Semantic Time: Representing Time and Temporal Transformations for Digital Audio in Interactive Computer Music Systems

Eric Lee and Jan Borchers
Media Computing Group
RWTH Aachen University
52056 Aachen, Germany
{eric, borchers}@cs.rwth-aachen.de

Abstract

Incorporating digital audio into computer music and multimedia systems with time-based interaction poses unique challenges. We describe some of these challenges, which we encountered whilst building previous systems, and describe our solution, which we call semantic time*. Semantic time is a theoretical framework based on theories of temporal intervals and denotational semantics. Semantic time defines semantic time* intervals*, such as beats of the music, that are more meaningful units of time than "audio samples";* time functions *describe the mapping between these semantic time intervals and presentation time (the absolute real time in which the media is output).* Function operators *manipulate and transform these functions, and* constraints *describe relationships between functions. All of these form an algebra for time, which creates interesting possibilities for time-based interaction with music and other digital media. We show how a software implementation of these theories applies to three different interactive computer music systems.*

1 Introduction

Most musicians will not dispute the key role that time plays in music expressivity; control over time and rhythm is something we often take for granted. Unfortunately, interaction with digital media today often does not take into account this temporal dimension, being limited to the VCR metaphors of "play", "stop", or "fast-forward". The important information in time-based media such as audio and video, however, must be perceived over time. In computer music, technologies such as MIDI and wavetable synthesis circumvent the problems of manipulating time in digitally sampled audio[1] by imposing more structure. However, there are arguments for working directly with digital audio recordings – today's synthesis techniques are still unable to reproduce the subtle nuances of a world-famous orchestra or jazz band, for example. Manipulating the time dimension of such recordings, however, can be challenging: simply resampling the audio to time-stretch it results in unwanted pitch shifts. Fortunately, recent research in high-quality, real-time, pitch-preserving time-stretching algorithms (Karrer et al. 2006) has partially addressed this problem.

A problem that remains largely unsolved is how to adequately represent time and temporal transformations for digital media. For designers and computer musicians working to build digital audio applications using multimedia frameworks such as Apple's Core Audio, Microsoft's DirectSound, or Max/MSP, time is referenced primarily by counting audio samples. The requirement for consistently and evenly spaced samples originates from the internal clock on audio hardware, but it is usually not the best unit in which to reference time. A more appropriate unit of time may be music beats; however, the mapping between music beats and audio samples is generally non-linearly to begin with, and changes as the audio is time-stretched.

For example, let us take one second/two beats of music sampled at 44.1 kHz (44100 samples). If the length of this audio snippet is time-stretched by a factor of two, we now have 88200 samples to represent these two beats. Modern well-known multimedia frameworks do not preserve the mapping between the "semantic time" of beats, and audio samples. The burden of maintaining this mapping falls upon application developers, needlessly distracting them from their primary task of designing the interaction and system logic.

In previous work (Lee et al. 2006), we introduced the *semantic time* concept, together with the Semantic Time Framework, a software framework for interactive orchestral conducting systems that use digital recordings. The Semantic Time Framework works in "semantic time units" (e.g., beats of the music), and preserves their mapping to real time, even after time scale modifications. We demonstrated how the design of an interactive conducting system can be made significantly simpler and more elegant.

We have since refined and expanded this idea of semantic time to include a more general class of computer music sys-

[1] To avoid confusion, the term "sampled audio" will always refer to PCM audio in this paper, which is a sequence of numerical values obtained by sampling an analog audio signal at some fixed sampling rate. Each numerical value in this sequence is referred to as a "sample", and should not be confused with a "sound sample", a snippet of audio used in modern synthesizers to recreate more realistic sounds.

tems, and also created an expanded set of theoretical tools for representing time and temporal transformations, which is the topic of this paper. Our theory of semantic time is inspired by previous research on temporal intervals (Allen 1983) and denotational semantics (Schmidt 1986). We demonstrate how we have applied this theory to prototype three different computer music applications: *Rhythmatic*, an application for interactively manipulating the rhythm pattern of digital recordings; *Personal Orchestra*, a system for conducting digital audio and video recordings using conducting gestures that first started in 2000, and is now in its fourth iteration; and *Beat Tapper*, a tool for browsing digital audio recordings and marking beats.

2 Related Work

In addition to common multimedia frameworks such as Core Audio, DirectSound, and Max/MSP, various other frameworks have been developed for computer music and multimedia:

Hudak et al. (1996) created *Haskore*, a language to describe music using functional programming. It includes data types such as notes and rests, and supports operations such as transposing and tempo scaling. The focus of our work is on the temporal aspect of media: units of time are not limited to "notes", nor is our work limited to music. More recently, Hudak (2004) proposed a polymorphic data type and theorems for representing temporal media; time is represented using intervals, and Hudak was able to prove that his theory is both sound and complete. Unfortunately, he does not discuss the implications of his theory on multimedia systems design.

ChucK (Wang and Cook 2003) is a programming language for music that includes mechanisms for interacting with time to provide on-the-fly, parallel composition. Interaction with time is primarily with musical notes, and scheduling these notes for playback. Our work, in contrast, represents time and temporal transformations for user interaction with prerecorded media.

Representing time in music has also been studied extensively; Honing (2001) gives an extensive overview of current work in this area, such as time maps (Jaffe 1985), and time warps (Dannenberg 1997a). Time warps are implemented in *Nyquist*, a sound/music synthesis language written in Common Lisp (Dannenberg 1997b). Honing (2001) has also developed generalized timing functions for music, and he shows a partial implementation in Common Lisp. All of these works seek to represent tempo changes and rhythmic timing (often differentiating between the two). Our work on semantic time, however, seeks to generalize beyond traditional musical concepts of beats and measures, making it applicable to other types of digital media (such as speech). We also use semantic time to describe time-based interactions, and the implications of such descriptions on system design.

A number of commercial applications, such as Ableton Live, Cakewalk Sonar, and Steinberg Cubase allow the user to modify the timing and rhythm of digital media and music. The interaction, however, remains specific to the application, and they do not introduce any mechanisms to discuss or facilitate building other time-based interactive systems.

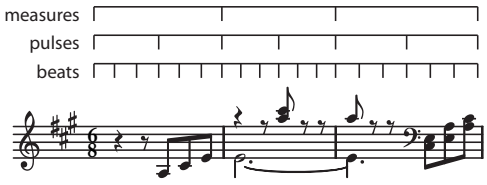

Figure 1: Example temporal interval structure for three bars of *Blue Danube Waltz* by Johann Strauss. The introduction of this piece is defined in $6/8$ time (six musical beats per measure). The pulses represent the "beat" perceived by a human tapping alongside the music (two pulses per measure).

3 Theory of Semantic Time

Semantic time is inspired by previous work on temporal intervals (Allen 1983) and denotational semantics (Schmidt 1986). We use a polymorphic *semantic time interval* as the basic unit of time, which can be recursively defined to represent different abstractions of time related to the temporal structure of a particular medium. In music, such a temporal structure could be in the form of beats, pulses and measures (see Figure 1); the beat and measure are defined by the musical score, and the pulse is defined to be what a human perceives as the "beat" (e.g., whilst tapping alongside the music). The structure could continue upwards to musical phrases, or downwards to the individual audio samples in the underlying PCM audio buffer.

One could imagine semantic time as described thus far as an extension of the MIDI time model to digital audio; however, as we will demonstrate with an example in the next section, the semantic time intervals are not limited to beats, pulses, and measures, nor is semantic time necessarily limited to music. It can be applied to other forms of media such as speech (which has a similar interval structure of phonemes, words and sentences). Such a discussion is beyond the scope of this paper, however.

The mapping between these interval sequences and presentation time (the absolute real time in which they occur) are continuous *time functions* (see Figure 2). For example, each beat of music performed at 120 beats per minute maps to half a second of presentation time. If, later in the performance, the music slows to 60 beats per minute, each beat interval now maps to one second of presentation time. These time functions are similar to time maps (Jaffe 1985), although we use this representation as a building block for describing time-based interaction, not just as a model for representing time in music.

Describing this mapping of media time (e.g., beats of the music) to presentation time as a mathematical function also

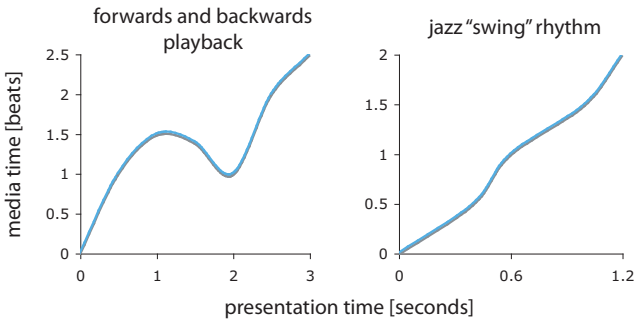

Figure 2: Left: Example mapping between media time and presentation time for forwards and backwards playback. Right: Mapping for a jazz "swing" rhythm.

allows us to formally represent not only position, but also rate (tempo) and acceleration as numerical time derivatives of these time functions. Temporal structures, such as the swing rhythm of jazz, can also be represented and visualized using this scheme (see Figure 2). Relationships between time functions can be described using *function operators* and *constraints*; these will be described in more detail in the next section.

4 Applications of Semantic Time

We illustrate how semantic time benefits designers and musicians building novel computer music systems with time-based interaction using three different interactive computer music applications that we have built[2]. While these applications differ in their purpose, design, and implementation, all of them respond to continuous user input to modify the temporal dimension of digitally recorded media.

4.1 Rhythmatic

Certain types of music, such as a Strauss waltz or jazz, have a characteristic off-beat swing or groove rhythm; such rhythms are often one of the more difficult aspects for musicians unpracticed in these genres to grasp. We created *Rhythmatic*, an application that allows the user to interactively manipulate the tempo and rhythm of such musical pieces by adjusting the relative timing of the beats. Adjustments can be made by applying a algorithmically generated groove pattern to the music, applying the rhythm pattern of an existing performance to another, or any scaled combination of these two.

This idea has been explored previously; commercial music sequencing applications such as Cakewalk Sonar and Steinberg Cubase allow composers and musicians to modify the rhythm of a MIDI recording. Cakewalk Sonar, for example, allows composers to apply rhythm patterns to MIDI recordings. Ableton Live provides a similar feature, called

[2]Select audio examples for these applications can be found at `http://media.informatik.rwth-aachen.de/~eric/time/`

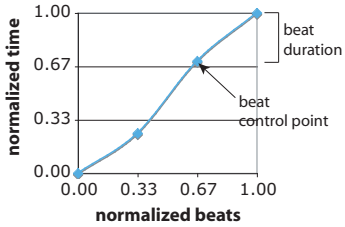

Figure 3: Example rhythm map from a Vienna Philharmonic performance of *Blue Danube Waltz*. The second beat is played early, and the third beat is slightly delayed. Both beats and time are normalized to values between $[0, 1)$.

"warp markers" for digital audio. Precomputed groove patterns can also be applied, although Live doesn't offer as much flexibility as Cakewalk with respect to applying arbitrary groove templates.

Unlike these applications, however, which are used primarily to edit the media offline, the aim of *Rhythmatic* is to allow users to *interactively* explore and learn about rhythm by experimenting and combining rhythm patterns from different sources. These interactions with rhythm are realized using semantic time.

We define a *rhythm map* to be a time function where each measure is normalized, such that each measure is a mapping from $[0, 1)$ to $[0, 1)$. The rhythm map remains a continuous curve defined by beat control points, which are determined from the normalized beat intervals $\tilde{b}_i$ of the measure (see Figure 3). Given the beat intervals b_i of a measure with n beats, the normalized beat intervals are calculated using:

$$\tilde{b}_i = \frac{b_i}{\sum_{j=0}^{n-1} b_j} \tag{1}$$

Choosing an appropriate interpolation scheme through these control points is an interesting research question in itself (it can be compared to selecting an interpolation scheme in computer graphics for drawing a continuous curve through the control points), and we reserve it for future work. Nevertheless, semantic time facilitates such experimentation; it also offers a more general representation than existing systems. Ableton Live, for example, linearly adjusts the tempo between warp markers placed at beat boundaries. This is equivalent to connecting the beat markers using straight lines, resulting in first-derivative discontinuities in the curve that manifest as sudden (and jarring) tempo changes at the beats.

We now define *function operators* to explore combinations of rhythm maps. The first operator is $f(t) :: g(t)$, the concatenation of the two rhythm maps $f(t)$ and $g(t)$ (see Figure 4). Concatenating rhythm maps do not suffer from the same issues as time maps (Honing 2001), because they are constrained to the start and end of a measure. The time function of a piece with n measures can be completely represented using concatenated one-measure rhythm maps $g_0(t) :: g_1(t) :: \ldots :: g_{n-1}(t)$ and their respective lengths

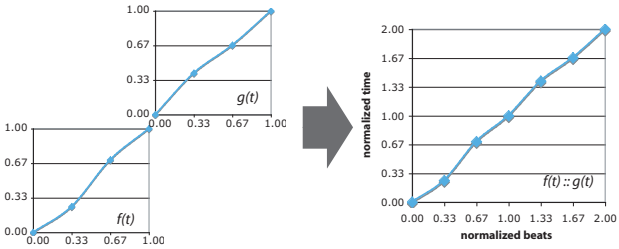

Figure 4: Visualization of rhythm map concatenation. Two rhythm maps, $f(t)$ and $g(t)$, are concatenated together.

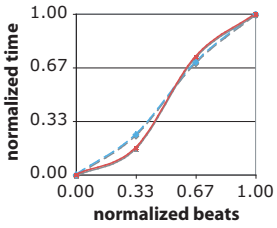

Figure 5: Visualization of a rhythm map scaled by $\alpha = 2.0$ (200%), resulting in a more accentuated swing. The dotted line shows the unscaled rhythm map.

$m_0 \ldots m_{n-1}$. Note that this formulation places no restrictions on the number of beats per measure, and even supports the alternating time signatures found in ancient folk music or Dave Brubeck's jazz compositions.

A rhythm map can also be *scaled* – the musical equivalent of accentuating (scale up) or easing (scale down) its swing. This scenario if very typical in jazz, since different artists swing differently – contrast Oscar Peterson's heavy swing with Bill Evans' lighter swing, for example. Let us define $\sigma_\alpha(f(t))$ as the rhythm map $f(t)$ scaled by α. To compute the scaled beat intervals for a rhythm map of k beats, we must first transform the beat intervals to the beat control points in the measure:

$$p_i = \sum_{j=0}^{i-1} b_j \; ; \; p_0 = 0 \tag{2}$$

Then, we scale the offset of these beat positions relative to a perfectly quantized beat:

$$p'_i = \alpha \left(p_i - \frac{i}{k} \right) + \frac{i}{k} \tag{3}$$

Finally, we transform the scaled beat positions p'_i back to beat intervals b'_i using an inverse of Equation 2. Figure 5 shows the effect of scaling a rhythm map.

A third operator is combining rhythm maps together using a weighted average. Rhythm maps can be averaged together only if the number of beats, and the distribution of beats within each measure, is the same. A rhythm map can be averaged from m other rhythm maps using the weights β_j:

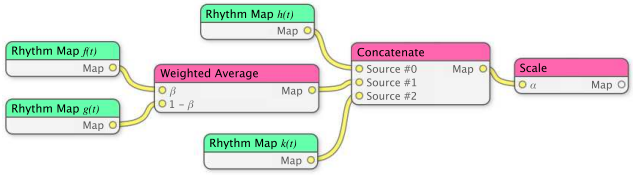

Figure 6: Visualization of a more complex rhythm map equation. Two maps, $f(t)$ and $g(t)$, are averaged with a weight β. The result is then concatenated on either side with $h(t)$ and $k(t)$. The concatenated rhythm map is finally scaled by α.

$$\sum_{j=0}^{m-1} \beta_j f_j(t) \; ; \; \sum_{j=0}^{m-1} \beta_j = 1 \tag{4}$$

The beat intervals of the averaged rhythm map are calculated as follows:

$$b'_i = \sum_{j=0}^{m-1} \beta_j \cdot b_{i,j} \tag{5}$$

Averaging two rhythm maps has the effect of "mixing" two performances together. For example, one could take a Vienna Philharmonic performance and a Boston Symphony Orchestra performance of the same piece, and create a new performance with the rhythm characteristics of both.

Concatenation, scaling, and averaging can also be arbitrarily combined; for example, we could take the concatenation of three rhythm maps, the second of which is a weighted average of two other rhythm maps, and scale the entire map (see Figure 6). These operators thus form an *algebra*, and we have begun to explore the properties of this algebra. Concatenation, for example, is *associative* (e.g., $(f(t) :: g(t)) :: h(t) = f(t) :: (g(t) :: h(t))$), averaging is *commutative* (e.g., $\beta f(t) + (1-\beta)g(t) = (1-\beta)g(t) + \beta f(t)$) and scaling is *distributive* over concatenation and averaging (e.g., $\sigma_\alpha(f(t) :: g(t)) = \sigma_\alpha(f(t)) :: \sigma_\alpha(g(t))$). These properties can be used to algebraically reduce complicated expressions to improve performance. While algebras have been proposed before for animation (Elliott et al. 1994), there appears to be no existing work that explores algebras for representing time and temporal transformations in computer music at this level.

The algebra can also be exposed directly to the end user as a visual language of interconnected building blocks, similar to Max/MSP (see Figure 6). Such an interface offers interesting possibilities for the user to interact with the tempo and rhythm of music; in our prototype implementation, for example, the weighting and scaling factors can be dynamically adjusted while the music is playing, and the user receives immediate feedback on their adjustments to the rhythm.

4.2 Personal Orchestra

Personal Orchestra is an ongoing project that started in 2000, with various iterations of the project result-

ing in a series of successful interactive museum exhibits around the world, including the House of Music Vienna (Borchers et al. 2004)[3], Boston Children's Museum (Lee et al. 2004)[4], and Betty Brinn Children's Museum in Milwaukee (Lee et al. 2006). Like many other interactive conducting systems, *Personal Orchestra* allows the user to control (amongst other parameters) the music tempo using conducting gestures; unlike other systems, however, *Personal Orchestra* continues to be one of few systems that guarantee *synchronous* playback of time-stretched digital audio and video recordings; the media is also synchronized in speed (tempo) and position (beat) with the user's conducting gestures.

Our first version of the Semantic Time Framework was used to design the media engine for *Maestro!*[5], the third generation *Personal Orchestra* (Lee et al. 2006). This version of the Semantic Time Framework addresses the problem that modern well-known multimedia frameworks do not distinguish and preserve the relationship between media time and presentation time; preserving this relationship is necessary to synchronize the time-stretched audio to the video, and to synchronize the media to the beats from the user's conducting gestures. The Semantic Time Framework preserves this mapping between media time and presentation time, which allowed us to simplify a system architecture that required two independent implementations of a synchronization algorithm. By introducing a generic "semantic time unit" model in the framework, we were able to create a reusable synchronization module that keeps one or more timebases (e.g., audio and video) synchronous with another timebase (e.g., user's beats, see Figure 7).

Using the semantic time theory presented in this paper, we can further generalize our synchronization algorithm. In (Lee et al. 2006), we specified synchronization *imperatively* by describing an algorithm for *how* synchronization is to be performed, rather than *what* the desired result is. This imperative approach to synchronization is difficult to generalize without exposing unnecessary details of the algorithm. For example, we may wish to relax the constraint that the audio precisely follow the user's beats, or have the audio follow the beat with some delay. Usa and Mochida (1998), for example, observed that professional conductors expect to lead the orchestra by some amount dependent on their cultural background and the tempo of the music; we have recently also studied this phenomenon in more detail for both conductors and non-conductors (Lee et al. 2005), and obtained some quantitative results. We found that conductors expect the orchestra to follow their beat with a fixed delay (150 msec for *Radetzky March*), while non-conductors lead the beat just

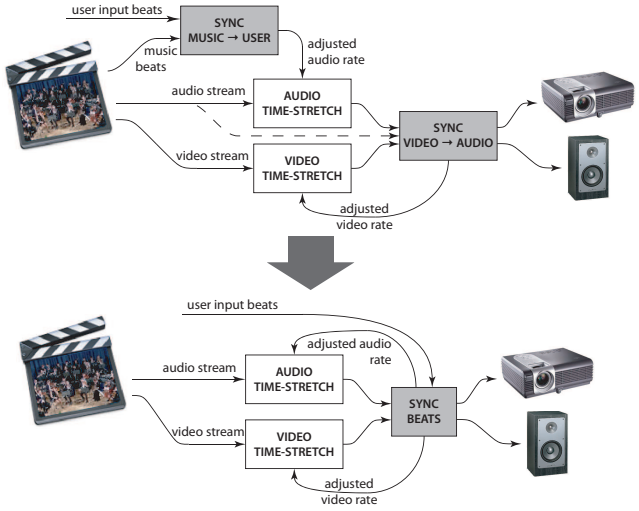

Figure 7: Using the Semantic Time Framework reduces the complexity of *Personal Orchestra*.

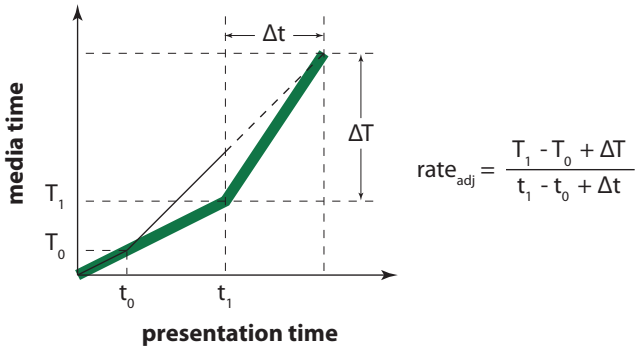

Figure 8: Algorithm for synchronizing a media stream (thick line) to user time (thin line). Details are presented in (Lee et al. 2006), and beyond the scope of this paper. Generalizing the algorithm to include other types of synchronization requires additional parameters, and requires developers wishing to synchronize media to understand the algorithm.

slightly (50 msec) on average, but alternate between leading and following the beat: a non-conductor's perception of the beat is different and not as precise as a conductor's. To support these users, we could modify the algorithm to support various levels of synchronization based on certain input parameters; this would, however, require the application developer to have some understanding of the synchronization algorithm itself (see Figure 8). We feel a more elegant solution is to describe synchronization not as an algorithm, but *declaratively* by imposing *constraints* on the independent timebases of audio, video, and user input.

Let the time functions for audio, video, and user input be $a(t)$, $v(t)$, and $u(t)$, respectively. Then, the drift between two timebases is the mathematical difference between the two functions (e.g., $a(t) - v(t)$). To preserve lip sync, video is usually synchronized as closely as possible to the audio,

[3]The first *Personal Orchestra* was coordinated by Max Mühlhäuser, now at Darmstadt University, and in cooperation with the Vienna Philharmonic.

[4]*Personal Orchestra* 2, also known as *You're the Conductor*, was a joint project in collaboration with Teresa Marrin Nakra at Immersion Music, and the Boston Symphony Orchestra.

[5]*Maestro!* was sponsored by the Betty Brinn Children's Museum, in cooperation with the Milwaukee Youth Symphony Orchestra.

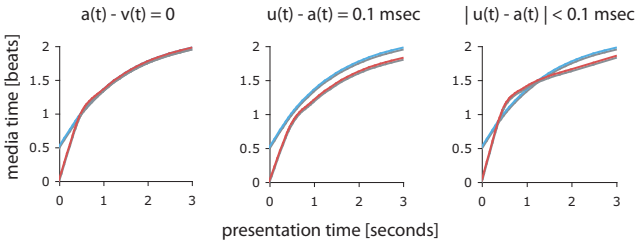

Figure 9: Visualization of three types of constraints for synchronization. On the left, the audio and video are exactly sychronized. In the middle, the audio lags behind the user gestures by 100 msec. On the right, the audio is allowed to lead or lag behind the user's gesture by 100 msec.

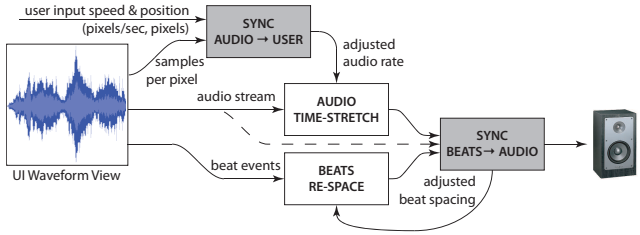

Figure 10: System architecture for audio scrubbing in *Beat Tapper*. The audio is first synchronized to the user input (speed and position, given in units of pixels/sec and pixels). The beats are then respaced and synchronized with the time-stretched audio before output. The dotted line shows that the "beats to audio" synchronization module has an interdependency on the unstretched audio stream.

equivalent to the constraint that $a(t) - v(t) = 0$. However, we can vary this constraint when synchronizing to the beats marked by the user. If the user is a trained conductor, we can set the music to follow her beat by, for example, $1/4$ of a beat ($u(t) - a(t) = 0.25$ beats). Or, if the user is not a trained conductor, we can allow their beat to lead or even lag behind the music beat by up to $1/8$ of a beat ($|u(t) - a(t)| = 0.125$ beats, see Figure 9).

While previous work has also introduced the idea of using "constraints" to describe synchronization (Bailey et al. 1998), the way in which we formulate constraints differs. The *Nsync* framework, for example, expresses temporal constraints as a conditional (e.g., `when (a(t) > v(t)) then reduce_video_playrate()`); we argue such a formulation is still imperative, compared to our declarative approach.

4.3 Beat Tapper

Automatic beat detection remains an active area of research in computer music, and algorithms have been developed that work well for many types of music (Dixon 2001). Nonetheless, certain types of music remain beyond the capabilities of these algorithms. For example, in a particular Vienna Philharmonic performance of *Blue Danube Waltz* that we analyzed, the tempo varies from 15 to 80 pulses per minute, and there is little percussion; we have found even humans often have trouble finding the pulse. Tracking the music beat, which is not only three times faster, but exhibits the characteristic Strauss waltz swing, is, unsurprisingly, beyond the capabilities of today's algorithms. Thus, we have found a tool like *Beat Tapper*, that allows a human to manually tag audio data with beat metadata, indispensable in our research.

Beat Tapper allows the user to load an audio file into a waveform view, and "tap along" to the beat while the audio is playing. The inserted beat markers can be manually fine-aligned in the waveform. *Beat Tapper* has features rarely seen on similar tools, however: users can, optionally, "hear" the beat (tapping sounds are played together with the music); users can arbitrarily adjust the audio playback speed; and users can "scrub" through the audio. Audible scrubbing is available in some audio editing applications, and allows the user to quickly "browse" through the audio by moving the cursor around in the waveform and hearing the audio at the same time. These applications, however, either simply skip to the current cursor position and play a short snippet (resulting in garbled audio), or play a resampled version of the audio (resulting in pitch-shifts). *Beat Tapper*, on the other hand, time-stretches the audio for continuous, high-fidelity feedback.

There are two main challenges to support this scrubbing interaction. Unlike a simple slider, where only the speed of the audio is synchronized to user input, or scrubbing by skipping where only the position of the audio is synchronized to user input, both the speed and the position must be synchronized in the pitch-preserving time-stretching scheme we introduced for *Beat Tapper*. Also, the time-stretched audio must be synchronized to the audible beats; these tapping sounds, being transient signals, are not time-stretched the same way as the audio. Instead, the time at which they are played must be adjusted according to the time-stretched audio. A system architecture to implement the scrubbing interaction using an existing multimedia framework, such as Core Audio or DirectSound, is shown in Figure 10.

This architecture suffers from design issues similar to that described in (Lee et al. 2006): there are multiple instances of conversion from one time unit to another, and the module for synchronizing the beats requires information not only from the time-stretched audio, but also from the original (unstretched) input. Moreover, there is a "daisy chain" of synchronization modules: the audio is synchronized to the input, and because of the processing latency in the time-stretching, the beats must be synchronized to the time-stretched audio to ensure precise synchronization. An implementation of this design becomes very specific to the application, and reusing these modules for another application is not possible.

In a redesign of *Beat Tapper* using the Semantic Time Framework, the semantic time interval is set to the number of audio samples that corresponds to one pixel along the x-axis of the graphical waveform view (see Figure 11). The beat

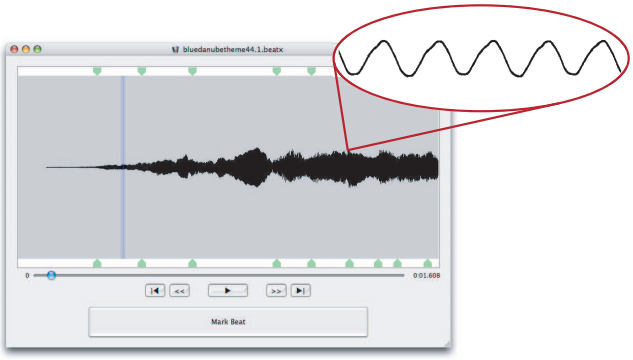

Figure 11: The *Beat Tapper* application. Each pixel along the x-axis of the waveform view corresponds to an interval of audio samples, which we use as the semantic time interval.

times are also expressed as points in this interval space. With this choice of the semantic time interval, we can simplify the system architecture to a design almost identical to the rendering engine of *Personal Orchestra* (see Figure 7). The differences between the two systems are the choice of semantic time units (pixels for *Beat Tapper*, beats for *Personal Orchestra*) and the media types (audio and beats for *Beat Tapper*, audio and video for *Personal Orchestra*).

Beat Tapper thus illustrates the need for a *polymorphic* semantic time interval data type, one that is not limited to only traditional musical notions of time, such as beats.

5 Discussion and Evaluation

How one evaluates a set of theories or a software framework is a question that is open to debate. For example, language theorists often use mathematical proofs to prove soundness and completeness (Hudak 2004). In our work, we have adopted an iterative design/implement/analyze approach. Our evaluation in the first iteration of the Semantic Time Framework (Lee et al. 2006) showed how using the Semantic Time Framework reduced the overall complexity of the system architecture of an interactive conducting system.

The goal of this next iteration is to both develop the semantic time theory further, and to expand the application domain beyond interactive conducting systems. We believe the three applications we presented here are all steps towards this goal.

Rhythmatic improves upon existing interactions with groove and digital audio, and incorporates an algebra for representing rhythm. Not only does such an algebra make the interaction with rhythm more general than existing implementations, but it also offers opportunities for real-time, interactive experimentation with rhythm; we believe such interaction has applications in, for example, music education.

Personal Orchestra and *Beat Tapper* illustrate how two rather different interactions (conducting and audio scrubbing) actually result in very similar requirements for a flexible-time media engine. And while the time-stretching and synchronization mechanisms for these two applications would normally be implemented differently using conventional multimedia frameworks, the Semantic Time Framework allows the same software modules to be reused for both applications.

Our expanded theory of semantic time has also allowed us to respecify a previously developed synchronization algorithm using constraints. Furthermore, we can also generalize this specification to include additional interactions based on the differences in how conductors and non-conductors mark beats relative to the music beat.

Finally, *Beat Tapper* is an example of where the semantic time interval is not set to the music beats or measures, but is instead set to an interval of audio samples as represented in a graphical user interface.

6 Future Work

As we continue with our work on semantic time, we would like to explore the following areas:

We have begun exploring an algebra for working with time that includes concatenation, scaling and averaging. We are working to develop this algebra further, with more operators and the implications of combining these operators. Using rules of algebraic simplification, we believe we can also apply algebraic reductions to a complex rhythm equation to improve performance.

Our current prototypes demonstrating the ideas presented in this paper were implemented in Objective-C. As we continue to develop the Semantic Time Framework, we are investigating the use of a functional programming language such as Haskell or ML. These languages have traditionally been criticized for their poor performance, and while recent trends show significant improvement in this regard, one promising approach is to use a functional programming-based front end that communicates with a high-performance back-end that performs the computationally expensive audio and video processing. Such a scheme has been adopted successfully before (Greenebaum 1997).

Finally, we are continuing to explore new application areas for the Semantic Time Framework. Research in comprehension of fast versus slow speech has shown, for example, that different aspects of pronunciation change when humans talk faster or slower (Zellner 1998). This implies that the naïve approach to changing the playrate of a speech recording by simply varying the playrate linearly could be improved. One possible approach is to define the semantic time intervals as the phonemes of a speech signal, and impose constraints on how these intervals are time-stretched relative to each other. This would, for example, reflect how we eliminate certain consonants when talking faster, or increase the relative length of pauses between words when talking slower.

7 Conclusions

We presented *semantic time*, a theory for representing time and temporal transformations in computer music systems. This theory is based on an interval structure tied to the semantics of the media, but defined according to the needs of a particular application. The mapping between media time and presentation time results in a time function. When normalized over a measure, these time functions form rhythm maps, and we introduced three rhythm map operators, scaling, concatenation and averaging, which form an algebra. We also showed how constraints imposed on time functions can be used to specify varying degrees of synchronization between media and user input. Finally, we showed an example where it is useful to define the semantic time interval as a unit other than a music beat.

While the ideas and concepts behind *Rhythmatic*, the fourth iteration of *Personal Orchestra*, and *Beat Tapper* are only incremental improvements over existing systems, each of these applications presents an improved temporal interaction with digital audio and video over previous work. Furthermore, semantic time enabled us to propose elegant solutions to the problem of designing and implementing these time-based interactions.

As we continue to develop semantic time, we hope that it will promote further research in the field of time-based interaction with music and multimedia.

8 Acknowledgements

The authors would like to thank Marius Wolf, Jonathan Diehl, Ken Greenebaum, Conal Elliott and Sidney Fels for their inspiring discussions on this topic.

References

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM 26*(11), 832–843.

Bailey, B., J. A. Konstan, R. Cooley, and M. Dejong (1998). Nsync - a toolkit for building interactive multimedia presentations. In *Proceedings of the MM 1998 Conference on Multimedia*, Bristol, UK, pp. 257–266. ACM Press.

Borchers, J., E. Lee, W. Samminger, and M. Mühlhäuser (2004, March). Personal Orchestra: A real-time audio/video system for interactive conducting. *ACM Multimedia Systems Journal Special Issue on Multimedia Software Engineering 9*(5), 458–465. Errata published in ACM Multimedia Systems Journal 9(6):594.

Dannenberg, R. (1997a). Abstract time warping of compound events and signals. *Computer Music Journal 21*(3), 61–70.

Dannenberg, R. (1997b). The implementation of Nyquist, a sound synthesis language. *Computer Music Journal 21*(3), 71–82.

Dixon, S. (2001). An interactive beat tracking and visualisation system. In *Proceedings of the ICMC 2001 International Computer Music Conference*, Havana, pp. 215–218. ICMA.

Elliott, C., G. Schechter, R. Yeung, and S. Abi-Ezzi (1994, July). TBAG: A high level framework for interactive, animated 3D graphics applications. In *Proceedings of the SIGGRAPH 1994 Conference on Computer Graphics and Interactive Techniques*, Orlando, USA, pp. 421–434. ACM Press.

Greenebaum, K. (1997). DirectAnimation - a new approach to multimedia. In *Proceedings of the ICAD 1997 Conference on Auditory Displays*.

Honing, H. (2001). From time to time: The representation of timing and tempo. *Computer Music Journal 25*(3), 50–61.

Hudak, P. (2004). An algebraic theory of polymorphic temporal media. In B. Jayaraman (Ed.), *Proceedings of the PADL 2004 Symposium on Practical Aspects of Declarative Languages*, Volume 3057 of *Lecture Notes in Computer Science*, pp. 1–15. Springer.

Hudak, P., T. Makucevich, S. Gadde, and B. Whong (1996, May). Haskore music notation - an algebra of music. *Journal of Functional Programming 6*(3), 465–483.

Jaffe, D. (1985). Ensemble timing in computer music. *Computer Music Journal 9*(4), 38–48.

Karrer, T., E. Lee, and J. Borchers (2006, November). PhaVoRIT: A phase vocoder for real-time interactive time-stretching. In *Proceedings of the ICMC 2006 International Computer Music Conference*, New Orleans, USA. ICMA, In Print.

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

Lee, E., H. Kiel, S. Dedenbach, I. Grüll, T. Karrer, M. Wolf, and J. Borchers (2006, April). iSymphony: An adaptive interactive orchestral conducting system for conducting digital audio and video streams. In *Extended Abstracts of the CHI 2006 Conference on Human Factors in Computing Systems*, Montréal, Canada. ACM Press.

Lee, E., T. Marrin Nakra, and J. Borchers (2004, June). You're the conductor: A realistic interactive conducting system for children. In *Proceedings of the NIME 2004 Conference on New Interfaces for Musical Expression*, Hamamatsu, Japan, pp. 68–73.

Lee, E., M. Wolf, and J. Borchers (2005, April). Improving orchestral conducting systems in public spaces: examining the temporal characteristics and conceptual models of conducting gestures. In *Proceedings of the CHI 2005 Conference on Human Factors in Computing Systems*, Portland, USA, pp. 731–740. ACM Press.

Schmidt, D. A. (1986). *Denotational semantics: a methodology for language development*. Dubuque, USA: William C. Brown.

Usa, S. and Y. Mochida (1998). A multi-modal conducting simulator. In *Proceedings of the ICMC 1998 International Computer Music Conference*, pp. 25–32. ICMA.

Wang, G. and P. Cook (2003). ChucK: A concurrent, on-the-fly, audio programming language. In *Proceedings of the ICMC 2003 International Computer Music Conference*, Singapore, pp. 217–225. ICMA.

Zellner, B. (1998). Temporal structures for fast and slow speech rate. In *Proceedings of the ESCA/COCOSDA International Workshop on Speech Synthesis*, Jenolan Caves, Australia, pp. 143–146.