First steps toward embedding real-time audio computing in Antescofo

Nicolás Schmidt Gubbins, Arshia Cont, Jean-Louis Giavitto

Journal I3
Investigación, Interdisciplina, Innovación

Journal de Investigación de Pregrado

FIRST STEPS TOWARDS EMBEDDING REAL-TIME AUDIO COMPUTING IN ANTESCOFO

Nicolás Schmidt Gubbins[1], Arshia Cont[2], Jean-Louis Giavitto[3]

[1] Pontificia Universidad Católica de Chile. Escuela de Ingeniería, Departamento de Ciencia de la Computación. 6to año. nschmid1@uc.cl
[2] IRCAM UMR STMS. INRIA - MUTANT TEam-project. Arshia.Cont@ircam.fr
[3] IRCAM UMR STMS. CNRS. INRIA - MUTANT TEam-project. Jean-Louis.Giavitto@ircam.fr

Abstract

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

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

Key words: Antescofo, UDOO, Faust, Real-time Audio Interaction, Interactive Music.

1. Introduction

a) Antescofo and Interactive Music.

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

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

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

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

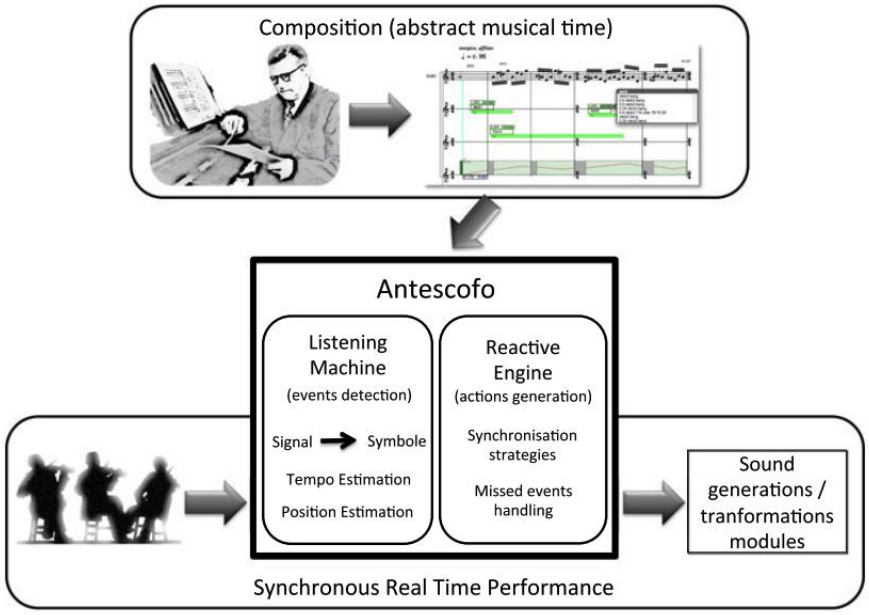

Figure 1.- Schema of Antescofo working system

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

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

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

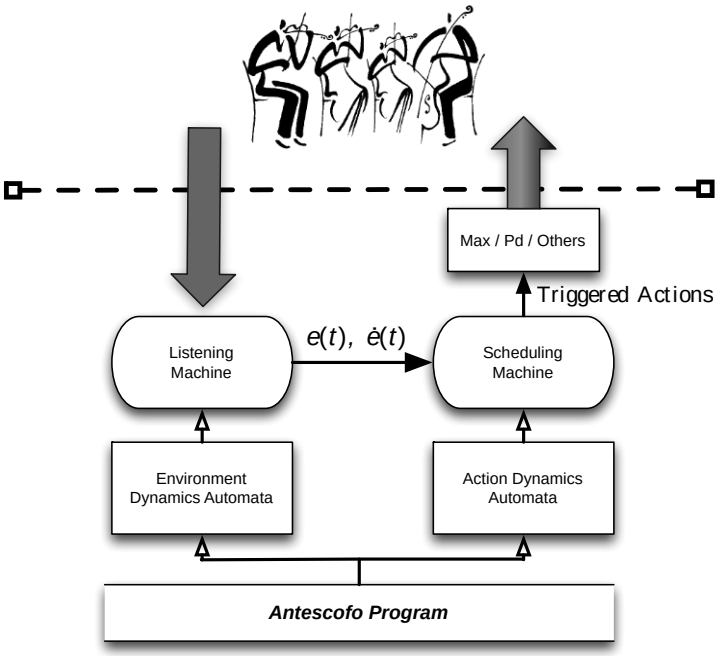

Figure 2.- Antescofo Synchronization

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

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

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

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

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

2. Metodology

a) Modular DSP Tools.

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

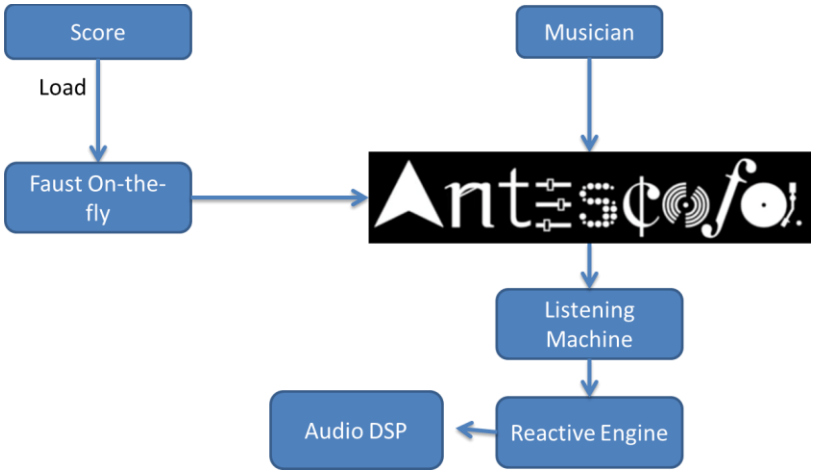

Figure 3.- Antescofo – Faust DSP chain

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

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

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

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

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

b) Preserving art Through Time.

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

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

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

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

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

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

c) Anthèmes 2 And Faust

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

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

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

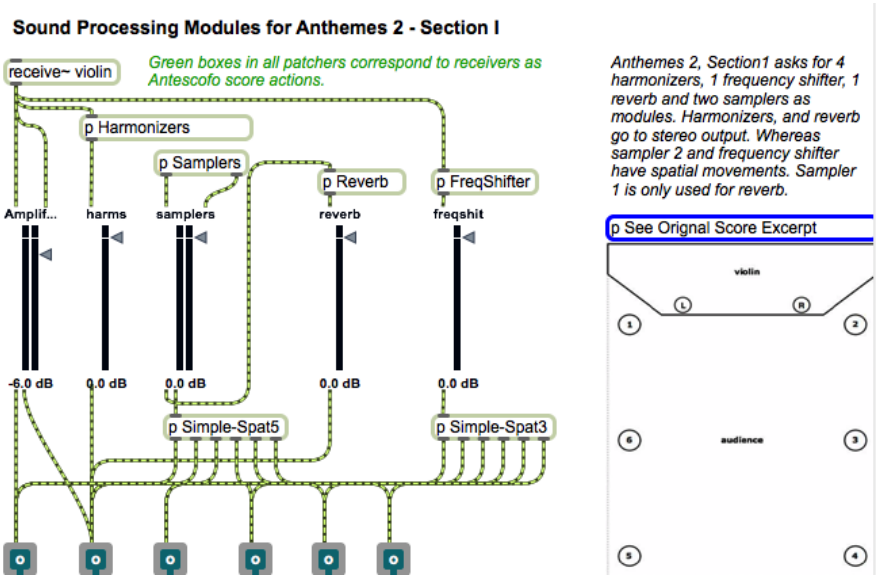

Figure 4.- Anthèmes 2 effects patch in Max MSP.

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

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

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

3. Results and Discussion.

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

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

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

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

After this change, the respective time of computing the dspchain_tick function were 5900ms in the case of the Anthèmes 2 score using the effects in external patches, and 3150ms in the case of the Antescofo-Faust version of the score. This number represents the sum of the time from all the times that the "dspchain_tick" was called during the non-real-time performance. This numbers represents an improvement of the performance of 46%, which is significant thinking on the improvement of the UDOO platform. As the UDOO computational resources are way fewer than the Mac computer used on this tests, we can say that the improvement of 46.5% will allow Antescofo to work on the mini - computer much faster and with better performance.

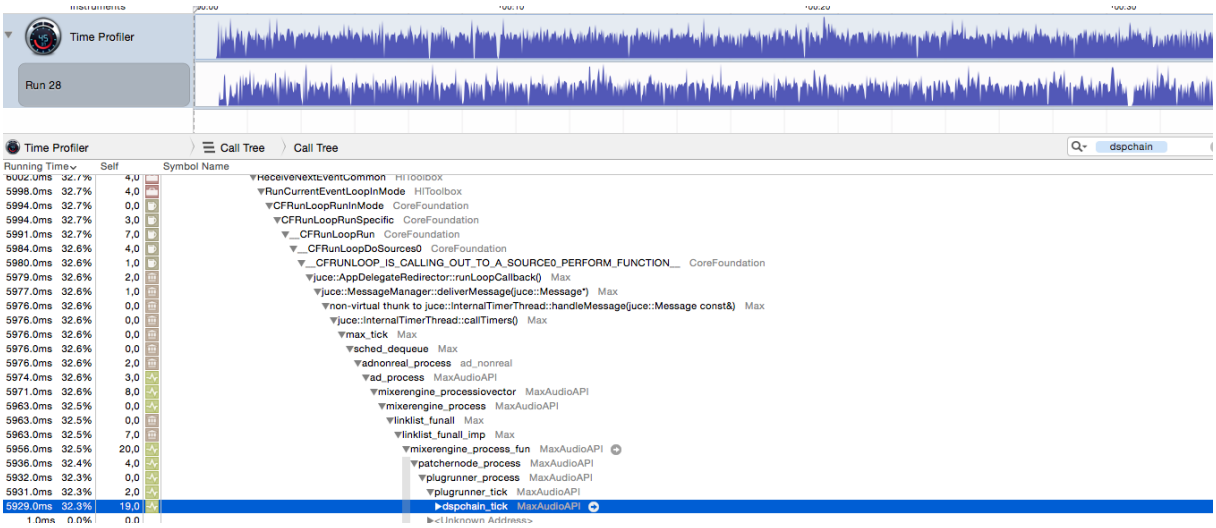

Figure 5.- Time profiler for the Anthèmes 2 score without Faust

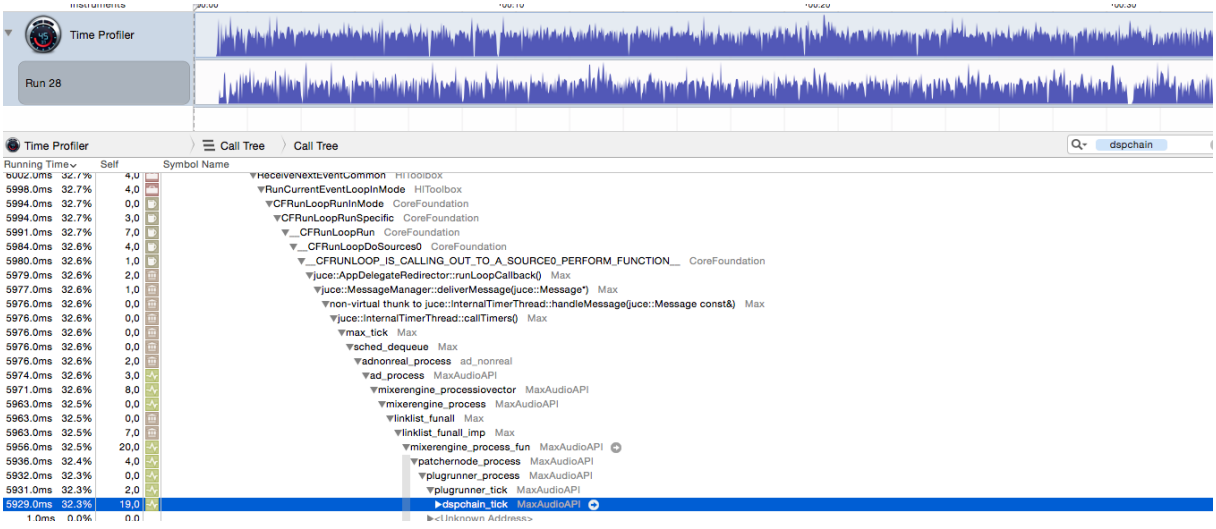

Figure 6.- Time profiler for the Anthèmes 2 using Faust

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

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

4. Conclusions

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

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

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

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

Acknowledgments

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

References

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

2. PUCKETTE, Miller. Pd Documentation. Available at http://puredata.info/docs/manuals/pd

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

4. FRIGO, Matteo. JOHNSON, Steven. 2012. FFTW. Available at http://www.fftw.org/fftw3.pdf

Glossary

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

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

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

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

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

DSP: Digital signal processor.

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

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

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

Scientific principle

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