TIMED SEQUENCES: A FRAMEWORK FOR COMPUTER-AIDED COMPOSITION WITH TEMPORAL STRUCTURES

Jérémie Garcia
Universit de Toulouse – ENAC
Toulouse, France
`jeremie.garcia@enac.fr`

Dimitri Bouche, Jean Bresson
UMR STMS
IRCAM-CNRS-UPMC Sorbonne Universités
Paris, France
`{bouche,bresson}@ircam.fr`

ABSTRACT

The software framework we present implements a simple and generic representation of the temporal dimension of musical structures used in computer-aided composition software. These structures are modeled as ordered sets of abstract "timed items" whose actual dates can be set and determined following different strategies. The timed items can be linked to an underlying action scheduling and rendering system, and can also be used as temporal handles to perform time stretching and hierarchical synchronization operations. A graphical user interface associated with this model can be embedded as a component within musical editors. We give several examples of musical objects implemented in this framework, as well as examples of time-domain operations and user interactions.

1. INTRODUCTION

Contemporary music composers commonly use computer systems to generate musical structures (scores, sounds, control data for signal and media processing). Most of these structures embed a fundamental notion of timing, which is expressed differently depending on their nature, on the tools used to create or manipulate them, or on the approach of the composer working and producing them. Working with time in consistent and efficient ways is therefore an important and challenging issue in computer music practice and research [1, 2].

Let us take the example of a compositional process involving the control of sound spatialization and the motion of sound sources defined with a set of 2D- or 3D-trajectories — sequences of pairs {*time, 3D-position*} for the different sound sources. Composers can face here several non-trivial time-related tasks such as the inner timing of the trajectories, their synchronization with the content of the sound sources or with other trajectories, etc. They may also want to integrate these trajectories within a higher-level time-structure along with other musical data — typically, in a score or in a Digital Audio Workstation. Besides, all these operations are likely to be repeated and spread at different levels of the compositional process, and applied to other types of musical objects with similar temporal characteristics.

Various approaches have been explored to unify and synchronize temporal dimensions of signal and symbolic structures in computer music environments, either from a graphical point of view [3] or from a more formal, logical perspective using temporal constraints [4], however without focusing explicitly on user-interaction.

From a composer's perspective, Stroppa and Duthen [5] introduced the concept of *pivots*, as "virtual anchors that are used to describe temporal objects [which] act as a skeleton of the temporal structure of the object and will be used to organise various elements together". Bel [6] followed a similar approach to describe musical objects and satisfy temporal constraints, but used a simpler model using a unique *pivot* for each object, thus not capturing any possible complexity in the internal time structures.

The present project takes place in the OpenMusic computer-aided composition environment [7]. It originates from previous work on the control of sound spatialization processes [8], and aims at facilitating the creation of musical objects by providing a unified framework to support both computational and graphical interaction for timing operations.

We propose a generic representation and software framework designed as a "temporal backbone" for musical structures, representing them as simple sequences of timed components, which we call *timed-items*. Timed-items can have different roles to make for explicit structuring, time transformations, or synchronization operations. They help handling time both at the micro level, to specify the inner structure of the musical objects, and at the macro level, to organize them in compound structures.

We describe the architecture and the main characteristics of the model in Section 2. We then present the corresponding graphical user interface and interactions in Section 3. In Section 4, we give several examples of musical objects and interfaces implemented in OpenMusic. In Section 5, we describe internal and external synchronization operations, and how this representation system supports the definition and manipulation of global time structures in a compositional process.

2. MODEL AND ARCHITECTURE

2.1 Timed Sequences

We consider timed musical objects as subclasses of an abstract superclass that we call a TIMED-SEQUENCE. This class holds an ordered set of "timed items" which represent basic components determining the temporal organisation inside the musical object. Public accessors allow the user (or programmer) to deal seamlessly with this list of items. Subclasses of TIMED-SEQUENCE can either use these accessors to maintain it during the life of an instance, or redefine them in order to redirect reading/writing and timing operations to existing attributes of the structure. A couple of additional methods can also be overridden to redefine or complement the addition or deletion procedures of timed items in a TIMED-SEQUENCE.

The main application programming interface (API) contains the following functions:

- **get-timed-item-list:**
  $\Rightarrow$ *returns the list of timed items of a* TIME-SEQUENCE
- **set-timed-item-list:**
  $\Rightarrow$ *sets a new list of timed items in a* TIME-SEQUENCE
- **make-timed-item-at-time:**
  $\Rightarrow$ *returns an item created at a given date in a* TIME-SEQUENCE
- **remove-timed-item:**
  $\Rightarrow$ *removes an given item from a* TIME-SEQUENCE

2.2 Timed Items

The class TIMED-ITEM is also provided as a default superclass for the elements in a TIMED-SEQUENCE. Each item has a time value (or *date*), which can be explicit (specified by the user) or implicit. It is not mandatory indeed, that all the elements in a TIMED-SEQUENCE have an explicit date: if we consider a trajectory describing a motion in space, for instance, a global duration might be specified for the motion, without a specific date assigned to every point in that trajectory: in this case, points' timing can remain implicit.

If its date is implicit, however, a TIMED-ITEM must be included in a sequence where an explicitly-dated item is present before and after it. The computation of a date is then possible provided a measure of distance exists between two items of a given type, which allows for time interpolation.[1] Implicit dates are therefore computed "on-deman" (as in a lazy-evaluation approach) and are implicitly updated by any change in the TIMED-SEQUENCE.

Finally, a special tag (*master*) can be assigned to the TIMED-ITEMs, meaning that they can be used as anchors for global time transformation or synchronization operations. Only explicitly-timed items can be tagged as *master*, and the first and last items of a sequence are always considered as *master*: they have a date assigned by default, and are privileged anchors to stretch, compress or synchronize the musical structures.

[1] The same measure of distance, in the case of spatial trajectories for instance, can be used to determine if two successive items are at the same position (distance = 0). This implies an idea of stability or steadiness of the structure over the duration defined by the two items.

2.3 Time-modification of Items

Interactions with the TIMED-SEQUENCE mostly consist in setting the date of the TIMED-ITEMs by different means: graphical or algorithmically. Depending on the type of the item, this operation will yield different effects as illustrated in Figure 1:

- Setting the date or moving an "untimed" item makes it explicitly *timed*.
- Changing the date of an explicitly timed item affects and updates the implicit date of its direct and indirect untimed neighbours.
- Changing the date of a *master* item changes the date of its direct or indirect timed neighbours and thus impacts the implicit date of their direct and indirect untimed neighbours as well.

The set of possible types of item: $\{untimed, timed, master\}$ is therefore strictly ordered, which introduces a hierarchy in the temporal structure that influences the way explicit or implicit dates are computed: the date of an item is systematically updated according to the dates of its closest left and right neighbours of a superior type.

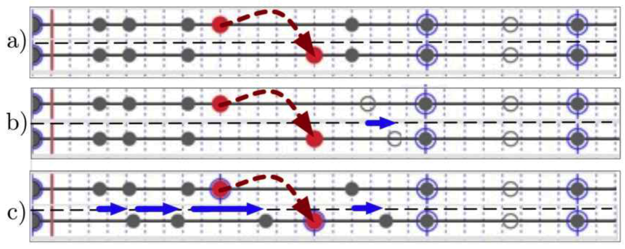


Figure 1. Impact of the time-modification of a timed-item (horizontal arrows indicate the changes yield by the initial item translation). a) Simple translation of a *timed* item (with timed neighbours — no side-effect). b) Translation with update of an item with no explicit date on the right (preserving a constant ratio). c) Translation and time stretching between *master* timed items.

2.4 Rendering

The TIMED-SEQUENCE model also allows for a seamless integration of musical objects in a generic rendering system [9]. In addition to the previous API, the method **collect-actions** allows a TIMED-SEQUENCE to return a list of actions and tasks to execute, related to the rendering of the structure in a given time interval. Actions can just be assigned to TIMED-ITEMs, so that playing the musical sequence can be modeled as reading through the list of TIMED-ITEMs in the corresponding interval, and triggering the corresponding actions. They can also be generated or interpolated to produce more complex sequences of actions dynamically: sampling can be achieved for instance by periodic calls to the **make-timed-item-at-time** function, not modifying the original sequence of TIMED-ITEMs.

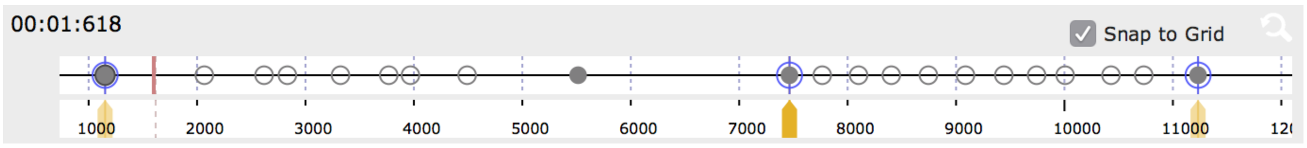


Figure 2. Main view of the *timeline-editor* associated to a TIME-SEQUENCE. The TIMED-ITEMs of different types are represented respectively as ○ (*untimed* / implicit time), ● (*timed*) and ⊙ (*master*).

3. GRAPHICAL EDITOR AND INTERACTIONS

A graphical user interface (GUI) is associated to the TIMED-SEQUENCE. This GUI allows the visualization of the TIMED-ITEMs and the execution of the most common operations on time structures. The main view (called *timeline-editor*, see Figure 2) lets the user select, add or delete timed items via calls to the API functions. For instance, adding a point in the *timeline-editor* (*command+click*) triggers a call to **make-timed-item-at-time** and adds the returned item to the sequence using **set-timed-item-list**. Setting/changing the date of TIMED-ITEMS is essentially done by translations on the timeline using the mouse. Keyboard short-cuts let user change the "type" of the items, mainly to turn standard items to master items and vice-versa.

Each master item creates a marker in the ruler visible at the bottom of the view. The marker, displayed as a yellow arrow, can be used as a proxy to move and synchronize the master item with the ruler. A cursor is also displayed as a vertical red line (around 1600 ms in Figure 2), which can be linked to the playback and rendering functionalities of the host environment to display or set the current playtime.

4. IMPLEMENTATION OF MUSICAL OBJECTS

A core set of objects can be defined in the generic framework introduced so far. It is possible to build such objects directly on top of the proposed architecture (Section 4.1), or by adapting existing objects to implement its API (Sections 4.2 and 4.3). The TIMED-SEQUENCE API and the *timeline-editor* handle the time representation and manipulations of the integrated objects.

4.1 Basic Timed Sequences

The TIMED-SEQUENCE model defines a generic type of object to use in computer music applications. We call DATA-STREAM a simple sequence of timed events directly inheriting from TIMED-SEQUENCE. The timed items in a DATA-STREAM are called DATA-FRAMEs. They contain data and perform specific actions to render these data. Subclasses of DATA-FRAME include for instance MIDI events or Open Sound Control (OSC) bundles, where data is a set of messages and the rendering transmits the messages via dedicated protocols.

Figure 3 shows a prototype editor created for DATA-STREAM objects, which plots DATA-FRAMES (in this case, representing OSC bundles) along the horizontal time axis. The *timeline-editor* at the bottom is included as a component in the main editor, and enables time manipulations on the data frames, such as local/piecewise stretch and compression, snap to grid, etc.

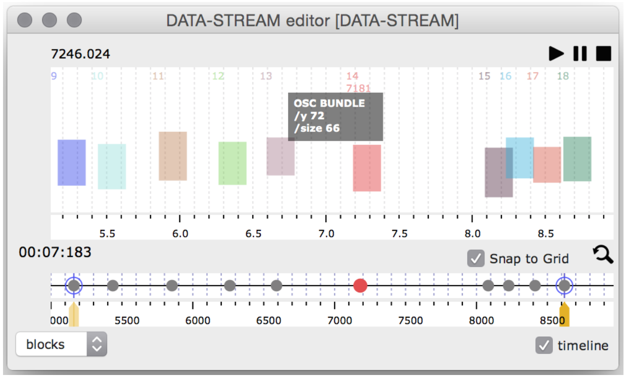


Figure 3. Editor for a sequence of data (class DATA-STREAM) containing DATA-FRAMES (in this case, OSC bundles). Each DATA-FRAME is represented as a coloured shape. The graphical parameters (shape, colour, size or vertical position of the frames) are assigned using an arbitrary mapping with the data. Note the *timeline-editor* GUI component at the bottom.

A specialization of the previous structures allows for the implementation of a PIANO-ROLL representation as shown on Figure 4. The class MIDI-NOTE is a specific DATA-FRAME containing pitch, velocity and duration information. When rendered, it produces two actions sending MIDI *key-on* and *key-off* events.

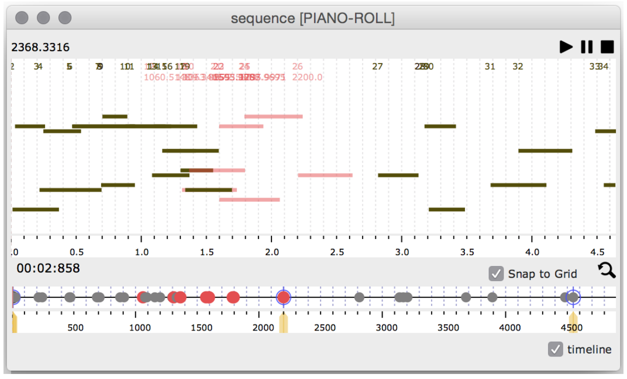
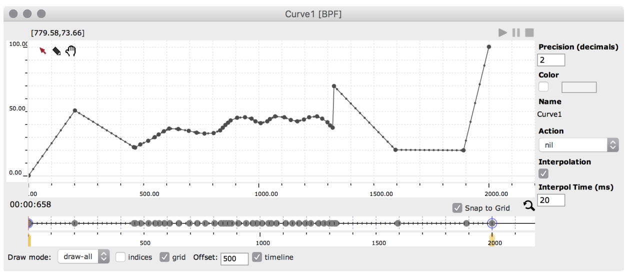


Figure 4. Editor for a sequence of MIDI notes (PIANO-ROLL) including a *timeline-editor*.

Since the model lets musical objects being built without exhaustive specific timing, one could consider specifying the date of two master items in a DATA-STREAM (DATA-FRAMES or MIDI-NOTES) and let the system interpolate intermediate time values.

4.2 Timed Controllers: Curves and Automations

The TIMED-SEQUENCE API can also be used to improve time manipulations in existing objects, such as the timed controllers and automations commonly used in computer music systems. It is straightforward to implement the accessors presented in Section 2 with a break-points function (BPF) object, which contains an ordered list of 2D points, and where the time dimension is implicitly associated to one of these dimensions (x / horizontal axis). Likewise, the *timeline-editor* can be embedded as a component in a BPF editor, as presented in Figure 5.


Curve1 [BPF]

[779.58,73.66]

00:00:658

Precision (decimals): 2

Color

Name: Curve1

Action: nil

Interpolation: ✓

Interpol Time (ms): 20

✓ Snap to Grid

Draw mode: draw-all; indices; ✓ grid; Offset: 500; ✓ timeline

Figure 5. Editor of a break-point function implementing the TIME-SEQUENCE API, including a *timeline-editor* component. Each point in the graph is considered as a TIMED-ITEM.

4.3 Trajectories

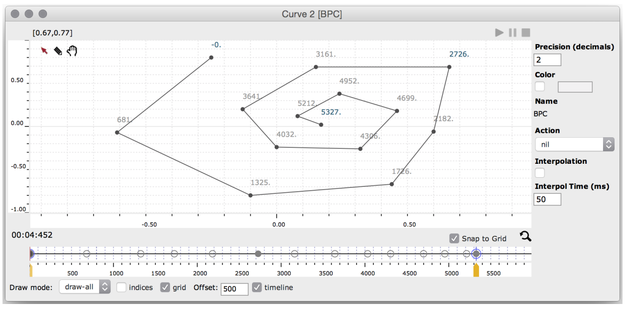
As mentioned in the introduction, timing in trajectory specification is often a delicate problem. Indeed, although efficient interfaces exist to draw and design 2D or 3D curves, time specification must generally be done piecewise or using a global duration given to the entire movement.

The integration of the TIMED-SEQUENCE model in 2D or 3D curves is done through the definition of "timed points", an extension of 2D/3D points (including x, y and z coordinates) and subtype of TIMED-ITEM. The embedded API and graphical interface allow the user to easily perform time manipulations on such graphical structures.

In Figure 6 two master points control the global scaling and synchronization of a trajectory. A timed point with a defined date near the middle of the sequence splits the overall morphology into two parts with relatively equivalent durations. The rest of the points have no explicit date (their positioning in time will be computed on-demand whenever needed, according to the dates of the timed and master points).

5. SYNCHRONIZATION

Synchronization is one of the main time-domain operations performed in musical software, and is often identified as a key element of computer music systems [10]. Our framework facilitates the implementation of intuitive synchronization tools by connecting multiple timelines.


Curve 2 [BPC]

[0.67,0.77]

00:04:452

Precision (decimals): 2

Color

Name: BPC

Action: nil

Interpolation

Interpol Time (ms): 50

✓ Snap to Grid

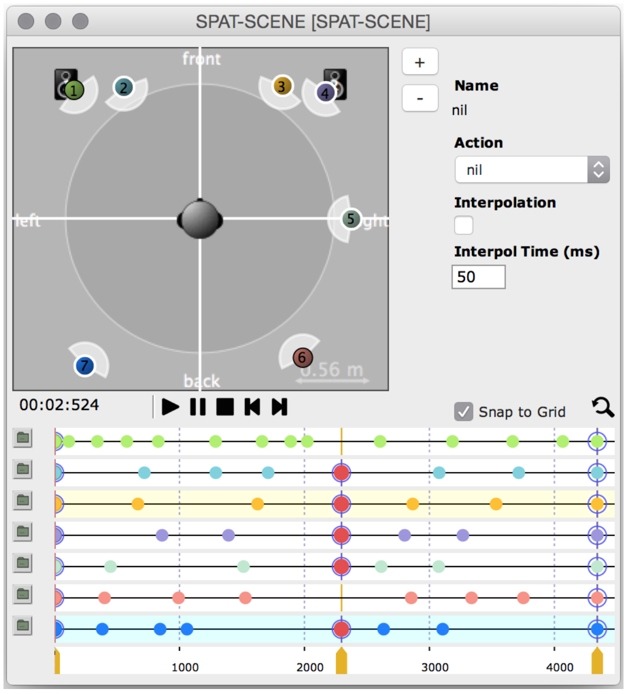
Draw mode: draw-all; indices; ✓ grid; Offset: 500; ✓ timeline

Figure 6. Editor of a 2D-curve in which only three points have an explicit date. The other (implicit) dates are deduced to preserve a constant speed across the different segments.

5.1 Internal Synchronization

Sticking with the sound spatialization example, let us now consider the case of multiple trajectories to control the motions of several sound sources. Time-synchronization strategies are crucial in this case: how to make specific (spatial) regions match in time?

The SPAT-SCENE is an interactive object/controller designed for sound spatialization processes, made up from a set of 3D trajectories (TIMED-SEQUENCEs) and connected to interactive (real-time) visualization and rendering [11]. Figure 7 shows the editor developed for this object, where the *timeline-editor* represents each trajectory in an individual track.


SPAT-SCENE [SPAT-SCENE]

Name: nil

Action: nil

Interpolation

Interpol Time (ms): 50

00:02:524

✓ Snap to Grid

Figure 7. SPAT-SCENE editor. Each sound spatialization trajectory is associated with a timeline in the *timeline-editor* component at the bottom.

The aggregation of timeline views dedicated to each trajectory in a single *timeline-editor* enables the implementation of local synchronization strategies in such compound objects. First, the snap-to-grid functionality supports the adjustment of the time positions to the closest items or grid element within a certain range. This helps users to precisely set items from different sequences at the same date. Second, the markers created by the master items fuse when they are at the same date, and therefore act as proxies to all master items situated at this particular date. This allows several master items from different timelines that have been synchronized to be moved simultaneously, and facilitates stretching and compression operation to the neighbour segments of all corresponding objects. Figure 7 illustrates this behaviour with several master items being selected and moved through the user interacting with a single marker.

5.2 External Synchronization

The synchronization of master items can also occur at a higher structural level, when several temporal objects are combined to form more complex musical structures, for instance within a sequencer.

Figure 8 illustrates a scenario in which the user decides to synchronize two hand-drawn audio effect automations with specific parts of a sound file. The scenario is implemented in a musical container interface currently developed in OpenMusic on top of the same *timed-sequence* model.

Before synchronizing the objects, the composer annotates the sound file with two markers in order to define the beginning of two "sections" A and B, and adds a master point in *BPF2* to synchronize with these markers. All *master* items are collected out of the objects to their container's context, and lifted to the time ruler at the top of the sequencer interface (represented as vertical yellow lines). From there, the targeted operation requires dragging only a couple of markers in the time ruler.

In Figure 8.a the beginning of both automation curves (considered master points by default) is synchronized with the first marker in the sound (beginning of section A). The user action ① then synchronizes the central master point of *BPF2* (marker displayed in red on the ruler at the top) with the last point of *BPF1* (also considered as a master/synchronization point by default). This action ties together the two items, which can then be manipulated as a single entity.[2] In Figure 8.b, the user performs action ② to position this "grouped" marker at the beginning of section B of the sound file, thereby modifying the length of *BPF1* and the relative lengths of the two segments in *BPF2*. Finally, in Figure 8.c, action ③ connects the end point of *BPF2* to the end of the sound file, in order to adjust the duration of the second segment of the automation. Figure 8.d shows the resulting sequence with synchronized markers and items.

[2] Similarly to the internal synchronization, a *snap* functionality allows any dragged marker to be adapted to the position of the closest one within a given time window.

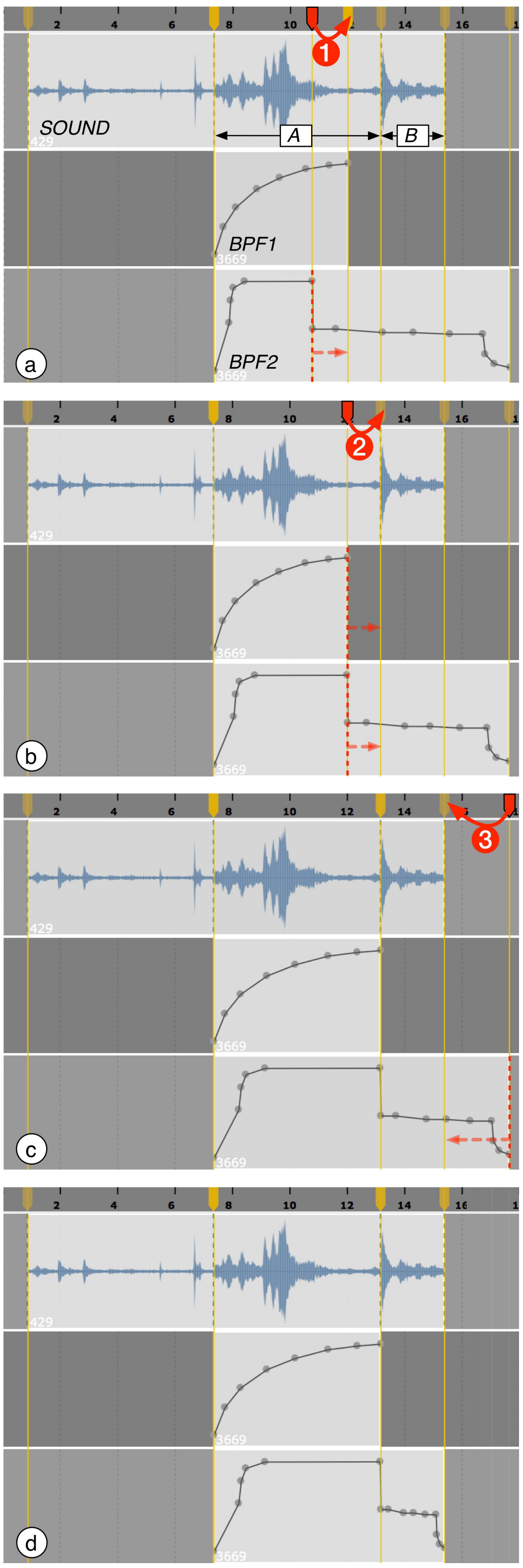


Figure 8. Synchronization of two effect automation curves with a sound. ①, ② and ③ represent user actions.

6. CONCLUSION

We introduced a programming and a graphical user interface framework for the representation of time in musical objects. We described the underlying concept of TIMED-SEQUENCE, an abstract representation containing an ordered set of TIMED-ITEMs, and the corresponding API used to represent musical objects through these simple structures. TIMED-ITEMs also facilitate explicit structuring of an object by defining temporal anchors that can be used to perform stretching and synchronization operations, both internally within an object, or externally with other objects.

This framework therefore enables expressive means to work with time either algorithmically or via graphical user interfaces, and provides end-users with consistent visualization and interaction mechanisms.

The TIMED-SEQUENCE model is currently implemented in the Common Lisp Object System [12] and used as a basis for the design of new interfaces and time structures in the OpenMusic environment.

Future work will focus on the representation of symbolic notation with our model in order to propose a comprehensive systems for composers to describe and process time structures in score-oriented frameworks. We also plan to explore more advanced interaction mechanisms, for instance considering weighted timed-items to control more sophisticated stretching and synchronization operations.

7. ACKNOWLEDGEMENTS

This work is supported by the French National Research Agency project EFFICACe ANR-13-JS02-0004.

8. REFERENCES

[1] R. B. Dannenberg, "Music Representation Issues, Techniques, and Systems," *Computer Music Journal*, vol. 17, no. 3, pp. 20–30, 1993.

[2] H. Honing, "Issues on the Representation of Time and Structure in Music," *Contemporary Music Review*, vol. 9, no. 1-2, pp. 221–238, 1993.

[3] J. Bresson and C. Agon, "Scores, Programs and Time Representations: The Sheet Object in OpenMusic," *Computer Music Journal*, vol. 32, no. 4, 2008.

[4] A. Allombert, M. Desainte-Catherine, J. Laralde, and G. Assayag, "A System of Interactive Scores Based on Qualitative and Quantitative Temporal Constraints," in *ARTECH 2008. Proceedings of the 4th International Conference on Digital Arts*, Porto, Portugal, 2008.

[5] M. Stroppa and J. Duthen, "Une représentation de structures temporelles par synchronisation de pivots," in *Colloque Musique et Assistance Informatique*, Marseille, France, 1990.

[6] B. Bel, "Time-setting of sound-objects: a constraint-satisfaction approach," in *International Workshop on Sonic Representation and Transform*, Trieste, Italy, 1992.

[7] J. Bresson, C. Agon, and G. Assayag, "OpenMusic: Visual Programming Environment for Music Composition, Analysis and Research," in *Proceedings of the ACM international conference on Multimedia – Open-Source Software Competition*, Scottsdale, USA, 2011.

[8] J. Bresson and M. Schumacher, "Representation and Interchange of Sound Spatialization Data for Compositional Applications," in *Proceedings of the International Computer Music Conference*, Huddersfield, UK, 2011.

[9] D. Bouche and J. Bresson, "Planning and Scheduling Actions in a Computer-Aided Music Composition System," in *Proceedings of the 9th International Scheduling and Planning Applications Workshop (SPARK)*, Jerusalem, Israel, 2015.

[10] P. Desain and H. Honing, "Towards a Calculus for Expressive Timing in Music," *Computers in Music Research*, vol. 3, pp. 43–120, 1991.

[11] J. Garcia, T. Carpentier, and J. Bresson, "Interactive-Compositional Authoring of Sound Spatialization," *Journal of New Music Research*, vol. 46, no. 1, 2017.

[12] R. P. Gabriel, J. L. White, and D. G. Bobrow, "CLOS: Integration Object-oriented and Functional Programming," *Communications of the ACM*, vol. 34, no. 9, pp. 29–38, 1991.