A Framework for Coordination and Synchronization of Media

Dawen Liang
Carnegie Mellon University
School of Music
5000 Forbes Ave, Pittsburgh, PA
dawenl@andrew.cmu.edu

Guangyu Xia
Carnegie Mellon University
School of Computer Science
5000 Forbes Ave, Pittsburgh, PA
gxia@cs.cmu.edu

Roger B. Dannenberg
Carnegie Mellon University
School of Computer Science
5000 Forbes Ave, Pittsburgh, PA
rbd@cs.cmu.edu

ABSTRACT

Computer music systems that coordinate or interact with human musicians exist in many forms. Often, coordination is at the level of gestures and phrases without synchronization at the beat level (or perhaps the notion of "beat" does not even exist). In music with beats, fine-grain synchronization can be achieved by having humans adapt to the computer (e.g. following a click track), or by computer accompaniment in which the computer follows a predetermined score. We consider an alternative scenario in which improvisation prevents traditional score following, but where synchronization is achieved at the level of beats, measures, and cues. To explore this new type of human-computer interaction, we have created new software abstractions for synchronization and coordination of music and interfaces in different modalities. We describe these new software structures, present examples, and introduce the idea of music notation as an interactive musical interface rather than a static document.

Keywords

Real-time, Interactive, Music Display, Popular Music, Automatic Accompaniment, Synchronization

1. INTRODUCTION

Computer music systems have been used extensively in interactive performances of cutting-edge electro-acoustic music, and also in some advanced systems that model the traditional role of the accompanist in Western art (or "classical") music [13]. In the realm of popular music, computers have had their largest impact through new instruments (almost every electronic instrument now has some sort of embedded computer). The concept of "instrument" has been extended to include the laptop computer, especially in loop-based music related to the DJ phenomenon. We believe that there are untapped possibilities in more traditional popular music forms such as rock, jazz, and folk music. There are opportunities here for innovative applications of highly intelligent and coordinated computer music systems [11]. In both rehearsal and live performance, computers could contribute to make new sounds possible, fill in for missing musicians, and ultimately to inspire new musical directions based on new capabilities and concepts from new technologies.

To bring computers into the realm of popular music performance, certain problems must be addressed. The main problem is that popular music timing is organized around a tight synchronization to beats. When live musicians are involved, the tempo is not perfectly steady, and humans have a difficult time synchronizing to an unyielding computer time-keeper. At the same time, computers cannot reliably adapt to human tempo variations. Another significant problem is the improvisation and decision-making that goes on in many live performances. It would be simple to prepare computers with fixed sequences, but what happens when the vocalist comes in a measure late or the bandleader signals to play another chorus? These problems are even more difficult given the amount of structure in popular music. Musicians and their audience know when performers are tightly synchronized in terms of rhythm and harmony. We cannot expect computers to improvise freely or "play by ear." Instead, they must understand, communicate, and synchronize at the level of beats, measures, and pre-determined musical structure such as sections and chord progressions.

Imagine a popular music performance system that could play different representations of music including MIDI, audio, guitar tabs, etc. as accompaniment and quickly adjust its tempo to follow the performer. Furthermore, the performance system could display an image of the score and automatically turn pages. In rehearsals, the computer could cover missing parts, especially for individual practice, and in live performance the computer could play additional parts not covered by human performers. The computer could be directed in part by pointing to locations in the score image, and the computer could confirm its location or intention to play by highlighting locations in the score.

To create such a system, we must coordinate time among different media. We would like to do this systematically and modularly so that new media can be added to the system without rewriting all the low-level, time-critical software. For example, one might want to synchronize video or lyrics to live music. How would this fit into an audio framework? This paper presents a flexible, beat-based "virtual time" framework to meet this challenge. One of the interesting aspects of this work is the two-way coordination of a visual score with a live computer performance, creating an interesting human-computer interface. Using a music notation display, the human can direct the performance to a location in the score, and the computer can give feedback to the human as to the current score location.

The next section presents related work. Section 3 describes how synchronization is achieved by scheduling computation according to piece-wise linear maps between time and beat

position. Techniques to keep these maps smooth in the face of latency and changing tempo estimates are presented. Section 4 describes a modular software framework for controlling multiple synchronized media player objects. Section 5 discusses the use of music notation as a bi-directional graphical interface for controlling music performance and monitoring the status of a computer performer. Our current implementation is discussed in Section 6. Finally, conclusions are presented in Section 7.

2. RELATED WORK

Much work has been done in the area of music performance systems. For example, automatic accompaniment systems for classical music performance [8], [9], [17], [18] and real-time music composition and performance systems [20] have been used and studied for many years. Related work exists in the area of music conducting. The work by Lee, Karrer, and Borchers [16] is especially relevant to our work in its discussion of synchronization of beats and smooth time map adjustment, and recent work [3], [14] discusses both tempo adjustment and synchronized score display, using an architecture similar to ours. However, the particular problems of popular music seem largely to be ignored. Of course, one simple way to incorporate computers in live popular music performance is to change the problem: humans can adapt to the steady time of the computer by listening to drums or a click track, and a fixed structure enables computers to play fixed sequences. Ableton Live [1] is an example of software that uses a beat, measure, and section framework to synchronize music in live performance, but the program is not well-suited to adapting to the tempo of live musicians. Robertson and Plumbley used a real-time beat tracker in conjunction with Ableton Live software to synchronize pre-recorded music to a live drummer [19]. Our goal is to create a more autonomous "artificial performer" that does not require a human operator sitting at a computer console, but rather uses more natural interfaces for direct control and more sophisticated listening and sensing for indirect control.

3. MEDIA SYNCHRONIZATION

The main role of our architecture 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 and the current tempo. Beats are represented by a floating point number, hence they are continuous rather than integers or messages such as in MIDI clock messages. Also, 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):

$$b = b_0 + (t - t_0) \times s \quad (1)$$

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.

One advantage of this 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 [5] 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 time among various processes, but this should be small given that tempo is normally steady. We will see below how these slight asynchronies can be smoothed and do not lead to long-term drift.

In our system, 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 Eq. 1 refers to the beat position of media which are being output currently, e.g. the beat position corresponding to the current output of a digital-to-analog converter (DAC). Time-dependent computation of media must of course occur earlier. For example, if the audio output buffer contains $0.01s$ of audio, then computation associated with beat b should be performed $0.01s$ 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:

$$t = t_0 + (b - b_0) / s - l \quad (2)$$

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.

3.1 Estimating the Mapping

Our current system relies on a simple foot pedal to tap beats. A linear regression over recent taps is 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.

We believe that audio analysis could be used to automate beat identification to a large extent, 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.

3.2 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 (possibly varying) tempo, as in Eq. 2. Previous architectures for handling tempo control and scheduling [2] have assumed a fixed and uniform latency for all processing. Under this assumption, there are some interesting fast algorithms for scheduling [9]. An important idea is that all pending events (callbacks) 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 Eq. 2, the earliest pending event can change when tempo changes. Therefore, we need to rethink scheduling structures of previous systems. The non-uniformity of latency is a real issue in our experience because audio time-stretching can have a substantial latency due to pre-determined overlap-add window sizes, page turning might 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. We do this using a piece-wise linear time-to-beat map, adjusting the slope occasionally so that the map converges to the most recent linear regression estimate of the mapping.

Figure 1 illustrates this process. The lower line represents an initial mapping according to Eq. 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 switching instantly to the new map could cause a sudden 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 0.1s 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. Since the current map passes through (t_1+l, b), we will choose this point as the origin for the new map (Eqs. 3, 4, 5) leaving only s_n to be determined.

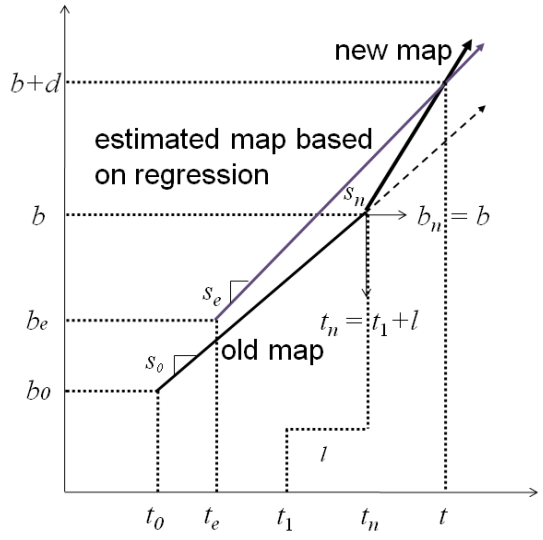

Figure 1. Modifying the local time-to-beat mapping upon receipt of a new regression-based mapping estimate.

$$b = b_0 + (t_1 + l - t_0) \times s_0 \quad (3)$$
$$t_n = t_1 + l \quad (4)$$
$$b_n = b \quad (5)$$

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 4 beats.) In practice, we expect a new linear regression every 2 beats (cut time), thus the new time map will only converge about half way to the estimated map before this whole process is repeated to again estimate a new map that "bends" toward the most recent time-to-beat map estimate.

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 Eq. 1 to obtain an equation for each line. This gives two equations (Eqs. 6, 7) 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 Eq. 8:

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

Under this scheme, we set (b_0, t_0, s_0) to (b_n, t_n, s_n) after each new estimated time map is received. Because of Eq. 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 Eq. 2. Equivalently, we can shift the time map left by l.

4. MODULAR STRUCTURE

Our system is 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 Eq. 2. The interface and interaction between the Conductor and Players is illustrated in Figure 2.

4.1 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 Player can also generate visual output, including page turning for music notation or an animated display of the beat.

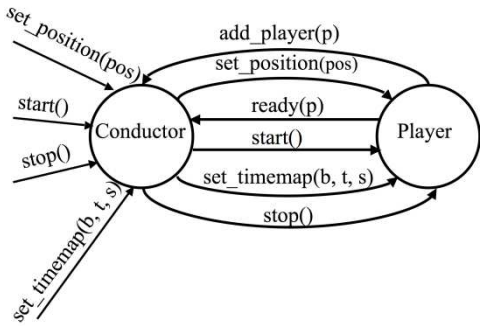

Figure 2. Interfaces for Conductor and Player objects.

Every "player" 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 pre-load data or to output certain data such as MIDI controller messages or a page of music notation. The s*tart*() 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 sustains, 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 to the linear function that passes through beat b at time t with slope s (in beats per second). This is how the new linear regression data (t_e, b_e, s_e) described in the previous section is transmitted to each Player.

Note that the external interface to Players 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.

4.2 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 that is 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.

4.3 Scheduling

We assume the existence of a real-time scheduler object [9] 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 Eq. 2 to determine the real time t at which to perform an action scheduled for beat position b.

4.4 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. In this section, we will discuss the details of time synchronization.

4.4.1 Shared Time System

As introduced in Section 2, 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. Audio and score images are more problematic.

For audio, we must have auxiliary information that encodes a mapping from beat position to audio time. An audio Player can then use time-stretching algorithms to adjust the audio speed to synchronize to a live performance. The audio can be recorded at constant tempo, e.g. using a click track, so that beat positions can be calculated directly from the known tempo. Alternatively, audio can also be labeled by manual tapping, by automatic beat tracking (for music where this is possible), or by automatic alignment [13] to audio or MIDI for which beat times are known.

For music notation, we can use structured documents such as MusicXML [6] or unstructured scanned images. In principle, structured score documents have all the information needed to map from beats to page numbers and positions, but in practice, rendering music notation is difficult and there is no readily available software that can be adapted to our purpose. Instead, we let users indicate the time signature (or by default 4/4) and manually label the start position of each measure to construct a mapping from beats to image position. Using this information, the score Player can convert beat positions to approximate page locations. In the future, we hope to adapt some optical music recognition (OMR) software to detect systems and bar lines to speed up the process of annotating score images. OMR combined with symbolic music to audio alignment is another promising approach to label scanned music notation [15].

4.4.2 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 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 3). 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 1ms. 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. More elaborate techniques based on averaging and estimating clock drift can even synchronize clocks to microseconds if needed [5].

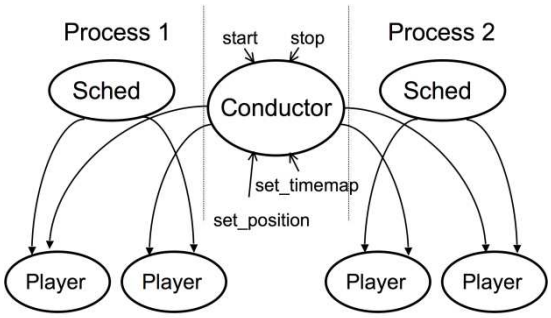

Figure 3. Distributed message-based implementation.

4.4.3 Static vs. Dynamic Scores

Even after providing mappings from beat position to specific time and spatial coordinates of different media, there is an important difference between scores and most other media that we must deal with. Scores are a bit like programs that must be "executed" to determine a music performance. Repeats and the "dal segno al coda" are forms of looping behavior. First and second endings and the coda are forms of conditional behavior based on the loop count. Thus, the score is a "static" representation of music in the sense of static code, and an audio file or MIDI file is a "dynamic" representation of music in the sense of dynamic or run-time program behavior. Because of repetition, there is a one-to-many association between static score position and dynamic beat position. Our current system implements conventional music control

structures, but real scores often resort to informal instructions that are difficult to formalize.

5. NOTATION AS INTERFACE

The idea of electronic display of music is not a new idea [4], [7] [15] [17], but we introduce the notion of active music notation as a bi-directional human-computer interface.

5.1 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 facilitates planning and synchronization. Given a mapping from beat position to score image location, it is easy to display the real-time beat position directly on an image of the score. 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 [17] 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 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(s) 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 which is proportional to the tempo, and scrolling an "infinitely wide" score horizontally. Our implementation presents multiple "slices" of the score on the screen (see Figure 4), but we plan to experiment with different approaches.

5.2 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, 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 will also indicate the position in the score, giving a confirmation to the user that the correct location was detected.

6. IMPLEMENTATION

We have implemented a prototype system in Serpent [10], a real-time programming language inspired by Python. Our system follows the architecture described earlier, with classes *Conductor*, *Player*, and *Time_map*. The *Player* class is subclassed to form *Midi_player*, *Score_player* (a music notation display program), and *Posn_player* (to display the current position). Each player implements methods for *set_position*, *start*, *stop*, and they all inherit a method for *set_timemap* that adjusts each local player time map to converge to that of the conductor.

The score player class is the most complex (about 2400 lines of Serpent code). It displays music notation, turning "pages" automatically 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 4.)

After annotating the score, the score player sorts measures, repeats, and other symbols to form a representation of the *static* score. It can then compute a *dynamic* score by "unfolding" the repeats and computing a list of dynamic measures. 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 3 sub-pages, in the order 1-2-3. When the player advances to the third sub-page, the display is updated to show 4-2-3. The player continues reading sub-page 4 at the top of the display, at which time the display updates to 4-5-3, etc.

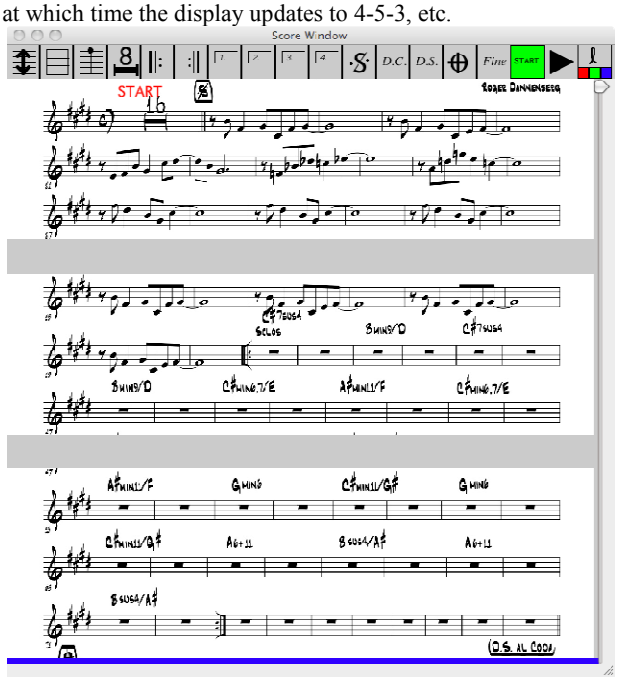

Figure 4. Score display showing editing toolbar at top and a vertical division into thirds.

We have also implemented a player for multi-channel audio that applies high-quality time stretching to each channel according to a time map [12]. However, this system is not yet integrated into the conductor/player framework. Finally, we have implemented a tempo control object that accepts beats from a space-bar or foot pedal, rejects outliers, and performs linear regression on recent taps to estimate a time map.

7. CONCLUSIONS AND FUTURE WORK

In conclusion, our framework implements a beat-based strategy for coordination and synchronization of media in real-time performance. We convert the music notation from images of score to a dynamic interactive display medium interface, which could cooperate with other forms of media to create a human-computer music performance system.

In the future, we plan to focus on applications that integrate music notation with audio playback, gaining practical experience using music notation as an interactive medium. We plan to test and evaluate different methods of music scrolling and assess whether music notation on a touch display can be used to control a computer musician during real performances.

Considering the complexity for prototyping and testing, our most recent system is based on MIDI and score images. As the framework structure becomes mature, we will integrate audio playback using PSOLA [21] and phase vocoder [16] time-stretching techniques. We will also need to implement editing techniques to label audio.

The scheduling and music notation framework described here is part of a larger overall architecture that facilitates music representation, preparation, and performance, and there are many more components required to achieve the kind of music production and performance flexibility that we envision. However, even based on this framework, many applications can be developed. For example, we could automatically align rehearsal recordings to MIDI files to quickly (and roughly) label audio. Then, we could listen to particular parts by pointing to the score, comparing the rehearsal performance of the same piece from two different days, etc. The flexibility of the framework provides many possibilities for future work.

8. ACKNOWLEDGMENTS

Thanks to Ryan Calorus, who implemented our first experimental music display, and Nicolas Gold for valuable discussions. Our first performance system and the music display work were supported by Microsoft Research and the Carnegie Mellon School of Music. Zplane kindly contributed their high-quality audio time-stretching library for our use. Current work is supported by the National Science Foundation under Grant No. 0855958.

9. REFERENCES

[1] Ableton. *Ableton reference manual (version 8).* http://www.ableton.com/pages/downloads/manuals (2011).

[2] Anderson, D. and Kuivila, R. A system for computer music performance. *ACM Transactions on Computer Systems*, Volume 8 Issue 1 (1990), pp. 56-82.

[3] Baba, T., Hashida, M., and Katayose, H. "VirtualPhilharmony": A Conducting System with Heuristics of Conducting an Orchestra. *Proceedings of the 2010 Conference on New Interfaces for Musical Expression (NIME 2010),* ACM Press, 2010, *263-270.*

[4] Bainbridge, D. and Bell, T. An ajax-based digital music stand for greenstone. *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries* (JCDL '09), ACM, New York (2009), pp. 463-464.

[5] Brandt, E. and Dannenberg, R. Time in distributed real-time systems. *Proceedings of the 1999 International Computer Music Conference*, ICMA, San Francisco (1999), pp. 523-526.

[6] Castan, G., Good, M. and Roland, P. Extensible markup language (XML) for music applications: An introduction. *The Virtual Score*, MIT Press, Cambridge, MA, 2001, pp. 95-102.

[7] Connick, H. Jr. System and method for coordinating music display among players in an orchestra. US Patent #6348648 (2002).

[8] Cont, A. ANTESCOFO: Anticipatory synchronization and control of interactive parameters in computer music. *Proceedings of International Computer Music Conference* (*ICMC*), ICMA, San Francisco, 2008.

[9] Dannenberg, R. Real-time scheduling and computer accompaniment. In *Current Directions in Computer Music Research*, edited by Max. V. Mathews & John R. Pierce, MIT Press, Cambridge, MA, 1989, pp.225-261.

[10] Dannenberg, R. A Language for Interactive Audio Applications. *Proceedings of the 2002 International Computer Music Conference*, ICMA, San Francisco, 2002, 509-515.

[11] Dannenberg, R. New interfaces for popular music performance. *Seventh International Conference on New Interfaces for Musical Expression: NIME 2007*, New York, NY, 2007, 130-135.

[12] Dannenberg, R. A Virtual Orchestra for Human Computer Music Performance. *Proceedings of the 2011 International Computer Music Conference,* (to appear).

[13] Dannenberg, R. and Raphael, C. Music score alignment and computer accompaniment. *Commun. ACM* 49, 8 (August 2006), pp. 38-43.

[14] Katayose, H. and Okudaira, K. Using an Expressive Performance Template in a Music Conducting Interface. *Proceedings of the 2004 Conference on New Interfaces for Musical Expression (NIME04),* (Hamamatsu), ACM Press., 2004, 124-129.

[15] Kurth, F., Müller, M., Fremerey, C., Chang, Y. and Clausen, M. Automated synchronization of scanned sheet music with audio recordings. *Proceedings of ISMIR*, Vienna (2007), pp. 261-266.

[16] Lee, E., Karrer, T. and Borchers, J. Toward a framework for interactive systems to conduct digital audio and video streams. *Computer Music Journal*, 30(1) (Spring 2006), pp. 21-36.

[17] MakeMusic, Inc. *SmartMusic interactive music software transforms the way students practice* (web page), http://www.smartmusic.com (2011).

[18] Raphael, C. Music Plus One: A system for flexible and expressive musical accompaniment. *Proceedings of the International Computer Music Conference*, (Havana, Cuba), ICMA, San Francisco, 2001.

[19] Robertson, A. and Plumbley, M. D. B-Keeper: A beat tracker for real time synchronisation within performance. *Proceedings of New Interfaces for Musical Expression* (*NIME 2007*), New York, NY, USA, (2007), pp 234-237.

[20] Rowe, R. *Interactive Music Systems*. MIT Press, Cambridge, MA (1993).

[21] Schnell, N., Peeters, G., Lemouton, S., Manoury, P., Rodet, X. Synthesizing a choir in real-time using Pitch Synchronous Overlap Add (PSOLA). *International Computer Music Conference (ICMC)*, (Berlin), ICMA, San Francisco, 2000.