Active Scores: Representation and Synchronization in Human–Computer Performance of Popular Music

Roger B. Dannenberg, Nicolas E. Gold, Dawen Liang, Guangyu Xia

Computer Music Journal, Volume 38, Number 2, Summer 2014, pp. 51-62 (Article)

Published by The MIT Press

Roger B. Dannenberg,* Nicolas E. Gold,† Dawen Liang, and Guangyu Xia***
*School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, Pennsylvania, 15213 USA
†University College London
Department of Computer Science
Gower Street, London WC1E 6BT, UK
**Department of Electrical Engineering
Columbia University
500 W. 120th Street, Mudd 1310
New York, New York 10027, USA
rbd@cs.cmu.edu, n.gold@ucl.ac.uk,
dliang@ee.columbia.edu, gxia@cs.cmu.edu

Active Scores: Representation and Synchronization in Human–Computer Performance of Popular Music

Abstract: Computers have the potential to significantly extend the practice of popular music based on steady tempo and mostly determined form. There are significant challenges to overcome, however, due to constraints including accurate timing based on beats and adherence to a form or structure despite possible changes that may occur, possibly even during performance. We describe an approach to synchronization across media that takes into account latency due to communication delays and audio buffering. We also address the problem of mapping from a conventional score with repeats and other structures to an actual performance, which can involve both "flattening" the score and rearranging it, as is common in popular music. Finally, we illustrate the possibilities of the score as a bidirectional user interface in a real-time system for music performance, allowing the user to direct the computer through a digitally displayed score, and allowing the computer to indicate score position back to human performers.

Popular music "scores" come in many forms of notation, from a full manuscript score to (more commonly) chord lists or lead sheets. Musicians improvise from these during rehearsal and performance, treating the score as a means of synchronising structure and harmony. Scores are typically sectional, allowing the dynamic reordering of sections during performance. When placed in the context of human–computer music performance (HCMP) of popular music, a computer-mediated approach to the management and representation of scores is required to allow a virtual (computer-based) "performer" to participate appropriately with the humans in the band.

The management and use of notation is a key aspect of HCMP, both for internal software representations of music that support automated performance systems and as a visible interface with which human performers interact (one of a number of usability and adoption aspects of HCMP recently identified, see Gold 2012). This article presents a foundation for coordinating media in multiple modalities, and then it explores two possible approaches to score management for HCMP. The first is a basic score representation language for well-formed HCMP scores (i.e., those that are "parsable" according to normal rules of music notation). Such a language could be used to encode HCMP scores from one of many traditional human-readable formats, e.g., lead-sheet, chord list, or full score. Second, we explore the idea of "notation as interface," based on the architecture first described by Liang, Xia, and Dannenberg (2011), which allows the performer to mark up a scanned score and use the resulting digital version for cueing in rehearsal and performance.

Although this work has many connections to previous work, we introduce several novel ideas. First, we present formulas for synchronizing media in the presence of latency, which may vary across different media and players. The approach is immune to communication latency between a central coordinating "conductor" and distributed "players" by expressing synchronization in terms of quasi-static mappings rather than time-sensitive messages. The methods ensure smooth tempo

Computer Music Journal, 38:2, pp. 51–62, Summer 2014
doi:10.1162/COMJ_a_00239

adjustments, as opposed to sudden jumps, when timing adjustments are required.

Second, we take a detailed look at the relationship between a conventional score and an actual performance. A conventional score may have nested repeated sections and other performance instructions, the interpretation of which can be unclear. Scores may be re-arranged before or during a performance. We describe a novel representation aimed at expressing and formalizing the metrical meaning of a score with respect to its performance.

Finally, we suggest that we can use these techniques to coordinate media with score displays to produce a new form of interactive music system that is well suited to HCMP. In particular, we show that a score can be used as a bidirectional interface in live performance, facilitating bidirectional communication between human and computer musicians.

Foundations of Media Synchronization

A key issue in HCMP is to synchronize media in multiple modalities. Because we assume popular music forms, we also assume a common structure of beats and measures across all media. Thus, time is measured in beats. The basis for synchronization is a shared notion of the current beat (called the *dbeat*, for dynamic beat number) and the current tempo (Dannenberg et al. 2014, this issue). A beat is represented by a floating-point number; hence, beats are continuous rather than integers or messages such as in MIDI clock messages. In addition, rather than update the beat number at frequent intervals, we use a continuous linear mapping from time to beat. This mapping is conveniently expressed using three parameters (b_0, t_0, s):

$$b = b_0 + (t - t_0) \times s \tag{1}$$

where tempo s is expressed in beats per second, at some time in the past beat b_0 occurred at time t_0, the current time is t, and the current beat is b. One could also solve for b_0 when $t_0 = 0$ to eliminate one parameter, but we find this formulation more convenient.

It should be pointed out that whereas Equation 1 (and the equations that follow) express tempo as a constant scale factor s, in practice, we expect frequent tempo estimations, e.g., on every beat, that make slight changes to s. One could handle this by numerical integration (current beat is the integral of tempo), but this leads to the accumulation of error and is not very efficient. One of our concerns will be how to obtain a smooth progression of beat position that synchronizes to external observations of tempo and beat.

One advantage of our approach is that it is almost independent of latency. One can send (t_0, b_0, s) to another computer or process and the mapping will remain valid regardless of the transmission latency. There is an underlying assumption of a shared global clock (t), but accurate clock synchronization is straightforward (Brandt and Dannenberg 1999) and can be achieved independently of media synchronization, thus making the system more modular. When parameters change, there can be a momentary disagreement in the current beat position among various processes, but this should be small given that tempo is normally steady. We will see subsequently how these slight asynchronies can be smoothed and do not lead to long-term drift.

Media players schedule computation to affect the output at specific beat times. For example, an audio player may begin a sample playback at beat 3, or a MIDI player may send a note-on message at beat 5. The current beat time b in Equation 1 refers to the beat position of media that are currently being output, e.g., the beat position corresponding to the current output of a digital-to-analog converter. Time-dependent computation of media must, of course, occur earlier. For example, if the audio output buffer contains 0.01 sec of audio, then computation associated with beat b should be performed 0.01 sec earlier than b. Thus, given a player-specific latency l, we need to compute the real time t at which to schedule a computation associated with beat b. The following formula is easily derived:

$$t = t_0 + (b - b_0) / s - l \tag{2}$$

We simply map the beat position b according to (b_0, t_0, s), and then subtract the latency l to get the computation time t.

Estimating the Mapping

One approach to acquiring a mapping from time to beat is the following: First, a simple foot pedal is used to tap beats. A linear regression over recent taps is then used to estimate the mapping from beat to time (i.e., to estimate t_0, b_0, and s). At this stage, successive beats are numbered with successive integers, but these start at an arbitrary number. Once the tempo and beat phase is established, there must be some way to determine an offset from the arbitrary beat number to the beat number in the score. This might be determined by a cue that tells when the system should begin to play. In other cases, especially with a foot-pedal interface, the system can be constructed to, say, start on the third foot tap (thus the pedal simultaneously fulfills the dual roles of beat acquisition and cueing system).

We believe that audio analysis could also be used to automate beat identification to a large extent (c.f. Robertson and Plumbley 2007), and we are investigating combinations of automated and manual techniques to achieve the high reliability necessary for live performance. The important point here is that some mechanism estimates a local mapping between time and beat position, and this mapping is updated as the performance progresses.

Tempo and Scheduling

Schedulers in computer music systems accept requests to perform specific computations at specific times in the future. Sometimes the specified time can be a "virtual" time in units, such as beats, that are translated to real time according to a (time-varying) tempo, as in Equation 2. Previous architectures for handling tempo control and scheduling (e.g., Anderson and Kuivila 1990) have assumed a fixed and uniform latency for all processing. Under this assumption, there are some interesting fast algorithms for scheduling (Dannenberg 1989). An important idea is that all pending events can be sorted according to beat time and then one need only worry about the earliest event. If the tempo changes, only the time of this earliest event needs to be recomputed. Unfortunately, when event times are computed according to Equation 2, a different event may become the earliest when tempo changes.

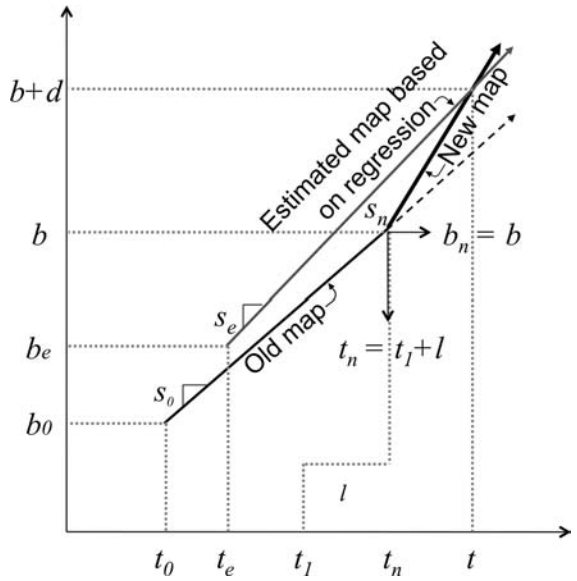
For example, consider an audio player with 0.3-sec latency, a MIDI player with 0.1-sec latency, and tempo $s = 1$ beat per second (BPS). An audio event at beat 1 is scheduled to be computed 0.7 sec in the future so that after 0.3 sec latency it will be heard at exactly 1 sec. A MIDI event at beat 0.7 is scheduled at 0.6 sec. Notice that we will compute the MIDI event first. Now suppose the tempo changes to $s = 2$ BPS. The audio event should now be computed at 0.2 sec and the MIDI event should be performed at $0.7/2 - 0.1 = 0.25$ sec. So, now the audio event must be computed first.

We need, therefore, to rethink scheduling structures of previous systems. The nonuniformity of latency is a real problem in our experience, because audio time-stretching can have a substantial latency due to predetermined overlap-add window sizes, page turning may need to begin seconds ahead of the time of the first beat on the new page, etc.

A second problem is that when the time-to-beat mapping is calculated from linear regression, there can be discontinuities in the time-to-beat-position function that cause the beat position to jump forward or backward instantaneously. Most media players will need to construct a smooth and continuous curve that approximates the estimated time-to-beat mapping. Previous systems have used elaborate rule-based or other models of tempo adjustment, especially for conducting or computer accompaniment where tempo changes might be sudden (Dannenberg 1989). We use a piecewise linear time-to-beat map, adjusting the slope occasionally so that the map converges to the most recent linear regression estimate of the mapping, and our formulation takes latency into consideration.

Figure 1 illustrates this process. The lower line represents an initial mapping according to Equation 1. Imagine that, at time t_1, a new beat has resulted in a new linear regression and a new estimate of the time-to-beat map shown in the upper line. This line is specified by an origin at (t_e, b_e) and a slope (tempo) of s_e beats per second. The problem is that

Figure 1. This graph shows mappings from real time to beat, which specify both tempo and synchronization. While performing according to "Old map," a new "Estimated map based on regression" arrives at t_1. *"New map" is computed to take effect later (due to latency* l*) and make a smooth transition from the old map to the new estimated map. See the text for equations describing "New map."*



switching instantly to the new map could cause a sudden forward jump in beat position. Instead of an instant switch, we want to "bend" our map in the direction of the new estimate. We cannot change the current (lower) map immediately at t_1 because output has already been computed until $t_1 + l$, where l is the latency. For example, if audio output has a 0.1-sec latency, then samples computed for beat position b at time t_1 will emerge at $t_1 + 0.1$. Thus, the earliest we can adjust the map will be at time $t_1 + l$ corresponding to beat b. Let us call the new map parameters t_n, b_n, and s_n. Because the current map passes through $(t_1 + l, b)$, we will choose this point as the origin for the new map (Equations 3, 4, and 5), leaving only s_n to be determined.

$$b = b_0 + (t_1 + l - t_0) \times s_0 \quad (3)$$

$$t_n = t_1 + l \quad (4)$$

$$b_n = b \quad (5)$$

We choose s_n so that the new time map will meet the estimated (upper) time map after d beats, where larger values of d give greater smoothing, and shorter values of d give more rapid convergence to the estimated time map (we use four beats). In practice, we expect a new linear regression every beat or two, depending on how often there is input from a beat detector or foot tap sensor. Thus, the new time map will converge only part of the way to the estimated map before this whole process is repeated to again estimate a new map that "bends" toward the most recent estimate for the time-to-beat map.

To solve for s_n, notice that we want both the upper regression line and the new time map to meet at $(t, b_n + d)$, so we can substitute into Equation 1 to obtain an equation for each line. This gives the two following equations in two unknowns (t and s_n):

$$b_n + d = b_e + (t - t_e) \times s_e \quad (6)$$

$$b_n + d = b_n + (t - t_n) \times s_n \quad (7)$$

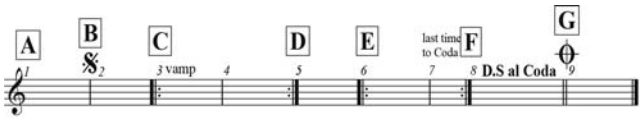
Solving for s_n gives us:

$$s_n = \frac{d}{t_e s_e - t_n s_e - b_e + b_e + d} s_e \quad (8)$$

Under this scheme, each media player sets (b_0, t_0, s_0) to (b_n, t_n, s_n) after each new estimated time map is received, ensuring that the media position converges smoothly to the "ideal" common time map. Because of Equation 3, these parameters depend on latency l, which can differ according to different players. It follows that different media will follow slightly different mappings. This can be avoided, and things can be simplified, by giving all media the same latency. For example, MIDI messages can be delayed to match a possibly higher audio latency. In any case, time-map calculation is still needed to avoid discontinuities that arise as new beat times suddenly change the linear regression, so we prefer to do the scheduling on a per-player basis, allowing each player to specify a media-dependent latency l. Note that (b_n, t_n, s_n) describes the output time for media. Given latency l, computation must be scheduled early according to Equation 2. Equivalently, we can shift the time map to the left by l.

Score Representation

Score representation is important for HCMP because scores can contain repetitions, alternate endings, optional repeats, and cuts. Media may exist only for certain sections of the score. Performers often alter the score, e.g., by improvising an introduction or skipping a verse. Finally, when things go wrong in performance, we would like both the human and



Static Score

```
Decl(A)
M1
End(A)
Begin Repeat (DS)
Decl(B)
M2
End(B)
Decl(C)
Begin Repeat (n)
M3
M4
End Repeat
End(C)
Decl(D)
M5
End(D)
Decl(E)
Begin Repeat (2)
M6
M7
End Repeat
End(E)
Decl Ending (DS,1)
Decl(F)
M8
End(F)
Decl Ending (DS,2)
Decl(G)
M9
End(G)
```

Dynamic Score

```
M1[A1]
M2[B1,A1]
M3[C1-1,B1,A1]
M4[C1-1,B1,A1]
M3[C1-2,C1-1,B1,A1]
M4[C1-2,C1-1,B1,A1]
M3[C1-3,C1-2,C1-1,B1,A1]
M4[C1-3,C1-2,C1-1,B1,A1]
(CUE)
M5[D1,C1-3,C1-2,C1-1,B1,A1]
M6[E1-1,D1,C1-3,C1-2,C1-1,...]
M7[E1-1,D1,C1-3,C1-2,C1-1,...]
M6[E1-2,E1-1,D1,C1-3,C1-2,...]
M7[E1-2,E1-1,D1,C1-3,C1-2,...]
M8[F1,E1-2,E1-1,D1,C1-3,C1...]
M2[B2,F1,E1-2,E1-1,D1,C1-3...]
M3[C2-1,B2,F1,E1-2,E1-1,D1...]
M4[C2-1,B2,F1,E1-2,E1-1,D1...]
M3[C2-2,C2-1,B2,F1,E1-2,E1...]
M4[C2-2,C2-1,B2,F1,E1-2,E1...]
(CUE)
M5[D2,C2-2,C2-1,B2,F1,E1-2...]
M6[E2-1,D2,C2-2,C2-1,B2,F1...]
M7[E2-1,D2,C2-2,C2-1,B2,F1...]
M6[E2-2,E2-1,D2,C2-2,C2-1,...]
M7[E2-2,E2-1,D2,C2-2,C2-1,...]
M9[G1,E2-2,E2-1,D2,C2-2,C2...]
```

Arrangement

```
A, B, C, D, E, F, B, C, D, E, G
```

Figure 2. Four score representations are shown. Common-practice notation (top) is translated directly to a machine-readable static score (left). The arrangement (bottom) shows the nominal interpretation of the score but could be altered to specify a different sequence of sections. The dynamic score (right) gives an expanded measure-by-measure performance sequence. Notice that the number of times section C is repeated is determined by cues during the performance. Also, the performance history preceding each measure is encoded (in reverse order) to provide context information required to perform instructions such as "play second time only on the D.S."

machine performers to recover gracefully. The score can provide a basis for this recovery.

The first approach to score management for HCMP that we present deals with abstract encoding of a score for use in performance. Such encodings need to be simple enough for non-expert users to create and use, but need to allow flexibility for rearrangement during performance. To achieve this, we adopt the notions of a static score, an arrangement, and a dynamic score.

A static score representation must be easy to encode from a printed score or lead-sheet, while also being amenable to arrangement and re-arrangement during performance. In our experience, popular music arrangement typically works by cutting, copying, and inserting whole measures (or sections). Therefore, the representations presented here operate on measures and groups of measures. We believe exceptions, such as pick-up notes, can be handled within this framework by considering partial measures to be part of the following measure. Although the language is, in essence, a formal programming language, it is intended to be representational (an artificial domain embedded language, see Gold 2011) with a clear correspondence to common-practice music notation.

Static Score

Figure 2 shows a short score fragment that will be used to illustrate the encodings proposed. The rehearsal letters designate sections of the piece. The fragment contains a number of structural complexities including a vamp repeat (section C) to be repeated as desired by the performers, a traditional repeat, and a *dal segno* (D.S.) repeat with coda. The corresponding static score representation consists of block declarations (`Decl(a)`) and terminations (`End(a)`), numbered measures (`Mx`), repeat declarations (numbered, unnumbered, dal segno), repeat terminations, and alternative ending declarations and terminations. This representation language allows the abstract structure of a score to be encoded without being concerned with the note-level specification of the musical material. The language for the static score thus encodes the score as written at the measure level and attaches sectional labels to groups of measures. Note that musicians need not learn or even be aware of this representation language, because it can be presented to the user in terms of music notation or other graphical representations.

Arrangement

The arrangement representation uses the sectional labels declared by the static score to specify the order of the sections to be performed. This is equivalent to the musicians noting the sectional structure of a song (e.g., intro, verse, chorus). It allows for easy rearrangement during rehearsal and performance,

simply by changing the section ordering and regenerating the dynamic score. An example arrangement based on the normal reading of the score (not a "re-arrangement") is shown in Figure 2.

Dynamic Score

The dynamic score provides a measure-level unfolding of the static score in accordance with the arrangement. It encodes a performance history using section names and the number of times they have been played (e.g., section C the first time through would be encoded C1). Where sections contain repeats these are indicated by hyphenated occurrence numbers (e.g., the second repeat of the first time through section C would be encoded C1-2. This allows a system to restart unambiguously from any point in the performance history and cue appropriate metadata.

Once an arrangement has been created, the measures to be played can be specified (as Mx where x is the measure number) in readiness for the rendering systems to schedule their data. Because it is important to be able to navigate through a piece during rehearsal (e.g., to respond to directions such as "let's go from the second time through section E"), each measure is attached to a state vector (in square brackets) describing the sectional progress (in reverse order) of the piece to that point.

This captures the notion of the dynamic score being both a prescription of what is to be played and subsequently a history of what has been played. Figure 2 shows a possible dynamic score for the example fragment and arrangement shown in the figure. This is a post-performance dynamic score, because pre-performance, the number of iterations of section C (the vamp section) cannot be known and it is only the receipt of a cue (as marked in the dynamic score) that causes the remainder of the score to be written as far as possible (until the next vamp is encountered). Unbounded repeats like this are counted during performance to support rehearsal direction (e.g., "twice through the vamp and then on"). In works without non-deterministic repeats, the entire dynamic score could be produced before the performance begins.

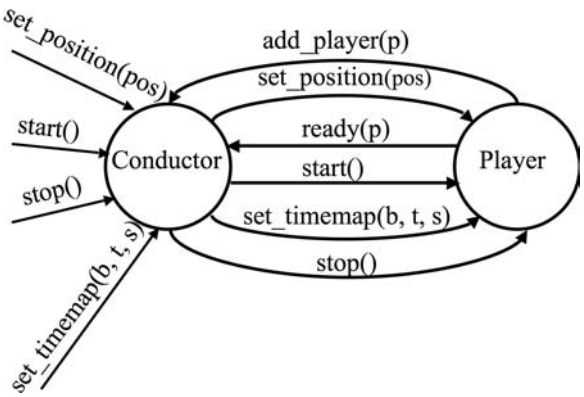


Figure 3. Interfaces for Conductor and Player objects include commands from sensors and user interfaces (left) and messages used by the Conductor to coordinate multiple instances of the Player class (center). The messages between Conductor and Player are shown in typical order from the top down.

Conductor and Players: An Instance of HCMP Architecture

Our second approach to score management involves the use of the score as an interface. We first describe an instance of HCMP architecture that supports the system.

We have implemented an HCMP system organized as a set of "Player" objects that interact with a "Conductor" object that controls the players. The Conductor provides a central point for system control. The Players also use a real-time scheduler object to schedule computation according to Equation 2. The interface and interaction between the Conductor and Players is illustrated in Figure 3.

The Player Class

A Player is any object such as an audio or MIDI sequencer that generates output according to the current tempo and beat position (a rendering system in terms of the architecture in Dannenberg et al. [2014]). A Player can also generate visual output, including page turning for music notation or an animated display of the beat.

Every Player object implements four methods used for external control: `set_position(pos)`, `start()`, `stop()`, and `set_timemap(b, t, s)`. The `set_position(pos)` method is a command to prepare to output media beginning at beat position pos. This may require the Player to preload data or to send certain data, such as MIDI controller

messages or a page of music notation, as output. The `start()` method is a command to begin output according to the current tempo and the mapping from time to beat position. The playback can be stopped with the `stop()` command. Note that stopping (sound will cease, displays indicate performance has finished) is different from setting the tempo to zero (sound continues, displays are still active), so we need explicit start and stop signaling. The `set_timemap(b, t, s)` method updates the mapping from real time to beat position, by changing it to the linear function that passes through beat *b* at time *t* with slope *s* (in beats per second).

Note that the external interface to Player objects concerns time, beats, and control, but says nothing about media details. In this way, new players can be added in a modular fashion, and the details of player operation can be abstracted from the overall system control. We will see in the subsequent section, Coordination of Media, how the beat position is mapped to media content.

The Conductor Class

The role of a Conductor is to provide a single point of control and synchronization for all players. The Conductor methods include the same `set_position(pos)`, `start()`, `stop()`, and `set_timemap(b, t, s)` methods as do Player objects. These methods are to be used by higher-level control objects. For example, a graphical user interface may have a conventional play/stop/pause/rewind interface implemented by Conductor methods. Alternatively, a more intelligent system might use automatic music listening, gestures, or other ways to determine when and where to start and stop. In addition, an `add_player(p)` method allows new Player objects to add themselves to the list of Players managed by a single Conductor.

Scheduling

We assume the existence of a real-time scheduler object (Dannenberg 1989) to be used by Players. A typical Player has computation to perform at specific beat times. Usually, a computation will perform some action needed at the present time, followed by the scheduling of the next action. The scheduler's role is to keep track of all pending actions and to invoke them at the proper time, thus eliminating the need for Players to busy-wait, poll, or otherwise waste computer cycles to ensure that their next computation is performed on time. Players use Equation 2 to determine the real time *t* at which to perform an action scheduled for beat position *b*.

Coordination of Media

An important feature of the framework is that it coordinates media of different forms (MIDI, audio, score, etc.) in real-time performance. As introduced earlier, the framework is based on a shared notion of beat position, i.e., all the players controlled by the Conductor share the same beat position. The beat information for most MIDI is easy to extract because it is normally encoded in a Standard MIDI File.

For audio, we must have auxiliary information that encodes a mapping from beat position to audio time. This mapping may be constructed by manual tapping or automatic alignment (Dannenberg and Raphael 2006) to audio or MIDI for which beat times are known.

For music notation, structured score documents such as MusicXML (Castan, Good, and Roland 2001) have all the information needed to map from beats to page numbers and positions, but for simplicity, we use scanned images and let users label the start position of each measure manually. Optical music recognition, combined with symbolic-music-to-audio alignment, is another promising approach to label scanned music notation (Kurth et al. 2007).

Distributed Computation

The framework supports distributed computation or computation in separate threads on multi-core computers. Coordination and synchronization is often difficult in distributed systems because of unknown communication latency. In our approach, communication latency is not critical. Communication latency certainly affects the responsiveness

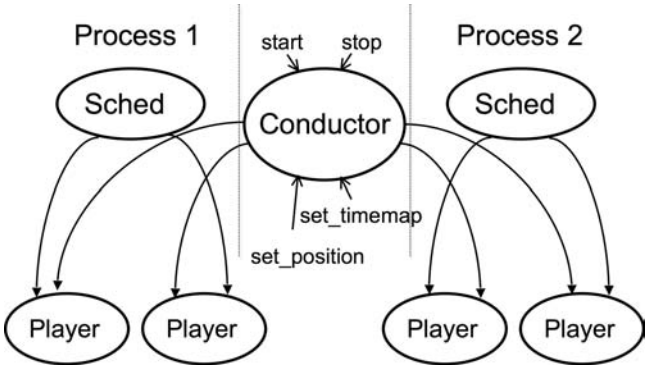


Figure 4. In a distributed message-based implementation, the Conductor communicates with Player instances over a network. Local scheduler ("Sched") objects enable Players to deliver accurately timed output. A clock synchronization protocol ensures that local clocks are synchronized. These design features substantially mask any effects of network latency.

of the system, but unless tempo changes drastically, beat positions are predictable in the near future. Instead of transmitting beat times, we transmit mappings from global time to beat position. These mappings are expressed with respect to a shared global clock, and they do not change even if their delivery is delayed. Any two processes that agree in terms of their real clock time and their mapping (t_0, b_0, s) will agree on the current beat position.

In a distributed implementation, the Conductor communicates via (reliable) messages with Players, and Players rely on local schedulers to activate timed computations (see Figure 4). If the schedulers are on separate computers, the computer real-time clocks must use a clock synchronization protocol to ensure that every scheduler agrees on the real clock time.

We have found it easy to synchronize clocks at the application level. For example, designated slave machines send a request to a master for the time, and the master time is returned. This round trip time is usually less than a few milliseconds, and the slave can set its clock assuming a communication latency of half the round trip time. This can easily produce synchronization to within 1 msec. If the round trip time is longer than normal, the slave simply assumes that an unexpected network delay has made the result unreliable, ignores the result, and tries again. Techniques that are more elaborate, based on averaging and estimating clock drift, can even synchronize clocks to microseconds if needed (Brandt and Dannenberg 1999).

Notation as Interface

The electronic display of music is not a new idea (Connick 2002; Kurth et al. 2007; Bainbridge and Bell 2009; MakeMusic 2013), but here we describe our use of active music notation as a bidirectional human–computer interface. Olmos and coworkers (2012) aptly describe this as "Score-Centric Control" in the context of their Open Orchestra system.

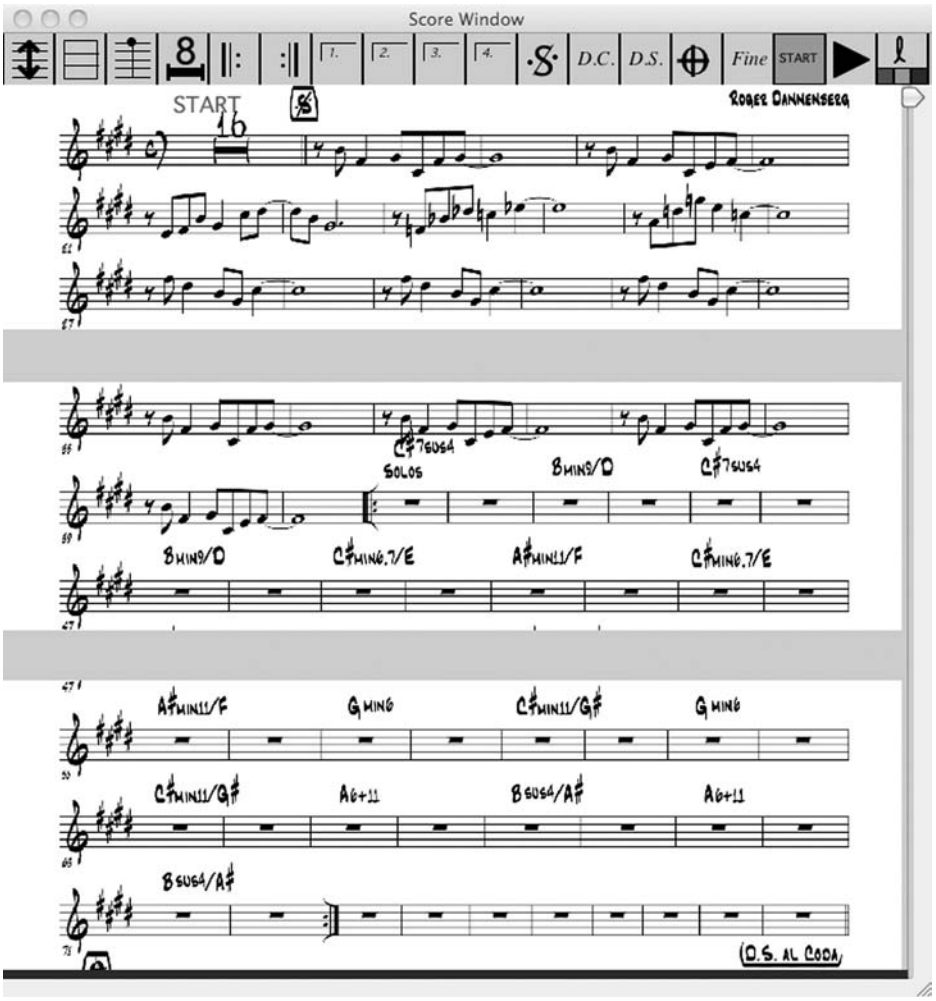
Location Feedback and Page Turning

In an interactive music system where synchronization is key, it is important for performers to communicate their coordination with the group. For example, when it is time for a guitar solo, the vocalist and guitarist might look at each other to acknowledge that both musicians expect the solo. If the vocalist's gestures instead indicate he or she will sing another chorus, the guitarist might hold off until later. In a similar way, it is important for the computer to signal its current position to human players so that they can either adapt to the computer or provide some override to steer the computer back into synchronization.

Music notation provides an attractive basis for communication, because it provides an intuitive and human-readable representation of musical time, it is visual (so that it does not interfere with music audio), and it provides both history and look-ahead that facilitate planning and synchronization. The computer can communicate its location to human performers by displaying a moving marker over a digital display of the score. We have already discussed how score display can be coordinated with MIDI and audio performance. Human musicians can then notice when the measure they are reading does not correspond to the measure that is highlighted and take corrective action.

Another possibility is automatic page turning, which was introduced in early computer accompaniment systems. For example, SmartMusic (MakeMusic 2013) uses the Finale notation engine to show scores and score position in real time as it follows a soloist in the score. In our framework, page turning

Figure 5. Score display showing editing toolbar (top) and a vertical division into thirds. The divisions allow incremental updates so that the performer can always see the current location and at least 1/3 page ahead.



is easily controlled by the Conductor. Just like scheduling an event from the MIDI player, the score player can also schedule a "scrolling-up" event.

Various schemes have been implemented for "page turning" on a display screen of limited size. It is well known that musicians read ahead, so it is essential to display the current music as well as several measures in the future. The most common approach is to split the screen into top and bottom halves. While the musician reads one half, the computer updates the other half to the next system (or systems) of music. Other solutions include scrolling up at a constant speed, scrolling up by one system when it is finished, scrolling at a variable speed that is proportional to the tempo, and horizontal scrolling of an "infinitely wide" score. Our implementation presented here displays multiple "slices" of the score on the screen (see Figure 5).

Selecting Locations from Notation

In addition to affording computer-to-human feedback, music notation can be used as an "input device," for example to indicate where to begin in a rehearsal. Our system has start positions for every measure stored as coordinates (*page, x, y*). When we

point to the position where we would like to start (either with a finger or an input device), the system can map the position to a beat number and use the Conductor's `set_position()` method to prepare all Players to start from that location. This action will also display a visual indicator of the position in the score, giving a confirmation to the user that the correct location was detected.

Implementation

We have prototyped components of the HCMP architecture in Serpent (Dannenberg 2002), a real-time programming language inspired by Python. Our system follows the approach described earlier, with classes Conductor, Player, and Time_map. Subclasses are derived from the Player class to form Midi_player, Score_player (a music notation display program), and Posn_player (which displays the current position). Each subclass of Player implements methods for `set_position()`, `start()`, `stop()`, and each inherits a method for `set_timemap()` that adjusts the local Player's time map to converge to that of the Conductor.

The score player class is the most complex (about 2,400 lines of Serpent code). It displays music notation, automatically turning "pages" according to score position given by the conductor. The music notation comes from image files (e.g., JPEG or PNG), which are manually annotated. The score player includes graphical annotation tools to: (1) indicate the staff height; (2) subdivide the score into systems; (3) mark bar lines; (4) mark repeat signs, endings, D.S., coda, and fine; (5) mark a starting measure; and (6) add arbitrary free hand and text annotations (see Figure 5).

After annotating the score, the score player sorts measures, repeats, and other symbols to form its internal representation of the static score. It can then compute a dynamic score by "unfolding" the repeats and computing a list of dynamic measures. Formalizing this process is the subject of a recent paper (Jin and Dannenberg 2013). The score player also scales the music notation images to fit the width of the display and divides the images into slices that are stacked vertically on the display.

There are many possibilities for music scrolling and page-turning. In the current implementation, we divide the screen into thirds and always display the previous, current, and next sub-pages. For example, the initial display shows the first three sub-pages, in the order 1-2-3. When the player object advances to the third sub-page, the display is updated to show 4-2-3. The player object continues reading sub-page 4 at the top of the display, at which time the display updates to 4-5-3, etc.

Evaluation

To our knowledge, there are no comparable systems that would enable a quantitative evaluation, but we can make a qualitative comparison between our work and many other related systems. Computer accompaniment (Dannenberg and Raphael 2006; MakeMusic 2013) cannot synchronize to performances with significant amounts of improvisation. Fixed media approaches such as Open Orchestra (Olmos et al. 2012) do not adjust tempo to synchronize to live musicians. Conducting systems (Katayose and Okudaira 2004) require the full attention of a human conductor to manage synchronization with live musicians. Interactive music systems to date are mostly designed to generate music in response to live inputs rather than play predetermined parts. Thus, they are not capable of performing conventionally notated compositions. Perhaps the work most closely related to ours is B-Keeper (Robertson and Plumbley 2007, 2013). Because B-Keeper relies on audio analysis for beat tracking, it is restricted to drumming, for which beat tracking is successful. This rules out much of the jazz idiom in which we have been working, at least until beat-tracking methods improve. Further evaluation, including current HCMP approaches, is discussed by Dannenberg and colleagues (2013). Overall, our work satisfies a set of interesting and practical musical requirements that have not been previously addressed.

Evaluation of software techniques is difficult because there are few data points and many extraneous factors. In our experience, scheduling based on time maps, as described here, offers a highly

effective approach to reasoning about timing. The main advantage is that problems can be addressed independently, in a modular fashion: What is the estimated actual tempo? How should performed tempo be adjusted to obtain synchrony? How can we compensate for real-time clock drift on separate systems? Given an event sequence specified according to beats, what is the real time of the next event? Each of these problems is handled in isolated software modules, making the software much easier to construct.

Working performance systems can be viewed through online video at www.youtube.com/watch?v=J_Z1GSltMPw and www.youtube.com/watch?v=R11u0S6uENA. The first example, described in a companion article (Dannenberg et al. 2014), is a large-scale performance with a live jazz band and a virtual string orchestra. The second shows a smaller ensemble (a quartet) where the trumpet player uses HCMP to add harmony and counterpoint to a melody.

Conclusions

Human–computer music performance has usually been explored in the context of experimental computer music, but we are only beginning to consider the possibilities of computers as "live" musicians performing popular music. Popular music poses interesting challenges for synchronization and music representation. We have described a modular implementation that synchronizes multiple media in the face of tempo changes and different amounts of latency.

Common-practice music notation with repeats and other structures (which we call *static scores*) must be reconciled with the "unfolded" linear representation (*dynamic scores*) seen in audio files, standard MIDI files, and the live performance itself. HCMP systems must also allow for changes at or near performance time. Musicians should be free to make new "arrangements" that alter the structure implied by the static score. We have suggested a representation to handle these requirements.

Musicians also need intuitive interfaces to communicate with HCMP systems. We described one interface based on music notation. The most interesting aspect of the interface is its bidirectional nature. The display can indicate the computer's position and future intentions (what music is next). At the same time, musicians can reset the computer's position or interactively give cues using a pointing device or touch-sensitive display.

We have built and used prototypes of the systems described here. In the future we aim for greater flexibility, synchronization to live players that is more accurate, improved sound, and tools to make HCMP "content" easier to develop and use.

Acknowledgments

Support for this work by the UK Engineering and Physical Sciences Research Council (grant no. EP/F059442/2) and the National Science Foundation (grant no. 0855958) is gratefully acknowledged. Our first performance system and the music display work were also supported by Microsoft Research. Thanks to Ryan Calorus, who implemented a precursor to the music display system described here. Portions of this article are based on earlier publications (Dannenberg 2011a, 2011b; Gold and Dannenberg 2011; Liang, Xia, and Dannenberg 2011).

References

Anderson, D., and R. Kuivila. 1990. "A System for Computer Music Performance." *ACM Transactions on Computer Systems* 8(1):56–82.

Bainbridge, D., and T. Bell. 2009. "An Ajax-Based Digital Music Stand for Greenstone." In *Proceedings of the ACM/IEEE-CS Joint Conference on Digital Libraries*, pp. 463–464.

Brandt, E., and R. Dannenberg. 1999. "Time in Distributed Real-Time Systems." In *Proceedings of the International Computer Music Conference*, pp. 523–526.

Castan, G., M. Good, and P. Roland. 2001. "Extensible Markup Language (XML) for Music Applications: An Introduction." In *The Virtual Score*. Vol. 12 of *Computing in Musicology*. Cambridge, Massachusetts: MIT Press, pp. 95–102.

Connick, H., Jr. 2002. System and Method for Coordinating Music Display among Players in an Orchestra. US Patent 6,348,648, filed 23 November 1999, and issued 19 February 2002.
Dannenberg, R. 1989. "Real-Time Scheduling and Computer Accompaniment." In M. V. Mathews and J. R. Pierce, eds. *Current Directions in Computer Music Research*. Cambridge, Massachusetts: MIT Press, pp. 225–261.
Dannenberg, R. 2002. "A Language for Interactive Audio Applications." In *Proceedings of the International Computer Music Conference*, pp. 509–515.
Dannenberg, R. 2011a. "A Vision of Creative Computation in Music Performance." In *Proceedings of the International Conference on Computational Creativity*, pp. 84–89.
Dannenberg, R. 2011b. "A Virtual Orchestra for Human–Computer Music Performance." In *Proceedings of the International Computer Music Conference*, pp. 185–188.
Dannenberg, R., et al. 2013. "Human–Computer Music Performance: From Synchronized Performances to Musical Partner." In *Proceedings of the Sound and Music Conference*, pp. 277–283.
Dannenberg, R., et al. 2014. "Methods and Prospects for Human–Computer Performance of Popular Music." *Computer Music Journal* 38(2):36–50.
Dannenberg, R., and C. Raphael. 2006. "Music Score Alignment and Computer Accompaniment." *Communications of the ACM* 49(8):38–43.
Gold, N. 2011. "Knitting Music and Programming." In *Proceedings of the IEEE International Working Conference on Source Code Analysis and Manipulation*, pp. 10–14.
Gold, N. 2012. "A Framework to Evaluate the Adoption Potential of Interactive Performance Systems for Popular Music." In *Proceedings of Sound and Music Computing Conference*, pp. 284–289. Available at smcnetwork.org/system/files/smc2012-155.pdf. Accessed December 2013.
Gold, N., and R. Dannenberg. 2011. "A Reference Architecture and Score Representation for Popular Music Human–Computer Music Performance Systems." In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 36–39.
Jin, Z., and R. Dannenberg. 2013. "Formal Semantics for Music Notation Control Flow." In *Proceedings of the 2013 International Computer Music Conference*, pp. 85–92.
Katayose, H., and K. Okudaira. 2004. "Using an Expressive Performance Template in a Music Conducting Interface." In *Proceedings of the Conference on New Interfaces for Musical Expression*, pp. 124–129.
Kurth, F., et al. 2007. "Automated Synchronization of Scanned Sheet Music with Audio Recordings." In *Proceedings of the International Symposium on Music Information Retrieval*, pp. 261–266.
Liang, D., G. Xia, and R. Dannenberg. 2011. "A Framework for Coordination and Synchronization of Media." In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 167–172.
MakeMusic, Inc. 2013. "SmartMusic: Music Education Software." Available at www.smartmusic.com. Accessed 22 October 2013.
Olmos, A., et al. 2012. "A High-Fidelity Orchestra Simulator for Individual Musicians' Practice." *Computer Music Journal* 36(2):55–73.
Robertson, A., and M. Plumbley. 2007. "B-Keeper: A Beat-Tracker for Live Performance." In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 234–237.
Robertson, A., and M. Plumbley. 2013. "Synchronizing Sequencing Software to a Live Drummer." *Computer Music Journal* 37(2):46–60.