MidiSpace: a Temporal Constraint-Based Music Spatializer

François Pachet
SONY CSL Paris
6, rue Amyot 75005
Paris FRANCE
(33) 1 44 08 05 16
pachet@csl.sony.fr

Olivier Delerue
SONY CSL Paris
6, rue Amyot 75005
Paris FRANCE
(33) 1 44 08 05 14
delerue@csl.sony.fr

1. ABSTRACT

We develop multimedia technology for enriching the music listening experience. We propose a system - MidiSpace - in which users may listen to music while controlling in real time the localization of sound sources, through a simple interface. We introduce the problem of *mixing consistency*, and propose a solution based on a constraint propagation mechanism. The proposed system contains both an authoring mode, in which sound engineers may specify spatialization constraints to be satisfied, and a listening mode in which listeners can modify the localization of sources under the supervision of a constraint solver that ensures the spatialization always satisfies the constraints. We describe the architecture of the system and report on experiments done so far.

1.1 Keywords

3-D sound, spatialization, constraints, interactive music.

2. ACTIVE LISTENING

We believe that listening environments of the future can be greatly enhanced by integrating models of musical perception into musical listening devices, provided we develop appropriate software technology to exploit them. This is the basis of the research conducted on “Active listening” at Sony Computer Science Laboratory, Paris.

Active Listening refers to the idea that listeners can be given some degree of control on the music they listen to, that give the possibility of proposing different musical perceptions on a piece of music, by opposition to traditional listening, in which the musical media is played passively by some neutral device. The objective is both to increase the musical comfort of listeners, and, when possible, to provide listeners with smoother paths to new music (music they do not know, or do not like). These control parameters create implicitly control spaces in which musical pieces can be listened to in various ways. Active listening is thus related to the notion of *Open Form* in composition [10] but differs by two aspects: 1) we seek to create listening environments for *existing* music repertoires, rather than creating environments for composition or free musical exploration (such as *PatchWork* [16], *OpenMusic* [2], or *CommonMusic* [25]), and 2) we aim at creating environments in which the variations always preserve the original semantics of the music, at least when this semantics can be defined precisely.

The first parameter which comes to mind when thinking about user control on music is the spatialization of sound sources. In this paper we study the implications of giving users the possibility to change dynamically the mixing of sound sources. We will first review previous approaches in computer-controlled sound spatialization, and propose a basic environment for controlling music spatialization, called MidiSpace. In section 5, we exhibit knowledge that can be used to define a semantic on sound spatialization, and we propose to use constraint propagation algorithms to represent faithfully a substantial part of this knowledge. After a review of basic approaches in constraint propagation for multimedia systems, we describe our algorithm, which handles cycles, non linear functional constraints, and inequalities. Section 6 describes the overall design and implementation of the resulting system.

3. MUSIC SPATIALIZATION

Music spatialization has long been an intensive object of study in computer music research. Most of the work so far has concentrated in building software systems that simulate acoustic environments for existing sound signals.

These works are based on results in psychoacoustics that allow to model the perception of sound sources by the human hear using a limited number of perceptive parameters [8]. These models have led to techniques allowing to recreate impression of sound localization using a limited number of loudspeakers. These techniques typically exploit difference of amplitude in sound channels, delays between sound channels to account for interaural distances, and sound filtering techniques such as reverberation to recreate impressions of distance.
For instance, The *Spatialisateur IRCAM* [15] is a virtual acoustic processor that allows to define the sound scene as a set of perceptive factors such as azimuth, elevation and orientation angles of sound sources relatively to the listener. This processor can adapt itself to any sound reproduction configuration, such as headphones, pairs of loudspeakers, or collections of loudspeaker. Other commercial systems with similar features have recently been introduced on the market, such as Roland *RSS*, the *Spatializer* (Spatializer Audio Labs) which allows to produce a stereo 3D signal from an 8-track input signal controlled by joysticks, or Q-Sound labs's *Q-Sound*, which builds extended stereophonic image using similar techniques. This tendency to propose integrated technology to produce 3D sound is further reflected, for instance, by Microsoft's DirectX API now integrating 3D audio.
These sound spatialization techniques and systems are mostly used for building various virtual reality environments, such as the Cave or *CyberStage* [9], [10]. Recently, sound spatialization has also been included in limited ways in various 3D environments such as *Community Place*'s implementation of VRML [17], or ET++ [1].
Based on these works, we are interested in exploiting spatialization capabilities for building richer listening environments. A main point of concern is to maintain some sort of consistency of musical pieces, while allowing the user to navigate freely in a control space. We will first describe our system MidiSpace, which precisely allows user to control in real time spatialization of sound sources, without any restriction. In section 5, we will show how to add some semantics to limit the range of user actions in a meaningful way.

4. THE BASIC MIDI SYSTEM

MidiSpace is a software system that embodies our ideas in active listening and user controlled music spatialization.

4.1 Overview

MidiSpace is a real time player of Midi files which allows users to control in real time the localization of sound sources through a 2D interface (extensions to audio and 3D are discussed in 7). MidiSpace takes as input arbitrary Midi files [14]. The basic idea in MidiSpace is to represent graphically sound sources in an editor, as well as an avatar that represents the listener itself. In this editor, the user may either move its avatar around, or move the instruments themselves. The relative position of sound sources and the listener's avatar determine the overall mixing of the music, according to simple geometrical rules illustrated in Figure 1. The 2D interface of MidiSpace is represented in Figure 2. Additional features such as muting sources are provided but not discussed here. The real time mixing of sound sources is realized by sending Midi volume and panoramic messages. More details on the implementation are given in section 6.

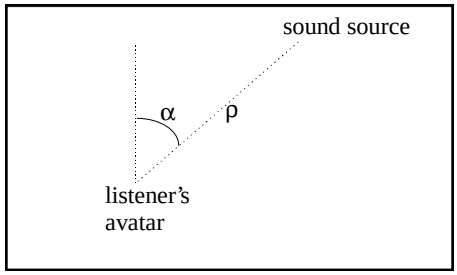

Figure 1. Volume of sound_source$_i$ = f(distance(graphical-object$_i$, listener_avatar)). f is a function mapping distance to Midi volume (from 0 to 127). Stereo position of sound source i = g(angle(graphical_Object$_i$, listener_avatar)), where angle is computed relatively to the vertical segment crossing the listener's avatar, and g is a function mapping angles to Midi panoramic positions (0 to 127).

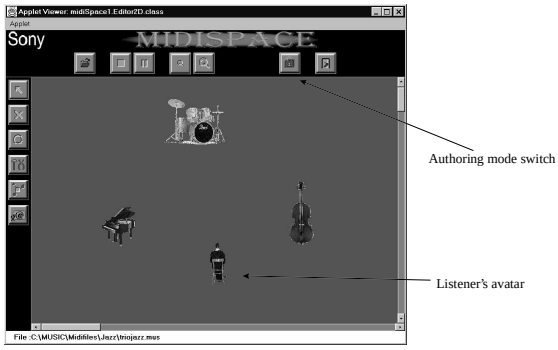

Figure 2. The 2D Interface of MidiSpace. Here, a tune by Bill Evans (Jazz trio) is being performed.

It is important to understand here the role of Midi in this research. On the one hand, there are strong limitations of

using Midi for spatialization *per se*. In particular, using Midi panoramic and volume control changes messages for spatializing sounds does not allow to reach the same level of realism than when using other techniques (delays between channels, digital signal processing techniques, etc.), since we exploit only difference in amplitude in sound channels to recreate spatialization. However, this limitation is not important for two reasons : 1) this Midi-based technique still allows to achieve a reasonable impression of sound spatialization which is enough to validate our ideas, and 2) more sophisticated techniques for spatialization can be added in MidiSpace, independently of its architecture (see section 7 about the extension to audio files).

4.2 The Temporal Aspect of MidiSpace Interaction

4.2.1 Annotations and Representation of Content

The need for representing and exploiting representation of content in multimedia systems is now widely acknowledged. The Mpeg7 project for instance aims at standardizing content representation of multimedia documents for future multimedia applications. Other standards in use or in progress are more dedicated to musical information, such as SMDL or HyTime. However, these formats are not primarily designed for real time applications.

In MidiSpace, we developed a simple format for representing annotations on musical pieces, that can be interpreted in real time to influence the spatialization. Examples of annotations useful for spatialization are : the structure of the musical piece (how a piece is divided into various segments such as introduction, chorus, coda, etc.), harmonic information (e.g. the chord sequence associated with the music), analytical information (e.g. the underlying keys or tonalities), etc. Our format is based on a time-tagged attribute/value representation. The time information is expressed in musical beats, and is therefore independent of the tempo. For instance, the structure of the musical piece illustrated in Figure 2 looks as follows:

```
[structure]
start=1.000     dur=8     type=Introduction instr=piano
start=9.000     dur=32    type=Exposition instr=piano
start=41.000    dur=32    type=Exposition instr=piano
start=73.000    dur=64    type=Chorus instr=piano
start=137.000   dur=32    type=Exposition instr=piano
start=169.000   dur=32    type=Exposition instr=piano
start=201.000   dur=8     type=Coda instr=piano …
```

Figure 3. Structure of a musical piece. Start time and durations are expressed in musical beats.

Similarly, the harmonic information is represented as a chord sequence, and is illustrated in Figure 4. Chords are represented using a format derived from the chord format of SMDL [23], [24], and adapted for tonal popular music (Jazz, rock, pop) tunes. Chord sequences are divided into parts (such as A, B, C), and the overall structure of the chord sequence is described in terms of these parts (e.g. AABA). This harmonic information is typically used for analytical process, such as harmonic analysis [20]. In our context, this information can also be used to emphasize particular musical segments, e.g. for pedagogical purposes.

```
[chordSequence]
parts = A, B
structure = AABA
begin part A
start=1.000  dur=4 chord = Bb maj7
start=5.000  dur=4 chord = Eb maj7
start=9.000  dur=4 chord = D min 7
start=13.000  dur=4 chord = G 7
start=17.000  dur=4 chord = C min
start=21.000  dur=4 chord = G 7
end part A
begin part B
start=1.000  dur=4 chord = C min
start=5.000  dur=4 chord = F 7
start=9.000  dur=4 chord = C min 7
…
```

Figure 4. A Chord sequence annotation

4.2.2 Authoring Mode

The existence of annotations makes it necessary to differentiate between two modes in MidiSpace: an *authoring* mode, in which the user may create annotations, and a *listening* mode, in which the annotation are used for spatialization. The switching between these two modes is triggered by a simple icon (see Figure 2). The only difference between the two modes is a facility to record, while playing, the user actions which generate annotations files. The user actions are analyzed in real time, and produce annotations in a format similar to the one described in the preceding sections. The recorded information is movements of sound sources, as illustrated in Figure 5. A specific editor of temporal structures (not

represented here) allows to edit manually the information once recorded.

[movements]
startBeat=7.692 duration=0 instr=bass x=127 y=329
startBeat=7.698 duration=0 instr=bass x=127 y=329
startBeat=7.881 duration=0 instr=bass x=127 y=329
startBeat=8.067 duration=0 instr=piano x=127 y=329
startBeat=8.214 duration=0 instr=piano x=127 y=329
startBeat=8.463 duration=0 instr=piano x=127 y=329
startBeat=8.631 duration=0 instr=piano x=127 y=329
startBeat=9.36 duration=0 instr=bass x=127 y=329
startBeat=9.531 duration=0 instr=bass x=127 y=329…

Figure 5. Temporal information representing movements of musical sources are represented as annotations.

The architecture of MidiSpace is illustrated in Figure 6. It mainly consists in a real time player which takes as input a musical data (a Midi file in the current version), and an annotation file containing all the annotations pertaining to the musical piece.

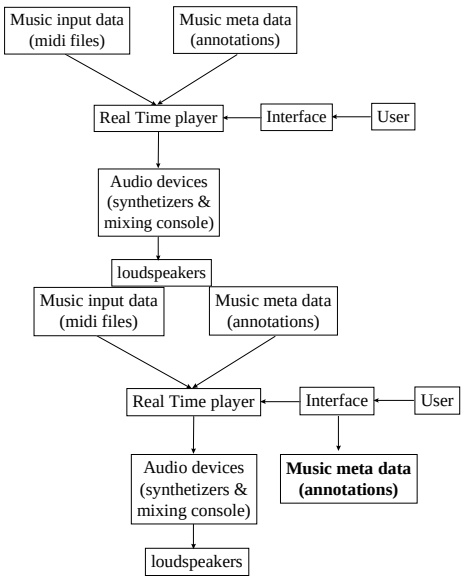

Figure 6. On the left, the architecture of MidiSpace in the listening mode. On the right, the architecture in authoring mode: the main output is an annotation file, which can be used in the listening mode by users.

The system described here has been fully implemented. Experiments conducted on various users have brought to evidence the need of including semantic information to our system. The next section introduces the semantic information we attempted to represent in the context of MidiSpace: mixing consistency.

5. INTRODUCING MIXING CONSISTENCY IN MIDISPACE

The notion of musical semantics in general has been extensively debated (see e.g. [18]). Without committing to a particular general semantic theory of music, we believe that it is possible to define a reasonable and pragmatic notion of musical semantics in the context of interactive systems, based on the properties of the navigational spaces that may be enforced automatically. The main claim of this paper is that in the context of spatialization, this semantics may be represented as a set of properties imposed on the overall mixing, and that these properties may in turn be represented as constraints between the sound sources and the listener's avatar. Moreover, we propose to represent these constraints as annotations, to be defined in real time, in order to produce dynamically evolving navigational spaces.

5.1 Mixing Consistency

The knowledge of the sound engineer is difficult to explicit and even more to represent as a whole. Its basic actions are modifications of mixing consoles controls, such as faders. However, mixing involves a set of actions that can often be defined as compositions of these atomic actions. For instance, sound engineers use knowledge on sound energy to ensure that the overall energy level of the recording always lies between reasonable boundaries. One effect of this property is that sound levels are usually not set independently of one another. Typically, when a fader is raised, another one, (or a group of other faders) should be lowered. Conversely, several sound sources may be logically dependent. For instance, the rhythm section may consist in the bass track, the guitar track and the drum track. Other typical mixing action is to assign boundaries to instruments or groups of instruments, and so forth.
The main proposal of this paper is to show how to encode this type of knowledge on sound spatialization as constraints, which are interpreted in real time by a constraint propagation algorithm, in the context of MidiSpace. We will first review the main approach in constraints for multimedia systems, and then give examples of typical constraints we need to represent. These examples in turn determine the requirements for the solving algorithm. We then propose an algorithm which achieves some of these requirements.

5.2 Constraints for Interactive Systems

Constraints may be defined as relations between objects that should always be satisfied. Constraints are interesting because they are stated declaratively by the programmer, thereby avoiding him to program complex algorithms.

Constraint techniques are traditionally divided into two categories: constraint satisfaction algorithms (CSP) and constraint propagation algorithms. CSP are used mostly for solving complex combinatorial problems, and are particularly efficient on finite domains, but are usually not usable for reactive systems, which makes them unsuitable in our context.
Constraint propagation algorithms are particularly relevant for building reactive systems (see e.g. [12] for a review), typically for layout management of graphical interfaces, from the pioneer *ThingLab* system [6], to *Kaleidoscope* [13] and more recently *OTI Constraint Solver* [4]. Currently, the choice of a constraint algorithm depends on the nature of constraints, and the nature of constraint sets. One can roughly divide constraint propagation systems into three families : 1) simple algorithms, based on propagation of degrees of freedom, but usually limited to acyclic dataflow constraints, such as *DeltaBlue* [22] or *QuickPlan* [26], 2) specific algorithms addressing particular classes of constraints, such as the algorithm for linear constraints and inequalities proposed in [7], and 3) hybrid algorithms, such as *UltraViolet* [5] or *DETAIL* [11], that attempt to cover all cases by using specialized and cooperating subsolvers.
In the next section we describe the constraints required for MidiSpace, and conclude on the requirement of the constraint propagation algorithm. In section 5.4 we will describe the algorithm designed to solve these constraints efficiently.

5.3 Constraints and Mixing Consistency

This section describes the main constraints needed to define mixing consistency in MidiSpace. Constraints are defined by relations holding on variables. We will first describe the variables needed, and then the relations.

5.3.1 MidiSpace Variables

In MidiSpace, the variables are the following. First there are as many variables as sound sources on the interface. More precisely, each sound source is represented by a point p_i, i.e. two integer variables (one for each coordinate):
p_i, where $p_i = \{x_i, y_i\}$ with $x_i, y_i \in [1, 1000]$ (a typical screen).
Moreover, there is one variable representing the position of the listener's avatar, itself consisting of two integer coordinate variables:
l, where $l = \{x_l, y_l\}$ with $x_l, y_l \in [1, 1000]$.

5.3.2 MidiSpace Constraints

Most of the constraints on mixing involve a collection of sound sources and the listener. We describe here the most useful ones.

- Constant Energy Level

The simplest constraint is the constraint stating that the energy level between several sound sources ($i = 1, .., n$) should be kept constant. According to our model of sound mixing, this constraint can be stated between variables pi, $i = 1, .., n$ as follows:

$$\prod_{i=1}^{n} \|p_i - l\| = Cte$$

Intuitively, it means that when one source is moved toward the listener, the other sources should be "pushed away", and vice-versa. The constant value on the right-hand side of the constraint is determined by the current values of p_i and l when the constraints are defined. In practice, the total energy level may be approximated by a linear expression, yielding:

$$\sum_{i=1}^{n} \|p_i - l\| = Cte$$

Note that this constraint is non linear. Moreover, the constraint is not functional, except in the case of two sources only.

- Constant Angular Offset

This constraint is the angular equivalent of the preceding one. It expresses that the spatial organization between sound sources should be preserved, i.e. that the angle between two objects and the listener should remain constant. It can be stated between variables p_1 and p_2 as follows:

$$(p_1, \hat{l}, p_2) = Cte$$

The constraint is generalized to a collection of objects between variables $p_1,\ldots, p_i\ldots,p_n$.

$$(p_1, \hat{l}, p_2) = Cte_{1,2}\ ;\ (p_1, \hat{l}, p_i) = Cte_{1,i}\ ;\quad (p_1, \hat{l}, p_n) = Cte_{1,n}$$

- Constant Distance Ratio

The constraint states that two or more objects should remain in a constant distance ratio to the listener:

$$\|p_1 - l\| = \alpha_{1,2} \|p_2 - l\|$$

This constraint can be generalized to n objects and the listener:

$$\forall i, j \le n : \|p_i - l\| = \alpha_{i,j} \|p_j - l\|$$

- Radial Limits of Sound Sources

This constraint allows to impose radial limits in the possible regions of sound sources. These limits are defined by circles whose center is the listener's avatar (as represented graphically in Figure 8).

$$\|p_i - l\| \ge \alpha_{\text{inf}} \quad \text{(lower limit)}$$

$$\|p_i - l\| \leq \alpha_{\sup} \quad \text{(upper limit)}$$

- Grouping constraint

This constraint states that a set of n sound sources should remain grouped, i.e. that the distances between the objects should remain constant (independently of the listener's avatar position):

$$\forall i, j \leq n : \left(x_i - x_j\right) = Ctx_{i,j} \text{ and}$$

$$\left(y_i - y_j\right) = Cty_{i,j}$$

5.3.3 More complex constraints

Other typical constraints include symbolic constraints, holding on non geographical variables. For instance, an "Incompatibility constraint" imposes that only one source should be audible at a time : the closest source only is heard, the others are muted. This constraint cannot be expressed as a relation between coordinates.

More complex constraints include the "Equalizing constraint", which states that the frequency ratio of the overall mixing should remain within the range of an equalizer. For instance, the global frequency spectrum of the sound should be flat.

5.4 Constraint algorithm

The examples of constraints given above show that the constraints have the following properties:

- the constraints are not linear. For instance, the constant energy level (between two or more sources) is not linear. This prohibits the use of simplex-derived algorithms, such as [7].
- The constraints are not all functional. For instance, geometrical limits of sound sources are typically inequality constraints.
- The constraints quickly induce cycles. For instance, a simple configuration with two sources linked by a constant energy level constraint and a constant angular offset constraint already yields a cyclic constraint graph.

There is no general algorithm, to our knowledge, which handles non linear, non functional constraints with cycles. *Indigo* [3] is an algorithm for functional constraints with inequalities, but does not handle cycles. Conversely, cycle solvers such as *Purple* (linear constraints) and *DeepPurple* for linear inequalities [5], do not handle non linear constraints. The general solution as proposed in the literature consists in using hybrid algorithms such as *Detail* or *UltraViolet* as mentioned in section 5.2. However, these algorithms add a considerable level of complexity: they are difficult to implement and tune, and may have unexpected behavior [3].

Instead, we designed a simple propagation algorithm which implements only a part of our requirements, but with predictable and reactive behavior. The current algorithm we use is based on a simple propagation scheme, and allows to handle functional constraints, inequality constraints. It handles cycles simply by checking conflicts. Each variable *v* is associated to the set of constraints holding on it (predicate *constraints(v)*). Each functional constraint has a set of methods, used to compute values of output variables from values of input variables. The algorithm is triggered by the modification of one variable, and is described below:

```
// Each variable holds a list of constraints, and each
// constraint holds the list of its variables
// The propagation depends on the type of the constraint
propagate (Constraint c, Variable v)
    if c is functional : propagateFunctional(c, v)
    if c is inequality: propagateInequality(c, v)

propagateFunctionalConstraint(Constraint c, Variable v)
    result = true
    for each variable v' in c. variables, such as v' ≠ v,
        new-value = perform-method (v', v, v.new-value)
result = result && perturbate(v', new-value, c )
    endfor
    return result

// Inequality constraints are just checked
propagateInequalityConstraint(variable v , perturbation v-
perturbation )
    return c.isSatisfied()

// Each variable holds a value (actual current value), and a
// new-value, which represents a perturbation, either triggered
// by the user or computed
perturbate(Variable v, Value new-value, Constraint c)
    result = true
    if  v.value ≠ v.new-value // v has already been perturbated
        //perturbation is the same
        return (v.new-value = new-value)
    endif
    v.new-value= new-value
    for each constraint c' in v.constraints such as c' != c
        result = result && propagate(c', v)
    enfor
    return result
```

Figure 7. Propagation algorithm of MidiSpace

5.5 The interface

The interface for setting constraints is straightforward: each constraint is represented by a button, and constraints are set by first selecting the graphical objects to be constrained, and then clicking on the appropriate constraint button. Constraints themselves are represented by a small ball, whose color depends on the constraint's type, linked to the constrained objects by lines. Some constraints have specific behavior, such as "limit constraints", which show a circle centered on the listener's avatar to display their scope (see Figure 8).

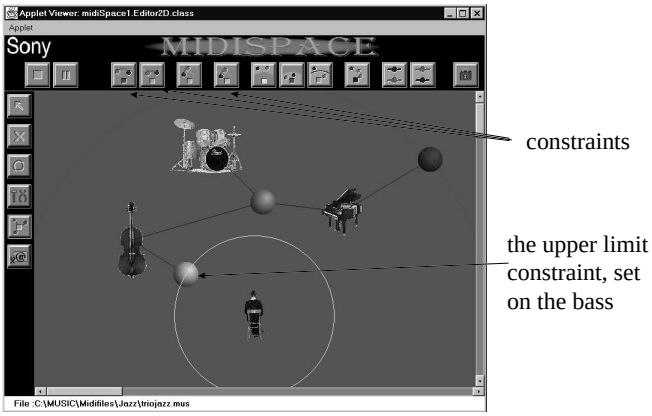

Figure 8. The MidiSpace authoring interface for specifying mixing constraints

Since constraints are themselves represented as graphical objects, they in turn can be constrained to form hierarchies of constrained objects. This allows, for instance, to constrain several groups of already constrained objects (as in Figure 8). Additionally, this mechanism may be used to specify higher-level specifications such as relative ambitus : constrain the upper limit and the lower limit constraints to remain grouped together, using the "constant ratio" constraint.

5.6 Constraints as annotations

The final step in MidiSpace is to allow constraint sets to be dynamically created in time, to reflect changes in music. A simple way to do so is by considering constraints as particular annotations. This requires two additions to MidiSpace : 1) a format for expressing constraints compatible with our annotation format, and 2) a scheme for adding and removing constraints in real time during listening. For the moment, we experimented with a simple approach in which the format of constraints is similar to the format for other annotations, i.e. a time tag, followed by a constraint type (out of a predetermined number of constraint types), and parameters when needed. In this scheme, constraints are represented as temporal objects with a start time and a duration (see Figure 9). It is to be noted that since constraints are considered as fully-fledged objects, they also can be moved, and the movements may in turn be considered as temporal annotations.

[constraints]
startBeat=7.692 duration=2000 type=CtEnergyLevel instr=bass, drums
startBeat=7.698 duration=10000 type=CtEnergyLevel instr=bass, piano
startBeat=7.881 duration=10000 type=CtAntiRelated instr=bass, piano
startBeat=8.067 duration=20000 type=CtUpperBound instr=piano parameter=45 …

Figure 9. Constraints as annotations

The difficult part is the real time handling of the constraint set, and raises two problems:

1) the incrementality of the algorithm. The constraint propagation algorithm should be able to incrementally add or remove constraints, without having to recompute too many variables.

2) the smoothness of the interface. A main concern in building mixing interfaces is that the user should not be lost because of too sudden movements of objects. When a constraint is added, it can be the case that some objects are in positions that violate the constraint. In this case, a natural solution would be to have the objects move smoothly from the old location to the new one. This second issue is not yet handled, and is the subject of current work.

6. IMPLEMENTATION

The implementation of MidiSpace consists in 1) translating Midi information and annotation files into a set of objects within a temporal framework, 2) scheduling these temporal objects using a real time scheduler.

6.1 The Parser

The Parser task is to transform the information contained in the Midi file and in the annotation files into a unified temporal structure. The temporal framework we use is described in [21], an object-oriented, interval-based representation of temporal objects. In this framework, each temporal object is represented by a class, which inherits the basic functionalities from a root superclass *TemporalObject*.

One main issue the Parser must address comes from the way Midi files are organized according to the General Midi specifications. Mixing is realized by sending volume and panoramic Midi messages. These messages are global for a given Midi channel. One must therefore ensure that each instrument appears on a distinct Midi channel. In practice, this is not always the case, since Midi tracks can contain events on different channels. The first task is to sort the events and create logical melodies for each instrument. This is realized by analysing program change messages, which assign Midi channels to instruments, thereby segmenting the musical structure. The second task is to create higher level musical structures from the basic musical objects (i.e. notes). The Midi information is organized into notes, grouped in melodies. Each melody contains only the notes for a single instrument. The total

piece is represented as a collection of melodies. A dispatch algorithm ensures that, at a given time, only one instrument is playing on a given Midi channel.

6.2 The Hierarchy of Playable Objects

These objects represent events that can be scheduled in time. Includes both musical events, such as notes, and also more abstract events such as movements.

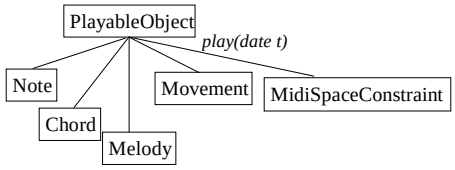

Figure 10. The class hierarchy of playable objects in MidiSpace.

6.3 Scheduling temporal objects

The scheduling of MidiSpace objects uses MidiShare [19], a real time Midi operating system, with a Java API. MidiShare provides the basic functionality to schedule asynchronously, in real time, Midi events, from Java programs, with 1 millisecond accuracy. The main loop of the Midi player consists in 1) creating a task that schedules all events that fall on the current date, and 2) rescheduling the task to the next date. When an event is scheduled, it is sent the method *play(t)*, with the date as parameter. This method is implemented in all subclasses of *PlayableObject*. Note objects implement the *play(t)* method by sending an appropriate Midi message. Thanks to this representation, it is straightforward to implement dynamically changing constraint sets, by simply ensuring that constraints implement the *PlayableObject* interface, and the method *play(t)*. This method will simply add the constraint to the current constraint set at the start date, and remove it at the end date.

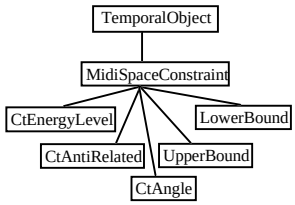

Figure 11. An excerpt of the constraint hierarchy

7. CONCLUSION, FUTURE WORK

The MidiSpace system shows that it is possible to give users some degree of freedom in sound spatialization, while preserving some semantics on the mixing of sound sources. The prototype built so far validates our approach, but future work remains to be done in several directions. First we are currently improving the constraint propagation algorithm to handle more complex constraints, such as proposed in 5.3.3. These constraints should allow users or composers to specify finer mixing configurations.

Second, we are experimenting with other interfaces for navigation. A 3D version of MidiSpace in VRML is in progress (see Figure 12), in which the VRML code is automatically generated from the 2D interface. Although the result is clearly stimulating, it is not yet fully satisfying because the 3D interface gives too little information on the overall configuration of instruments, which is a crucial parameter for spatialization, but this problem is a general problem with 3D interface, and is not specific to MidiSpace.

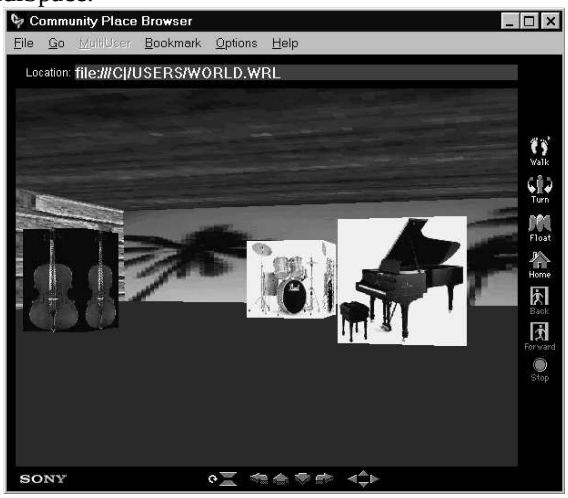

Figure 12. MidiSpace/VRML on the Jazz trio.

Finally, an audio version is in progress, to 1) enlarge the repertoire of available music material to virtually all recorded music, and 2) improve the quality of the spatialization, using more advanced techniques such as the ones sketched in Section 3.

8. REFERENCES

[1] Ackermann P., Developing object-oriented multimedia software, Dpunkt, Heidelberg, 1996.

[2] Assayag G., Agon C., Fineberg J., Hanappe P., "An Object Oriented Visual Environment For Musical Composition", Proceedings of the International Computer Music Conference, pp. 364-367, Thessaloniki, 1997.

[3] Borning A., Anderson R., Freeman-Benson B., "Indigo: A Local Propagation Algorithm for

Inequality Constraints", Proceedings of the ACM Symposium on User Interface Software and Technology, pp. 129-136, 1996.

[4] Borning A., Freeman-Benson B., "The OTI Constraint Solver: a Constraint Library for Constructing Interactive Graphical User Interfaces", Proceedings of the First International Conference on Principles and Practice of Constraint Programming, pp. 624-628, 1995.

[5] Borning A., Freeman-Benson B., "Ultraviolet: A Constraint Satisfaction Algorithm for Interactive Graphics", Constraints, Special Issue on Constraints, Graphics, and Visualization, Vol. 3 No. 1, pp. 9-32, April 1998.

[6] Borning A., "The Programming Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory", ACM Transactions on Programming Languages and Systems, 3, pp. 353-387, 1981.

[7] Borning A. Lin R., Marriott K., "Constraints for the web", Proceedings of ACM Multimedia Conference, Seattle, pp. 173-181, 1997.

[8] Chowning J., "The simulation of moving sound sources", JAES, vol. 19, n. 1, p. 2-6, 1971.

[9] Dai P., Eckel G., Göbel M., Hasenbrink F., Lalioti V., Lechner U., Strassner J., Tramberend H., Wesche G., "Virtual Spaces: VR Projection System Technologies and Applications", Tutorial Notes, Eurographics '97, Budapest, 1997.

[10] Eckel G., "Exploring Musical Space by Means of Virtual Architecture", Proceedings of the 8th International Symposium on Electronic Art, School of the Art Institute of Chicago, 1997.

[11] Hosobe H., Matsuoka S., Yonezawa A., "Generalized local propagation: a framework for solving constraint hierarchies", Proceedings of CP' 96, Boston, 1996.

[12] Hower W., Graf W. H., "a Bibliographical Survey of Constraint-Based Approaches to CAD, Graphics, Layout, Visualization, and related topics", Knowledge-Based Systems, Elsevier, vol. 9, n. 7, pp. 449-464, 1996.

[13] Lopez G., Freeman-Benson B., Borning A., "Kaleidoscope: A Constraint Imperative Programming Language", In Constraint Programming, B. Mayoh, E. Tougu, J. Penjam (Eds.), NATO Advanced Science Institute Series, Series F: Computer and System Sciences, Vol 131, Springer-Verlag, pages 313-329, 1994.

[14] IMA, "MIDI musical instrument digital interface specification 1.0", Los Angeles, International MIDI Association, 1983.

[15] Jot J.-M., Warusfel O., "A Real-Time Spatial Sound Processor for Music and Virtual Reality Applications", Proceedings of ICMC, 1995.

[16] Laurson M., Duthen J., "PatchWork, a graphical language in PreForm", Proceedings of ICMC, San Francisco, 172-175, 1989.

[17] Lea R., Matsuda K., Myashita K., *Java for 3D and VRML worlds*, New Riders Publishing, 1996.

[18] Meyer L., *Emotions and meaning in music*, University of Chicago Press, 1956.

[19] Orlarey Y., Lequay H., "MidiShare: a real time multi-tasks software module for Midi applications", Proceedings of ICMC, San Francisco, 1989.

[20] Pachet F., "Computer Analysis of Jazz Chord Sequences. Is Solar a Blues ?", Readings in Music and Artificial Intelligence, Harwood Academic Publishers, to appear, 1998.

[21] Pachet F., Ramalho G., Carrive J. "Representing temporal musical objects and reasoning in the MusES system", Journal of New Music Research, vol. 25, n. 3, pp. 252-275, 1996.

[22] Sanella M., Maloney J., Freeman-Benson B., Borning A., "Multi-way versus one-way constraints in user interfaces: experiences with the DeltaBlue algorithm", Software Practice and Experience, 23(5):529-566, 1993.

[23] Sloan Donald, "Aspects of Music Representation in Hytime/SMDL", Computer Music Journal, Cambridge, MA, MIT Press, 17:4, Winter 1993.

[24] SMDL, Draft International Standard, ISO/IEC CD 10743, 1995.

[25] Taube H., "Common Music: A Music Composition Language in Common Lisp and CLOS", Computer Music Journal, vol. 15, n° 2, 21-32, 1991.

[26] Vander Zanden Brad, "An incremental algorithm for satisfying hierarchies of multi-way dataflow constraints", ACM Transactions on Programming Languages and Systems, 18(1):30-72, 1996.