Contemporary Music Review
Vol. 28, No. 1, February 2009, pp. 31–42

Routledge
Taylor & Francis Group

Split Levels: Symbolic to Sub-symbolic Interactive Music Systems

Robert Rowe

The first wave of interactive music systems relied almost exclusively on the Musical Instrument Digital Interface (MIDI) standard, a symbolic representation of music modeled closely on the behavior of a piano keyboard, as well as traditional concepts of music notation. The second wave takes as its input raw audio signals, a sub-symbolic data representation that is far more flexible while being far less structured. This article will consider the issues involved with building a pathway back up from sub-symbolic systems to the symbolic approaches of the first wave, and the relative advantages of the two.

Keywords: Interactive Music System; Audio Analysis; Symbolic and Sub-symbolic

Introduction

Interactive music systems are computer programs that participate in live performances of music; they are characterized by their ability to change their behavior in response to musical input (Chadabe, 1984; Rowe, 1993). As computer programs, they are necessarily algorithmic. As participants in live performance, the algorithms that are possible are necessarily constrained. For one thing, they must depend only on information that is currently present, or that arrived in the past—they must be causal. This disqualifies music analysis systems that can look anywhere in a representation of a performance, as human analysts do when considering a musical score, despite active and compelling research in this area (Pople, 2004; Temperley, 2001). There are many important interactive music technologies that do know what is coming: score following is one obvious example. In this discussion I will nonetheless limit the algorithms under consideration to those that are causal, and fast enough to operate in the real-time environment of a live stage performance, possibly including improvisation.

Such algorithmic approaches to composition and performance have been discussed in numerous publications over the past twenty years (Jordà, 2002; Rowe, 2001; Winkler, 1998). This review will look at the algorithms employed in interactive systems with reference to the input representations that are assumed to be

ISSN 0749-4467 (print)/ISSN 1477-2256 (online)
DOI: 10.1080/07494460802664015

available—for example, audio signals (sub-symbolic) as opposed to Musical Instrument Digital Interface (MIDI) information or other symbolic forms. In particular we will address the parallel universes of symbolic and sub-symbolic interactive systems, and how new designs and techniques might help us to bridge the two.

Interactive music systems in early implementations usually made use of the MIDI standard. The MIDI standard has been recognized since its inception to be slow and limited in its scope of representation (Moore, 1988). Nonetheless, MIDI was the dominant protocol for the design and implementation of interactive music systems for a long time, and its use persists today. There were good reasons for this dominance, particularly at the end of the twentieth century when these systems were first being developed. MIDI-enabled synthesis, sampling, and effects gear produced high-quality digital audio at an affordable price. Moreover, offloading the synthesis duties onto a dedicated piece of hardware freed the CPU of the computer to concentrate on control-level analysis and composition.

Working with MIDI as a musical representation afforded interactive music systems access to a very high-level, symbolic description of the music being played into the system and manipulated within it. Because 'notes' were already clearly defined in terms of pitch, amplitude, onset and offset times, high-level analyses including beat tracking (Desain & Honing, 1999), key induction (Toiviainen & Krumhansl, 2003), segmentation (Cambouropoulos et al., 2001), style identification (Dannenberg et al., 1997) and more could be performed from a relatively secure foundation. In terms of interaction design, the small, fast, robust, and clear MIDI format made an era of rapid progress possible.

On the other hand, MIDI transmission rates are too slow to handle large control streams. Synthesizer manufacturers retreated into bland repetitions of the same sampling-based architectures. The need to stimulate demand by introducing new models every few years meant that works written for commercial gear faced technical obsolescence in a very short period of time as specific boxes broke down and could no longer be replaced. Already, compositions written as recently as ten years ago may require a substantial undertaking of technical archeology to be brought into a state of readiness for renewed performance.

Small wonder, then, that composers have embraced a new generation of technology that allows the rendering part of interactive music systems (synthesis, sampling, and effects) to be handled by the CPU of the same computer calculating larger control structures. The most widespread instance of this phenomenon is the Max/MSP platform, which itself reflects the evolution of the technology: Max alone was developed first and is a MIDI-processing environment (Puckette, 1988), while the later MSP extensions incorporate real-time digital audio signal processing into the control-level structures organized by Max (Puckette et al., 1998).

The issue for composers of interactive music is that adopting the power and flexibility of audio-based systems means exchanging a high-level, symbolic representation for a low-level, sub-symbolic one. All of the hard-won analytical constructs based on MIDI inputs suddenly disappear when the sub-symbolic level

splits off from the symbolic one. In this brief review, we will look at how systems receiving inputs of raw audio are different from or convergent with those built on symbolic inputs.

Digital Audio Platforms for Interactive Music

A wide range of platforms for analyzing and rendering digital audio interactively on personal computers have arisen. Miller Puckette's open source Pd environment is itself a primary reference for the development of MSP (Puckette et al., 1998). A phenomenal outpouring of open source digital audio applications is listed on Sourceforge (sourceforge.net) and other repositories. Beyond the fervor of commercial and individual developers, which has led to such influential packages as ChucK (Wang & Cook, 2003), SuperCollider (McCartney, 2002), and the CLAM library (Amatriain et al., 2002), there have been several efforts to standardize digital audio rendering protocols: The Open Sound Control standard (OSC) is one of the most direct approaches to resolving the networking and representational limitations of MIDI (Wright & Freed, 1997). Other platforms have been shaped by international standards organizations, or by their connection to existing languages. Two of these are the Structured Audio Orchestra Language (SAOL) (Vercoe et al., 1998) and JSyn (Burk, 1998).

Given this extensive range of choices, composers may be forgiven for hesitating over which way to turn. The author's solution has been to develop a personal library of audio sampling, synthesis, and effects routines in C++. While starting anew with such a project inevitably entails reinventing several wheels, it facilitates coordination with an existing control layer (Rowe, 2001) and seems the most expressive way to develop new ideas: a set of personal effects.

As we have already seen, a wealth of well-established languages and programming environments exist for coding interactive music applications. Why then produce one more? I have produced my own libraries not because I think I have better ideas for managing these issues than do the many computer scientists who have looked at it from a much more fundamental perspective. In any case, the library I use has not been developed entirely from scratch, or in a void: I often consult open source projects in designing particular functions, and am in the process of preparing the Personal FX library for open source distribution as well.

There are several reasons for this choice, which range from the compositional to the utilitarian: (1) a lifetime of experience with programming interactive systems at a low level, which makes it easier for me to work in a language like C++ than it is to learn someone else's formalisms; (2) concomitant with that experience, the conviction that design and implementation of every aspect of the computer program for a given piece of music becomes, for me, part of the composition of that piece; (3) a desire to minimize the impact of version changes imposed by others on the maintenance of the software needed to perform my music; and (4) a preference for a very lightweight, easily maintained body of code that does exactly what I need for a

given composition and no more. Further, having virtually all of the software for a given compositional project available in C++ source code greatly simplifies the process of exhuming old works for new performances. Those parts of the system that are subject to changes in the underlying operating system (audio I/O and file handling) are completely localized into two files, which typically are the only things that need updating when a work is revived five or ten years after its most recent performance.

Without going into a long exposition of every part of the Personal FX library, a brief outline of its operation is in order here. A Unit stores some number of channels of audio, and has an optional pointer to an input Unit. At each call from the operating system for a new batch of samples, all allocated Units are executed in order, starting from sound-generation modules (microphones, sound file readers, synthesis routines), through various forms of mixing and effects processing, and finally out to the operating system's output buffers. The most fundamental class in the system, then, does little more than provide a standard way of receiving and storing samples.

Making simple sound-processing objects from the Unit base class, then, becomes primarily a matter of defining a routine that will take an input sample, process it, and return an output sample.

```
double Allpass::Sample(double inSample)
{
  // read delayed sample
  double temp=delayLine[ current] ;
  // add (input sample * gain) to (delayed sample * gain)
  delayLine[ current] =gain * temp + inGain * inSample;
  // calculate output
  double output=temp - gain * delayLine[ current] ;

  // update delay line pointer
  if (++current >= bufferLength)
  current=0;

  return output;
}
```

More involved effects-processing classes can be easily constructed by combining lower-level units. In the example below, the 'Dinosaur' effect is created by allocating two delay lines that feed into two pitch shifters. Delay times and pitch shift amounts are independently variable for the two channels.

```
Dinosaur::Dinosaur(void)
{
  numChans=kStereo;
```

```
  for (int i=0; i<kStereo; i++)
  {
    delay[i]=new DelayLine(200L + (long)(133*i));
    delay[i]->TurnOn();

    pShift[i]=new PitchShift;
    pShift[i]->SetInputUnit(delay[i]);
    int downer=-1 - i;
    pShift[i]->SetSemitones(downer);
    pShift[i]->TurnOn(1.0);
  }
}
```

When a new buffer of samples is called for by the operating system, 'Dinosaur' adds each input sample to a gain-shifted previous output to provide dampened feedback, runs the result of that through the delay line, and changes the pitch. The whole thing produces an effect that takes the input sound and seems to melt it down into nothing, as feedback through a downward-biased pitch shifter will do.

```
for (i=0; i<kStereo; i++)
{
  double* in=inputUnit->OutputSamples(i);
  for (j=0; j<bufferSize; j++)
  {
    double changeme    = in[j] + (outputSamples[i][j] * 0.33);
    double temp        = delay[i]->Sample(changeme);
    outputSamples[i][j]=pShift[i]->Sample(temp);
  }
}
```

There is nothing particularly special going on here—indeed, that is the point. For a great deal of interactive signal processing, quite simple and straightforward techniques are all that is required. Beyond the nuts and bolts of audio processing, other parts of the library handle the invocation of various processes at the indicated moments in the musical score, provide hooks into control-level compositional algorithms, and take care of the scheduling of program state changes over time.

A 'score' in this sense refers to a combination of traditional music notation and cadenzas—opportunities for improvisation within the scope of a particular piece. During notated sections, the relationship between the computer output and the human performance will stay relatively constant from performance to performance. During the improvisations, the computer listener attempts to discern structure in the human performance such that it can add meaningful material during the cadenza

itself. The structures it can 'hear' make possible a set of family resemblances between improvisations—as the human player makes use of consistent gestures or material, the computer will find that continuity and react in similar ways from one rendition to the next.

Analysis Techniques for Live Interaction

The discussion so far has centered on those platforms that are available for realizing real-time interactive algorithms, followed by an exposition of my effects-processing library and a simple example of how it is used. Now we will turn to the input analysis phase of such systems, and consider how we might derive musically meaningful information from an input of raw audio samples.

At first blush, one would think that taking audio signals as the only input representation would be simply an enhancement of earlier, symbolic techniques. After all, we can just find all the notes in the audio and be right back where we were with a MIDI score. Unfortunately, the reality is not so simple. A long period of research has led to some success in automatic music transcription (Klapuri & Davy, 2006), but the intensive computation required to transcribe raw audio, particularly of polyphonic material, moves these techniques well outside the realm of what is possible in the context of interactive performance.

Further, it is not at all clear that what happens in a human brain when listening to music is the formation of some intermediate representation that is equivalent to the notes of a MIDI score (Scheirer, 2000). Certainly we do recognize melodic, harmonic, and rhythmic changes, and we are able to learn to notate heard music with a relatively high degree of proficiency—that is what ear-training class is all about, after all. But the fact that it requires, for most students, years of training to be able to convert by ear a relatively straightforward four-part harmony played slowly on a single instrument into a reasonable facsimile of the score that was played would indicate that such audio-to-score transfers come with difficulty even for humans, if they ever come at all (Hainsworth, 2004).

There is a large body of research concerned with describing the musical content of audio signals for various purposes, including genre identification, mood estimation, and more—indeed, such work forms the core of the field of Music Information Retrieval (MIR) (Aucouturier & Pachet, 2004). Indeed, much of this article is concerned with how to best adapt MIR research to interactive performance. My point here is that there is no straightforward way to take an arbitrary audio signal and transform that into the equivalent of a MIDI representation in real time, thereby allowing the front end of first-wave interactive systems to be replaced with a microphone instead of some MIDI-compatible musical instrument. Moreover, it is not clear that the most effective output from an audio front end would be the equivalent of a MIDI score in any case.

It would be wonderful to know what intermediate representations the brain actually does use in recognizing melodic, harmonic, rhythmic, and timbral patterns

in music, and then to program those representations in our software. As there exist many competing and partial accounts of what those representations might be (Deutsch, 1990; Lerdahl, 2001; Narmour, 1992), we are left to use those tools we have as best we can. Luckily, there is a wide range of techniques to choose from that have varying levels of success for various purposes. The challenge from a compositional standpoint is to choose those techniques that best produce information able to inform a given piece of music or performance situation in a meaningful way. In fact, we may consider the choice of analyses and their interpretation as something of a compositional choice in itself.

Tristan Jehan argues for a similar approach in his dissertation, *Creating Music by Listening*: 'Analysis and synthesis of musical audio can share a minimal data representation of the signal, acquired through a uniform approach based on perceptual listening and learning' (Jehan, 2005, p. 24). Jehan's thesis is about elaborating and implementing a perceptual model that is used both to characterize incoming audio streams and to direct resynthesis techniques for fashioning an output in response. Much of Jehan's analysis work has become available online through a company he co-founded called the Echo Nest, and results from that engine are finding their way into several other analytical systems (Corthaut et al., 2008).

Jehan's work points to the central issue facing audio analysis in the context of interactive performance: how can we approximate the mental representations of music formed by human listeners from an examination of an audio stream? That is the broadest possible statement of the objective—any particular musical composition may require the generation of representations much more limited than the whole of human musical cognition. Given a composition-specific orientation, there is much that can be done and has been done already—for a thorough review, see Collins, 2006. Within the limited scope of this discussion, let us examine a few representative examples.

Segmentation

One of the most important problems we face, and often one of the earliest, is how to meaningfully segment the audio stream as it arrives. We often want to know where the perceptual boundaries are in an incoming audio stream, such that we can identify objects or events within that stream, characterize them, and react to them appropriately. We might extend the notion of segmentation to include many of the fundamental building blocks of a symbolic representation of music: beat tracking (Ellis, 2007), chord recognition (Chew & Chen, 2004), pattern recognition (Shmulevich et al., 2001), and so forth.

Recent work in real-time music analysis has been applied to this question as it pertains to interactive composition and performance (Brossier et al., 2004). The authors review a number of methods for identifying note onsets, including measures of high-frequency content, spectral difference, and phase deviation.

Any of these measures is followed by a peak-picking phase, which examines the analysis output and identifies those transients that will best qualify as a segment boundary.

In my own work I have used the simple high-frequency content method (Masri, 1996) to detect note onsets in a composition for harp and interactive music system, *Moon on one side, Sun on the other*, premiered at the 2007 International Computer Music Conference by Sofia Asuncion Claro. The heart of the calculation is shown in the code fragment from the Personal FX library below:

```
for (i=0; i<_ampSpecBuf.count(); i++)
{
  sample_t amp=_ampSpecBuf[ i] ;
  hfc += (amp*amp) * i;
}
```

After taking a short-term Fourier transform, we simply take the square of the energy in each bin and multiply by the bin number. Then this high-frequency content measure is compared to a threshold, and a note onset is posited when the threshold is passed in a positive direction after a given time duration has passed since the previous onset (to eliminate multiple positives around a single attack). These onsets were used in various ways to influence responses in the work. One of the most important was an ongoing measure of onset density (onsets per second) that was used to control such aspects of the output as sample selection and the variation of parameters in a granular synthesis process that was reading from prerecorded sound files of the harp part.

Treating onset density in this way is similar to a density feature I used earlier in more symbolically, MIDI-oriented work, particularly in my program *Cypher* (Rowe, 1993). Seen this way, onset detection and a related density estimation begin to bridge the levels between sub-symbolic and symbolic systems. The main analytic difference between the audio-based onset density estimate and the *Cypher* density feature is that MIDI Note On commands were much more reliable indicators of note onsets. The compositional difference between the two has to do with how they can be used: in my MIDI work, I often employed what I called *transformative* operations on the input representation (Rowe, 1993). That is, because the signal arriving from a human player already consisted of notes identified by pitch, dynamic, and duration, the response of the program could be formed from manipulations of those parameters. In audio-level work, because much less is known about the pitch, dynamic, duration and even grouping of frequencies found in the input signal, the compositional expression of the output tends toward effects processing of the signal as a whole, rather than transformations based on collections of discrete 'notes'. Another compositional example uses an analysis of the frequency content of an incoming signal to guide various spectral components to different locations in a spatialization process (Torchia & Lippe, 2004).

Timbral Variation as a Control Signal

The idea behind *adaptive audio effects* is that analyses of an audio stream can be used as control signals for other processes (Verfaille et al., 2006). This research area can be seen as directly related to the algorithmic approaches emphasized in this article: analysis techniques are used to derive various forms of musically or perceptually meaningful signals from an incoming audio data stream, and the derived signals are then used to drive some compositional or sound-processing algorithm in response.

A recent audio effects plug-in developed by Sourcetone, LLC, provides a platform for using audio analyses adaptively through a visualization and control generation tool called *Optic* (Figure 1). Four standard audio analysis processes are available: zero crossing rate (a simple measure of the 'noisiness' of the signal); RMS amplitude (a common expression of the physical amplitude of the signal); spectral centroid (the 'center of gravity' of the spectral content of a signal, showing brightness); and spectral flatness (a measure of how evenly energy is distributed across the spectral bands). Any of the four measures may be selected, and the red line on the interface will display the corresponding value. To use the measures adaptively, the user may direct the output from any of the analyses through MIDI to a receiving application—the host application for the plug-in itself, or any other program able to receive MIDI signals via the widely supported ReWire standard. Once the MIDI version of the analysis has arrived at the target application, the user can map this information onto a performance or editing scenario in any way he or she sees fit: spectral centroid could be used to control the modulation index of a frequency modulation patch in

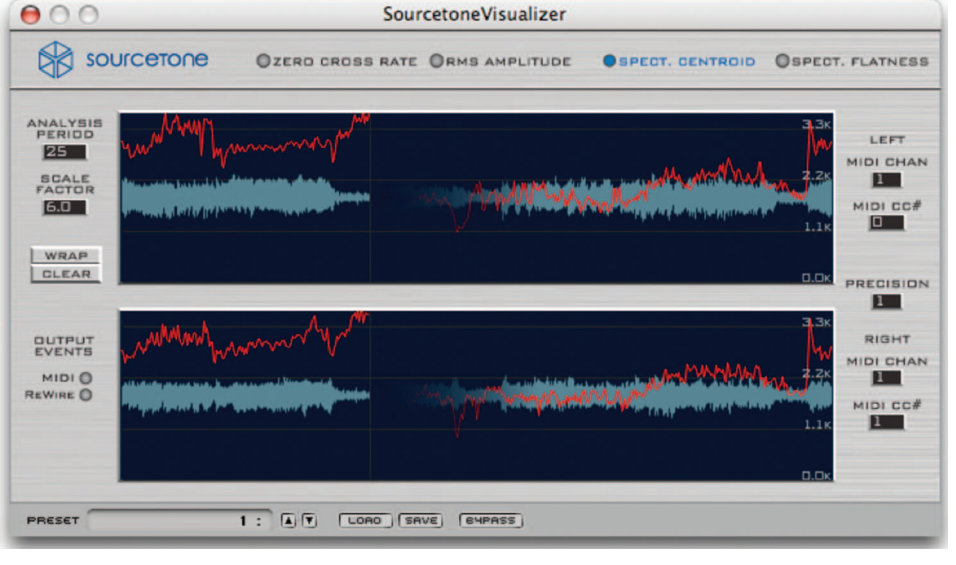

Figure 1 Sourcetone Optic plug-in.

Max/MSP, for example, or the RMS amplitude could be used to adjust the gain in Peak. Finally, the user can launch a standalone version of the plug-in that will accept an audio file as input and deliver an output file of time-stamped analysis files, giving a sample-synchronous record of the full-resolution analysis values for the entire file.

Clearly such analysis signals can be used to control many aspects of compositional algorithms, effects processing, spatialization, and sound synthesis. A sub-symbolic description (in the case of these particular features) can be mapped onto a full range of output manipulations. Again we see the levels split: audio analysis guides output processing, but in such a way that the interaction follows low-level spectral variations, rather than higher-level musical constructs.

Conclusion

Interactive music systems rely on finding musically relevant information in a stream of data produced by a human musician. In many of their early implementations, such systems were built on the relatively high-level but constricted MIDI representation. Current systems are more likely to reference an audio signal, and use analysis techniques from various sources, importantly including the field of music information retrieval, to characterize the content on several levels. Both approaches have strengths and weaknesses: symbolic systems afford a ready pathway to high-level descriptions of musical content, but are limited by their dislocation from the pure sound to which they correspond. Sub-symbolic systems are rich in spectral information, but deliver fewer high-level objects for manipulation. As audio-based systems develop, we are seeing a growing convergence between the two approaches as better content descriptions evolve. Given the state of the art, the choice and implementation of analysis methods and the mapping of their output onto compositional, synthesis, and effects algorithms may well be considered as part of the compositional act itself. We may never return to all the same constructs as those that existed in purely symbolic systems, but the hybrids now emerging provide fertile new ground for compositional design.

References

Amatriain, X., Arumi, P. & Ramirez, M. (2002). CLAM—yet another library for audio and music processing? In *Proceedings of the 17th Annual ACM Conference on Object-oriented Programming, Systems, Languages and Applications*. Seattle: Association for Computing Machinery.

Aucouturier, J. J. & Pachet, F. (2004). Improving timbre similarity: How high's the sky? *Journal of Negative Results in Speech and Audio Sciences*, *1*(1), 1–13.

Brossier, P., Bello, J. P. & Plumbley, M. D. (2004). Real-time temporal segmentation of note objects in music signals. In *Proceedings of the 2004 International Computer Music Conference*. San Francisco: International Computer Music Association.

Burk, P. (1998). Jsyn—a real-time synthesis API for Java. In *Proceedings of the 1998 International Computer Music Conference.* San Francisco: International Computer Music Association.

Cambouropoulos, E., Crawford, T. & Iliopoulos, C. S. (2001). Pattern processing in melodic sequences: Challenges, caveats and prospects. *Computers and the Humanities*, *35*(1), 9–21.

Chadabe, J. (1984). Interactive composing: An overview. *Computer Music Journal*, *8*(1), 22–27.

Chew, E. & Chen, Y. C. (2004). Real-time pitch spelling using the spiral array. *Computer Music Journal*, *29*(2), 61–76.

Collins, N. (2006). *Towards autonomous agents for live computer music: Realtime machine listening and interactive music systems.* Unpublished doctoral dissertation, Music Department, University of Cambridge.

Corthaut, N., Govaerts, S., Verbert, K. & Duval, E. (2008). Connecting the dots: Music metadata generation, schemas and applications. In *Proceedings of the 2008 International Symposium on Music Information Retrieval.* Philadelphia: Drexel University.

Dannenberg, R., Thom, B. & Watson, D. (1997). A machine learning approach to musical style recognition. In *Proceedings of the 1997 International Computer Music Conference.* San Francisco: International Computer Music Association.

Desain, P. & Honing, H. (1999). Computational models of beat induction: The rule-based approach. *Journal of New Music Research*, *28*(1), 29–42.

Deutsch, D. (1990). Grouping mechanisms in music. In D. Deutsch (Ed.), *The Psychology of Music* (2nd ed.). (pp. 299–348). San Diego: Academic Press.

Ellis, D. P. W. (2007). Beat tracking by dynamic programming. *Journal of New Music Research*, *36*(1), 51–60.

Hainsworth, S. W. (2004). *Techniques for the automated analysis of digital audio.* Unpublished doctoral dissertation, Department of Engineering, Cambridge University.

Jehan, T. (2005). *Creating music by listening.* Unpublished doctoral dissertation, MIT Media Laboratory.

Jordà, S. (2002). Improvising with computers: A personal survey (1989–2001). *Journal of New Music Research*, *31*(1), 1–10.

Klapuri, A. & Davy, M. (2006). *Signal processing methods for music transcription.* New York: Springer-Verlag.

Lerdahl, F. (2001). *Tonal pitch space.* Oxford: Oxford University Press.

Masri, P. (1996). *Computer modeling of sound for transformation and synthesis of musical signals.* Unpublished doctoral dissertation, University of Bristol.

McCartney, J. (2002). Rethinking the computer music language: SuperCollider. *Computer Music Journal*, *26*(4), 61–68.

Moore, F. R. (1988). The dysfunctions of MIDI. *Computer Music Journal*, *12*(1), 19–28.

Narmour, E. (1992). *The analysis and cognition of basic melodic structures: The implication–realization model.* Chicago: University of Chicago Press.

Pople, A. (2004). Modeling musical structure. In E. Clarke & N. Cook (Eds.), *Empirical musicology: Aims, methods, prospects.* Oxford: Oxford University Press.

Puckette, M. (1988). The patcher. In Proceedings of the 1988 International Computer Music Conference (pp. 420–429). San Francisco: International Computer Music Association.

Puckette, M., Apel, T. & Zicarelli, D. (1998). Real-time audio analysis tools for Pd and MSP. In Proceedings of the 1998 International Computer Music Conference. San Francisco: International Computer Music Association.

Rowe, R. (1993). *Interactive music systems: Machine listening and composing.* Cambridge, MA: MIT Press.

Rowe, R. (2001). *Machine musicianship.* Cambridge, MA: MIT Press.

Scheirer, E. (2000). *Music-listening systems.* Unpublished doctoral dissertation, MIT Media Laboratory.

Shmulevich, I., Yli-Harja, O., Coyle, E., Povel, D.-J. & Lemström, K. (2001). Perceptual issues in music pattern recognition: Complexity of rhythm and key finding. *Computers and the Humanities*, *35*(1), 23–35.

Temperley, D. (2001). *The cognition of basic musical structures.* Cambridge, MA: MIT Press.

Toiviainen, P. & Krumhansl, C. (2003). Measuring and modeling real-time responses to music: The dynamics of tonality induction. *Perception*, *32*, 741–766.

Torchia, R. & Lippe, C. (2004). Techniques for multi-channel real-time spatial distribution using frequency-domain processing. In *Proceedings of the 2004 New Interfaces for Musical Expression conference.* Singapore: National University of Singapore.

Vercoe, B., Gardner, W. & Schierer, E. (1998). Structured audio: Creation, transmission and rendering of parametric sound representations. In *Proceedings of the IEEE*, *86*(5), 922–940.

Verfaille, V., Zölzer, U. & Arfib, D. (2006). Adaptive digital audio effects (A-DAFx): A new class of sound transformations. *IEEE Transactions on Audio, Speech, and Language Processing*, *14*(5), 1817–1831.

Wang, G. & Cook, P. (2003). ChucK: A concurrent, on-the-fly, audio programming language. In *Proceedings of the 2003 International Computer Music Conference.* San Francisco: International Computer Music Association.

Winkler, T. (1998). *Composing interactive music.* Cambridge, MA: MIT Press.

Wright, M. & Freed, A. (1997). Open Sound Control: A new protocol for communicating with sound synthesizers. In *Proceedings of the 1997 International Computer Music Conference.* San Francisco: International Computer Music Association.