SMC18
sonic crossings

15th Sound & Music Computing Conference

Sonic Crossings

4-7 July
Limassol, Cyprus

PAPER PROCEEDINGS

Proceedings of the 15th Sound and Music Computing Conference (SMC 2018), Limassol Cyprus, 2018

Editors

Dr. Anastasia Georgaki and Dr. Areti Andreopoulou

ISBN

978-9963-697-30-4

Towards Flexible Audio Processing

Thilo Koch, Marcelo Queiroz
Computer Science Department
Institute of Mathematics ans Statistics
University of São Paulo
tiko@ime.usp.br, mqz@ime.usp.br

ABSTRACT

In this paper we introduce a flexibilization mechanism for audio processing that allows the dynamic control of a trade-off between computational cost and quality of the output. This mechanism can be used to adapt signal processing algorithms to varying CPU load conditions in critical real-time applications that require uninterrupted delivery of audio blocks at a fixed rate in the presence of processor overload. Flexibilization takes place through the control of parameters in elementary signal processing modules that are combined to form more complex processing chains. We discuss several instances of audio processing tasks that can be flexibilized along with a discussion of their flexibilization parameters and corresponding impact in costs and quality, and propose an implementation framework for plugin development that provides the necessary mechanisms for control of the cost-quality trade-off.

1. INTRODUCTION

Many sound processing applications, such as digital audio workstations (DAW), spatialization environments and sound synthesis frameworks, use fixed values for internal algorithmic parameters, such as audio block size, precision and sample rate. This ensures that their algorithms have fixed computing costs for processing each audio block. In real-time mode these applications typically either underuse or overload their computational platform most of the time. Underuse implies that some configurations might be improved, such as increasing sample rate or sample size, or tweaking other internal parameters that would lead to improved audio quality. Overload often lead to unsatisfactory results, such as audio artifacts (clicks and noise) or worse (segmentation faults or system halt).

In this paper we present a methodological approach towards flexibilization of sound processing computing costs, in order to adapt these costs to the available computational resources of the underlying platform. A key element of this methodology is the definition of a trade-off between computing costs and user-experienced sound quality, which is embedded into time-varying *flexibilization parameters* of the sound processing algorithms. These parameters allow the sound processing algorithm to dynamically tilt the above trade-off between computing costs and sound quality, in an attempt to use computational resources more efficiently. This means that more quality might be obtained whenever possible, and less computing costs might be demanded whenever needed. The same flexibilization parameters could also be used statically to adapt an application originally developed for a specific computational platform (e.g. a high-end desktop computer) to another one with different computational power (e.g. a smartphone). Another possibility for trade-off exploration would be to trade quality for better energy consumption (of the involved computations), which is a great concern in mobile applications.

It should be obvious that not every sound processing application could be subjected to such a flexible approach. In particular, critical audio applications (e.g. in professional recording studios) that do not allow sound quality to be lowered under any circumstances will not be considered as suitable candidates for this methodology. Better examples of suitable sound processing applications will be readily found in the contexts of experimental music, mobile- and network-based music performances, and circuit hacking or low-fi audio artistic scenes.

1.1 Related Work

Trade-offs are a recurring theme in computer science, and are found under many guises, usually trying to replace a short-supplied or expensive resource with another abundant or cheap one. Well-known examples include data compression (computer time pays for storage space), redundant data transmission (network band usage pays for transmission reliability or quality) and numerical algorithms that produce approximate solutions (more iterations reduce numerical error). In the audio compressing domain, perceptual coding [1] has shown that a trade-off between sound quality and data size is feasible in many applications.

The term *flexible computation* [2] refers to real-time applications that are designed to dynamically control the trade-off between some measure of quality of the results (e.g. numerical error) and the resources (e.g. time, memory, etc.) they require to produce these results. Flexibilization is achieved by partitioning processing tasks into mandatory and optional components, where the latter might be partially executed or even left aside entirely according to individual timing constraints.

Feng et al. [3] extend this approach to account for error propagation in such systems, where imprecise output

of one processing task contaminates the inputs for subsequent tasks. These authors do not consider individual timing constraints for each task, but rather end-to-end timing constraints, which make their model better applicable to audio processing where typically there are several cascading processing steps (tasks) applied to a single audio block.

Fouad et al. [4] applied scheduling algorithms within an experimental audio server, adapting the audio's sampling rate to load conditions. Tsingos [5] proposed a perceptual-based and scalable approach for filtering and mixing a large number of audio signals. In his approach signals are processed in the frequency domain, where the number of spectral bins processed is optimized based on criteria referring to audibility and importance of each bin in the final mix. Tsingos shows that this approach allows certain applications to speed up significantly; nevertheless, in order to adapt this approach to existing code, radical changes are required, especially in audio applications that process the signal in the time domain, where most filters and processing elements would have to be completely rewritten.

Another approach to parametrize computing costs for mixing sounds was presented by Kleczkowski et al. [6]. The proposal of selective mixing is based on an analysis in the frequency domain to determine which input sources could be discarded, depending on the relative energies of all contributing sources. They concluded that, depending on the thresholds used, perceptual quality was not affected, although listeners seemed to perceive selectively mixed sounds as less noisy.

Sound spatialization applications, which typically use many virtual sound sources and many output channels (e.g. wave field synthesis) with high computing costs, could evidently benefit from a flexible computing approach, especially in dynamic auditory scenes. Herder [7] and Tsingos et al. [8] proposed several methods for dynamic resource management which are based on a level of detail approach. Both use psychoacoustic measures such as loudness and perceptual salience to dynamically adapt computing costs. Their proposals concern specific applications, and presuppose the knowledge of the complete data flow pathways, which is not always the case in interactive sound processing applications; on the other hand, spatialization plug-ins are used as subtasks for many audio processing applications.

The methodology proposed in the sequel naturally requires that existing code should be adapted. For the adaptation of existing software we first consider simple and isolated audio processing steps, in order to analyze them and look for potential flexibilization parameters. Later on we consider the application audio processing graph to look for flexibilization parameters referring to process interconnections and data flow pathways. These case studies are considered a preliminary step towards defining a protocol for flexible audio applications that would allow seamless integration between protocol-abiding applications, and minimal intrusion in order to adapt existing code.

1.2 Limits

Applicability Various scenarios in audio processing are not compatible with our approach. This concerns applications which do not permit variable quality of the processing result because their quality is predefined and fixed, as is the case of commercial audio CD production or high-fidelity systems.

Feasibility Our approach to adapt existing applications and plug-ins in the least invasive way limits certain flexibilization options. This is the case, for example, when the framework used in an application does not permit sample-rate changes or adaptation of scheduling.

Quality limits The trade-off between quality and computing costs reaches its limits when the quality degradation becomes perceptually unacceptable and, at the same time, the computational overload can not be avoided. This is a limiting but intrinsic characteristic of the flexible computation approach.

2. AUDIO PROCESSING STEPS / ELEMENTS

In this section we discuss selected examples of synthesis algorithms which are commonly used as building blocks in audio processing software. In these algorithms we aim at identifying parameters which affect their computational costs, and at analyzing the theoretical impact of these parameters on the resulting audio quality. This relationship between computational cost and resulting quality represents the main trade-off upon which we would like to have control.

2.1 FIR filters

FIR filters are used ubiquitously in audio processing applications, from simple low-pass filtering to auralization and spatialization.

FIR filtering is defined as the application of a convolution equation

$$y_n = \sum_{i=0}^{M-1} a_i x_{n-i}, \tag{2.1}$$

where x is the input, y the output and a is the coefficient vector which characterizes the filter.

For the computation of FIR filters there are two main approaches: 1. Convolution in the time domain; or 2. Multiplication in the frequency domain (including forward and backward Fourier Transforms).

The first approach is used for filter with few taps (nonzero coefficients), whereas the second is more efficient for larger-order filters but requires a delay corresponding to the filter size (which may be an issue for low-latency systems).

2.1.1 FIR filtering in the time domain

The execution time for convolution in the time domain, derived directly from the equation above, is

$$t \propto MN, \tag{2.2}$$

where N is the size of the input vector and M the size of the filter vector.

Among several alternatives for thinning digital filters, the proposals from Baran et al. [9], Wu et al. [10] and Ye et

al. [11] are particularly useful for our flexibilization purposes and do not require cascading FIR design. The authors divide the thinning process in two phases, 1. select a tap to zero; and 2. re-optimize remaining taps. The re-optimization step consists in the definition of an objective criterion (e.g. harmonic distortion) and also the filter design constraints (e.g. linear phase). It is useful to experiment with different thinning approaches, especially because we have to account not only for the convolution costs but also the costs of the re-optimizing step (2).

2.1.2 FIR filtering in the frequency domain

The fast convolution algorithm transforms the input and the coefficient vectors into the frequency domain, multiplies the spectra element-wise and transforms the result back to time domain (using circular convolution with zero-padding and overlap-add to compensate for temporal aliasing). The execution time when using the Fast Fourier Transform (FFT) is

$$t \propto N \log M, \tag{2.3}$$

where M is the filter length and N the number of samples to process.

Parametrizing the filter length for this algorithm does not lead to expressive computational savings since the costs depend logarithmically on this quantity, but the number of samples to process can be flexibilized in the context of processing chains (see section 3.1).

Another approach was proposed by Queiroz et al. [12] which is based on the approximation of FIR filters with low-order IIR filters in the time and frequency domain. It is especially useful for large FIR filters as their occur as room impulse responses or Head-Related Transfer Functions (HRTFs).

2.2 Room simulation

Most of the room simulation algorithms are based on geometrical acoustics and use mathematical models to describe the geometry of the room, positions of sound sources and listeners, sound propagation, etc. Rooms are treated as LTI systems with acoustic behavior characterized by their impulse responses [13], according to the equation below:

$$y(t) = \int_{-\infty}^{\infty} x(\tau) h(t-\tau) \mathrm{d}\tau. \tag{2.4}$$

where $x(t)$ is a source audio signal, $h(t)$ is an impulse response and $y(n)$ is the resulting signal. This is again a convolution filter, where the impulse response depends not only on room dimensions and acoustic configuration (such as revesting materials, pieces of furniture, etc) but also on sound source and listener positions. Accordingly, room simulation algorithms have to calculate new impulse responses and new audio convolutions for every change in sound source or listener positions.

2.2.1 Room models

Room models are derived from CAD representations and have to consider the simulation goals when modeling acoustic configurations, since every additional detail increases the complexity of the simulation and its computational costs. Not all details of a room are useful for its acoustic characterization, since geometrical acoustics is based on absorbing and scattering geometrical elements which interact with soundwaves only when geometrical elements and sound wavelengths have the same order of magnitude.

One parametrization approach would correspond to work with different room models with varying degrees of details. Although certain algorithms require the transformation of CAD room models into other representations (e.g. *meshes* [14] and *binary space partitioning [15]*) to decrease computing costs, the level of detail is critical for execution time. Thakur et al. show that there are many open issues in automatic CAD simplification [16], but it is nevertheless possible to manually derive different complex models beforehand and later select the appropriate model for real-time processing, according to load conditions of the computing platform. There is however no closed form expression to relate the complexity of the model to computing costs or simulation quality.

The most relevant geometrical algorithms for calculating the room impulse response from a given room model are the image source model and ray tracing. Many implementations use a hybrid approach combining both algorithms.

2.2.2 Image source model

The image source model derives virtual sources by iteratively mirroring sound sources through each reflective surface. The impulse response is then obtained considering the distances of the mirrored sources to the listener and also the attenuations the sound ray undergoes after each reflection. As reflections occur in 3-dimensional space, computing costs will increase cubically with the length of the desired impulse response [17]:

$$t \propto t_{ir}^3. \tag{2.5}$$

2.2.3 Ray tracing

Ray tracing is a stochastic method based on simulating the trajectory of a large number of independent sound rays, which propagate linearly away from the sound source. For each ray, reflections, absorption and scattering are calculated until either the ray is absorbed completely or the maximum simulation time is reached. Vorländer [13] determines the execution time of the algorithm as

$$t \propto N t_{ir} \bar{n} \tau \tag{2.6}$$

where N is the number of simulated rays, t_{ir} the size of the desired impulse response, $\bar{n}$ the mean number of reflections. The necessary time for reflection and detection test is $\tau = t_w \log_2 n_w + t_d \log_2 n_d$ when using *binary space partitioning*, where n_w and n_d are the number of reflection and detections tests (point-in-polygon) and t_w and t_d their executions times.

There are several options to parametrize room simulation algorithms. Besides controlling the complexity of the room model, the most critical parameter to affect computing costs is the size of the desired impulse response. Any flexibilization of this parameter has to consider its strong

influence on psychoacoustic properties such as intelligibility, spacial impression, and envelopment. For hybrid algorithms, which use the image source model for the first part of the impulse response and ray tracing for the later part, it is possible to reduce the very time-consuming first part at the cost of calculating a longer ray tracing part which is less costly. Additionally, it is possible to control the number of simulated rays, which affects the resolution and robustness of the obtained impulse response, but not the precision of the result.

2.3 Spatialization

2.3.1 Wave field synthesis

Wave field synthesis was formulated by Berkhout [18] in 1988 and was employed in many installations once the necessary computing power for real-time processing was available (from the 1990s on, see for example Baalman et al. [19]). This method is based on the interference of a large number of loudspeakers to synthesize a sound wave field in a certain environment. The signal attenuation and delay has to be computed separately for every loudspeaker, leading to a huge amount of computations.

The execution time for WFS is

$$t \propto napc_{a,s,f} \tag{2.7}$$

where n is the number of samples, a the number of loudspeakers, p the number of virtual sources and $c_{a,s,f}$ represents the costs for attenuation, superposition and filtering for each loudspeaker. Herder [7] proposes an approach to reduce the number of virtual sources by estimating each source's audibility in the final mix, thus deciding whether it is rendered at all. Furthermore, distant virtual sources are grouped into clusters dependent on their localization and distance, that are going to be processed as a single virtual source.

Spatial aliasing, which produces spatial artifacts, comes into effect only for virtual sources with spectral content above $f_{al} = c/2\Delta x$, where c is the speed of sound and Δx the distance between neighboring loudspeakers. Correspondingly, virtual sources with, for example, spectral content below $f_{al}/2$ have to be computed only for every other loudspeaker without introducing spatial artifacts. Therefore, it is possible to parametrize the number of loudspeakers in the execution time equation above by dynamically deciding which source has to be rendered for which loudspeaker.

Although the flexibilization approaches proposed by Tsingos [5] are not applicable as a whole, because they require completely rewriting existing WFS software, it should be possible to use auditory culling and clustering, as proposed by earlier papers from Tsingos et al. [8] and Herder [7], to parametrize the number of virtual sources that have to be rendered.

2.3.2 Binaural synthesis

In binaural synthesis, complete acoustic scenes are rendered for two channels, and all virtual sources have to be convolved with an impulse response that corresponds to the position of the source and listener (room characteristics, LTI system) and also depends on the physical characteristics of the listener (Head related transfer function (HRTF)).

The execution time is

$$t \propto npc_{a,s,f}, \tag{2.8}$$

where n is the number of samples, p the number of virtual sources and $c_{a,s,f}$ are the constant costs to apply attenuation, source superposition and HRTF filtering to one sample. The number of virtual sources can be parametrized by adapting auditory culling and clustering sources. In moving scenes, extra costs appear from mixing signals with different impulse responses for different localizations or from applying transition functions. The angular resolution defines the rate in which impulse responses switch when the incident angle changes, and can therefore be parametrized to control switching and mixing costs.

3. FLEXIBLE AUDIO PROCESSING

3.1 Processing chain

Most audio applications combine several processing steps to produce the desired output; therefore, they can be described with directed graphs, where vertices represent each processing step, and arcs the digital audio data stream. This approach facilitates the investigation of flexibilization parameters both for subgraphs or for the application as a whole. Such parameters can be the sampling rate, the number of (virtual) sources or the resolution (bandwidth) of the audio stream. This approach enables the investigation of parametrizations that would not have a cost benefit when applied to a single processing step, but whose advantage would appear in longer processing chains.

Sample rate changes, for example, have their own extra computing costs, which in many cases can be amortized when the converted audio is subject to more than one processing step. An analysis of this layer permits to identify processing subgraphs with band-limited signal flux, where processing at lower sampling rates is sufficient and introduces only minimal errors. As the computational costs for all processing elements depend heavily on the sample rate, multi-rate implementation of signal processing elements clearly leads to flexible computing costs [20].

3.2 Real time constraints and flexible management

The approach we propose uses the graph-analytical view of the DSP processing chain to discover the flexibilization potential of each processing step and of the chain as a whole, which enables the control of the overall processing costs. For this approach to be effective, such a flexible system has to obtain information about platform conditions and about the running plugins and their specific behavior, in order to generate a priority queue of flexibilization steps. When system load conditions require, such steps can be applied in constant time, and since they can be reversed, a flexible system can move along paths of different computational costs, making the audio processing adaptive and flexible.

The fundamental relation that a decision-making instance has to uphold is given by the inequality below,

$$\sum_{k=1}^{K} \phi_k(n) + o(n) \leq \Phi(n), \tag{3.9}$$

where K is the number of processing elements, each with processing time ϕ_k, $o(n)$ is an additional overall processing time for the flexible system, and $\Phi(n)$ represents the time budget allowed for the computation of n samples. A *flexible* audio system has to contend with situations in which this inequality does not hold. Reducing the time conceded to each processing element can reduce overall costs, but will introduce differences or imprecisions in the resulting audio signal.

To mitigate this problem, the so called *flexible manager* considers these differences and tries to minimize them. With a priori knowledge on how parameter configurations relate to these imprecisions and to computing costs, paths with the best quality-cost-ratio can be found and used during application execution. In real time audio processing applications without this a priori knowledge, the decisions to adapt costs to resource availability have to be based on ad hoc data and heuristic rules.

4. A FRAMEWORK FOR FLEXIBLE AUDIO PROCESSING

4.1 Flex protocol

We propose an architecture for experimentation and prototyping which consists of a server (flexible audio manager) and clients for each processing element. The server works as central monitoring and controlling instance which communicates with clients through a protocol that allows the server to request static and dynamic data from clients. Static element specific data enclose information about control parameter options, their range, and mappings describing computing costs as a function of parameter values. Dynamic data provided by the clients upon request are imprecision measures, actual parameter values and execution time statistics.

Moreover, control messages allow the server to instruct clients to change their processing parameters. As these parameters control the computing costs of the elements, the established mechanism can be understood as an implementation of the *multiple version method* described by Liu [21]: each parameter set represents another version. Correspondingly, the flexible audio manager does not interfere with task scheduling already established by the audio framework or by the operating system, but controls the execution time of each processing step through parametrization.

Collected information from the clients allows the server on one hand to make decisions about client parameters to meet the above time budget inequality constraint and, on the other hand, to minimize audible effects. Clients in this architecture establish an interface between the flexible audio manager and the audio processing elements (or *plugins* in most contemporary software parlance). The necessary functions for communication, data collection and control are added to existing plugins through libraries. Code changes inside the plugins are necessary for the parametrization and parameter transition mechanisms.

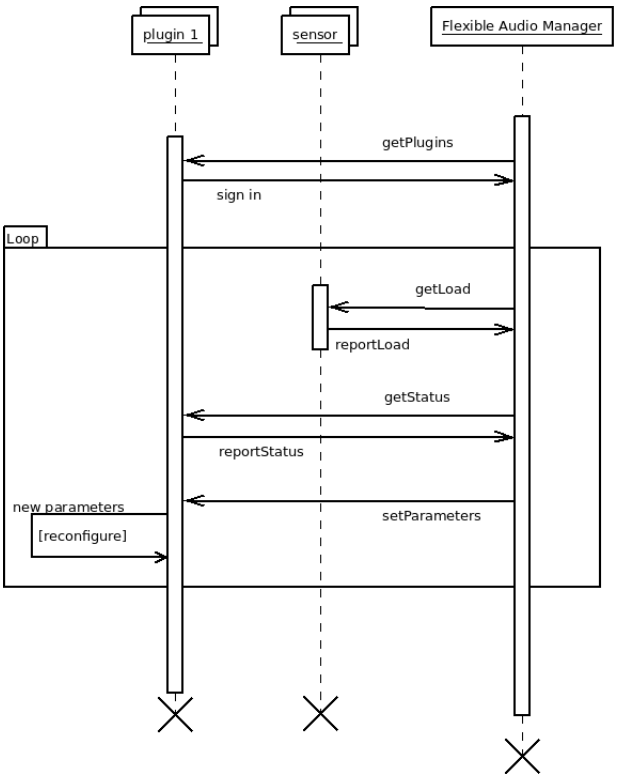

Figure 1. UML2 Sequence diagram of the interaction between the components of a flexibilized audio processing application

4.2 Plugins

To adapt an existing plugin to our flexible approach, first the potential for a cost-imprecision trade-off must be investigated. To parametrize the plugin, the relations between parameters and execution time costs and imprecision measures have to be described as mappings/functions.

The adaptation of software to our proposed flexibilization approach depends fundamentally on how easily and how exactly existing code can be adapted. Consequently, the necessary code changes should be as small and less intrusive as possible. As depicted in Figure 1, each plugin has to implement 3 functions:

- a request response sign in - to inform the manager of the flexible plugin availability and to send static descriptive data about its characteristics and properties;
- a request response to send dynamic data about run time statistics to the manager;
- a request response to change processing parameters (reconfiguration).

The *getPlugins* function identifies all flexible plugins available, i.e. all plugins abiding to the Flex protocol defined. The *signIn* function, issued by a flexible plugin, allows the

manager to control the plugin through scheduling actions within its main loop (the box in Figure 1) and also to learn from the plugin how its flexibilization parameters affect both the quality and the computational processing costs as a function of the parameters. Such static description is supposed to have been mapped through previous experiments with the plugin, and it is used by the manager as a model to define a preliminary policy on how to dynamically adjust this plugin's settings.

Inside the loop in Figure 1 there is also a two-way interaction (*getLoad/reportLoad*) with a sensor, who is responsible for measuring the load of the processing hardware, allowing the manager to estimate the computational headroom for adding flexible plugins to the DSP chain and increasing the quality of the produced signal without degrading quality. This measurement is made once per DSP processing block.

The *getStatus/reportStatus* functions will establish the current computational cost of the plugin and its estimated quality, which are used to update the model of this plugin's behavior (in terms of its empirical cost-quality trade-off), and to decide on possible reconfigurations, performed by the *setParameters* function, given the current system load. This last function does not return data to the manager, but requires a transition of the plugin towards the new set of flexible parameters, while avoiding audible artifacts (this is a responsability of the plugin). The manager can see these parameter changes with its next *getStatus* request.

4.3 Application: Flexible Mixer

To experiment with a reference implementation of the described protocol, we developed the *Flexmix* prototype application; a virtual mixer which processes a number of sound sources and mixes them down into a mono mix in real time. Each audio source passes through various processing steps: pre-amplification, an equalizer and a spatialization plugin, while after the mix down the audio stream passes through a 10-band-equalizer and a compressor/limiter. The prototype serves as a proof of concept of some aspects of our approach and is inspired by typical audio processing workflow for many musicians and audio professionals.

Flexmix is implemented using the Gstreamer [1] library as audio framework. Gstreamer is a library for constructing graphs of media-handling components that offers many characteristics which facilitate the implementation of a flexible system, such as a communication channel between plugins and application, and foremost a dynamic pipeline autoreconfiguration mechanism. There are many native gstreamer plugins, and also other compatible audio plugin formats like LV2, which are available for immediate use, and many of which are open-source, and thus modifiable. Many of the already existing audio plugins have properties which relate to trade-offs between execution time costs and imprecision, which can be easily adapted for the flexmix application. Furthermore, the Gstreamer library already provides a mechanism to measure and propagate messages about quality-of-service events. The downside of these otherwise convenient properties of Gstreamer is its relative complexity for implementation and testing in certain scenarios.

Figure 2 shows the Flexmix data flow graph with computing time measurements for each element and size of audio data flow for one execution of a sample run. Each source is obtained from FLAC audio files (*filesrcN*) which passes through a number of non-flexible Gstreamer elements (*GstFlacParse*, *GstFlacDec* and *GstAudioConvert*) before it enters the flexible DSP chain, represented by a shaded rectangle. In this flexible chain, each audio channel is separately processed by the flexible Gstreamer plugins *GstAudioAmplify* (a pre-amplifier), *GstIIREqualizer* (a 3-band equalizer based on recursive filters) and *GstAudioFIRFilter* (a spatialization plugin). After that, all channels are mixed down to a single channel by *GstAudioMixer* (a flexible mixing plugin), and this mono signal is further processed by a flexible 10-band equalizer (*GstIIREqualizer*) and a flexible compressor/limiter (*GstAudioDynamic*; the remaining two elements in the chain are format modifiers).

In this experiment, the main qualitative concern was to ensure that all elements would produce an uninterrupted data flow (i.e. no x-runs in Linux audio jargon), despite variations on CPU load. This was so designed in order to have an objectively measurable way of verifying that the system policy and parameter reconfigurations was working within prescribed limits. The Flexmix application reacts on CPU stress and adapts accordingly decreasing computing costs and ensuring audio stream processing without glitches in many settings. This allows testing several components of our approach, especially through gathering of statistics to develop heuristics for trade-off decisions. Imprecision is currently measured as frequency weighted spectral distortion, a design choice which aims at allowing objective experiments to be carried out before subjecting the system to human evaluation.

In future experiments with this application we aim to acquire specific experimental knowledge about plugin implementations and their parametrizability, and to gain more insight into the following questions:

- what is the implementational effort and what are the computational costs for the manager;
- what type of information is needed by the manager to reasonably control the overall processing.

A first step towards a fully subjective evaluation of the system is to work with automatic perceptual models [2] that could be inquired in a timely fashion and provide quality evaluations for each DSP block (something humanly infeasible). After these new experiments, a more realistic model of obtaining human feedback may be designed, e.g. by selecting portions of audio output for which the automatic evaluation is not very trustworthy and presenting them to human listeners for a detailed qualitative feedback.

[1] https://gstreamer.freedesktop.org/

[2] Such as PEAQ (Perceptual Evaluation of Audio Quality), a ITU-R standard described in `http://www.itu.int/rec/R-REC-BS.1387/en`

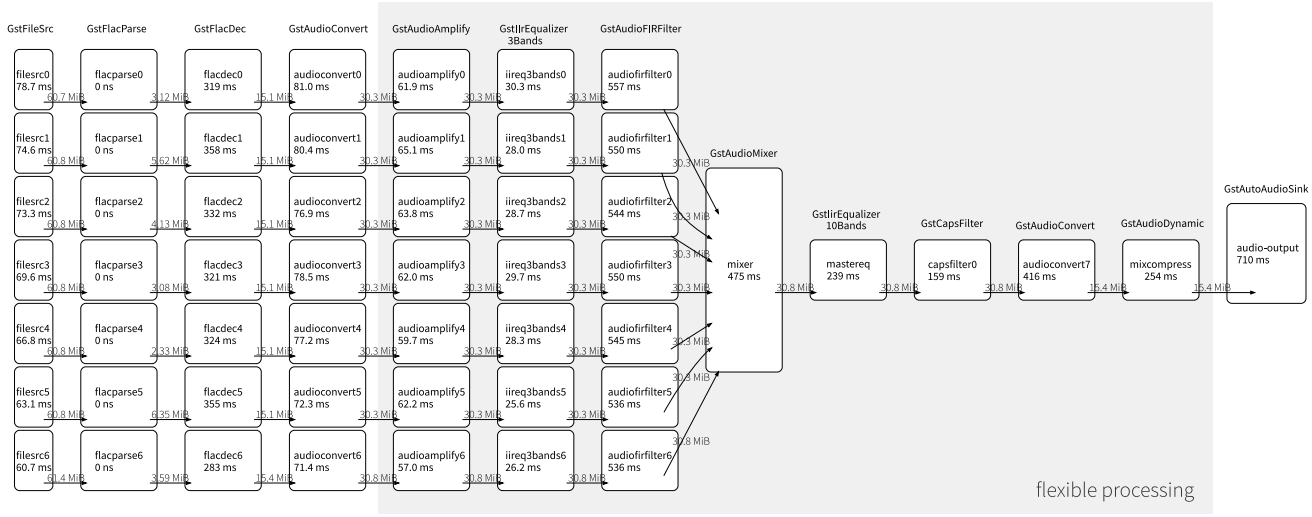

Figure 2. Pipelines and plugins of the flexmix application and measured execution times

5. CONCLUSION AND FUTURE WORK

This paper proposes a framework for flexible real time audio processing, which dynamically adapts computing costs of a DSP chain according to a varying CPU load, by managing a complex chain of plugins through communication channels, status reports and reconfiguration messages. Our parametrization approach is applicable to many common elementary processing steps (plugins) with different potential trade-off parameters, and also to the paths that connect such elements within a sound processing graph. In order to lower the implementation burden, we propose a modification on the communication and control mechanisms on existing audio processing software, instead of having to completely rewrite both plugins and DSP host.

The implemented prototype is currently being subjected to experiments in different scenarios, in order to explore the relationship between parameter settings and perceived quality and to experiment with different kind of imprecision measures. In order to ensure that flexible sound processing entails no or minimal impact on the audible result, we are exploring perceptual representation spaces in which audio signal modifications can be quantitatively assessed. We also intend to develop other parametrization strategies, especially for processing elements that appear in these practical scenarios.

Another focus of future work is to investigate error propagation through processing pipelines to discover practical patterns for controlling execution times through parameter settings with minimal impact on the audible result. One approach will be to use the auto tuning techniques [22] to explore valid parameter spaces and parameter settings with less imprecision.

Acknowledgments

During this work the first author received scholarship from CAPES, and the second author acknowledges the support of FAPESP grant 2018/09373-8 and CNPq grant 309645/2016-6.

6. REFERENCES

[1] T. Painter and A. Spanias, "Perceptual coding of digital audio," *Proceedings of the IEEE*, vol. 88, no. 4, pp. 451–515, 2000.

[2] L. K.-J., S. Natarajan, and J. W. S. Liu, "Imprecise results: Utilizing partial computations in real-time systems," in *Proceedings IEEE Eighth Real-Time Systems Symposium*, 1987, pp. 210–217, madrid, Spain.

[3] W.-C. Feng and J. Liu, "Algorithms for scheduling real-time tasks with input error and end-to-end deadlines," *Software Engineering, IEEE Transactions on*, vol. 23, no. 2, pp. 93–106, Feb 1997.

[4] H. Fouad, J. Hahn, and J. Ballas, "Perceptually based scheduling algorithms for real-time synthesis of complex sonic environments," *ICAD'97 Intl. Conf. Auditory Display*, pp. 77–82, 1997.

[5] N. Tsingos, "Scalable perceptual mixing and filtering of audio signals using an augmented spectral representation," in *Proceedings of the International Conference on Digital Audio Effects*, September 2005, madrid, Spain.

[6] P. Kleczkowski and M. Pluta, "Perceptual evaluation of the effect of threshold in selective mixing of sounds," *Acta Physica Polonica A*, vol. 125, pp. A–117–A–121, Apr 2014.

[7] J. Herder, "Optimization of sound spatialization resource management through clustering," *The Journal of Three Dimensional Images, 3D-Forum Society*, pp. 59–65, 1999.

[8] N. Tsingos, E. Gallo, and G. Drettakis, "Perceptual audio rendering of complex virtual environments," INRIA, REVES/INRIA Sophia-Antipolis, Tech. Rep. RR-4734, Feb 2003.

[9] T. Baran, D. Wei, and A. Oppenheim, "Linear programming algorithms for sparse filter design," *Signal Processing, IEEE Transactions on*, vol. 58, no. 3, pp. 1605–1617, Mar. 2010.

[10] C. WU, Y. ZHANG, Y. SHI, L. ZHAO, and M. XIN, "Sparse fir filter design using binary particle swarm optimization," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E97.A, no. 12, pp. 2653–2657, 2014.

[11] W. Ye and Y. J. Yu, "Greedy algorithm for the design of linear-phase fir filters with sparse coefficients," *Circuits Syst. Signal Process.*, vol. 35, no. 4, pp. 1427–1436, Apr. 2016.

[12] M. Queiroz and G. H. M. de Sousa, "Efficient binaural rendering of moving sound sources using hrtf interpolation," *Journal of New Music Research*, vol. 40, no. 3, pp. 239–252, 2011.

[13] M. Vorländer, *Auralization: Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality*, 1^a ed., ser. RWTHedition. Berlin Heidelberg: Springer-Verlag, 2008.

[14] B. De Moor, A. Nackaerts, and B. Schiettecatte, "Real-Time acoustics simulation using mesh-tracing," in *Proceedings of the International Computer Music Conference 2003*, 2003.

[15] D. Schröder, "Physically based real-time auralization of interactive virtual environments," Ph.D. dissertation, Berlin, 2011.

[16] A. Thakur, A. G. Banerjee, and S. K. Gupta, "A survey of cad model simplification techniques for physics-based simulation applications," *Computer Aided Design*, vol. 41, no. 2, pp. 65–80, Feb. 2009.

[17] L. Cremer, *Die wissenschaflichen Grundlagen der Raumakustik*, 1^a ed. Stuttgart: Hirzel-Verlag, 1948, vol. 1.

[18] A. Berkhout, "A holographic approach to acoustic control," *Journal of the Audio Engineering Society*, vol. 36, pp. 977–995, 1988.

[19] M. A. J. Baalman, T. Hohn, S. S., and T. Koch, "Renewed architecture of the swonder software for wave field synthesis on large scale systems," in *Linux Audio Conference*, Berlin, 2007, pp. 76–83.

[20] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing*, ser. Prentice-Hall Signal Processing Series. Upper Saddle River, NJ: Prentice-Hall, 1983.

[21] J. W. S. Liu, *Real-Time Systems*, 1^a ed., ser. Modern Acoustics and Signal Processing. Upper Saddle River, NJ: Prentice Hall PTR, 2000.

[22] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe, "Opentuner: An extensible framework for program autotuning," in *International Conference on Parallel Architectures and Compilation Techniques*, Edmonton, Canada, August 2014.