

Flicker Composer’s Assistant (FCA) - Timbre and Hierarchy - Meaningful Selection of Resources in a Large Database

Kenneth Newby, December 2016

Hierarchy is a very useful and, if constructed well, a very meaningful way of breaking a large set of things into smaller groups of things with meaningful, i.e. semantic relations to other things within a group as well as relations to other groups and their populations.

Steffen asked for some feedback on his design of a template that had as its objective a musical optimization of the VSL libraries. I made some notes on that which I’ll lay out first. Then I’m going to go into a fair bit of detail about my approach to the issue in my design of the Flicker Composer’s Assistant.

Musical decisions, (Steffen’s terminology) This is an essential thought to keep close at hand when working with computers. It’s easy to let this slide because the machine has a way of imposing its relatively inflexible machine-like way of expressing things and performing actions that easily impose too much abstraction onto our creative processes. Another way of looking at this is that much of the software we use is not designed by artists. Don’t get me wrong here... many of the gifted software engineers responsible for the tools we use are artists of a sort... but their medium is software, not necessarily the medium the tool is designed to emulate and provide affordances for. A great example of this is Photoshop. When you go to save a .tiff file you’ll be presented with a dialog that asks you if you want your image saved as a 24-bit file or a 32-bit file. That’s it. No other context or language provided to assist the artist to decide what the answer to this question might be. I’ve seen a couple of generations of young digital artists go blank when asked what the right answer to the question is. Now the real question, from the artist’s perspective is: do you want to save the alpha (transparency) layer with your image? That layer accounts for the extra 8-bits of file size per pixel. That’s already a fairly technical bit of knowledge about the architecture of the digital image that’s required of the “digital artist”. But it’s grounded in the nature of the digital visual medium and how it works. Simply asking to choose between a 24-bit or 32-bit bit depth is simply too abstract. It’s actually an example of a bad programming practice known as “magic numbers”, but more on that later. Here one can see how it might make sense to the designer of the software. You need more bits to encode the extra information. You need to manage memory, format a file properly to save it out successfully, etc. None of that is of concern to the digital artist... at least it needn’t be. This is a classic example of the “Procrustean Bed” problem in which humans are forced to adapt to a system’s requirements rather than the systems adapting to the human need. If you wanted to sleep at Procrustes’s Inn for the night you’d find there was only a bed of one peculiar size. Procrustes would solve any problems of bad fit between guest and bed by either lopping off some length of your legs if you were too long for his bed, or stretch you out in case you were too short for it.

The primary approach I’ve taken in the design of FCA. is to organize things in a musical way, which more often than not, turns out to be some kind of recursive, self-similar hierarchical structure. Music.

A second important design tool has been the use of “symbolic” names for things, elements, group names, individual instances of things so that the composer can communicate with the system in the language of music and most, if not all, actions and organizing is done along musically meaningful metrics.

And thirdly, I created a naming convention that packs as much information into the name of the articulation and its higher level containers as possible using a system of underscore-delimited sub-attributes of the articulation that collect everything we need to know about that name and what it points to. More on this in the more detailed description of FCA’s way of organizing the VSL libraries.

A very good idea to provide an option for the mute/sordino articulations to be accessed through Slot-SFade. One will be able to achieve much more timbral variation that way.

I make use of the Filter in VIPro to get a similar wide range of variation from articulation playing categories such as ponticello, which can be filtered quite a lot and still sound very good. In the case of the ponticello articulations it’s simulating a variable distance of the bow from the bridge with the un-Filtered ponticello being the closest to the bridge and successively Filtered versions played further away (having less emphasis of the high partials of the timbre).

With respect to Steffen’s way of ordering his resources, I think this is a very important design decision to have made in that it preserves the musical/cultural information that’s embedded in our musical terminology. As much as I’m fluent in the languages the computer does its work in—thirty-five years of coding practice will do that to you—I don’t want to have my musical language framed that way. It’s akin to the proscription in programming against the use of what are called “magic numbers”, which can take various forms such as a number that occurs in the code that represents the value of something that remains unnamed in the code. It’s hard to read, and if it goes wrong, even harder track down as a problem. It’s always better to make use of a symbolic container, a variable, to hold such values with a descriptive name that informs the reader as to its meaning in the code. So, in my template, everything is given a symbolic name. Instruments are `grouped by family (strings, woodwinds, brass, etc.), articulations are grouped according to the way the sound is produced (e.g. bow position on a string instrument,

use of a mute, what kind of mute, etc., what kind of mallet is used, soft, wood, metal, etc.).

The matrices, are another clear example and have symbolic names (shorts, dynamics, repetitions, etc.), when they could just as easily have been numbered Matrix 1, Matrix 2, etc. But this latter, of course, represents a loss of meaningful musical information. In our project (music composition, orchestration, performance, et al), which is one of the cultural activities of the highest level of sophistication we know, the machine must adapt to our needs and understanding not the other way around. Digital tools need to aim for “Intelligence Amplification”—empowering us to get more work done and better quality work at that. Too many tools are designed to somehow make it easy, or provide a door into the project for people without the basic knowledge to really know the richness of what they’re dealing with in all its glorious multi-dimensionality.

The Structure of the FCA Articulation Hierarchy

So... there’s a master document that specifies that big tree that represents the VSL libraries (or the big chunk of them that I have to work with). That document is very detailed and specific and has grown to almost 750 pages in length. But it can’t be too specific or you lose flexibility in its use so I had to play with different designs for a while before I found the right balance between structure and the potential for variation. The current version has been stable for almost two years now and makes use of a seven-level hierarchy. Starting from anywhere in the hierarchy you can follow a line to either the root structures or, in the opposite direction, to the surface where the individual articulations live. I enriched this structure with a multi-valent naming convention that encodes multiple characteristics into each level allowing its entries to be queried and “best fit” answers to be returned in cases where something is asked for that doesn’t exist.

The My main objective in the labour involved in creating this huge description was to provide an ordering of the articulation resources that encompasses every available sample of every instrument. I ended being able to do this six matrices: shorts, sustains, performances, dynamics, repetitions and specials. In the case of instruments that have a complete set of articulations played with a mute, such as the solo trumpets, tenor trombones, solo violin family, chamber strings, etc. I used 12 matrices with the matrices 7 - 12 mirroring the organization as much as possible of the un-muted matrices 1-6.

Matrices are chosen using MIDI program change messages 1 - 12.

Using keyswitches I found it possible to get everything into a matrix that was 8 x 12 (96 cells per matrix).

So that’s 20 keyswitches plus the two a/b switches for a total of 22 MIDI keys needed to do the job. It just fits if you start the keyswitches at the lowest C~ key of the keyboard (MIDI pitch number 0). And instruments with the lowest range just fit without collisions.

Some Matrices are much smaller than the 8 x 12 maximum, but I found that it’s best to set all every matrix to that largest 8 x 12 size due to the possibility of VIPro receiving the cell-position keyswitches before the program change moving from one of the smaller matrices to a larger one. If you send a keyswitch for a cell that doesn’t exist in a matrix VIPro will simply ignore that setting, a logical approach, except if a program change now tells it to move to the larger matrix, VIPro will end up in the cell of the previous articulation in the smaller matrix. This was happening to me at one point and took me a few hours of frustrating trouble-shooting before I finally realized what’s going on. I not make it failsafe by enforcing the matrix change (MIDI program change) first, then the two x, y keyswitches. Plus, I set all of my matrices to the largest 8 x 12 size so it will still work even if that first rule is broken (the first “out-of-bounds” keyswitch will just select an empty cell in the “smaller” matrix. then the program change message will find that empty cell now occupied by the desired articulation in the second matrix.

Other than the blending of standard and muted articulations of the same instrument into a kind of “super-instrument” I kept each instrument as a separate channel. I also broke up a lot of the percussion into separate instruments or groups to facilitate “cherry-picking” the things I want to work with. This results in a lot of channels so I’m making use of the IAC Driver Bus system on the Mac OS, making as many buses, each with 16 MIDI channels as needed. An instrument is fixed in its relationship to a particular IAC Driver Bus+MIDI channel. I made duplicates where needed such as multiplying the Vienna Horns by four, doubling oboe, bassoons, clarinets, etc. to the maximum size of orchestra I thought I would want to work with. All in all I ended up with 190 individual tracks in the master template which translates into 12 IAC Driver Buses x 16 MIDI channels each, with two channels to spare, perhaps solo violin 2 and solo cello 2. At any rate it’s a simple matter to add another Bus and get another 16 channels to work with. I, of course, load the master template with cells disabled so it doesn’t overwhelm the computer resources.

To clarify all of the above we can, for example, look at how the hierarchy represents a group of related articulations taken from the dynamics articulations in a library such as the Appassionata Violins. Lets take the medium dynamics played with vibrato at several lengths for this example. In the Appassionata Violins there are two versions of these, One set is played arco and the other sordino (with a mute). In each of these two playing categories there are three variations with lengths of 2, 3, and 4 seconds respectively.

These articulations dwell in the hierarchy as follows: (you can see the tree structure which, when represented this way in text, is lying on its side.

The trunk of the tree is rooted at the leftmost entry (the Orchestra) and elaborates large branches, smaller branches, twigs, etc. as it subdivides smaller and smaller groups to the right. Individual articulations represent the surface of the hierarchy, the leaves as it were. I’m using a programming convention that the double-forward slash is the beginning of a comment, useful for documenting the form on the left. That means the the statements beginning with // are not part of the hierarchy per se, but explanations about the entry to the left.

Orchestra	// the root level of the hierarchy “everything”, the Universe // you could add another layer to capture different libraries in addition to VSL (a multidimensional universe?)
Strings	// instrumental section (strings, woodwinds, brass, percussion, mallets, keyboards, voices, special instruments)
Appassionata Violins	// specific instrument or group (choose any instrument) In my template there are two sections of Appassionata Violins 1 & 2 // they are fixed to play on IAC Driver Bus 4, channel 6 and 7 respectively
arco	// playing category (arco, ponticello, tastò, sordino, etc. in the case of strings, other sections will vary accordingly)
dynamics	// the type of articulation (shorts, sustains, performances, dynamics, repetitions, specials) // A type corresponds to one of the Matrices in VIPro and is accessed by MIDI Program Change messages.
articulationSubType: crescDim_vib_medium_2-3-4	// the sub-type of articulation (these are groupings of articulations along a symbolically meaningful ordering or metric // in the case of these dynamics it made sense to order them according to their length, shortest to longest // you’ll notice that the articulationSubType name has four underscore-delimited parts, the final part of which // is, in itself, an ordered sequence of duration values, again there’s hierarchy at several levels encoded in the name // this name says its a dynamics sample that can be a/b controlled to be crescendo or diminuendo, it’s played with vibrato // and of medium intensity. The fourth element of the name is the 2-3-4 hyphen-delimited group that is easily taken apart // to determine how many individual aritulations are in the group and what their specific values are. This makes it possible // to do really cool things like query the database for a desired articulation say a 2.5 second crescendo played with vibrato // the name gives that answer that, while the specific length is not available, there are two close options, at 2 or 3 seconds // in length. The final choice is made on whether the resulting texture should allow for connecting to the next articulation // (round up and select the 3 sec one) or can be disconnected from the next one (round down, select the 2 sec one)
M_crescDim_2 0 0	// these are the keyswitches specifying the cell position of the articulation in the matrix, // it has a symbolic name that describes three of its musical qualities (intensity, direction, length)
M_crescDim_3 1 0	
M_crescDim_4 2 0	
sordino	// here’s the sordino variations of the same articulation, in this case, however, the sordino keyword causes // a different MIDI program change to select the dynamics matrix containing the sordino dynamics articulations // otherwise the hierarchical structure is the same
dynamics	
crescDim_vib_medium_2-3-4	
M_crescDim_2 0 0	
M_crescDim_3 1 0	
M_crescDim_4 2 0	

It's in the definition of the sub-types that the most power lies. This is because the designer can decide on any ordering that makes musical sense. We could have ordered the dynamics articulations along the intensity metric, in which case a group might hold: L_crescDim... M_crescDim... S_crescDim.... for example. The beauty of this way of thinking is that one doesn't have to choose one or the other. If more than one ordering is useful define them all. There's no penalty for including the same articulation in several orderings. In fact there are huge benefits to be gained by having a variety of ways of making use of an articulation in a different context or group in which its musical function might be significantly different that the first. So your resources become polyvalent... able to be used in a variety of contexts, while still retaining an anchor to a larger set of resources being used for the same purpose.

This in itself might not seem all that significant (although I find in practice it *is*), but there's one more trick to this system that leverages these orderings into a flexible approach to their selection that produces performances that “breathe” with life and meaningful variation. I defined a number of contextual variables that can influence the selection of articulations within one of the sub-type groups. These are:

speed, connectedness, emphasis, accent, expressivity (e.g. expressivity is an ordering that moves from senza vibrato through progressive vibrato, to vibrato, to espressivo)

Here's an example of a sub-type group ordered according to speed:

articulationSubType: speed_5		
legato_medium	0	0
detache_short	1	0
legato_fast	0	1
spiccato_medium	3	0
spiccato_fast	3	1

In this sub-type group the articulations are ordered along a slower-faster metric. And, depending on how fast the line is being played with this sub-type, the appropriate articulation will be selected. These contextually sensitive groupings always end with a underscore-delimited number that specified how many options are contained in the group. That way it's easy to map the contextual driver to the range of options exposed by the group.

This speed example is basically the same approach used in VIPro to switch between articulation sets based on the inter-offset delays between pitches in a melody, and it was, indeed, initially inspired by the VSL design. I simply defined a few more contextual “states” that could be modulated by higher-level compositional concerns such as melodic shape, phrase structure, section position, etc. etc. Then as those compositional concerns change over time the selection of articulations will follow.