# PROOF OF CONCEPT OF A GENERIC TOOLKIT FOR SONIFICATION: THE SONIFICATION CELL IN OSSIA SCORE

*Maxime Poret*
Univ. Bordeaux, CNRS,
Bordeaux INP, LaBRI, UMR 5800,
F-33400 Talence, France
maxime.poret@labri.fr

*Jean-Michaël Celerier*
Concordia University
1515 St. Catherine St. West
Montreal, QC, Canada
jeanmichael.celerier@gmail.com

*Myriam Desainte-Catherine*
Univ. Bordeaux, CNRS,
Bordeaux INP, LaBRI, UMR 5800,
F-33400 Talence, France
myriam@labri.fr

*Catherine Semal*
Univ. Bordeaux, CNRS,
INCIA, UMR 5287,
F-33000 Bordeaux, France
catherine.semal@ensc.fr

## ABSTRACT

A popular topic in sonification research is the development of a complete, user-friendly tool for sonification creation. A study of existing attempts at such tools highlights what seems to be the main challenge of this endeavor: exhaustiveness with regards to the great diversity of approaches for auditory display of information. Most tools are designed to allow for selecting among a few typical modalities, but could not really be used to create any kind of sonification. In order to tackle this issue, we proposed a theoretical model of the sonification process, which is intended to take all of its properties as a data observation technique into account. The goal for this model is to be translated into a user interface and programming approach as part of a sonification toolkit. In the present paper, we report our work in creating a proof of concept of such a toolkit using the ossia musical sequencing environment, chosen for its proximity to our objectives in terms of user interaction and library of functionalities. This prototype was tested to recreate two of our previous data sonification works. Most of the specificities of these case studies could be recreated properly, though some of the planned features, notably for grain synthesis, are currently missing from the ossia environment. For our future works, we will consider that the current state of this proof of concept is sufficient to start studying the user experience of sonification designers interacting with the toolkit.

## 1. INTRODUCTION

The use of non-verbal sounds to convey information, or "sonification" as modality for displaying data has taken many forms over the years, depending on the types of tasks being carried out, the types of data being displayed or the target audiences for the displays. One of the oldest and main research topics in this field has been the development of a common software framework for data sonification: something Kramer et al. called a "sonification shell" in the 1999 *Sonification Report* [1]. Over the years, this research has led to multiple theoretical models and software toolkits proposals, which were intended to enable encapsulation of any and all sonification tasks. As far as we know, none has been widely adopted yet. In section 2, we review the works specifically designed for sonification, especially in light of the traditionally listed criteria for such tools [1]. Our focus in particular is on their flexibility, as one of our main concerns is the exhaustive inclusion of all sonification approaches in a common framework. In his recently-published doctoral thesis [2], the first author of the present paper proposed that a fully-inclusive toolkit for data sonification may be created by first describing an abstract model for all sonification programs. This model, dubbed "Sonification Cell" is described in section 3.

It is expected that an interactive software toolkit for sonification design may then be created based on the model. Following the proposed template, a user would be able to design any sonification as a diagram, in such a way that the drawing may then be translated into a real-time data-to-audio processor, or compiled e.g. as a plug-in or as a standalone program. In this paper, we report on a proof of concept for the implementation of such a toolkit. That proof of concept was built using elements of the ossia score environment [3], whose graphical layout and affordances for data acquisition, sound synthesis and parameter mapping are already relatively close to the ones envisioned for our purposes. This choice is explained in more detail in section 4. We perform evaluation by recreating two of our previously-designed sonifications in our proof-of-concept toolkit. In section 5, we highlight the differences in sonification approaches that those two case studies represent, before describing how they were recreated. Finally, we discuss in section 6 how this implementation of the model performs: it is rather encouraging overall as it includes most of the necessary affordances for recreating two different sonification projects in the same environment. Still, a few other affordances will need to be added in the future, and a broader sample of sonification approaches will have to be experimented upon in order to confirm the model's exhaustiveness.

## 2. RELATED WORKS

Our goal in this section is to highlight the main trends in sonification toolkit designs, as well as the underlying models of sonification they seem to be based upon. With this review, we aim to assess the features that appear to still be missing when cross-examined with the original criteria listed by the *Sonification Report* [1], and explain why – to our knowledge – none of those frameworks has been widely adopted for sonification design.

### 2.1. Portability

Portability is a criterion that requires the provided software to remain functional on most operating systems. Aside from a few anomalies (such as SonifYer [4], which was created exclusively for use on macOS), almost all the existing generic sonification toolkits properly satisfy this criterion. A popular solution is to program these tools with cross-platform systems like Java (Sonification Sandbox [5], xSonify [6]), SuperCollider (SonEnvir [7]) or Pure Data (Interactive Sonification Toolkit (ISonTK) [8]), with the Web also becoming an increasingly viable option in recent years (Web Sandbox [9], Rotator [10]).

### 2.2. Integrability

The toolkit should be able to handle various data sources and types, and should be synchronizable with – or plugged into – other display modules (e.g. visual). Most of the time, the toolkits are simply meant to directly showcase the sonifications being designed within their user interfaces, with no way of integrating the audio engine or mapping choices into other environments. Still, some examples of tools which reportedly allowed themselves to be integrated into other environments were Porsonify [11], Listen [12] and MUSE [13], although we were not able to find information on how that was achieved.

### 2.3. Ease of Use

As a user interface, the toolkit should provide intuitive interaction for control over sound synthesis and for the mapping of data properties to perceptual parameters. For most of the existing toolkits, the main motivation is to offer simple sonification tools for users who may not be experts at computer science or sound design, so the criterion for ease of use is usually rather well satisfied. The interaction is designed to be self-explanatory, with real-time auditory feedback allowing sonification designers to quickly understand "which button does what". Although, as noted by Phillips and Cabrera in [14], ease of use is often provided at the cost of flexibility as more powerful, general-purpose languages tend to be less accessible to novice users. These authors attempted to address this issue by offering an interactive creation of mappings through a visual programming language [14].

### 2.4. Experimentation

The *Sonification Report* states that a toolkit should provide a framework for perceptual testing. Yet, besides the quick prototyping functionalities afforded by most real-time audio mapping toolkits, nothing appears to be in place to facilitate formal perceptual testing. Indeed, no system that we could find included, for example, usability testing forms or experimental frameworks for signal detection.

### 2.5. Flexibility

The customization allowed by the toolkit should give users full control over each and every aspect of the sonification. Sound-computing-savvier users should also be able to "peek under the hood" to edit the provided functions and extend the toolkit to fit their own needs.

Due to the aforementioned trade-off between ease of use and flexibility, and the fact that ease of use usually prevails as a concern in the design of sonification toolkits for non-experts, the flexibility criterion seems to have been somewhat lacking in most attempts. For a more thorough analysis of flexibility in generic sonification toolkits, we give in the following paragraphs an overview of some of the main affordances that tend to be provided, to varying degrees, by sonification toolkits: mappings, sound design, data structures and interactive sonification.

#### *2.5.1. Mappings*

The perceptual scales for sound parameters are not always linear, as famously illustrated by the logarithmic perception of pitch [15], so it is important to account for any and all mapping functions when defining a sonification. Though, the often-implied "default" mapping approach, proposed most notably in the Sonification Sandbox [5], seems to consist only in selecting input and output scales to define a linear transfer function from one data dimension to one sound parameter. In some cases, it is also specified that the mapping functions can include non-linear (usually boolean) operations, as is the case for Porsonify [11], Listen [12], MUSE [13] or SonART [16]. The Sonification Workstation provides an interesting case where the user can type out custom expressions to describe how several input signals should be fused together upon entering a computational block [14].

#### *2.5.2. Sound Design*

Sonification toolkits usually include a few basic functions for sound synthesis, or sometimes MIDI, with finite numbers of parameters to choose from when building mappings (Personify [17], Listen [12], Sonification Sandbox [5], xSonify [6], SonifYer [4], Sonification Workstation [14]).

The sound design affordance can sometimes be more flexible due to the sonification tool being part of a host environment already dedicated to sound programming: SonEnvir with SuperCollider [7], the ISonTK with Pure Data [8] or MUSE with CSound [13]. In these cases, the toolkit is provided as a way to simplify most tasks related to sonification design – except for sound, which is thus programmed using the interactions and modalities of the host environment.

#### *2.5.3. Data Structure*

Sonification toolkits tend to be built around the assumption that the input data is sequential and should be "played out" as such: each data point gets mapped to the parameters of one sound event, so that a sequence of sound events conveying the sequence of data points can be heard. This approach at least accounts for techniques like parameter mapping sonification and, in a more extreme form, audification. Toolkits that seem to work under this assumption are the Sonification Sandbox [5], the ISonTK [8], xSonify [6] and the Sonification Workstation [14].

Some toolkits can handle more diverse input data structurations: in Barrass' PerSonify [17] for example, the input data is assumed to be spatially organized in 2D. SonifYer [4] and Rotator [10] are two instances of tools that can be used for displaying geometrically-organized streams of sequential data.

Several other sonification toolkits seem to position themselves solely at the sound mapping stage of the display design, with no assumption of the data structure itself: Listen [12], MUSE [13], MUSART [18], SonART [16]. Presumably, the part of the final sonification application which handles data acquisition and structuring then has to be programmed through other means, outside of the toolkit.

#### 2.5.4. *Interactive Sonification*

The kind of user interaction afforded by a toolkit usually depends on the assumed structure of the input data. In the case of sequential data, it is sometimes possible to position a playback cursor in the sequence through mouse interaction (e.g. the ISonTK [8]). For spatially-organized data, the interaction usually consists in probing the data space (e.g. Personify [17]). Beyond these assumptions though, as far as we know, no custom sonification tool lets users entirely define their interaction modalities.

### 2.6. Remaining Challenges

While questions of portability, integrability and ease of use have seemingly been successfully answered in existing works, and surely will be again in the future, it seems that the main remaining challenge with regards to the *Sonification Report*'s criteria is true flexibility. Indeed, a lot of toolkits appear to have been built around assumptions on what a sonification "should be", especially in terms of input data organization. We submit that the best way to achieve flexibility is to first describe a fully generic model of the sonification process.

## 3. THE SONIFICATION CELL

Borrowing from a notation introduced by Hermann et al. in 2007 [19], we propose that sonification should be formulated as a mathematical function of its input data: let $X$ the data to be displayed, $\theta$ the variable mapping parameters, $T$ the numerical values denoting user interaction, and $t \in \mathbb{R}^+$ a parameter for time progression, then a sonification is a signal synthesis function $s(t, X, \theta, T)$.

To detail the actual semantics of this function, we turn to the notion introduced by Roddy and Bridges in [20] that a sonification is a combination of meaningful perceptual units, or "sonic complexes", and that each of these complexes is dynamically generated based on its own subset of perceptual parameters, or "sonic dimensions", computed from the data dimensions. Let $M$ be the number of sonic complexes in a display, $(\phi_i)_{i\in[1,M]}$ the functions for generating those complexes and $(\vec{p_i})_{i\in[1,M]}$ the functions calculating vectors of sonic dimensions for those complexes. We propose the following mathematical expression of a sonification:

$$s\left(t, X, \theta, T\right) = \sum_{i=1}^{M} \phi_i\left(t, \vec{p_i}\left(t, X(t), \theta(t), T(t)\right)\right) \qquad (1)$$

Note that, as far as signal synthesis is concerned, $\sum$ really is the actual sum of several numerical signals for audio mixing. In the case of device control signals such as MIDI for example, this "sum" is more akin to a concatenation of the messages being generated. For the sake of simplicity in this paper, we will only consider the output of a sonification to be a numerical audio signal.

The goal then is to propose a framework for sonification design which allows its users to provide more detail on the elements involved in this function. In the following subsections, we describe how the data inputs, sonic outputs and sonic dimensions can be organized into a diagrammatic layout, then connected using a visual programming language to form parameter mappings and sound synthesis functions.

### 3.1. Input/Output Layout

During the planning or design phases, the structuring of the elements involved in a system can take the form of a diagram, in which various categories of information fit into different blocks and cables or arrows between these blocks can represent streams flowing or actions taking place. The visual layout proposed here is intended to play such a role for the description of all the variable elements in the functional expression of sonifications: the data to be displayed, the interaction modalities and custom parameters provided to the end user, and the parameterized sonic output.

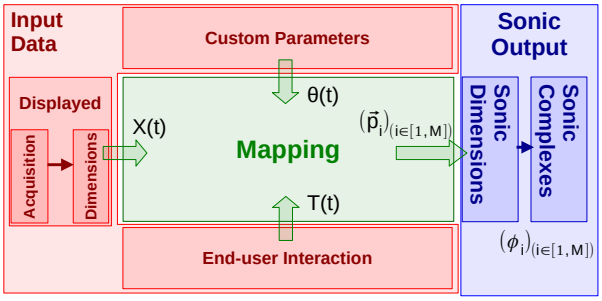


Figure 1: An overview of the visual layout in the proposed model.

#### 3.1.1. *Input Data*

Three separate sections can be used to describe data sources in the model, each representing a different type of role within the sonification process, as well as different relationships between the end-user and the sonification tool.

The data being displayed by the sonification per se is placed in the section on the left of the pipeline (see "Displayed" in Figure 1). This section is used to describe the source of the data (e.g. reading from a file, listening to a stream) and the structures of numerical values that they form (e.g. an $N$-dimensional space, a temporal sequence of $N$-dimensional data slices, sets of objects with given properties...).

In situations requiring data-space exploration, a sonification tool usually offers means for the end-user to interact with the data or the mappings in ways that enhance the feeling of embodiment or immersion within the display [21]. In our model, end-user interaction is defined in the section at the bottom of the diagram (see "End-user Interaction" in Figure 1). This section is used to describe user inputs by means of interaction with the system (e.g. keyboard, mouse, dedicated peripherals) as well as the structures those values form (e.g. positions, events, specific movements).

In data observation tools, it is sometimes useful to provide ways for the end-users to input their own values for parameters, for example in cases where the input or output scales of a perceptual mapping are customized to allow for varying sensitivity. The section for custom parameters, which we place at the top of the diagram (see Figure 1), is used to describe explicit numerical end-user input denoting choices in the data-to-sound transformations, (e.g. perceptual ranges, selecting intervals in the data).

The data ports defined in all three of these sections can be written as functions of time (respectively $X(t)$, $T(t)$ and $\theta(t)$ in Figure 1 and expression 1), such that at time $t$, a vector of numerical values representing the data sources state is outputted.

#### *3.1.2. Sonic Output*

We propose that, in the diagram, the block for the description of the sonic output of a display be placed on the right side of the pipeline (see Figure 1). In it, designers can describe the functions involved in the generation of sonic complexes ($\phi_i$ in expression 1), their parameters ($\vec{p_i}$ in expression 1), and the way they should run in parallel and be combined (e.g. additive mixing of all elements in the case of signal synthesis, or concatenation in the case of MIDI).

#### *3.1.3. Mapping Section*

We propose that the relation between the properties of the input data and the parameters of the sonic output be described within the central section of the pipeline (see "Mappings" on Figure 1). In it, a designer describes the transfer function that maps input data to sonic dimensions: $\forall i, \vec{p_i}(t, X(t), \theta(t), T(t))$ in expression 1. The visual programming language envisioned for this purpose is described in section 3.2.1.

### 3.2. Specificities

#### *3.2.1. Visual Programming Language*

In our proposed model, a mapping function is constructed by defining operational blocks which contain numerical functions, either as named functions which are defined elsewhere or as explicit mathematical expressions of input variables, and connecting them into one another's input ports using cables indicating the dataflow.

This "patching" approach presents a number of well-known advantages and disadvantages in terms of learnability, flexibility and ease of use [22, 23, 24]. It has already been successfully implemented in several popular sound programming languages, such as Pure Data, Max/MSP, Kyma [25], and is also similar to the interaction offered by libmapper's graphical tool for data mapping [26]. It has even been proposed in at least one sonification toolkit in the past: the Sonification Workstation [14].

#### *3.2.2. Temporality*

As exemplified by cases where sonifications are designed for the exploration of the spatial organization of data rather than their sequential progression (e.g. model-based sonifications), the temporal aspect of an auditory display does not always depend on the temporal evolution of the data. Conversely, sound is an inherently temporal phenomenon. As such, it seems necessary that the temporality of an abstract model of sonification programs be based on that of the sonic output rather than on the data's.

In our model we consider that, for each time step of the program's runtime, audio is rendered based on the state of the sound parameters, which are themselves synchronously computed based on the state of the input data for that time step. The value of that time step can be used as a read-only variable by all the functions throughout the model.

#### *3.2.3. Variables*

In order for displays to take consecutive temporal states of the system into account, an additional internal set of dynamic data, which we call "variables", may be used in the mapping section. Those variables are defined by the initial values they take upon startup, and may be updated or consulted over the program's runtime. In the case of the sonification for process monitoring described in 5.1 for instance, a variable allows us to memorize the last time an alarm was triggered, so as to wait ten seconds before triggering it again, should the related issue still be present.

#### *3.2.4. Factorization*

Sonifications can sometimes be built using granular synthesis techniques: several simultaneous sound grains being synthesized according to the same process, and then mixed together. This is a technique often found in model-based approaches to sonification [27].

In order to take those techniques into account, it becomes necessary to be able to factor out processing paths in the model. The syntax we propose in the diagram format consists in writing the recipe for one grain, from data input to sonic output, and indexing it over a range (for example "$\forall i \in [1, 4]$") to denote its repetition. See Figure 4 for an application in one of our case studies.

### 3.3. Towards a Generic Toolkit for Sonification

The aim of this very diagrammatically-oriented model is to allow for a natural progression from theoretical model to practical applications: if a user can draw a sonification following our model, this sonification should be implementable within the toolkit. We describe thereafter the envisioned interactive design tool, based on the Sonification Cell and following existing recommendations for the creation of a generic sonification toolkit (Kramer et al. [1], De Campo et al. [28]):

- The graphical user interface is made up of rectangular frames according to the model's layout of inputs, mappings and outputs.
- Data inputs can be defined by picking from a set of ready-made functions: among other things, parsing data from some file types, or inter-operation with external devices or software.
- Operational blocks can be dragged-and-dropped into the mapping section from a library of predefined operations. Additionally, it is also possible to include blank blocks in which custom expressions can be typed out freely (though with some basic syntactic rules).
- Blocks have one or several input ports on their left and a single output port on their right. A mapping is created by drawing a cable from one block's output to another block's input.
- Sound synthesis can be achieved from a selection of options, notably importing audio files, using pre-made DSP or pro-

gramming new ones. The interface explicitly presents the parameters for these sonic elements as input ports for mappings.

- A lower-level development tool can be accessed by users willing to code additional, more complex operational blocks from within the environment.
- The toolkit's release includes a collection of pre-made project files showcasing some of the more commonplace sonification techniques, such as auditory graphing or interactive data space exploration.
- Based on the diagram created in the interface, the toolkit can produce a real-time rendition of the resulting audio.
- A compiled version of the resulting sonification program can be exported, in order to plug it into other programs and environments or use it as a standalone application.

## 4. PROTOTYPING IN OSSIA SCORE

Ossia has been chosen as the environment for us to build an approximation of the sonification toolkit proposed. A hybrid between sequencer and data-flow patcher supporting various multimedia data formats, it satisfies several of the criterion discussed in section 2.

Ease of use has been validated through years of workshops with novice students and artists which have been shown to become proficient in the basics of the software in a few hours of classes.

Flexibility is the main strength of the system: it has been designed from the ground up with a very extensive plug-in API in order to serve experimental research needs, and embeds multiple programming languages that can be edited on the fly to adapt the operation of the software to one's needs: JavaScript, C++, GLSL, Faust, Bytebeat and math expressions among others. In this work, for instance, the plug-in API has been leveraged to enable us to easily import arbitrary CSV data into the system, as part of the project file (i.e. the software code itself did not require modifications).

Integrability is another strength: this cross-platform software can handle audio and video streams, as well as many communication protocols: OSC, DMX, MIDI and others. It also supports embedding various existing audio plug-in formats, such as VST, VST3, LV2, JSFX. Being free software helps ensuring its perennity. In terms of differentiating factors with other common audio mapping environments, such as Max/MSP, PureData or SuperCollider, a fundamental difference of ossia is its built-in support for timelines: conceptually, the software is organized as a "temporal graph" of individual patches. For instance: a specific sub-patch can run for a few seconds, then another patch will start and run until an external event happens, after which the execution could loop back to the beginning or start executing a third sub-patch. This is all presented to the user in a "standard" sequencer fashion akin to Cubase or Pro Tools, where the timeline is executed – at least initially – from left to right at a constant rate, and allows to trivially introduce interactive elements into the sonifications.

### 4.1. A Project Template Based on the Model

Leveraging ossia score's workspace organization, we recreated our proposed I/O layout in a simple project, shown in Figure 2. This instance of the model contains nothing by default, but it constitutes an approximation of the expected appearance and behavior for our user interface.

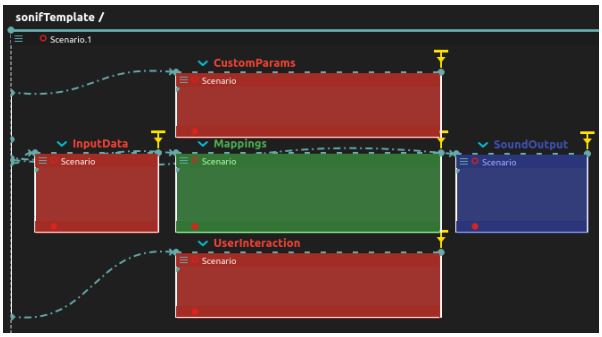


Figure 2: A project template for sonification in ossia score, based on our model

We were also able to use the built-in audio patching environment and on-the-fly programming tools to recreate most of the features intended for our sonification toolkit. The features relating to interactive definition of parameter mappings and sound synthesis are directly available by using ossia score's "vanilla" environment.

The custom processing blocks can be emulated either by using ExprTK process blocks to write simple algorithms for up to 4 parameters, or by coding and compiling "Just-In-Time" (JIT) some of the more complex processes that required more than 4 parameters. That second option was especially useful for us to quickly add the functions that had been missing for data sonification in ossia, such as a linear mapping function with the option to parameterize input and output intervals in real time, as well as processes capable of reading and parsing CSV files for data acquisition.

The variables announced in section 3.2.3 can be recreated using OSC devices in the ossia environment. The interaction allowing users to give initial values to these devices before manipulating them as input and output ports is relatively close to the one intended for our toolkit.

As addressed in the second case study and in the discussion for this paper, a few of the intended features of the toolkit – most notably factorized sound synthesis – could not be implemented in the proof of concept for now, but are expected to be added in the future.

## 5. CASE STUDIES

Project files, resources and video demos for the following case studies are provided in the supplementary material for this paper.

### 5.1. Additive Manufacturing Process Monitoring

In this example, the purpose of sonification is to facilitate the real-time monitoring of an industrial process for its operators. The process in this case is Wire+Arc, a form of additive manufacturing which consists in the controlled projection of molten metal onto an item being constructed layer by layer [29].

The display needs to allow its listeners to detect faults in the item, both at the point where matter is being deposited (the "weld pool") and overall (the "part"). Faults can be detected based on a set of metrics acquired in real-time during production, which we named as follows: WPH the weld pool height, WPW the weld pool width, WPT the weld pool temperature, PH the cumulative part height, $PH_0$ the expected cumulative part height, and PT the average part temperature.

#### 5.1.1. *Proposed Solution*

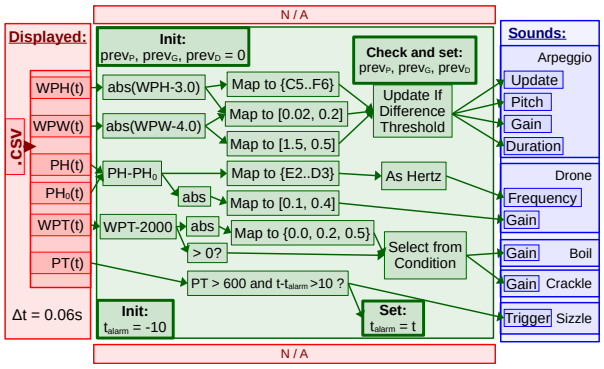


Figure 3: Abridged depiction of the mappings in the sonification for process monitoring

The display studied in this example has been published in [30]. It consists in a soundscape made up of a combination of prerecorded natural sounds conveying temperature alerts, and some simple musical elements conveying geometry faults. Figure 3 gives an overview of the mappings in place for this display. *Arpeggio* is a continuously-repeating sequence of notes whose starting pitch is mapped to WPH, whose duration is mapped to WPW, and whose volume gain is mapped to the average of both. These three parameters are updated at the same time if at least one of them exceeds a certain tolerance threshold. *Drone* is a continuous synthetic tonal sound whose pitch and gain are mapped to PH discrepancy. *Boil* and *Crackle* are prerecorded natural sounds which occur respectively in case of over- or under- heating of the weld pool (WPT), with a gain factor conveying the amount of temperature discrepancy. *Sizzle* is a sound that gets triggered as an alarm in case PT passes its threshold, and can only be activated again after ten seconds.

This example constitutes an interesting case study in parameter mapping sonification as it involves a diverse panel of techniques for sound creation, parameterization and mapping. In the next subsection, we cover the way those specificities were recreated in the toolkit.

#### 5.1.2. *Discussion*

A process was written in C++ and live-compiled within the ossia scoreinterface to handle the data acquisition from CSV files. Every sonic complex defined for this sonification was successfully recreated using the toolkit. The tonal elements (*Drone* and the piano-like instrument that plays the notes for *Arpeggio*) were programmed as Faust [31] signal processors and imported into the project. For *Arpeggio* in particular, the sequencing of notes according to parameterized duration and pitch, which had originally been achieved using SuperCollider's pattern tool, was handled in the sonic output section by a patch mimicking pattern behavior. The natural sounds for *Boil*, *Crackle* and *Sizzle* could be imported into the project using ossia score's built-in sound sampling, and parameterized using an audio gain control object. Finally, the parts of the parameterization which require variables, for *Arpeggio* and *Sizzle*, could be recreated through ossia score's built-in OSC-based device tree.

### 5.2. Interactive Exploration of Protein Trajectories

The goal of this display is to facilitate the study of protein diffusion data acquired using super-resolution nanoscopy [32]. For each time frame Plane$(t)$, each of the M$(t)$ active proteins is a structure Point$_i$ defined by its coordinates in 2D space Pos$_i(t)$, and its fluorescence Lum$_i(t)$. The diffusion rate is an additional metric that is computed from the accumulated displacement of a protein over time. It characterizes a point's type of movement and thus its role within the cell. For each point, a diffusion rate is computed for the whole trajectory (GDiff$_i$), and another one is computed for the displacement between one frame and the next (Diff$_i(t)$).

A visual display technique that is often in use for these data consists in rendering the positions of the proteins as color-coded dots in an animated 2D image. Our addition of an auditory modality to that display is intended to make the observation of this data more informative by leveraging the listener's ability to detect temporal patterns in sounds, and more inclusive by providing an alternative for visually impaired observers.

#### 5.2.1. *Proposed Solution*

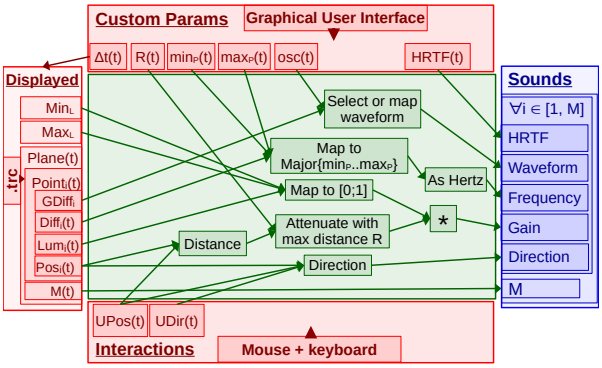


Figure 4: Abridged depiction of the mappings in the sonification for trajectory exploration

The interactive exploration tool we propose consists in building a soundscape of sound grains that are spatialized in real-time around a virtual listener. The virtual listener is positioned and oriented by the user via a mouse and keyboard interaction through the visual display window. Each of the M$(t)$ points in the frame contributes to this soundscape by "emitting" a sound grain (indexed by $i$ between 1 and M$(t)$) that is parameterized according to its diffusion rate, fluorescence and position relative to the virtual listener. As illustrated in Figure 4, the instantaneous diffusion rate is mapped to the grain's pitch, while the fluorescence and distance to the listener are mapped to the sound volume of the grain.

A number of parameters for these mappings can be custom-selected by the user. The extrema min$_p$ and max$_p$ give the start and end degrees of the major scale used for conveying instantaneous diffusion. R is the maximum distance of audibility in the attenuation of volume. It is also possible to select a collective timbre osc for the grains, either from four fixed wave forms (sine, sawtooth, square and triangle), or such that each grain's timbre is selected based on its trajectory's global diffusion rate GDiff$_i$.

This case study constitutes an opportunity to recreate a more complex kind of sonification, in comparison with the previous one,

where all three types of data inputs are necessary, where the auditory display is synchronized with a visual display, and where the sound synthesis uses factorization.

#### *5.2.2. Discussion*

Since it is not part of ossia score's toolkit yet, factorization of the sound grains could not be recreated here. Instead we built a simpler, asynchronous version of the display where each trajectory is played out fully before the next one starts. The recipe for parameterized sound grain synthesis itself was successfully reconstructed. The custom process for CSV parsing that had been developed for the previous case study was modified for the more specific preprocessing phase needed here, so that minimum and maximum fluorescence values, as well as diffusion rates, could be extracted from the data. The interaction is done by dragging the cursor on a graphical window on which the evolving position of a point is displayed. As of now, the parameter that gives the listener's orientation is handled as an explicit custom parameter, but it would likely be possible to also recreate the interaction for listener orientation in a future version. All the input fields for custom mapping parameters were successfully added to the upper part of the workspace layout, including the choice for sound grain timbres as either a fixed waveform or as one depending on $\text{GDiff}_i$.

## 6. DISCUSSION

Over the course of our experimentation, we were able to recreate two different sonifications in a single software environment, while relying on our intended programming approach and Sonification Cell model. This is encouraging in terms of confirming the feasibility of a generic sonification toolkit based on our proposed model.

One of the main objectives of the model and the toolkit is exhaustiveness. Even though our case studies represent two very different approaches to sonification in terms of data structure and end-user interaction, they are not quite enough to confirm that this objective is reached. Work towards confirming the exhaustiveness of the model and toolkit will require experimentation on a broader sample of cases, especially including some of the more unusual types of sonification like model-based or audification.

Most of the features we consider to be required for the dedicated tool (see section 3.3) are successfully accounted for in this proof-of-concept: the graphical user interface is able to implement our proposed visual layout for the model, pre-built operational blocks can be dragged-and-dropped into the workspace, new blocks can be defined quickly, mappings can be created by patching outputs into inputs, multiple integrated lower-level programming tools (ExprTK, Faust and C++) can be accessed quickly, audio can be rendered in real-time, the temporality is dissociated from that of the data, and variables can be created and updated.

Still, a few of the more complex features could not be implemented in this proof of concept yet: the factorization of data-flow paths and the compilation of sonification projects into programs or modules. Factorization is known to be difficult to translate into visual programming paradigms ([33, chap. 1]), but it still seems to be possible since, notably, the operational block "poly" provided by Max/MSP allows for the creation of multiple instances of a given patch. Compilation of sonification DSP into native code is also feasible as exemplified by existing audio programming tools such as the online IDE for the Faust language.

While the current state of this proof of concept is still missing a few of the planned features, it seems reasonable to expect that they could be added in the near future. As of now, it should at least be possible to use this tool to create any sonification where the number of sonic complexes is known in advance, granted that all repeating paths are manually re-written each time they are needed.

## 7. CONCLUSION

After proposing a generic model of sonification programs, we tested the feasibility of its translation as an interactive design toolkit by writing two of our case studies into that model's format, then recreating them in a basic sonification template, made in the ossia score sequencer and based on the model's layout. Most of the intended features for the toolkit were present in this proof of concept, and the few missing ones are expected to be added in a near future. By keeping up with future updates to the sequencer and engaging in a collaborative feedback loop orienting its development towards our needs, we should be able to construct a full prototype of the intended toolkit eventually. On longer timescales, our goal will be to develop the toolkit as a stand-alone application, with a workspace and a library of functionalities that are specific to an implementation of our model rather than inherited from the ossia environment and with a streamlined user interface for sonification. Using the prototype presented here, we are now able to start user testing, in order to check whether people with more or less experience with sonification actually find the Sonification Cell and its dedicated toolkit useful – at least starting with the non-factorized types of sonifications that it supports so far.

Our long-term expectations are that the right toolkit will facilitate the creation of sonification prototypes, as well as help with setting up experiments on auditory perception, while involving the various actors of the domain with as little prerequisite technical knowledge as possible. Such openness should overall help the field of sonification by leading to faster prototyping, creation and iteration of more adequate tools, for sonification to be considered a go-to display method when faced with data observation requirements.

## 8. REFERENCES

[1] G. Kramer, B. N. Walker, T. Bonebright, P. Cook, J. H. Flowers, and N. Miner, "The Sonification Report: Status of the Field and Research Agenda. Report prepared for the National Science Foundation by Members of the International Community for Auditory Display," in *International Community for Auditory Display (ICAD)*, Santa Fe, NM, 1999.

[2] M. Poret, "Vers un Modèle pour la Sonification de Données Scientifiques," Theses, Univ. Bordeaux, Dec. 2022.

[3] J.-M. Celerier, P. Baltazar, C. Bossut, N. Vuaille, J.-M. Couturier, and M. Desainte-Catherine, "Ossia: Towards a unified interface for scoring time and interaction," in *TENOR 2015-First International Conference on Technologies for Music Notation and Representation*, 2015.

[4] F. Dombois, O. Brodwolf, O. Friedli, I. Rennert, and T. Koenig, "SonifYer – A Concept, a Software, a Platform," in *Proceedings of the 14th International Conference on Auditory Display*, Paris, France, June 2008.

[5] B. N. Walker and J. T. Cothran, "Sonification Sandbox: a graphical toolkit for auditory graphs," in *Proceedings of the*

*9th International Conference on Auditory Display*, Boston, MA, July 2003, pp. 161–163.

[6] R. Candey, A. Schertenleib, and W. Diaz Merced, "xSonify sonification tool for space physics," in *Proceedings of the 12th International Conference on Auditory Display*, London, UK, June 2006, pp. 289–290.

[7] A. de Campo, C. Frauenberger, and R. Höldrich, "Sonenvir – a Progress Report," in *Proceedings of the 2005 International Computer Music Conference*. Barcelona, Spain: Michigan Publishing, Sept. 2005.

[8] S. Pauletto and A. Hunt, "A toolkit for interactive sonification," in *Proceedings of the 10th Meeting of the International Conference on Auditory Display*, Sydney, Australia, July 2004.

[9] Z. Kondak, T. A. Liang, B. Tomlinson, and B. N. Walker, "Web sonification sandbox-an easy-to-use web application for sonifying data and equations," Aug. 2017.

[10] J. Cherston and J. A. Paradiso, "Rotator: Flexible Distribution of Data Across Sensory Channels," in *Proceedings of the 23rd International Conference on Auditory Display*, Pennsylvania State University, USA, June 2017, pp. 86–93.

[11] T. M. Madhyastha and D. A. Reed, "Data sonification: Do you see what i hear?" *IEEE Software*, vol. 12, no. 2, pp. 45–56, 1995.

[12] C. M. Wilson and S. K. Lodha, "Listen: A data sonification toolkit," in *Proceedings of the 3rd International Conference on Auditory Display*, Palo Alto, California, Nov. 1996.

[13] S. K. Lodha, J. Beahan, T. Heppe, A. Joseph, and B. Zane-Ulman, "MUSE: A musical data sonification toolkit," in *Proceedings of the 4th International Conference on Auditory Display (ICAD1997)*, Palo Alto, California, Nov. 1997.

[14] S. Phillips and A. Cabrera, "Sonification workstation," in *Proceedings of the 25th International Conference on Auditory Display*, Northumbria University, Newcastle upon Tyne, UK, June 2019, pp. 184–190.

[15] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.

[16] O. Ben-Tal, J. Berger, B. Cook, G. Scavone, M. Daniels, and P. Cook, "SonART: the sonification application research toolbox," in *Proceedings of the 8th International Conference on Auditory Display*, Kyoto, Japan, July 2002.

[17] S. Barrass, "Personify: A Toolkit for Perceptually Meaningful Sonification," *ACMA*, vol. 95, 1995.

[18] A. J. Joseph and S. K. Lodha, "MUSART: Musical audio transfer function real-time toolkit," in *Proceedings of the 8th International Conference on Auditory Display*, Kyoto, Japan, July 2002.

[19] T. Hermann, K. Bunte, and H. Ritter, "Relevance-based interactive optimization of sonification," in *Proceedings of the 13th International Conference on Auditory Display*, Montréal, Canada, June 2007, pp. 461–467.

[20] S. Roddy and B. Bridges, "Mapping for meaning: the embodied sonification listening model and its implications for the mapping problem in sonic information design," *Journal on Multimodal User Interfaces*, vol. 14, no. 2, pp. 143–151, 2020.

[21] T. Hermann and A. Hunt, "The importance of interaction in sonification," in *Proceedings of the 10th Meeting of the International Conference on Auditory Display*, Sydney, Australia, July 2004.

[22] A. Brocker, R. Schäfer, C. Remy, S. Voelker, and J. Borchers, "Flowboard: How seamless, live, flow-based programming impacts learning to code for embedded electronics," *ACM Trans. Comput.-Hum. Interact.*, aug 2022, just Accepted.

[23] I. Ouahbi, F. Kaddari, H. Darhmaoui, A. Elachqar, and S. Lahmine, "Learning basic programming concepts by creating games with scratch programming environment," *Procedia - Social and Behavioral Sciences*, vol. 191, pp. 1479–1482, 2015, the Proceedings of 6th World Conference on Educational Sciences.

[24] C.-Y. Tsai, "Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy," *Computers in Human Behavior*, vol. 95, pp. 224–232, 2019.

[25] C. Scaletti, "The Kyma/Platypus Computer Music Workstation," *Computer Music Journal*, vol. 13, no. 2, pp. 23–38, 1989.

[26] J. Malloch, S. Sinclair, and M. M. Wanderley, "Libmapper: (a library for connecting things)," in *Proceedings of the Conference on Human Factors in Computing Systems (ACM CHI)*. ACM, 2013.

[27] T. Hermann and H. Ritter, "Listen to your Data: Model-Based Sonification for Data Analysis," in *Advances in Intelligent Computing and Multimedia Systems*. Windsor, Ontario: Int. Inst. for Advanced Studies in System research and cybernetics: G.E. Lasker and M.R. Syed, eds., 1999, pp. 189–194.

[28] A. de Campo, C. Frauenberger, and R. Höldrich, "Designing a generalized sonification environment," in *Proceedings of the 10th International Conference on Auditory Display*, Sydney, Australia, July 2004.

[29] S. W. Williams, F. Martina, A. C. Addison, J. Ding, G. Pardal, and P. Colegrove, "Wire + Arc Additive Manufacturing," *Materials Science and Technology*, vol. 32, no. 7, pp. 641–647, 2016.

[30] M. Poret, S. Ibarboure, C. Semal, M. Desainte-Catherine, and N. Couture, "Peripheral Auditory Display for 3D-Printing Process Monitoring," in *Proceedings of the 18th Sound and Music Computing Conference*, D. A. Mauro, S. Spagnol, and A. Valle, Eds., Turin, Italy, June 2021, pp. 268–275.

[31] Y. Orlarey, D. Fober, and S. Letz, "Syntactical and semantical aspects of Faust," *Soft Computing*, vol. 8, no. 9, pp. 623–632, 2004.

[32] E. Betzig, G. H. Patterson, R. Sougrat, O. W. Lindwasser, S. Olenych, J. S. Bonifacino, M. W. Davidson, J. Lippincott-Schwartz, and H. F. Hess, "Imaging Intracellular Fluorescent Proteins at Nanometer Resolution," *Science*, vol. 313, no. 5793, pp. 1642–1645, 2006.

[33] M. Fouilleul, "A Temporal Programming Environment for Live Shows and Art Installations," Theses, Sorbonne Université, Jan. 2023.