Perceptual Parameters—
Their Specification, Scoring and Control
Within Two Software Composition Systems

Daniel V. Oppenheim
Center for Computer Research in
Music and Acoustics (CCRMA)
Department of Music
Stanford University, Stanford, CA 94305
Dan@CCRMA.Stanford.EDU

Tim Anderson,
Ross Kirk
Music Technology Group
University of York
YORK YO1 5DD, UK
tma@chm.york.ac.uk

Composers think in terms of perceptual concepts whereas synthesis requires the specification of physical device parameters. Composers would like to work by defining and specifying perceptual parameters within a given sound, rather than synthesis parameters. This requires a system to translate from such user-defined perceptual aspects to the larger numbers of parameters which control the synthesis of that aspect. We describe the implementation of this concept within two compositional systems: DMIX and E-Scape.

The Concept of Perceptual Mapping

Composers think about their music in perceptual terms whereas computers control synthesis parameters that typically model acoustic parameters. The correlation between human perception and the related acoustic parameters is complex and not clearly understood. This raises many problems for composers, particularly when working with the timbral aspects of a musical event. A simple example will clarify this. Let us assume a composer is working with a sound, and finds within it two qualities he would like to work with at the compositional level. He names the first "smooth" and the second "grainy". Note that unlike parameters such as pitch or duration, smooth and grasy are meaningless unless associated with the specific sound and the musical context in which it is embedded. He would now like to work by specifying and controlling the perceptual aspects of the sound that he defined.

Changes in such a perceptual aspect may require corresponding changes in many synthesis parameters. Furthermore, the manner of change in each synthesis parameter might be different. The composer must first discover how each synthesis parameter should change in relation to a desired perceptual change. Then he can control the synthesis by specifying the correct physical-parameter values. It can be extremely tedious to input this data via a score, even if using a high-level programmeable music description language, such as Quill (Oppenheim, 1990). Real-time control of many parameters is also hard to accomplish as it requires exceptional performance skills, especially if parameters change independently of each other. Clearly, a mechanism is needed that will allow composers to specify control only the perceptual aspects they define and hide the complex non-linear mapping into the many corresponding synthesis parameters.

Perceptual Mapping Design

An important design consideration is the location of such perceptual-mapping within the overall system—whether in the code defining the synthesis instrument, in the high-level sound-object (such as a note-event), or in an intermediate layer between the two. In this paper we describe the implementation of such mechanisms in two composition systems, both implemented in Smalltalk-80: DMIX (Oppenheim, 1993a and 1993b) and E-Scape (Anderson, 1990 and 1992). Both systems enable users to define the perceptual aspects within a sound that they are interested in and then control it via a score; DMIX also enables real-time control.

E-Scape implements the first approach to perceptual mapping, having 'Instruments' which possess an intermediate Processor object within which users can define perceptual parameters. Perceptual data from high-level ScoreEvents is then processed into data for input to synthesis structures. The relationship between perceptual and device parameters is defined by writing (programming) short scripts. This approach requires E-Scape to incorporate the instrument definition of synthesis structures. Hence it will be inappropriate for other systems which do not incorporate such definitions.

DMIX employs an abstract OneToMany object that can be placed in a note-event for real-time control, in various intermediate software levels, or even used merely as a compositional tool for its mapping capabilities. If incorporated into a note-event, there will be a OneToMany for each perceptual parameter.
Whenever the Perceptual Parameter changes in value, each of its outputs will update the appropriate synthesis parameter in real-time. The user can specify the perceptual parameters interactively via an interface that provides real-time audio and graphical feedback.

**Implementation in DMIX—PeRRY**

PeRRY is a simple mechanism that maps one (perceptual) input to any number of outputs. Each time PeRRY receives an input it calculates a new value for each of its outputs and updates it. The transform function between the input and each output is unique and can be a Function (table), DMIX Modifier (much more flexible than a table), or Smalltalk BlockClosure i.e. any algorithm expressed in Smalltalk syntax. A graphic interface aids in interactively defining and testing each output individually. Thus, the composer may vary several states in the perceptual entity (a process we term teaching), and PeRRY will then interpolate to derive intermediate values. Another important feature is that each output can itself be a OneToMany, allowing the composition of hierarchies of transformations.

**Design and User Interface**

![Figure 1: The user interface](image)

Figure 1 displays a graphic editor that can be used to define, set and test a PeRRY. It is implemented in DMIX via two classes: OneToMany and PhysicalNode. The OneToMany has two instance variables a value - this is the perceptual parameter - (1, numbers in brackets refer to the numbers in the two figures) and a set (2). The set is a collection of PhysicalNodes, one for each synthesis parameter. Each PhysicalNode has three instance variables: name, mapper, and seeEvent. The mapper maps a given perceptual value into the synthesis parameter in the seeEvent. The mapper can be a table, a Function (6, 7), or a user-defined algorithm. It can be edited by clicking on the edit button. The seeEvent updates the synthesis device as soon as it receives a new synthesis parameter.

**Typical types (classes) of seeEvent are:**

- MIDIExclusive
- MIDIExclusive, MIDIResetting, MIDIExclusive, DSPNote, or DSPUpdate. This arrangement allows for interactive testing and setting of each PhysicalNode, provided the real-time synthesis is available.

![Figure 2: Mapping pitch to formant/frequencies](image)

The user can set the perceptual parameter value by moving the slider (1). As he does so the sliders of each PhysicalNode in the set also moves to display their new synthesis parameters, and sound is produced to provide real-time feedback. After setting a perceptual parameter, the user can fine-tune each PhysicalNode; when satisfied with the result he can click on the set button. We term this process teaching the set.

**Example:**

Formant Synthesis in MIDI

Figure 2 is an example of using a PeRRY to implement formant synthesis of the human voice via MIDI on a Yamaha SY-77 synthesizer. We chose this example because it is extremely hard to do via MIDI. The human voice is synthesized by having three formants at fixed frequencies. In FM this is can be modeled with three carriers, that emulate the three formants, and one modulator that determines the fundamental frequency. The modulator’s frequency is determined by the MIDI key number and the trick is to have the carrier frequency-ratio be inversely proportioned to the fundamental frequency so that formants stay at fixed frequencies. Needless to say, the Yamaha SY-77 does not allow such a flexible frequency-setting of the carriers so this method has a limited acceptable range of roughly 1 to 4 semi-tones. Using PeRRY we were able to extend the range extensively by having a OneToMany calculate a frequency-ratio for each formant in relation to the note event’s pitch (MIDI key-number). PeRRY makes the needed calculations and sends SystemExclusive messages to the synthesizer.
messages to set the frequency-ratio of each carrier before the actual Midi-note is played. Note that here the setEvent in the first OneToMany (4) is itself a OneToMany (5) that automatically deals with the hardware requirement to set the carriers' frequency-ratio via two parameters: coarse and fine frequency. The first OneToMany (4) calculates the desired frequency-ratio and the second (5) trims this value into the two SY-parameters and sends the correct SystemExclusive messages. The transfer functions are numbered 6 and 7.

PeRRY in the DMIx system

As DMIx is a true object-oriented environment, using music-events with perceptual parameter is no different than using any other music-event. Hence, they can be created algorithmically in Quill, edited graphically with DMIx's powerful editors, get slapped onto Functions (Oppenheim, 1993b), be modified by Functions that get slapped onto them, be transformed and edited in real-time, and much more.

Implementation in E-Scape

E-Scape is a graphical composition system also implemented in Smalltalk-80. Hence it is able to reuse significant objects classes from DMIx. As described above, one of its emphases is on providing score specification of multiple sonic parameters via a processing layer which is designed into each instrument. For this purpose, the essential components of an E-Scape instrument are one or more DCTs and a set of PspProcessors.

A DCT is a specification of one or more "device-level" synthesis modules (processes equivalent to a Music-N unit generator) on a single device. Each synthesis module can be controlled independently. Examples of such modules would be a "LFO" in the MDAS synthesizer [Kerk, 1990] or a "VOICE" in a Yamaha SY77 synthesizer. A DCT can be constructed by a user out of higher-level modules, which are then unrolled hierarchically.

Each PspProcessor specifies one or more perceptual parameters which are available to a composer for scoring, and translates these into 'device-level' synthesis input parameters within a DCT.

An example instrument could contain three DCTs and two PspProcessors. The DCTs could contain ten MDAS "LFOs", two D110 "PART" and four SY77 "VOICE" modules respectively. In this example, one PspProcessor provides two (interrelated) Perceptual Parameters ("pitch" and "random spread") and the other provides one ("energy"). They will then translate values of these parameters to various DCT inputs (which are connected to various inputs of their device-level modules).

A PspProcessor has inputs, outputs and a userProcess. Its inputs effectively describe the Perceptual Parameters which a composer may specify for a score event that uses this Instrument. Each input defines the range and default value of a parameter as well as providing one or more different units which can be selected by the user. Each output of the PspProcessor connects to one or more inputs of a DCT.

A PspProcessor's userProcess contains user-entered Smalltalk source-code which is compiled to a code block (see example below). The code can include specialized system functions which either provide dedicated functions (such as reading data from one or more perceptual parameters of a score event, and assigning data to an available input of a DCT), or utility functions for analyzing or processing data (e.g. find the integer center value of a set of points, or quantize a value to the nearest available on the device). New functions can also be cussed by a user.

Creating ScoreEvents using an E-Scape Instrument

A new ScoreEvent is constructed using a specified Instrument. Each PspProcessor in the Instrument creates a holder to contain "input" Perceptual Parameter values (which may be time-varying). It also creates a holder to contain "output" processed data assigned to the input of a DCT.

When Perceptual Parameter values are entered or changed in the score, the PspProcessor calls its userProcess block, sending itself as a block parameter (parameterHolder). The block can then access the "input" Perceptual Parameter data, and "output" DCT inputs.

Example: Pitch processing by a PspProcessor

The Smalltalk code entered by a user for a simplified example userProcess is given in figure 3. The PspProcessor has two Perceptual Parameters named "pitch" (whose basic units are "semitones from A440"), and "detuning spread" (whose basic units are cents). It has three connected DCT inputs (named "BENDER RANGE", "pitchbend amount", and "at. pitch") which are each connected to several modules within a DCT.

Each input parameter may be time-varying, thus whenever Pitch Parameter changes, the block calculates the absolute pitch by adding the "pitch" value to the "detuning spread" value / 100. It then determines the integer nearest value, quantized to the nearest allowable value of the DCT input above this on the device. This value is then loaded (with time = 0) to the output data holder for the "BENDER RANGE" DCT input. Similar processing results in values which are loaded to holders for the other DCTInputs. When the score is played, this data will be sent to the appropriate modules within the DCT as instantiated on a device (inserted within the assigned message for that input of that device).
NB. Temporary block variables commence by convention with lower case 'y'

[pitchFunc |
  pPitch <- (pspProcessor pointsOf PerceptualParameterNamed: 'pitch')).
(dpitchFunc <- (pspProcessor pointsFor PerceptualParameterNamed: 'returning spread')).
(PitchPoints <- pPitchProcessor allTimes collect: [T | @ ((pitchFunc valueAt: T) + 100) ]).
(CentreVal <- EFUnction imageCentreOf dpitchPoints).
(Deviation <- EFUnction imageMaxDeviationOf dpitchPoints fromCentreValue: CentreVal).
(IPBSensitivity <- (pspProcessor outputNames: 'BENDER RANGE') nearestDeviceValueAbove: Deviation abs).
pspProcessor removeAll.
pspProcessor loadVal: (IPBSensitivity toDCTInputNamed: 'BENDER RANGE').
pitchPoints do: [eachPt | newPt <- eachPt processVal: [val | (val - CentreVal) divBy: (IPBSensitivity) * 63 + 64].
  pPitchProcessor loadPoint: newPt toDCTInputNamed: 'pitch bend amount'].
pspProcessor loadVal: (CentreVal + 68) toDCTInputNamed: 'st. pitch'.

Figure 3: code entered in a userProcess for pitch-processing

Conclusions

The implementations in both DMIX and E-Scape enable composers to specify complex real-time changes using musically meaningful Perceptual Parameters. Each approach has strengths and limitations. E-Scape's approach to specifying the process between score parameters requires the user's code (albeit a restricted subset). However it allows quite complex algorithmic mechanisms to process, if desired, several Perceptual Parameters which interact; to analyze time-varying score parameters, and to interrogate device specifications.

The approach in DMIX whereby the user 'teaches' a OneToMany is more intuitive, easier to use and provides real-time audio and visual feedback. It is especially useful for specifying mappings which are more empirical (as the 5V77 example above) and which would be highly laborious to enter in the E-Scape implementation. However, the DMIX implementation supports a less flexible relationship between inputs and outputs. The ability to replace the function mapping with a code block facilitates more complex mappings, but still assumes a tree structure starting with a single Perceptual Parameter, and provides no access to device input settings or other Perceptual Parameters.

It is planned to incorporate the DMIX OneToMany object into the E-Scape PspProcessor object, in order to use the more intuitive DMIX implementation in cases where there is only a single Perceptual Parameter to be processed. Parameter data can be sent to a DMIX OneToMany owned by the PspProcessor. Each of its bottom-level PhysicalNodes could then have a DCT input as its m2Event and load mapped data into the corresponding holder. The task of creating userProcesses blocks in E-Scape will also be made easier with a menu-driven interface allowing functions, input and output names to be selected by the user.

Bibliography