# A Database System For Organizing *Musique Concrète*

**Christopher Bailey**

## Abstract

When writing *musique concrète*, there may arise a question of how to deal with a massive bank of heterogeneous source sounds. One possible solution involves a simple, powerful and flexible database engine built in FileMaker Pro. A composer enters data about sounds in the collection and then specifies musical gestures abstractly by creating parametric profiles called models. Models, realized as sequences of actual soundfiles via database queries, can be written out as mix-files which can then be displayed, modified, processed, and mixed with the Ardour mixing application.

**Keywords**: *musique concrète*, database, Filemaker, gesture, Ardour.

## 1.Introduction

An old teacher of mine once said that there were two routes a composer could take with a composition: create a lot of variety out of a small piece of material; or, take maximally varied materials and try to find ways that those materials can cohere and flow together, as if they, too, arose from single source. When it comes to *musique concrète*, I have always enjoyed the latter approach: starting with a large bank of heterogeneous sounds, I attempt to weave them into a coherent sound tapestry. This paper is a about a tool I developed to help me do this. I begin with some personal aesthetic history, then describe the tool and give examples of its use, and finally discuss some possible future developments.

## 2.Aesthetic Context

My early works in the *musique concrète* genre, *Ow, My Head* and *Duude*, (both of which somehow managed to sneak onto former ICMC programs), were composed in 1996 and 1997 respectively, and partake of the same aesthetic approach. Both works used large banks of source sounds (which I recorded myself), and involved little or no sound processing. I was interested in coherence, but not (necessarily) unity: it was more important to me that one gesture led into another smoothly (or abruptly, if appropriate), than that all gestures were derived from the same basic bit of material. I was interested in the possibility of creating new timbres solely from superposition/combination and rhythmic arrangement of unprocessed audio. Finally, I enjoy the way a sound can be referential (we know what the source is), while at the same time being an abstract part of a musical utterance (a "note" in a "phrase"). Heavy processing eliminates the referential function of a sound, and thus its dual functionality, leaving only it's abstract musical purpose. For some, (and for myself, at certain times) this is desirable, but in most of my works I aim for a situation where sounds can be heard playing as many roles and fulfilling as many functions as possible.

For both *Ow, My Head*, and *Duude*, I began with between 300-500 source soundfiles. Members of the collections varied profusely: vocal sounds, speech, banging, scraping, shaking, a few 'field recordings' (e.g. crickets, beaches, practice room hallways), and so forth. For a given work, the sounds were related only in that they were all recorded in the same geographical location (for example, in and around a certain house or apartment.)

The works were, for the most part, written by stringing together a sequence of smaller sub-mixes, with just a few musical gestures in each. I would compose these sub-mixes by first assembling a more-or-less random set of soundfiles, and then attempting to arrange the gathered soundfiles in time and stereo space to create a musical gesture. By musical, I mean a gesture that coheres, that sounds more like a whole entity than a mere sequence of randomly juxtaposed sounds.

This was what I refer to as a "junk sculpture" paradigm of composition: give me some sonic detritus, and I will make music out of it for you. I enjoy this way of working very much, but I was, at times, a bit uncomfortable with it. I sometimes wanted to be able to work in a more deterministic fashion. I began to realize, especially during the composition of *Duude*, that 'deterministic' might refer to the ability to conceive a musical gesture, somewhat abstractly, and then find the sounds from among my collection of 300-500 to realize that gesture.

My first attempt was a system built in LISP, as part of a larger system made for my work *Sand*. *Sand*'s *concrète* layer worked well, and was a successful "proof of concept," of the ideas I had been toying with in my head since *Duude* and *Ow* were composed. However, these LISP tools lacked an interface of any sort, and at this point, I really knew nothing about how database systems were built, so the programming was rather clumsy.

In 2004 I began working as a database programmer using FileMaker Pro. For those not familiar with FileMaker, it can roughly be described as a GUI-driven, all-in-one (structural and UI-design tool) database design application, complete with its own scripting language; a sort of MAX/MSP of database design, if you will (one view even involves 'patching' data tables together in a very similar fashion to what one does with MAX/MSP objects.) I instantly saw the possibility of realizing, in a more user-friendly and cleanly-programmed form, the ideas I had put to work in *Sand*'s LISP engine. In 2007, I received a grant/ residency from Harvestworks in New York City to work on this project. The initial result was my composition *Harvest Kitchen*, and the system itself, which I used extensively in realizing the work. After composing *Harvest Kitchen*, I composed the smaller work (*Divertimento in Eb*) using the same sound bank and set of tools. Currently in process is *Harvest Kitchen Part II*, which continues to expand on the materials and processes discussed in this paper.

## 3.The System

### 3.1.Modules

The system consists of 4 modules: Sound Data Storage, Gestural Scoring, Gestural Realization, and Output.

### 3.2.Sound Data Storage Module

The Sound Data Storage module consists of a set of data tables to hold data about soundfiles. There are 10 parameters for each soundfile record (See Figure 1). Most of the parameters have a numerical range between 1 and 7. This might seem rather coarse: that is intentional. The system works best with a very heterogeneous bank of source sounds, but because of that heterogeneity, it is difficult to compare sounds with any degree of exactitude. We can say for sure that the attack hardness of a hammer hitting a nail is a lot greater than the attack hardness of, say, a field recording of crickets at night, but how does the attack-hardness of that field recording compare with the attack-hardness of a wispy tweet from a dove? I thought originally of making the parameters go simply from 1-3, (low-medium-high), but in the end, 1-7 seemed like a workable compromise that would allow for some ambiguity and room for error.

Parameters of special note include: PC, (pitch-class), which I separated out as a different parameter from [pitch-] register for maximum compositional flexibility later on, and which can be noted in any equal-temperament, (important for me given my interest in microtonal music); and Material/Category/Semiotic, where one can 'tag' a sound with any one or more words (description of the action taken to make the sound, the material of the object making the sound, and so on) in order to group it with other sounds—in the example we see "Gesture", "Grunge", "Scrape", and "Squeak."

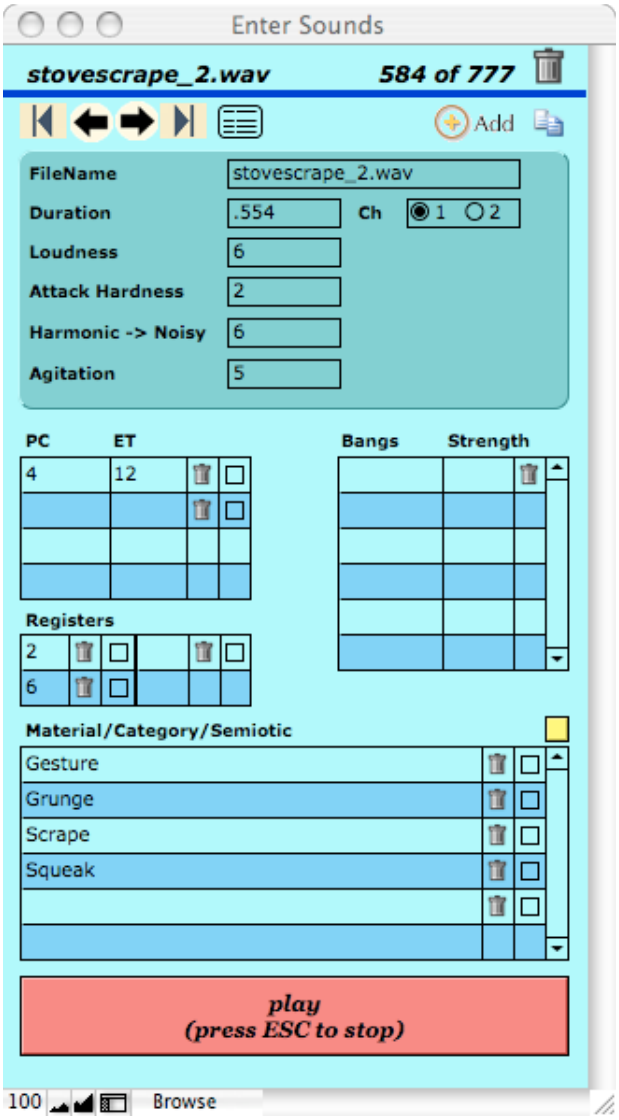


**Figure 1. The Sound Data Storage and Entry Module**

For some parameters, there is the possibility of multiple values. In the example shown in Figure 1, scraping the stove created spectral energy at the low end (2) and the high end (6) of the audio spectrum. I therefore assigned to this sound registers 2 and 7. Another familiar example of a type of sound (not shown in Figure 1) involving more than one value in a parameter is a sound in which one hears more than one prominent "pitch-class."

In thinking through many possibilities, I have found that these 10 parameters cover all or most of the possible sounds I use. On occasion people have mentioned parameters not covered here, but in most cases they are accounted for as hybrids or combinations of the parameters found in the system already.

### 3.3.Gestural Engine

Having entered data on the sounds in the sound bank, one can proceed to the most interesting part of the system: the compositional tools.

The basic concept is as follows: one begins with an abstract idea for a sonic gesture, which I call a **model**; a model consists of one or more **elements**, or soundfile-specifications, in the gesture. A given model, when realized, can produce one or more **sequences** – actual sequences of specific soundfiles, elements now realized as specific **events**. From such a sequence we can produce a mix-file. The system currently renders mix-files as XML which can be used with the Ardour open-source mixing application.

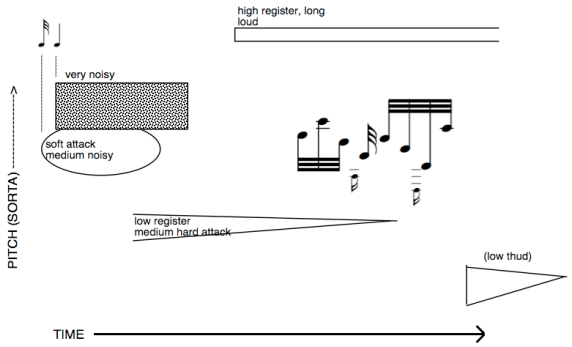


**Figure 2. A graphical score, an idea for a sonic gesture**

Let us look at an example. Figure 2 shows a sketch of a sonic gesture. We now need to define each sound element in this gesture more specifically. For example we begin with a sound, which, most importantly, should be of a relatively soft attack-hardness (within a range of 1-3, for example), and medium noisiness (probably 3-5). In the sketch, the 32nd-notes will be interpreted as sounds with a short duration ( <1.5' ), register according to position on the page (sort of), hard attacks (5-7), and low Agitation (1-3) (in other words, hit and let ring). Similar determinations of necessary parameters are made for the other sound events in the gesture. It is not necessary, and almost never desirable, that a value for every parameter be entered for an element. (In fact, as I quickly discovered even with the LISP version of this system, the reverse is the case: for the most successful results, one should specify an element with as few parameters as possible.) Having carefully thought through the sound events in the sketched-out gesture, we can then translate them into actual elements within a model. Figure 3 shows the list of elements on the left, and 1 (highlighted) element's parameters on the right.

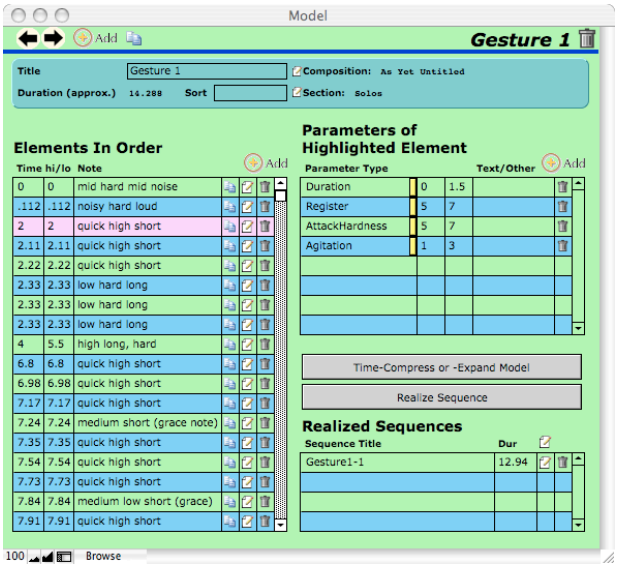


**Figure 3. The same data, entered as a model into the system.**

Notice the left 2 columns in the left most list in Figure 3. These specify a range for the start-time of an element. The actual start-time of a realized event will be randomly chosen between these values (To specify an absolute or determinate time, just put the same value in both fields).

Once we hit the "Realize Sequence" button, the database system will find sounds in the sound bank whose parameters match those specified in the model's elements.

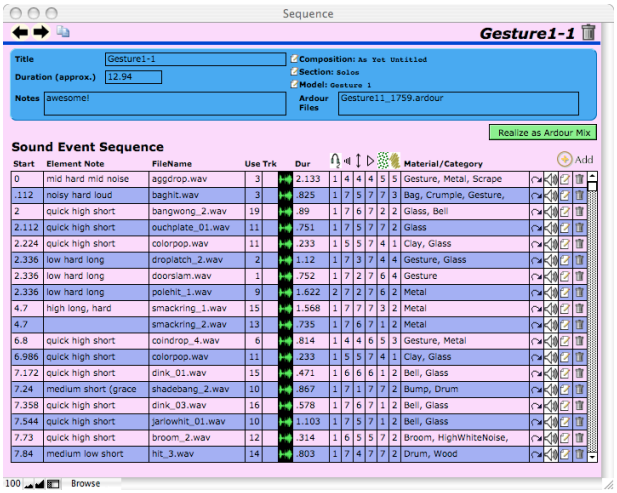


**Figure 4. The model in Figure 3 now realized as a sequence.**

In most cases, more than one soundfile will satisfy the requirements of a given element; therefore, many different realized sequences are possible for any given model. One can realize as many sequences as one wants. One can also check over the chosen soundfiles in the sequence-view window (see Figure 4) and re-calculate one or more of the elements (clicking the curly-arrow symbol, 4 columns

from the right side), in case you need to just fix one or two events in a given mix which was otherwise satisfactory.

Once you've reviewed the component soundfiles, and you think a sequence will work, (or at least, has potential), you can hit the "Realize As Ardour Mix" button to create the Ardour file.

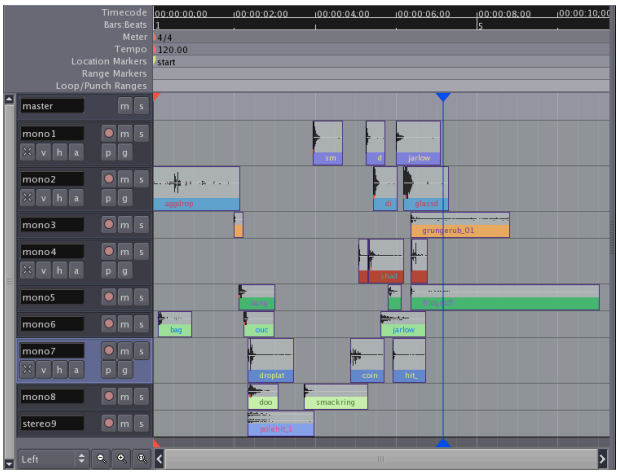


**Figure 5. The sequence in Figure 4 realized as a mix in Ardour.**

Figure 5 shows an Ardour-realized mix of the gesture, model and sequence given above. When the system creates a mix, sounds are randomly placed into one of 8 or more tracks, which are arranged, in the mixing window, from top to bottom, corresponding to stereo position from left to right. Once the mix is loaded into Ardour, one can adjust rhythm, timing and stereo position, add processing, and so forth.

### 3.4.Some Additional Features and Fallout

A few small features were added as I put the system to use composing *Harvest Kitchen* and later in the *Divertimento*. It's possible to ask for a completely random soundfile by not specifying any parameters for a given element. Hence, the method I used to compose much of *Duude* and *Ow, My Head* ('junk sculpting' with random sub-collections of sounds) is still possible with this system.

Another feature I implemented soon after starting to use the system was a 'used' field that will tell you how many times a given sound is used in all of the realized sequences in the database. In the sequence view, (see Figure 4) one can consult the 4th field from the left; one might notice, for example, that a given sound has been used 19 times, and then opt to re-calculate it, hopefully choosing a lesser-used sound in the database. Many composers of *concrète* music desire to keep close control over the repetition of specific soundfiles; this calculation field helps with that task. After composing the *Divertimento,* I made this part of the engine itself: by default, it always chooses among the least-used sounds that fit the parameters asked for.

Recall the first 2 columns in the left list in Figure 2: these were the minimum and maximum start-time for an element. I added a few features with which to quickly manipulate rhythm: for example, if one decides that 'tempo' of a model needs to be a bit faster, one can multiply all the start-times by a factor of .83, for example, to get the desired effect.

## 4.Future Plans and Conclusion

The system is not complete and continues to grow and change. A top future programming priority is to set up some sort of GUI "scoring" system for entering the model/element data. It is hard to know exactly what such a score would look like. It would be hard enough to assign 10 different visual "dimensions" to the 10 parameters, but the problem is, as I mentioned above, for any given element, usually you don't care about most of the parameters, only one or two of them. So any scoring system should show values for only the parameters attached to a given element —only the parameters that the user "cares" about in choosing that sound from the collection. Thus, for example, assigning vertical visual height to pitch, as is traditional in musical scores, is misleading—sometimes we don't care what the pitch is, as long as the selected sound has, say, a very hard attack. In that case, we don't want a vertical dimension aspect to "mean anything"—we don't want information displayed that is not relevant to the kind of sound we are seeking through the agency of that element.

Currently, entering sounds into the database is done manually. Another programming priority for this system is to set up automated analysis to determine some or all of the 10 parameters for sounds in a collection, and thereby cut down on data entry. The sound collections I use are typically extremely heterogeneous, however, which would, I believe, make this difficult (and/or render the results of such automation more or less useless).

I would also like to see the system output files for other mixing systems besides Ardour (such as Logic, DP, or even Pro Tools), although I have been quite impressed with Ardour and am very happy to be contributing something its growing community of users. Ardour keeps getting better and better, and I hope that the application I describe in this paper provides yet another reason for people to think about using it.

Finally, although until recently most of my *concrète* music uses little or no processing, I have been easing up on that aesthetic stance lately, and in fact the gesture engine now outputs scores for the RTCMIX package, in order to accomplish some simple processing such as transposition, filtering, delays, etc. In fact, in the *Divertimento,* there is already quite a bit of processed sound to be heard.