
42 been present.

43 Antescofo is a software created by Arshia Cont and the composer Marco Stroppa in the
44 year 2007, and its development has been continued by the INRIA's MUTANT team as a
45 common project between INRIA, IRCAM and the CNRS since 2012. The main goal of this
46 software is aid with the synchronization between the electronic system and the musician in
47 an interactive music composition.

48 Antescofo has mainly two components: one **machine listening** compound and a reactive
49 engine system [1]. The machine listening is responsible of everything related to the score
50 following: identifying where exactly in the score is the musician at any instant. It also has
51 functions of **pitch and beat tracking**. The reactive engine system is responsible of
52 triggering the actions specified in the score as a response to certain events produced by the
53 musicians.

54



55

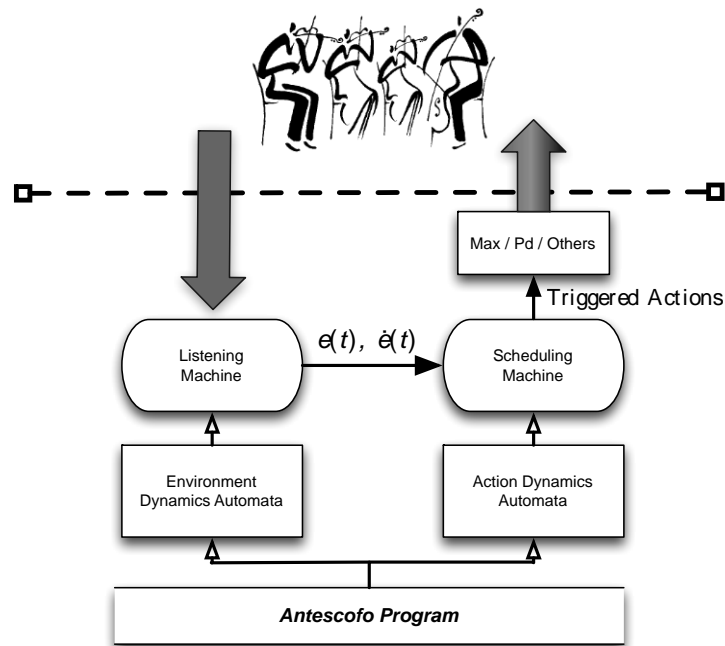
56 **Figure 1.-** Schema of Antescofo working system

57

58 The current Antescofo augmented score language has been mainly developed by Jean-
59 Louis Giavitto and José Echeveste since 2012, and it is highly inspired by Synchronous
60 Reactive languages such as ESTEREL and Cyber-Physical systems. It allows the composer
61 to write both, musician and electronic parts in the same **score**, with a very intuitive
62 language based on actions triggered by the musician. Antescofo also allows the composer
63 to program different actions, variables, operations, error handling, function declaration, etc.

64 Antescofo was developed as a C++ software compiled as a patch for PureData and Max
65 MSP host environments.

66 With Antescofo, the problem of synchronization between the musician and the electronic
67 parts of an interactive music piece is solved. The way that the sounds are processed and all
68 the effects are generated is done in the following way. The score is loaded to Antescofo and
69 the listening machine compound waits for the musician. When the musical events are
70 identified, the reactive engine triggers the actions specified in the score. Then, Antescofo
71 communicates with the host environment (Max MSP or Puredata) via messages, so the
72 sounds and effects are processed and generated in an external **patch** inside the host
73 environment and controlled by the messages.



74

75

Figure 2.- Antescofo Synchronization

76

77 This current **DSP** computational chain presents the following problems:

78

- Non – optimal performance
- Host environment dependencies
- Preservation through time

79

80

81

82 As the DSP chain depends on the host environment, all the DSP processes performance
83 depends on how the platform manages the **system resources**. Max MSP and Puredata do
84 not manage resources the in the most optimal way. The **scheduling** of these platforms is
85 flat and they are not aware of the musical context, so if the patch is not good or needs a lot
86 of resources for the DSP processing, it will not be able to process all the buffers and
87 maintain a real time performance.

88 As the DSP processes are computed by the host environment, there are a lot of
89 dependencies to the platform. This means that if the host environment change from one
90 version to another, the whole musical piece could lose some parts of the electronic
91 compounds. This causes a distributed score that could cause problems of preservation of all
92 the dependencies.

93 The final problem is the preservation of interactive music pieces. As we know, technology,
94 platforms and protocols of communication of the applications are continuously changing. In
95 the technology and informatics field there are a lot of examples of platforms, formats and
96 protocols that had become deprecated and obsolete. This means that there is no guaranty
97 that one interactive musical piece composed today is going to be able to be played in the
98 future, and this is a big problem for contemporary musicians and the transcendence of their
99 art.

100

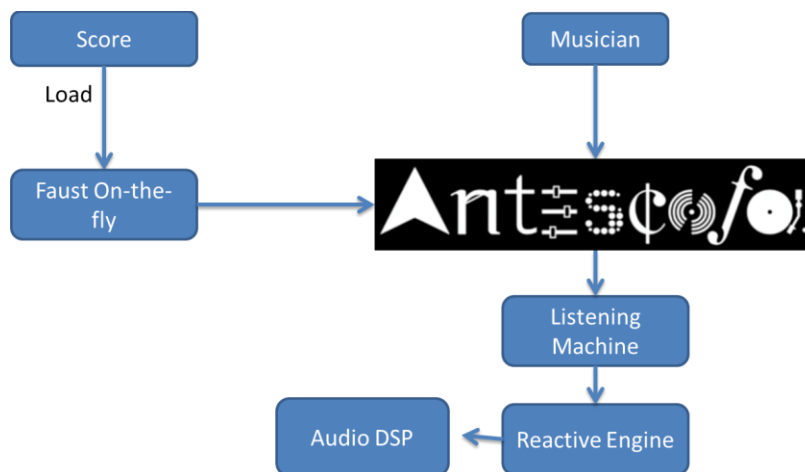
101 2. Metodology

102

103 a) Modular DSP Tools.

104 As a solution for the potential problems with the current version of Antescofo, Mutant
105 started developing a version that has different modular DSP tools embedded. The main goal
106 of this is to change the DSP chain process so the responsibility of generating the sounds
107 and processing the effects relays on the Antescofo patch and not in the host environment.
108 This way, all the sounds and effects can be processed in a more optimal way than the way
109 done by Max or PureData.

110



111

112 **Figure 3.-** Antescofo – Faust DSP chain

113 The integration of the modular DSP tools gives also the possibility to the composer to write
114 his own effects and DSP inside the Antescofo score. Also, the use of the different DSP
115 tools makes the Antescofo score much more self-contained, because it does not depend on
116 the host environment to generate the effects and sounds. This means that the effects and
117 sounds will remain exactly the same as in the moment that the piece was created regardless
118 of any changes applied by the host environment from one version to another.

119 Another advantage of this new version is the possibility of changing in real time the DSP
120 network. This means that using the Antescofo control variables, different effects can be
121 redirected from one link to another creating an infinity of dynamic possibilities on the
122 routing of the different effects, just like plugging inputs and outputs from and to each
123 effect. This was possible with the host environments but once again it depends on the way
124 that the platform implements the different signal routing through the different patches.

125 The different tools that Mutant wants to implement on Antescofo are Faust, Fluidsynth,
126 Csound and SuperCollider. Faust is already embedded in a new version of Antescofo and it
127 is already working.

128 Faust is a functional programming language for real-time signal processing that has been
129 developed by Grame since 2012. It allows programming signal processing software and
130 compile them as plugins or standalone applications [3]. Faust translates a high-level
131 description of a signal transformation to a C++ program that has been optimized by a
132 lambda-abstraction-based mechanism. The fact that Faust produces a C++ program, allows
133 the user to compile it to generate plugins and standalone applications such as MAX/MSP or
134 PureData patches, VST plugins, ALSA or Coreaudio applications.

135 Faust effects are written in Faust language inside the Antescofo score. These effects are
136 compiled with an on-the-fly compiler when the score is loaded to Antescofo. Thus
137 Antescofo calls the Faust functions already compiled only when they are needed, so the
138 performance of the application is much better than passing messages to external Max
139 patches to do the signal processing.

140

141 **b) Preserving art Through Time.**

142 MUTANT has decided to implement a version of Antescofo for the UDOO platform in
143 order to preserve interactive music pieces through time. The high speed developing in
144 technology, and the permanent changes in the paradigms of storage and reading digital
145 information, could produce that an interactive music piece composed today could not be
146 able to be played in the future. The goal of preserving one hardware with Antescofo
147 embedded is to create a piece of art that can be preserved over time despite all the
148 technological changes.

149 UDOO is a mini computer developed by Aidilab srl and SECO USA Inc. Its hardware
150 includes an ARM Cortex A9 processor (corresponding to two CPU Freescale i.MX 6), 1
151 GB of RAM memory and one Arduino interface. The biggest advantage of this device is its
152 small size, great performance and versatility, because it allows the developing of
153 applications that integrates Arduino sensors and devices, in a high performance mini-
154 computer. This platform is one of the most powerful mini computers that are currently in
155 the market, so the challenge of being able to run in an efficient mode Antescofo in this
156 platform, and also perform a score such as Anthèmes 2 are the first steps towards the
157 computation of real-time audio on this kind of platforms.

158 After compiling a PureData version for UDOO and Antescofo for PureData in a Linux
159 operative system, we tried the Anthèmes 2 example patch for PureData to test the
160 performance of the old version of Antescofo. The results were the expected: The

161 performance of Antescofo in terms of the score following system and the reactive engine
162 worked fine, but after passing the messages to PureData, the system got too slow to
163 perform correctly the signal processing (*i.e.* with the adequate timing), so a lot of **clicks**
164 were generated.

165 Antescofo manages the electronic compounds of the score (such as sounds and effects) by a
166 message passing system to the host environment. Thus, the responsibility of the DSP relies
167 on the host environment and it is not managed directly by Antescofo. The problem with this
168 is that host environments such as Max MSP and PureData, do not manage the resources in
169 an efficient way. Because of this, the CPU usage and the time used to compute the effects
170 in real time can make the whole process too slow. Taking in consideration that this kind of
171 software is made to be used in real-time, every non optimal computation can generate a
172 poor real-time performance. In the case of UDOO, the system resources are fewer than in a
173 computer, so this system is very sensitive to the optimization of all the DSP chain.

174 As the new version of Antescofo uses the integrated modular DSP tools, the performance in
175 the UDOO platform should be much better than the old one. Improving the performance of
176 the signal processing processes and optimize the whole system would solve the resource
177 managing problems of Puredata. That is why we compiled this new version for UDOO.

178

179 **c) Anthèmes 2 And Faust**

180 Anthèmes 2 is an interactive music piece composed in 1997 by Pierre Boulez, founder of
181 the IRCAM. It is an 18 minutes piece for violin and live electronics, that was created based
182 on Anthèmes 1, which is a piece composed by the same Boulez in the year 1991 for violin
183 solo. Part of the live electronics compounds of Anthèmes 2 are samplers, frequency
184 shifters, harmonizers, reverbs and spatialization compounds.

185 MUTANT wrote an adaptation of the Anthèmes 2 score for Antescofo, and created a patch
186 in PureData and Max MSP that includes all the live electronics effects, and that are
187 controlled by the message passing system embedded in Antescofo to interact with Max or
188 PureData. This piece of music is used to show all the potentiality of Antescofo in live
189 performances, because it works as an example score in which the user can see different
190 examples of the Antescofo programming tools and interaction with real-time input. This
191 patch uses as input a 1 minute length part of the human part of the score for violin. This
192 sound file works as the real time input of Antescofo, so the results are always the same.

193 To benchmark the two versions of Antescofo, with and without the modular DSP tools
194 (Faust), an Anthèmes 2 score adaptation for Faust was written. This score had to implement
195 in Faust the 5 effects that originally were implemented in an external Max patch. The
196 effects are a 4 channel harmonizer, a Frequency shifter, a Sampler a Reverb and Panners.

197

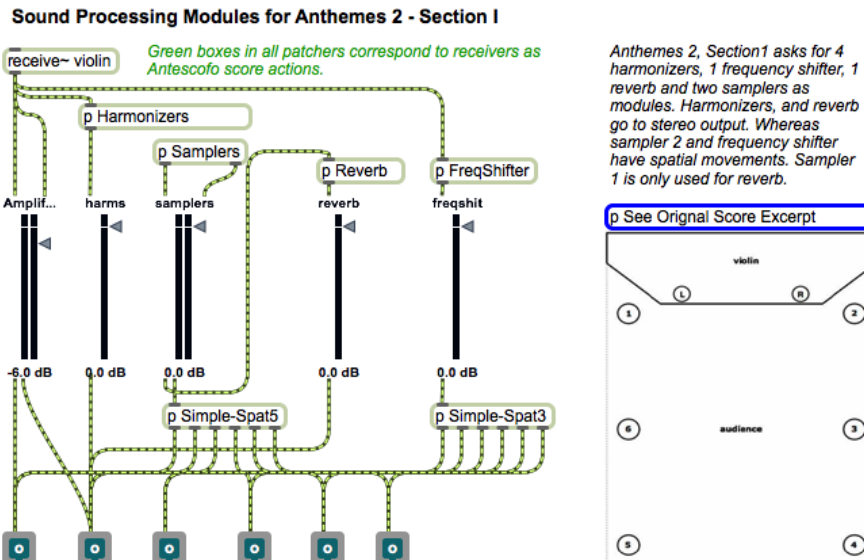


Figure 4.- Anthèmes 2 effects patch in Max MSP.

198

199

200

201 The implementation of the Faust effects used in Anthèmes 2 is the following:

- 202 • Harmonizer: was originally taken from the source code of the pitch shifter
203 implemented by Grame in the examples of the Faust repository. The code was
204 modified in order to have a method that receives the same parameters than the
205 Anthèmes 2 score, *i.e.* semitones. The values of the 4 channels must be integers
206 expressing semitones that represent the amount of semitones from the original
207 frequency that the harmonizer have to reproduce. The harmonizer also receives an
208 amplitude parameter that goes from 0 to 1 to control the amplitude.
- 209 • Frequency shifter: As the harmonizer, this effect was also modified from the source
210 of the pitch shifter code, developed by Grame on the Faust repository. It receives as
211 parameters the value in hertz of the frequency to be shifted from (integer), the value
212 of the Antescofo variable \$PITCH for computing the frequency ratio, and the value
213 of the out amplitude (from 0 to 1).
- 214 • Reverb: this effect is the “freeverb” made by Grame and it was taken from the
215 examples folder in the Faust repository. The parameters this effect receives, are the
216 damp value, the size of the room, the wet of the signal and the out amplitude. All
217 those values goes from 0 to 1.
- 218 • Panner: The two panners are the same. They receive the values of amplitude of each
219 of the 6 channels. The audio input is divided into 6 channels and each output
220 channel is multiplied by the corresponding control parameters (from 0 to 1). With
221 this system it is possible to control the amplitude of the audio input for any channel,
222 and using Antescofo elements such as curves, it is possible to generate panner
223 effects.
- 224 • Mixer: additionally a Faust mixer was implemented to be able to mix down all the
225 audio effects to the corresponding channels. The only parameters are the audio
226 signals, and the outputs are the 6 final audio signals.

227

228 The sampler could not be implemented with Faust because it does not work in the spectral
229 domain, only makes transformations in the domain of time. It is going to be implemented
230 when the Fluidsynth tool is embedded to Antescofo.

231

232 **3. Results and Discussion.**

233

234 After the implementation of the Faust effects, the Anthèmes 2 score was adapted to respond
235 to these new tools. The Anthèmes 2 patch example from the Max MSP host environment
236 was modified, deleting the message passing system and the external patches
237 implementation. The main idea of this was comparing the performance of the two versions
238 of Anthèmes 2: the one with the effects implemented in the host environment, and the one
239 with the effects implemented in Faust language embedded inside Antescofo. Also, the
240 sampler of the original score was removed so it can be compared with the new one.

241 For the profiling tests between the two versions of Antescofo we used the Time Profiler
242 tool from the Xcode developer tools. The Time Profiler allows the user to “record” a
243 sequence of usage of a specific software and it allows him to see in detail how much
244 computer time it takes for any computation or process in the program in real time. The
245 choice of using Xcode Time Profiler is because it allows to see in detail the computational
246 power needed to achieve the signal processing and all the processes involved in the
247 performance of the software. The different calls to all the processes are represented in a
248 tree-like diagram, and all the details including the CPU usage and time needed to achieve
249 the process is detailed. In this special case, the function of interest between the two
250 Anthèmes 2 scores was “dspchain_tick”. This function, written by MUTANT, is the
251 Antescofo function that is in charge of all the DSP processes and sound generation.

252 As Antescofo is a real-time audio application, it was very difficult to measure performance
253 between two versions. This is because the main objective of a real-time audio application is
254 to finish the computation processes before the deadline of the entrance of the next audio
255 buffer, and not to run as fast as possible. For this purpose, the Max MSP mode selected was
256 the Non – Real - Time mode, which allows the program to run as fast as possible, reading
257 the audio input from an audio file. This function allowed us to simulate a real time
258 interaction between a musician and the electronic part through the Antescofo score. The
259 performance of this mode and the time needed to achieve with the end of the
260 “dspchain_tick” function is a real indicator of the optimization between the two versions of
261 the Antescofo software with or without the Faust DSP tool embedded.

262 At the first try, the Antescofo - Faust version of Anthèmes 2 was not better than the one
263 without Faust. The main reason of this was the compilation flags of the Faust compiler
264 inside Antescofo, the size of the vector was 32 samples and the size of the vector on Max
265 host environment was 64 samples. This difference causes a non-optimal performance, so
266 we added the proper Antescofo compilation flags (flags -vec -vec-size 64) to the Faust
267 compiler. Doing this, the size block of the buffers were the same and the performance was
268 much better.

284

285 As the tests were run in a Mac platform, the part depending on the audio architecture could
286 not be compared. Because of this the profiling tests in the UDOO platform are still pendent
287 for the porpoise of profiling the real performance of the software with the DSP tools
288 embedded. However, the present results are valuable and represents a guaranty that using
289 Faust as DSP instead of using a message passing system in Max and PureData hosts
290 environments is better in terms of the computational power needed to achieve the signal
291 processing.

292 With these results, MUTANT started to develop a new version of the Antescofo scheduling
293 algorithm optimized exclusively for the UDOO platform. The results of this article showed
294 that the UDOO platform is powerful enough to run a standalone version of Antescofo, but
295 the scheduling algorithms were not optimal for this platform. With the integration of the
296 new scheduling algorithm and the DSP tools in Antescofo, the potential of the software
297 grows and reaches a new platform, which also allows, for its size and potentiality, to
298 preserve the interactive musical pieces through time.

299

300 **4. Conclusions**

301

302 As a conclusion of the work presented in this article, it can be said that the use of modular
303 DSP tools embedded in Antescofo optimizes the performance of the signal processing in
304 terms of computational power needed in about 46%. This result can be used as a part of the
305 optimization for the version that MUTANT team is going to develop for the UDOO
306 platform. This improvement will allow Antescofo software to run in a standalone version,
307 creating this way a new developing platform for composers of interactive music.

308 Also, the use of modular DSP tools in Antescofo, allows the composer to create his own
309 effects and sounds, having also the freedom to change the link network dynamically during
310 performance. This can be done inside the very Antescofo score and it integrates pre-
311 existing popular DSP tools such as Faust, Fluidsynth, Csound and Supercollider. These
312 DSP tools creates the possibility to the composer to write sophisticated effects in real time,
313 but most important: maintaining the same language and the same score.

314 Another benefit of using this new version of Antescofo is the fact that it will allow the
315 musical community and contemporary musicians to preserve their pieces through time. The
316 UDOO platform is small, independent and powerful enough to be loaded with a standalone
317 version of Antescofo and the scores and be saved for years. It creates the unique possibility
318 to play the pieces just plugging in a microphone and playing the human part of the score.

319 Finally, the profiling tests of the performance of the different Faust effects in the Anthèmes
320 2 score in a Mac environment shows a first approach to the optimization of the UDOO
321 platform and the Antescofo software. However, when the other DSP tools were
322 implemented and the re-scheduling process of Antescofo for UDOO is done, more profiling
323 tests must be done to show more accurate results on the UDOO platform optimization.

324

325 **Acknowledgments**

326

327 I would like to thank to all the people that in one way or another contributed to earn the
328 scholarship that allowed me to go to the IRCAM, for this internship. In particular to my
329 family, for all the support; Pierre Donat-Bouillud, for all the help and patience; everyone in
330 MUTANT Team, for the amazing experience; and all my friends.

331

332 **References**

333

334 **1.** CONT, Arshia. GIAVITTO, Jean-Louis. ECHEVESTE, José. 2015. Antescofo, a not-so-
335 short introduction to version 0.x. Available at
336 <http://support.ircam.fr/docs/Antescofo/AntescofoReference.pdf>

337 **2.** PUCKETTE, Miller. Pd Documentation. Available at
338 <http://puredata.info/docs/manuals/pd>

339 **3.** GAUDRAIN, Etienne. ORLAREY, Yann. 2003. A Faust Tutorial. Available at
340 http://faust.grame.fr/images/faust-doc/Faust_tutorial.pdf

341 **4.** FRIGO, Matteo. JOHNSON, Steven. 2012. FFTW. Available at
342 <http://www.fftw.org/fftw3.pdf>

343

344 **Glossary**

345

346 Machine Listening: Technique to obtain meaningful information from audio signals using
347 software and hardware.

348 Pitch Tracking: Technique to obtain the pitch or the value in frequency or tones of the
349 fundamental sound reproduced at every moment in an audio signal.

350 Beat Tracking: Technique to obtain the tempo in beats per minute at every moment in an
351 audio signal.

352 Score: The file that contains all the musical events and actions of a musical piece.

353 Patch: A sub-program used in software like PureData or Max MSP.

354 DSP: Digital signal processor.

355 System resources: Any physical or virtual component of limited availability within a
356 computer system.

357 Scheduling: Method by which work specified by some means is assigned to resources that
358 complete the work.

359 Click: Sound produced by a sound card when a process is not able to be processed in real
360 time.

361

362 **Scientific principle**

363

364 The scientific principle used for this article is based on increasing efficiency for both,
365 resource use and decrease of the execution time of methods and functions used to process
366 the sounds and effects. The change in the DSP chain architecture of Antescofo, from a
367 software in which the signals were processed in external patches by a message passing
368 system, to the integration of specialized modular DSP tools within the software allowed a
369 substantial increase in the runtime efficiency. The main argument for this is that by
370 integrating modular DSP tools, the architecture changed and the system became more
371 efficient in compiling, comparing to the previous message passing architecture. The use of
372 Faust and its on-the-fly compiler, allows the functions to be compiled at the time the score
373 is loaded into the system, and not at runtime. This increases the efficiency considerably.

374

375

376