# A visual framework for dynamic mixed music notation

Grigore Burloiu, Arshia Cont, Clement Poncelet

# A visual framework for dynamic mixed music notation

Grigore Burloiu[1], Arshia Cont[2] and Clement Poncelet[2]
[1]University Politehnica Bucharest, Romania – gburloiu@gmail.com
[2]INRIA and Ircam (UMR SMTS - CNRS/UPMC) Paris, France

**Abstract**

We present a visual notation framework for real-time, score-based computer music where human musicians play together with electronic processes, mediated by the *Antescofo* reactive software. This framework approaches the composition and performance of mixed music by displaying several perspectives on the score's contents. Our particular focus is on *dynamic* computer actions, whose parameters are calculated at run-time. For their visualization, we introduce four models: an extended *action view*, a staff-based *simulation trace*, a tree-based *hierarchical display* of the score code, and an out-of-time *inspector* panel. Each model is illustrated in code samples and case studies from actual scores. We argue the benefits of a multifaceted visual language for mixed music, and for the relevance of our proposed models towards reaching this goal.

**Keywords:** dynamic scores, visualisation, notation, mixed music, *Antescofo*

## 1 Introduction

We approach the issue of notation for the authoring and performance of *mixed music*, which consists of the pairing of human musicians with computer processes or electronic equipment, where each side influences (and potentially anticipates) the behaviour of the other. The term has been used along the history of electronic music, referring to different practices involving tape music, acousmatic music or live electronics (Collins et al., 2014).

In the 1990s, real-time sound processing gave birth to various communities and software environments like Pd (Puckette, 1997), Max (Cycling '74, 2016) and SuperCollider (McCartney, 1996), enabling *interactive music* situations between performers and computer processes on stage (Rowe, 1993). In parallel, computer-assisted composition tools have evolved to enrich data representation for composers interested in processing data offline to produce scores or orchestration, such as *OpenMusic* (Assayag et al., 1999) and *Orchids* (Nouno et al., 2009).

Composer Philippe Manoury has theorised a framework for mixed music (Manoury, 1990), introducing the concept of *virtual scores* as scenarios where the musical parameters

are defined beforehand, but their sonic realisation is a function of live performance. One example is the authoring of music sequences in beat-time and relative to human performer's tempo; another example is the employment of generative algorithms that depend on the analysis of an incoming signal (see (Manoury, 2013) for an analytical study).

Despite the wide acceptance of interactive music systems, several composers have provided insights into the insufficient musical considerations and potential abuse of the term "interactive" in such systems. Among these, we would like to cite the work of Marco Stroppa (Stroppa, 1999), Jean-Claude Risset (Risset, 1999) and Joel Chadabe (Chadabe, 1984). In his work, Stroppa asks the musical question of juxtaposing multiple scales of time or media during the composition phase, and their evaluation during performance. He further remarks on the poverty of musical expressivity in then state-of-the-art real-time computer music environments as opposed to computer-assisted composition systems or sound synthesis software. Risset takes this one step further, arguing that interactive music systems are less relevant for composition than performance. Finally, Chadabe questions the very use of the term "interaction" as opposed to "reaction". Effectively, many such systems can be seen as *reactive systems* (computers reacting to musicians' input), whereas interaction is a two-way process involving both specific computing and cognitive processes.

To address the above criticisms and to define the current state of the art in computational terms, we turn to the concept of *dynamic* mixed music, where the environment informs the computer actions during runtime. In this paradigm, computer music systems and their surrounding environments (including human performers) are integral parts of the same system and there is a feedback loop between their behaviours. An ubiquitous example of such dynamics is the act of collective music interpretation in all existing music cultures, where synchronisation strategies are at work between musicians to interpret the work in question. Computer music authorship extends this idea by defining processes for composed or improvised works, whose form and structure are deterministic on a large global scale but whose local values and behaviour depend mostly on the interaction between system components—including human performers and computers/electronics. The two-way interaction standard in (Chadabe, 1984) is always harder to certify when dealing with cognitive feedback between machine generated action and the human musician, and many of the programming patterns might be strictly described as *reactive* computing. For this reason we argue that the computationally *dynamic* aspect of the electronics should be their defining trait.

A uniting factor in the diversity of approaches to mixed music can be found in the necessity for *notation* or *transcription* of the musical work itself. A survey on musical representations (Wiggins et al., 1993) identified three major roles for notation: recording, analysis and generation. We would specifically add performance to this list. Despite advances in sound and music computing, there is as yet no fully integrated way for composers and musicians to describe their musical processes in notations that include both *compositional* and *performative* aspects of computer music, across the offline and real-time domains (Puckette, 2004), although steps in this direction can be seen in the OSSIA framework for the i-score system (Celerier et al., 2015).

This paper attempts to provide answers to the problem of musical notation for dy-

namic mixed music. The work presented here is the outcome of many years of musical practice, from composing to live performance of such pieces and in the context of the *Antescofo* (Cont, 2008) software used today in numerous new music creations and performances worldwide.[1] We start with a brief survey of the current state of mixed music notation. In this context we look at the *Antescofo* language and its basic visual components, before turning to the main focus of the paper: notational models for dynamic processes.

Note: throughout this paper, we use the word *dynamic* and its derivatives in the computer science sense, signifying variability from one realisation to another. We occasionally use *dynamics* as a short-hand for dynamic values or processes. Please do not confuse this with the musical sense of the word, which refers to the intensity of playing.

## 1.1 Mixed music notation at a glance

To facilitate the understanding of the field, we distinguish notational strategies for mixed music into three categories, before noting how composers might combine them to reach different goals.

### 1.1.1 Symbolic graphical notation

Early electroacoustic music scoring methods evolved in tandem with the expansion of notation in the mid-20th century towards alternative uses of text and graphics. Written instructions would specify, in varying degrees of detail, the performance conditions and the behaviours of the musicians. In addition, symbolic graphical representations beyond the traditional Western system could more suggestively describe shapes and contours of musical actions. While these methods can apply to purely acoustic music, the introduction of electronics came without a conventional performative or notational practice, and so opened up the space of possibilities for new symbolic graphical notation strategies.

Major works of this era include Karlheinz Stockhausen's *Elektronische Studie I* and *II* (1953-54), the latter being the first published score of pure electronic music (Kurtz, 1992). These examplified a workflow where the composer, usually aided by an assistant (such as G. M. Koenig at the WDR studio in Cologne), would transcode a manually notated symbolic score into electronic equipment manipulations or, later, computer instructions, to produce the sonic result. An instance of this paradigm, which continues today, is the collaboration of composer Pierre Boulez and computer music designer Andrew Gerzso for *Anthemes II* (1997), whose score is excerpted in Figure 1.

### 1.1.2 Graphics-computer integration

Before long, composers expressed their need to integrate the notation with the electronics, in order to reach a higher level of expressiveness and immediacy. Xenakis' UPIC (Xenakis, 1992) is the seminal example of such a bridging technology: his *Mycenae-α* (1978) was first notated on paper before being painstakingly traced into the UPIC. The impetus to enhance the visual feedback and control of integrated scoring led to systems such as the

[1] An incomplete list is available at `http://repmus.ircam.fr/antescofo/repertoire`.

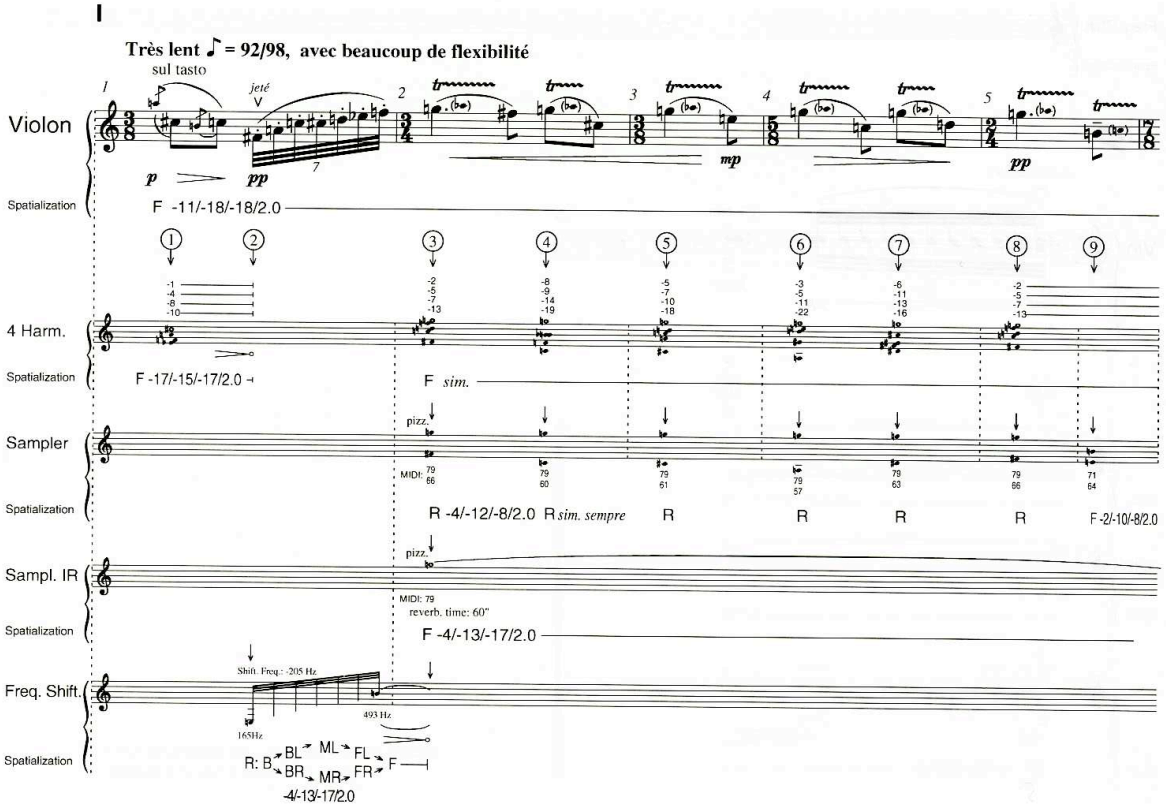



**Figure 1:** *The first page of the* Anthemes II *score for violin and electronics, published by Universal Edition. The marked cues correspond to electronic action triggerings.*

SSSP tools (Buxton et al., 1979), which provided immediate access to several notation modes and material manipulation methods. Along this line, the advent of real-time audio processing brought about the integrated scores of today, underpinned by computer-based composition/performance environments. Leading examples in the field are the technical documentation databases at IRCAM's Sidney[2] or the Pd Repertory Project[3], hosting self-contained software programs, or *patches*, which can be interpreted by anyone with access to the required equipment. These patches serve as both production interfaces and *de facto* notation, as knowledge of the programming environment enables one to "read" them like a score. Since most real-time computer music environments lack a strong musical time authoring component, sequencing is accomplished through tools such as the *qlist* object for Max and Pd (Winkler, 2001), the Bach notation library for Max (Agostini and Ghisi, 2012), and/or an electronics performer score such as the one for *Anthemes II* pictured in Figure 2.

[2] `http://brahms.ircam.fr/sidney/`
[3] `http://msp.ucsd.edu/pdrp/latest/files/doc/`

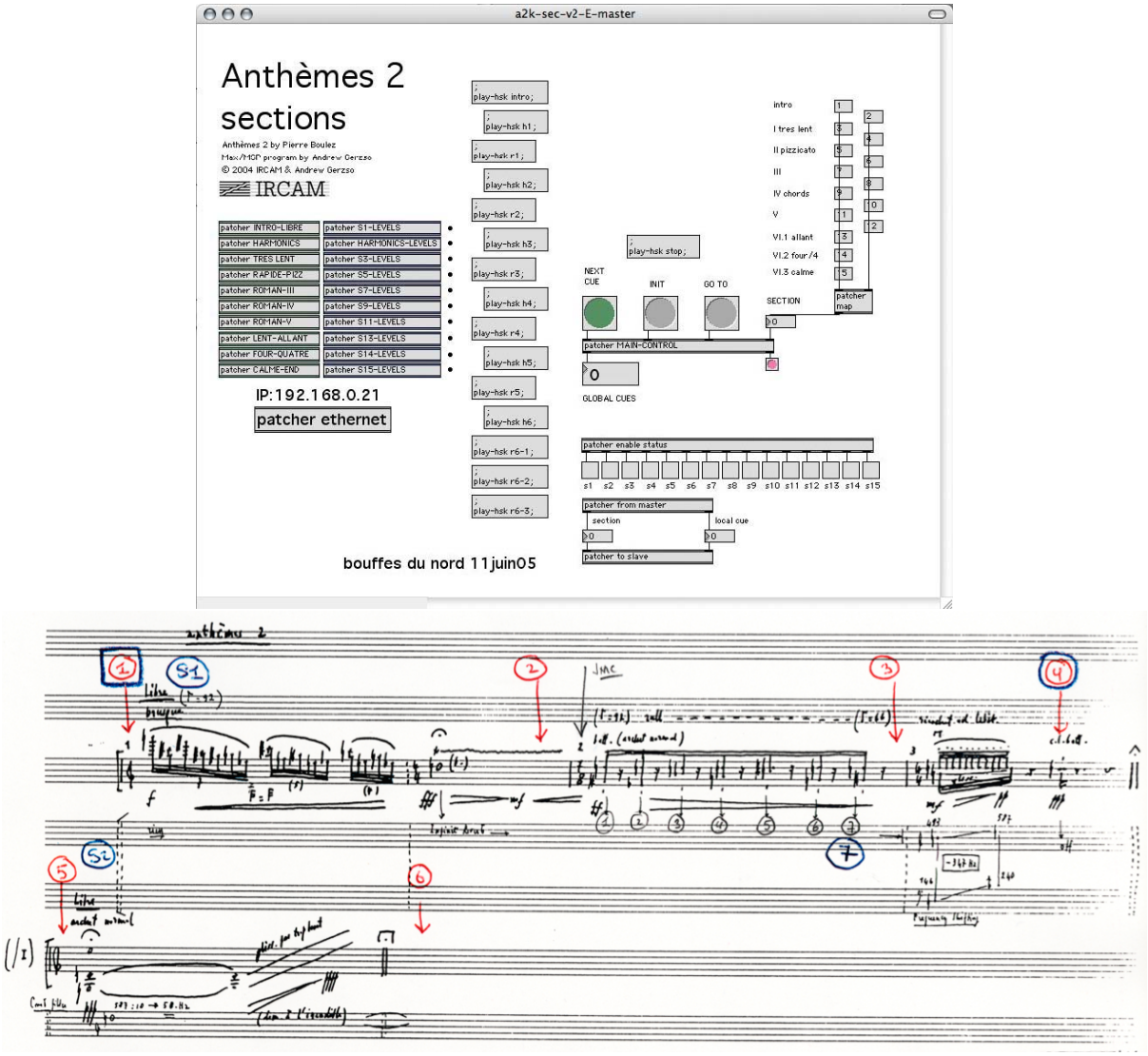



**Figure 2:** *Top: A portion of the* Anthemes II *Max/MSP patch from 2005. All electronic processes are triggered from a list of cues.*
*Bottom: Excerpt from the first page of the* Anthemes II *computer music performer score. The marked cues correspond to interaction points with the dedicated patch.*

### 1.1.3 Dynamic notation

Finally, a third category is represented by *dynamic scores* – these are Manoury's virtual scores (Manoury, 1990), also known in the literature as interactive scores due to their connection to real-time performance conditions (D. Fober, 2012). Interpretations range from prescribed musical actions to "guided improvisations", where the musicians are free to give the notation a personal reading (Clay and Freeman, 2010), or the electronic algorithms have a degree of nondeterminism. Here, composers are responsible for creating a dynamic roadmap whose acoustic realisation could be radically[4] different from one

[4](in terms of structure, duration, timbre etc)

performance to another.

A question arises: how are such dynamic interactions to be notated? While regular patches can already reach high levels of complexity, the problem of descriptiveness increases exponentially once time and decision-making are treated dynamically. The low-level approach is typified by the Pd software, which was designed to enable access to custom, non-prescriptive data structures for simple visual representation (Puckette, 2002). One step higher is an environment like INScore, which provides musically characteristic building blocks while retaining a high level of flexibility through its modular structure and OSC-based API (D. Fober, 2012). More structured solutions are provided by OSC[5] sequencers with some dynamic attributes such as IanniX (Coduys and Ferry, 2004) and i-score (Allombert et al., 2008). In particular, i-score uses a Hierarchical Time Stream Petri Nets (HTSPN)-based specification model (Desainte-Catherine and Allombert, 2005), enabling the visualisation of temporal relations and custom interaction points. The dynamic dimension is however fairly limited: an effort to enhance the model with conditional branching concluded that not all durations can be preserved in scores with conditionals or concurrent instances (Toro-Bermúdez et al., 2010). By replacing the Petri Nets model with a synchronous reactive interpreter, (Arias et al., 2014) achieved a more general dynamic behaviour, accompanied by a real-time i-score-like display of a dynamic score's performance. Still, this performance-oriented visualisation does not display the potential ramifications *before* the actual execution. This is generally the case with reactive, animated notation[6]: it does a good job of representing the current musical state, but does not offer a wider, out-of-time perspective of the piece. One significant development to the i-score framework enables conditional branching through a node-based formalism (Celerier et al., 2015). Meanwhile, more complex structures (loops, recursion etc.) still remain out of reach.

Naturally, much contemporary music makes use of all three types of notation outlined above. Composers often mix notational strategies in an effort to reach a two-fold goal: a (fixed) blueprint of the piece, and a (dynamic) representation of the music's indeterminacies. On the one hand, a score should lend itself to analysis and archival; on the other, notation is a tool for composition and rehearsal, which in modern mixed music require a high degree of flexibility. But perhaps most importantly, the score serves as a guide to the musical performance. As such, the nature of the notation has a strong bearing on the relationship of the musician with the material, and on the sonic end result.

## 1.2 Dynamic mixed music composition in *Antescofo*

In the realisation of *Anthemes II* shown in Figure 2, the temporal ordering of the audio processes is implicit in the stepwise evaluation of the score's data-flow graph, based on the human operator's manual triggering of cues. But the audio processes' activation, their control and most importantly their interaction with respect to the physical world

[5] OpenSoundControl, a multimedia communication protocol: `http://opensoundcontrol.org/`.

[6] in the literature, this kind of "live" notation is sometimes called dynamic notation (Clay and Freeman, 2010), regardless of the underlying scenario being computationally dynamic or not. In this paper, we use the term "dynamic" with regard to notation for the scoring *of* dynamic music in general. While the notation itself may be dynamic, this is not a necessary condition.

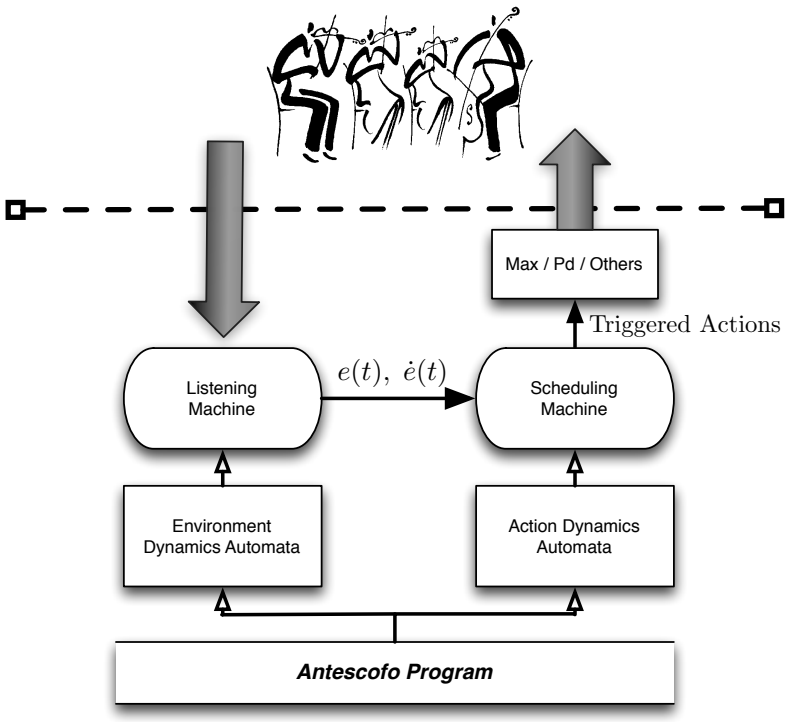



**Figure 3:** Antescofo *execution diagram.*

(the human violinist) are neither specified nor implemented.

The authoring of time and interaction of this type, and its handling and safety in real-time execution is the goal of *Antescofo* system and language (Cont, 2008, Echeveste et al., 2013a), which couples a listening machine (Cont, 2010) and a reactive engine (Echeveste et al., 2013b) in order to dynamically perform the computer music part of a mixed score in time with live musicians. This highly expressive system is built with time-safety in mind, supporting musical-specific cases such as musician error handling and multiple tempi (Cont et al., 2012, Echeveste et al., 2015). Actions can be triggered synchronously to an event $e(t)$ detected by the listening machine, or scheduled relative to the detected musician's tempo or estimated speed $\dot{e}(t)$. Finally, a real-time environment (Max/MSP, Pd or another OSC-enabled responsive program) receives the action commands and produces the desired output. The *Antescofo* runtime system's coordination of computing actions with real-time information obtained from physical events is outlined in Figure 3.

*Antescofo*'s timed reactive language (Cont, 2011) specifies both the expected events from the physical environment, such as polyphonic parts of human musicians (as a series of EVENT statements) and the computer processes that accompany them (as a series of ACTION statements). This paper touches on several aspects of the language; for a detailed specification please consult the *Antescofo* reference manual (Giavitto et al., 2015). The syntax is further described in (Cont, 2013), while a formal definition of the language is available in (Echeveste et al., 2015).

To facilitate the authoring and performance of *Antescofo* scores, a dynamic visualisation system was conceived, with a view to realising a consistent workflow for the

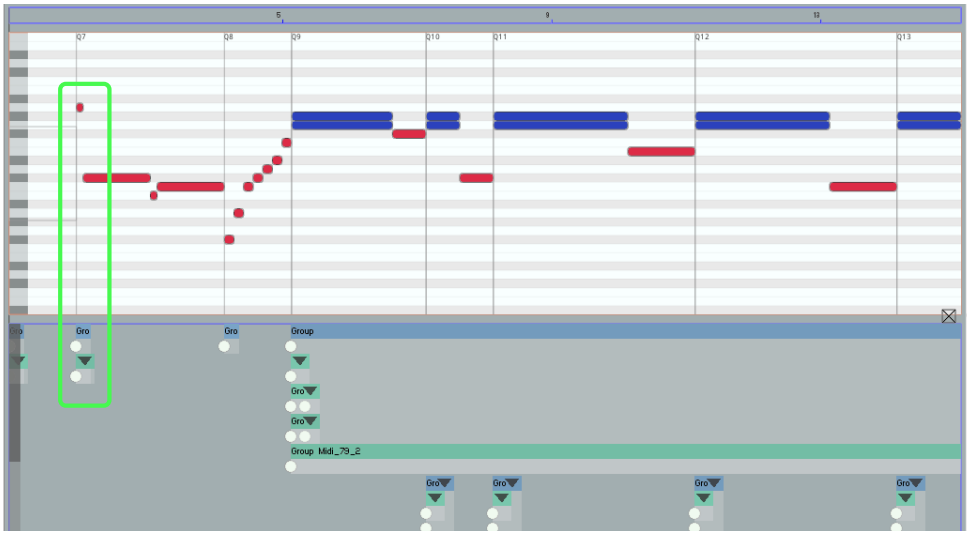


**Figure 4:** AscoGraph *visualisation for* Anthèmes II *(Section 1) from the* Antescofo Composer Tutorial. *Top: piano roll; Bottom: action view. The left-hand portion highlighted by a rectangle corresponds to Listing 1.*

*compositional* and *execution* phases of mixed music. *AscoGraph* (Coffy et al., 2014) is the dedicated user interface that aims to bridge the three notational paradigms described in section 1.1: *symbolic*, *integrated* and *dynamic*. We demonstrate the first two aspects with a brief study of *AscoGraph*'s basic notation model (Burloiu and Cont, 2015) in section 1.3, before laying out different strategies to tackle the dynamic dimension.

## 1.3 The basic AscoGraph visualisation model

Since its inception, *AscoGraph*'s visual model has been centred around the *actions view* window, which is aligned to the instrumental *piano roll* by means of a common timeline. Electronic actions are either discrete (visualised by circles) or continuous (curves) but they are all strongly timed[7] (Cont, 2010). Figure 4 displays the implementation of *Anthèmes II* (Section 1) from the *Antescofo Composer Tutorial*[8]. This layout facilitates the authoring process by lining up all the elements according to *musical time*, which is independent of the *physical (clock) time* of performance. Thus, in cases where the score specifies a delay in terms of seconds, this is automatically converted to bars and beats (according to the local scored tempo) for visualisation.

Generally, atomic actions in the *Antescofo* language have the following syntax: [<delay>] <receiver_name> <value>. An absent <delay> element is equivalent to zero delay, and the action is assumed to share the same logical instant as the preceding line.

[7]Wang (Wang, 2008) defines a *strongly timed* language as one in which there is a well-defined separation of synchronous logical time from real-time. Similarly to Wang's *ChucK* language, *Antescofo* also explicitly incorporates time as a fundamental element, allowing for the precise specification of synchronisation and anticipation of events and actions.

[8]available at `http://forumnet.ircam.fr/products/antescofo/`.

```
NOTE 0 1.0          ; silence
NOTE 8100 0.1              Q7
        Annotation "Harms open"
        hr-out-db -12.0 25          ; bring level up to -12db in 25ms
        group harms      {          ; four harm. pitches:
                hr1-p (@pow(2., $h1/12.0))
                hr2-p (@pow(2., $h2/12.0))
                hr3-p (@pow(2., $h3/12.0))
                hr4-p (@pow(2., $h4/12.0))
        }
```

**Code 1:** *Opening notes and actions in the* Anthèmes II *augmented score.*

Code listing 1 shows the starting note and action instructions of the score. After one beat of silence, the first violin note triggers the opening of the harmonizer process, by way of a nested group of commands. The resulting hierarchy is reflected in the green-highlighted section of Figure 4's action view: the first action group block on the left contains a white circle (representing two simultaneous messages) and a sub-group block, which in turn includes a circle (containing four messages to the harmonizer units). Note the absence of any time delay: all *atomic actions* mentioned above are launched in the same logical instant as the note detection.

This visualisation model meets the first notational requirement we specified at the end of section 1.1: it acts as a graphic map of the piece, which reveals itself through interaction (e.g. hovering the mouse over message circles lists their contents).

The visual framework presented in this paper is the outcome of a continuous workshopping cycle involving the mixed music composer community in and around IRCAM. While some features are active in the current public version of *AscoGraph*[9], others are in the design, development or testing phases. This is apparent in the proof of concept images used throughout, which are a blend of the publicly available *AscoGraph* visualisation and mockups of the features under construction. We can expect the final product to contain minimal differences from this paper's presentation.

In the remainder of this paper, we lay out the major dynamic features of the *Antescofo* language (section 2) and the visual models for their representation (timeline-based in section 3 and out-of-time in section 4), in our effort towards a comprehensive dynamic notation that supports both authorship and execution of augmented scores. Specifically, the four models are the extended action view (section 3.1), the simulation trace staff view (3.2), the hierarchical tree display (4.1) and the inspector panel (4.2). We test our hypotheses on real use-case scenarios (section 5) and conclude the paper with a final discussion (section 6).

[9]at the time of writing, the newest release of *Antescofo* is v0.9, which includes *AscoGraph* v0.2.

# 2 Dynamic elements in the *Antescofo* language

Dynamic behaviour in computer music can be the result of both interactions during live *performance*, and algorithmic and dynamic compositional elements prescribed in the score itself. Accordingly, in an *Antescofo* program, dynamic behaviour is both produced by real-time (temporal) flexibility as a result of performing with a score follower, and through explicit reactive constructs of the action language.

In the former case, even though the temporal elements can be all statically defined in the score, they become dynamic during live performance due to the tempo fluctuations estimated by the listening machine. We alluded to this basic dynamic aspect in section 1.3, where we noted the implicit interpretation of physical time as musical time.

The second case employs the expressive capabilities of the strongly timed action language of *Antescofo*. Such explicitly dynamic constructs form the topic of this section.

## 2.1 Run-time values

Variables in the *Antescofo* language can be run-time, meaning that their values are only determined during live performance (or a simulation thereof). The evaluation of a run-time variable can quantify anything as decided by the composer, from a discrete atomic action to breakpoints in a continuous curve, as shown in code listing 2.

In this basic sample, the value of the level output can be the result of an expression defined somewhere else in the code, whose members depend on the real-time environment. In the example on the right, the output level is defined by $y, which grows linearly over 2 beats from zero to the run-time computed value of $x.

```
NOTE C4 1
level $x
```

```
NOTE D4 1
curve level {
  $y {
       ( 0 )
    2 ( $x )
  }
}
```

**Code 2:** *Dynamic amounts.*
*Left: atomic value. Right: curve breakpoint.*

Thus, while for now the circle-shaped action message display from section 1.3 is adequate for the dynamic atomic action, for the curve display we need to introduce a visual device that explicitly shows the target level as being dynamic. Our solution is described in section 3.1. Additionally we propose an alternative treatment of both atomic actions and curves, in the context of performance traces, in section 3.2.

## 2.2 Durations

On *AscoGraph*'s linear timeline, we distinguish two kinds of dynamic durations. Firstly the *delay* between two timed items, such as an EVENT and its corresponding ACTION,

or between different breakpoints in a curve; and secondly, the number of *iterations* of a certain timed block of instructions.

The examples in code listing 3 show the two kinds of temporal dynamics: On the left, the run-time value of $x determines the delay interval (in beats, starting from the detected onset of NOTE C4) until level receives the value 0.7, and the duration of the (continuous) increase from 0 to 0.5. On the right, the duration of execution of the loop and forall structures depends, respectively, on the state of $x and the number of elements in $tab. In these cases the terminal conditions of loop and forall are reevaluated on-demand.

```
NOTE C4 1
$x level 0.7
NOTE D4 1
curve level {
        $y {
                ( 0 )
                $x ( 0.5 )
        }
}
```

```
NOTE E4 1
loop L 1 {
$x:=$x+1
} until ($x > 3)

NOTE F4 1
forall $item in $tab {
$item level ($item * 2)
}
```

**Code 3:** *Dynamic durations.*
*Left: delay durations. Right: number of iterations.*

A particular extension of iterative constructs are *recursive processes*. A process is declared using the @proc_def command, and can contain calls to itself or other processes. Thus, the behaviour and activation interval (*lifespan*) of a process, once it has been called, can be highly dynamic. The example in code listing 4 produces the same result as the loop block in code listing 3. See (Giavitto et al., 2015) for in-depth specifications of all iterative constructs.

```
@proc_def ::L() {
  if ($x <= 3) {
    1 ::L() ; new proc call
  }
  $x:=$x+1
}
NOTE E4 1
::L() ; initial proc call
```

**Code 4:** *Dynamic recursive process.*

We introduce a graphic solution for representing dynamic durations in section 3.1. Going further, the action view model is not well suited for dynamically iteration-based repetition. We propose three alternatives: "unpacking" such constructs into an execution trace (section 3.2), detailing their structure in a tree-based graph (section 4.1), and monitoring their status in an out-of-time auxiliary panel (section 4.2).

## 2.3 Occurence

The examples we have shown thus far, while pushing the limits of *AscoGraph*'s action view, can still be represented along a linear compositional timeline. There is a third category which could not be drawn alongside them without causing a breakdown in temporal coherence, as shown in code listing 5.

```
whenever ($x) {
  level 0
} during [2#]
```

**Code 5:** *Dynamic occurence point.*

Here, the action block is fired whenever the variable $x is updated. Moreover, this is set to occur only twice in the whole performance, hence the during [2#]. From here, it is easy to imagine more complications – recursive process calls, dynamic stop conditions etc – leading to a highly unpredictable runtime realisation.

In most cases, since such occurence points are variable, nothing but the overall lifespan of the whenever (from entry point down to stop condition fulfilment) should be available for coherent representation; this might still be helpful for the composer, as we show in section 3.1.

Since whenever constructs are *out-of-time* dynamic processes[10], being detached from the standard timeline grid, they require new methods of representation beyond the classic action view. We discuss the "unpacking" of whenever blocks onto traces in section 3.2.3, and their non-timeline based representations in section 4.

## 2.4 Synchronisation strategies

In *Antescofo*, the tempo of each electronic action block can be dynamically computed relatively to the global tempo detected by the listening machine. Through attributes attached to an action group, its synchronisation can be defined as @loose, @tight, or tied to a specific event @target; the latter enabling timing designs such as real-time tempo canons (Trapani and Echeveste, 2014). Since @target-based relationships define temporal alignment between a group and event(s), we can visualise this by connecting the piano roll to the action tracks (see section 3.2.2), or by drawing the connections in an out-of-time model (see section 4.1).

Additionally, the *Antescofo* language allows for dynamic targets, acting as a moving synchronisation horizon. In this case, the tempo is aligned to the anticipation of the @target event at a specific distance in the future, computed either by a number of beats

[10]Our use of the term "out-of-time" is derived from the sense coined by Xenakis (Xenakis, 1992) to designate composed *structures* (and the methods used to generate them), as opposed to sonic form. In an analogous fashion, we distinguish "in-time" electronic actions that are linked to specific acoustic events from "out-of-time" constructs, which are not. Just like Xenakis' structures, during the performance, the out-of-time constructs are actuated (given sonic form) in time. The difference from Xenakis' approach is that *Antescofo* out-of-time actions do not reside on a separate plane of abstraction from in-time actions: only the nature of their activation is different.

or a number of events. We can indicate this synchronisation lookahead in relation to the timeline; see section 5.2 for an example.

## 2.5 Score jumps

The instrumental events in an *Antescofo* score are inherently a sequence of linear reference points, but they can be further extended to accommodate jumps using the @jump attribute on an event. Jumps were initially introduced to allow simple patterns in western classical music such as free repetitions, or *da capo* repetitive patterns. However, they were soon extended to accommodate composers willing to create open form scores (Freeman, 2010).

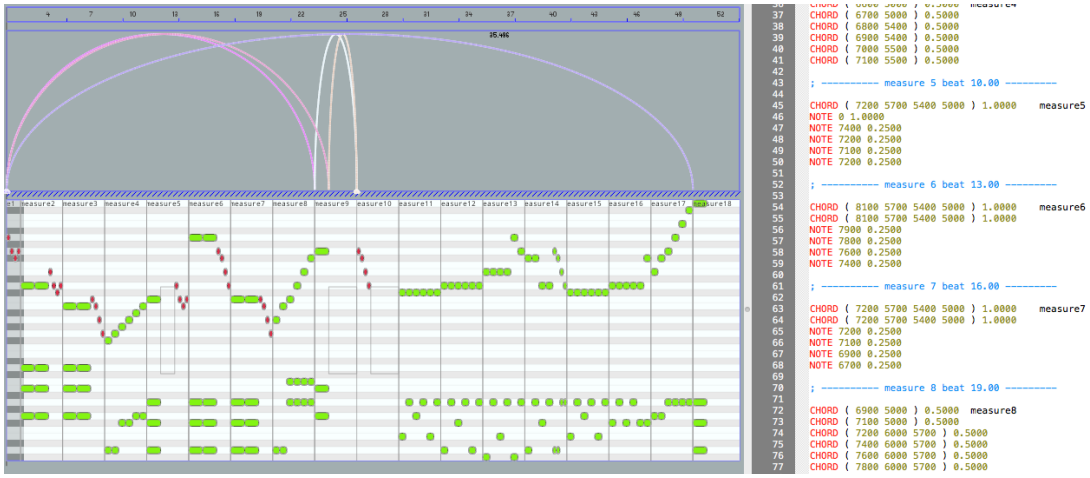



**Figure 5:** Ascograph *with static jumps: A classical music score (Haydn's* Military Minuet*) with* Antescofo *jumps simulating* da capo *repetitions during live performance.*

For the purposes of visualisation, we distinguish two types of open form scores: in the first, jump points are fixed in the score (static), while their activation is left to the discretion of the performer. This scheme is more or less like the *da capo* example in Figure 5. Its success in live performance depends highly on the performance of the score follower. Such scoring has been featured in concert situations such as pieces by composer Philippe Manoury realised using *Antescofo* (Manoury, 2016). Figure 5 shows the treatment of static jumps in the action view, which would be similarly handled in the staff view (section 3.2).

The second type is where the score elements and their connections are dynamically generated, such as in the work of composer Jason Freeman (Freeman, 2010). In this case, similarly to the dynamic @targets from section 2.4, we are dealing with an attribute, this time that of an event. For now, we choose to print out the algorithm for jump connection creation in a mouse-over popup, since its runtime evaluation can be impossible to predict.

# 3 Timeline-based models

In the following, we put forward visualisation solutions for the dynamic constructs from section 2, anchored to the linear timeline. Since so much compositional activity relates to timelines, be they on paper or in software, it makes sense to push the boundaries of this paradigm in the context of mixed music.

## 3.1 Representing dynamics in the action view

The main challenge in displaying dynamic delay segments alongside static ones is maintaining horizontal coherence. Dynamic sections must be clearly delimited and their consequences shown. To this end we introduce *relative timelines*: once an action is behind a dynamic delay, it no longer synchronizes with actions on the main timeline; rather, the action lives on a new timeframe, which originates at the end of the dynamic delay.

To avoid clutter, a relative time ruler appears only upon focus on a dynamically delayed section. Also, we add a shaded time-offset to depict the delay, as seen in Figure 6. Since by definition their actual duration is unpredictable, all such shaded regions will have the same default width.

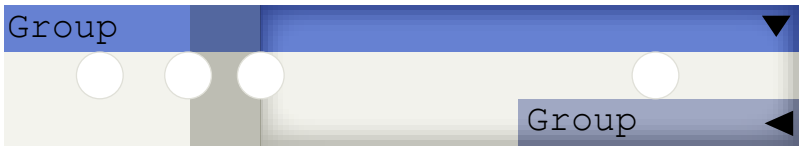



**Figure 6:** *A group with a dynamic delay between its second and third atomic actions. The subsequent action and subgroup belong to a relative timeline, whose ruler is hidden.*

These concepts apply to the display of curves as well. As discussed in section 2.1, dynamic breakpoint heights now come into play. Our solution is to randomly generate the vertical coordinate of such points, and to mark their position with a vertical shaded bar, as in Figure 7.

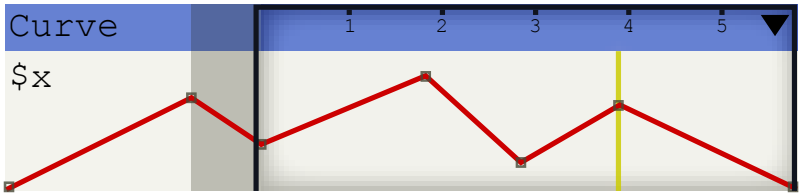



**Figure 7:** *A curve with a dynamic delay between its second and third breakpoints. The 6th breakpoint has a dynamic value. The relative timeline ruler is drawn.*

In our investigations with the core user group at IRCAM, a need was identified for the possibility of "local execution" of a dynamic section, to be able to compare a potential realisation of the dynamics with their neighbouring actions and events. To this end, we are developing a *simulate* function for any dynamic block, which transforms it into a classic static group, eliminating its relative timeline. The process makes a best-possible

guess for each particular situation, in the context of an ideal execution of the score[11], and can be undone or regenerated. See Figure 8 for an example of such a local simulation result. The underlying mechanisms are part of the *Antescofo* offline engine, similarly to the full simulation model in section 3.2.

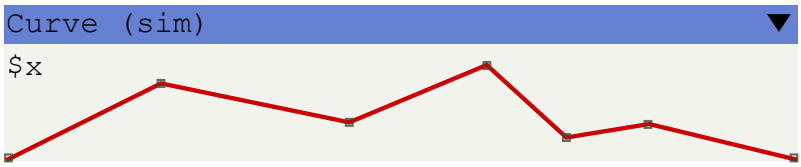



**Figure 8:** *A best-guess simulation of the previously shown curve. The dynamic delay and value have both been actuated.*

For the constructs involving a number of iterations over a set of actions, we propose a specific striped shading of the block background, as well as a model for depicting the group's lifespan along the timeline. We chose vertical background stripes for loops and horizontal ones for forall s, according to their sequential or simultaneous nature in standard usage, respectively[12]. For the activation intervals of the constructs, we distinguish three situations with their respective models, depicted in Figure 9: (1) a *definite* lifespan, when the duration is statically known, (2) a *dynamic, finite* lifespan for dynamically determined endpoints, and the (3) *dynamic, infinite* lifespan for activities that carry on indefinitely. These graphic elements are all demonstrated in the examples in section 5, Figures 18a and 19a.

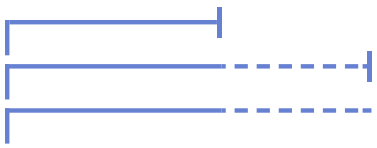


**Figure 9:** *Definite lifespan (top). Dynamic finite lifespan (mid). Dynamic infinite lifespan (bottom).*

## 3.2 Tracing performance simulations

From its conception, *AscoGraph* has included an experimental *simulation mode* that "prints" the whole piece to a virtual execution trace (Coffy et al., 2014). Much like a traditional score, electronic action *staves* would mark the firing of messages or evolution of continuous value streams along a common timeline. We now present a perfected simulation model, to be implemented into the next version of *AscoGraph*, that more robustly handles the dynamic aspects of the score language and also supports the recently developed *Antescofo* test framework (Poncelet and Jacquemard, 2015).

[11] Details on how such ad'hoc trace generation and execution is accomplished can be found in (Poncelet and Jacquemard, 2015).

[12] Of course, a loop can be made simultaneous through a zero repeat period, while a forall can function sequentially by way of branch-dependant delays.

The general aim is to produce the equivalent to a manually notated score, to be used as a reference for performance and analysis, as well as a tool for finding bugs and making decisions during the composition phase.

Our main design inspiration is a common type of notation of electroacoustic music (Xenakis, 1992), as exemplified in Figure 10. The standard acoustic score is complemented by electronic action staves, along which the development of computerised processes is traced.

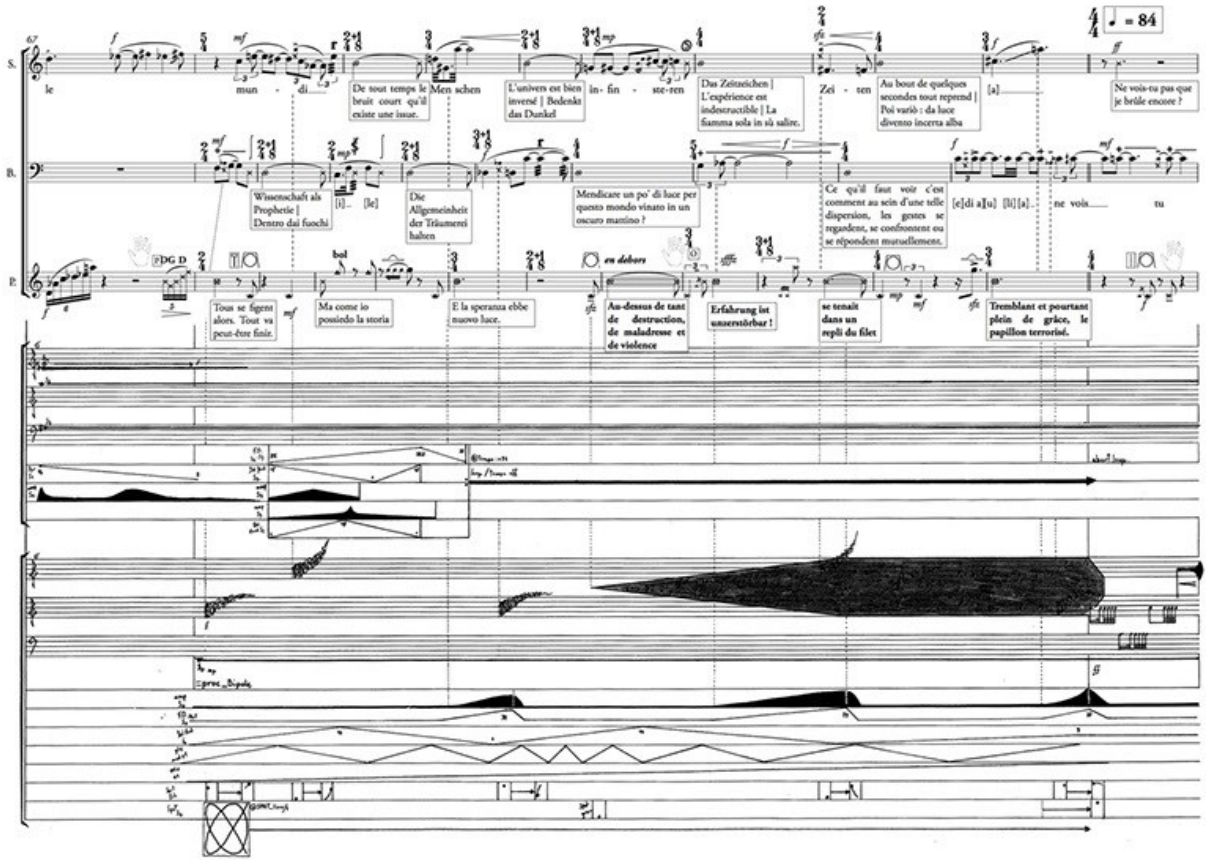



**Figure 10:** *Electroacoustic staff notation: Nachleben (excerpt) by Julia Blondeau.*

The new display model accommodates all concepts introduced so far: atomic values, curves, action groups and their lifespans. Dynamic values and durations are still indicated specifically; this time we use dotted lines, as the examples in section 3.2.1 show. Horizontally, distances still correspond to musical time, but, as was the case of the shaded areas in section 3.1, the dotted lines representing dynamic durations produce disruptions from the main timeline.

Electronic action staves can be collapsed to a "closed" state to save space, where all vertical information is hidden and all components are reduced to their lifespans.

### 3.2.1 Defining staves

Unlike the action view (see sections 1.3 and 3.1), in the simulation mode the focus is on reflecting the score's output, not its code structure. While the action view is agnostic

with regard to the content of the coded actions, the simulation mode is closely linked to electronic action semantics. Thus, it is likely that commands from disparate areas in the code will belong on the same horizontal staff.

In this simulation trace model, staff distribution is closely linked to the *Antescofo tracks* that are defined in the score, using the @track_def command.

```
@track_def track::T {
  "level*"
}
```

**Code 6:** *Track definition.*

In the example in code listing 6, the track T contains all score groups or actions whose label or target start with the prefix level, and their children recursively. The model will attempt to print all the corresponding actions to a single staff. Should this prove impossible without creating overlaps, the following actions will be taken, in order, until a "clean" layout is obtained:

1. collapse overlapping action groups to their lifespan segments. These can then be expanded, creating a new sub-staff underneath or above the main one.
2. order the open action groups and curves by relative timeline (see section 3.2.2), and move them to sub-staves as needed.
3. order the open action groups and curves by lifespan length, and move them to sub-staves as needed.

If track definitions are missing from the score, the staff configuration will simply mirror the action group hierarchy. Figure 11 shows a possible reflection of the group from Figure 6, whose corresponding score is Group g1 from code listing 7 below. The height of a point on a staff is proportional to the atomic action value, according to its receiver[13]. It is possible to have several receivers on a single staff, each with its own height scale (as in section 5.1), or with common scaling (as in the present section).


**Figure 11:** *Staff representation of a group containing a dynamic delay and a subgroup.*

Since the same item can belong to several tracks, this will be reflected in its staff representation. By default, a *primary* staff for the item will be selected, and on the remaining staves the item will only be represented by its lifespan. The user can then expand/collapse any of the instances. The primary staff is selected by the following criteria:

[13]Recall the general atomic action code syntax: [<delay>] <receiver_name> <value>. A receiver might get no numeric value, or a list of values. We use the first value if any, or else a default height of 1.

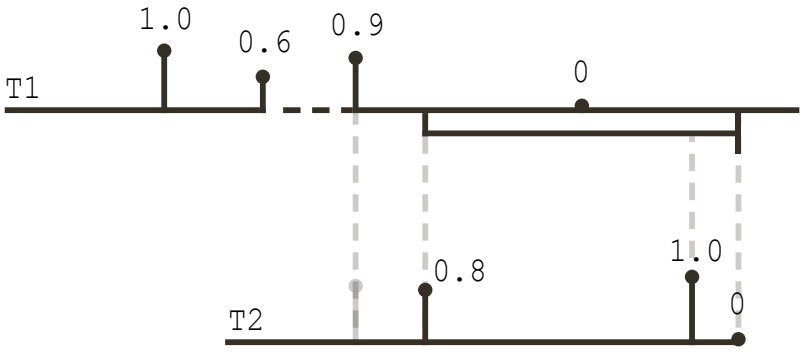



**Figure 12:** *The elements of T2 are also part of T1 (in collapsed form).*

1. least amount of overlapping items
2. smallest distance to staff's track definition: an identical label match will be a closer match than a partial match, which is closer than a parent-child group relationship.

In Figure 12 we show the same group from Figures 6 and 11, now with two tracks defined: T1 for the main group and T2 for the subgroup. The subgroup and its contents also fall under the track T1 definition, which is why the subgroup lifespan is represented on the T1 staff.

Note that, while the subgroup's timeline starts simultaneously with the m3 0.9 atomic action, its first triggering is m21 0.8, which comes after a .4 beat delay. Since, as we have explained, the simulation view focuses on reflecting the *execution* of the score, the subgroup lifespan on the T1 track is represented as starting with the m21 0.8 event.

Similarly to the action view (section 2.3), whenever blocks are represented by their lifespans. However, here the user can expand the contents of the whenever block on a new staff, marked as being out-of-time – much like populating the auxiliary inspector panel, as we describe in section 4.2. We present a practical approach to visualising whenever constructs as part of the example in section 5.1.

An alternative to the automatic layout generation function is adding tracks one by one using context menu commands. Naturally, users will be able to employ a mix of both strategies, adding and removing score elements or entire tracks or staves to create a desired layout.

While some layouts produced with this model might prove satisfactory for direct reproduction as a final score, we will also provide a vector graphics export function, where a composer can subsequently edit the notation in an external program. Finally, the layout can be saved in XML format and included alongside the *Antescofo* score file of the piece.

### 3.2.2 Representing sync points

We showed in Figure 12 how the start and end of a subgroup's relative timeline (coinciding with the actions m3, valued at 0.9, and m23, valued at 0) are marked by vertical shaded dotted lines. Similarly we signify a return to the main timeline, by synchronising to a certain event, as in Figure 13, where curve A is no longer part of the relative timeline

before it; it synchronises to an event, depicted by the red rectangle on the piano roll. The corresponding *Antescofo* score is presented in code listing 7.

```
; [PREVIOUS EVENTS]
NOTE C4 0.8 e2 ; (length: 0.8 beats. label: e2)
Group g1 {
        0.5 m1 1.0
        0.5 m2 0.6
        $x m3 0.9
        Group g2 {
                0.4 m21 0.8
                1.2 m22 1.0
                0.3 m23 0
        }
        0.8 m4 0
}
NOTE 0 2.7 e3    ; silence
NOTE D4 0.8 e4
Curve A @Action := print $x {
        $x {      {1.0}
                0.4 {0.3}
                0.2 {0.7}
                0.2 {0.0}
        }
}
NOTE E4 1.0 e5
m5 0.9 @local
```

**Code 7:** *Score for Figures 13, 14.*

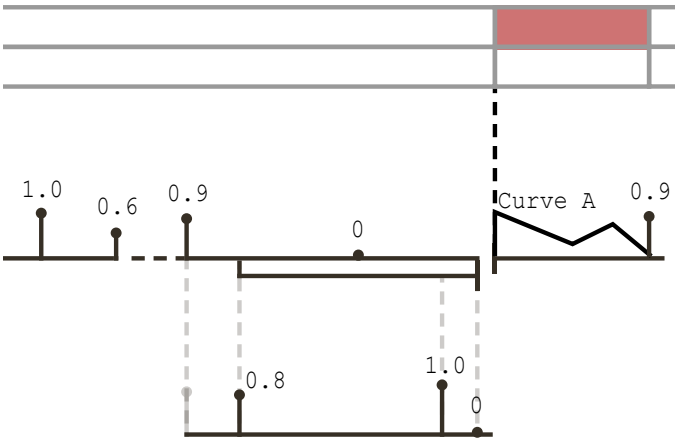



**Figure 13:** *Synchronising to an event: the piano roll is focused on the D4 note which triggers* Curve A.

We take a similar approach for dynamic synchronisation targets, as exemplified by the case study in section 5.2. Again the sync relationship will be represented by a dotted line, this time parallel to the timeline.

### 3.2.3 Visualising test traces

Beyond the *ideal* trace produced by executing a score with all events and actions occurring as scripted in the score, the simulation view extends to support the Model-Based Testing workflow (Poncelet and Jacquemard, 2015), which builds a test case by receiving a timed input trace, executing the piece accordingly and outputting a verdict. Such an input trace describes the behaviour of the listening machine, by way of the deviations of the detected musician activity from the ideal score. For each deviation, *Antescofo* computes a new local tempo, based on the last detected note duration.

We propose an example partial test scenario in Table 1[14], again corresponding to code listing 7. Since the last line of code is an atomic action @local-ly synced to NOTE E4, and

[14]This table and the corresponding trace listings are an example of the automatically generated output of the Test framework.

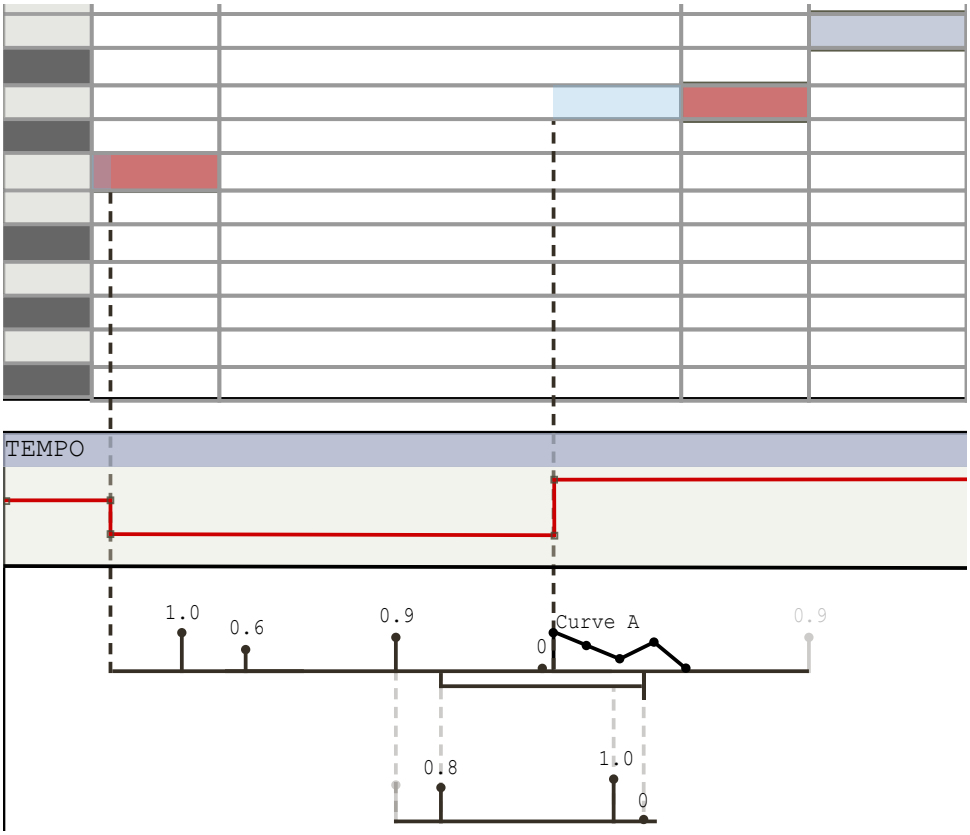



**Figure 14:** *Partial test scenario: the event* e2 *(C note) is .125 beats late, the event* e4 *(D) is .75 beats early, the event* e5 *(E) is missed. The curve is quantised into 5 actions:* c1–c5.

in our example input trace (see code listing 8) the event is missed, then the action will remain untriggered.

Timed input traces, as lists of event symbols and their corresponding timestamp and local tempo, can be loaded from text files and visualised on the piano roll, as in Figure 14. The time distance between the ideal event and its detected trace is highlighted, and missed events are greyed out. The user is able to edit the input trace on the piano roll and save the modifications into a new text file. The computed *tempo curve* $\tau$ connects all the local tempo values and spans the duration of the piece; it is displayed along the timeline.

Output traces are produced by computing physical time from musical time. For instance, the timestamp of event e2 from code listing 8 is the result of multiplying its input beat position with the corresponding relative tempo: $t(\mathsf{e2}) = 1.125 \times (60/102)$.

Once the input trace has been processed, the simulation view updates accordingly. The offline engine does a best-effort attempt to produce a veridical realisation of the electronic score. For instance, any whenever blocks are fired just as they would be during a real performance, by monitoring their activation condition. This allows their contents to be displayed on the appropriate staves alongside the regularly timed actions – which would be impossible without a known input trace.

| | Musician | | *Antescofo* estimation | | Event durations | | |
|---|---|---|---|---|---|---|---|
| cue | timestamp | tempo | timestamp [s] | tempo | real | *A.* estimate | Relative duration |
| e1 | 0.0s | 90.7 | 0.0 | 102 | 0.661s | 0.588s | 1.125beats [long] |
| e2 | 0.661s | 115.4 | 0.58 zzz 0.66 | 90.7 | 1.819s | 2.315s | 2.75beats [short] |
| e4 | 2.481s | N/A | !2.481 | 115.4 | irrelevant | | |
| e5 | [missed] | N/A | | | | | |

**Table 1:** *Partial test scenario. In* Antescofo*'s listening estimation, "zzz" denotes the wait for a late event detection, and "!" is a surprise detection of an early event. The real tempo and duration of event e4 are irrelevant, since the following event is missed.*

```
IN: <e1, 0.0, 102> <e2, 1.125, 90.7> <e4, 3.875, 115.4>
OUT: <e1, 0.00> <e2, 0.661> <m1, 0.992> <m2, 1.323> <m3, 1.653>
<m21, 1.918> <m4, 2.183> <e4, 2.481, 60> <c1, 2.481>[1.0]
<c2, 2.585>[0.65] <m22, 2.662> <c3, 2.689>[0.3]
<c4, 2.793>[0.7] <m23, 2.818> <c5, 2.897>[0.0]
```

**Code 8:** *Test case: example input and output traces. We assume an initial event* e1*, ideally 1 beat long, with an initial tempo of 102bpm. The input trace format is* <label, timestamp(in beats), tempo>*. The output trace format is* <label, timestamp(in seconds)>[value]*. The* curve *triggerings* [c1 ... c5] *are determined by keyframe timings and lookup grain.*

Effectively, a visualisation of the test's output trace is produced, with any missed actions being greyed out. Any staff layout previously configured by the user is preserved. The corresponding test verdict can be saved as a text file.

# 4 Models complementary to the timeline

We have shown so far how the introduction of dynamic processes makes linear timeline-based models partially or wholly inadequate for coherent representation. Along with addressing this issue, alternative models can provide the added benefit of improving focus and offering a fresh perspective on the score.

In this section, we propose two solutions: a tree-based display of the score's hierarchical structure and internal relationships, and an auxiliary panel that focuses on specific, possibly recurring actions or groups. We note that these two models are currently under development as part of the roadmap towards the next major version of *AscoGraph*. The final implementations may vary slightly from the specifications presented hereon.

## 4.1 The Hierarchy View

There are significant precedents of graphic tree representations for grammars in the computer music literature, such as for Curtis Roads' *TREE* specification language (Roads, 1977). In a similar way, we can interpret the *Antescofo* language as a Type 2 context-free grammar, and construct the hierarchical tree of a score as follows. The primary nodes are the instrumental EVENTs. Their siblings are the neighbouring events, vertically aligned. Should a jump point be scripted, then one event node can have several downstream siblings.

ACTIONs are secondary nodes, connected to their respective event nodes in a parent-child relationship. The branch structure of the action nodes mirrors the groupings in the score. We have designed a set of glyphs for all *Antescofo* score elements; see Figure 15.

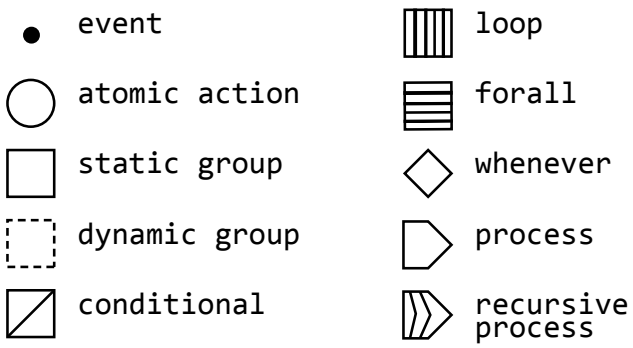



**Figure 15:** *Glyphs used in the hierarchy tree model.*

Aside from the parent-child and sibling relationships defined so far, we also provide ways to indicate internal relationships. These include:

- common variables or macros (colour highlighting);
- common process calls (colour highlighting);
- synchronisation targets (dotted arrow).

The user can selectively activate them permanently, or they can appear upon mouse hover. Figure 16 shows an example of all three types of relationships between nodes.

To avoid cluttering the tree display, we have decided not to show lifespans in this model. However, whenever and @proc_def nodes are persistently displayed at the top of the frame, next to the score tree, for as long as the current zoom view intersects with their lifespan. A click on a whenever node expands its contents in place, and clicking on a @proc_def node expands its currently visible instances within the tree.

## 4.2 The Inspector Panel

In this auxiliary visualisation mode, the user can observe the contents and/or monitor the state of groups, actions, or variables, which shall be selected from the other views (text editor, action view, hierarchy view). Once inside the inspector, the item state

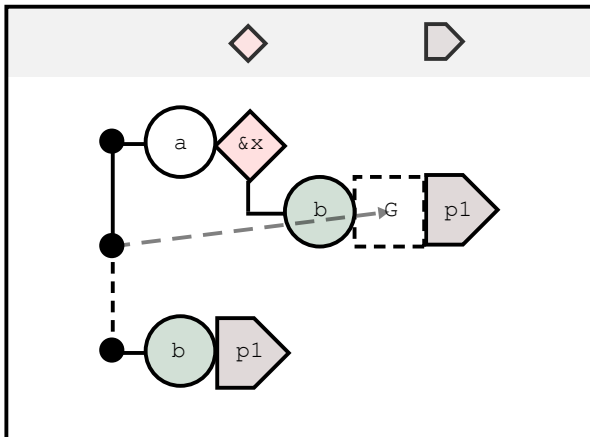



**Figure 16:** *Example of a hierarchy tree. Group G synchronises to the second event.*

will synchronise, via a local simulation estimate, with the current position in the score from the other views. This behaviour is consistent with the visualisation principle of explicit linking (Roberts, 2007), which is maintained in *AscoGraph* along the diagram in Figure 17.

The inspector displays a combination between the timeline-based designs from section 3. For action groups, we retain the block display (e.g. for showing whenever groups outside the timeline) and we use horizontal staves to visualise the values in variables and action receivers, along with their recent histories. The added value is two-fold: block display of out-of-time constructs, and persistent monitoring of values (even when they have lost focus in the other views).

The hierarchy view and the inspector panel are both depicted in a working situation in the following section; see Figure 18.

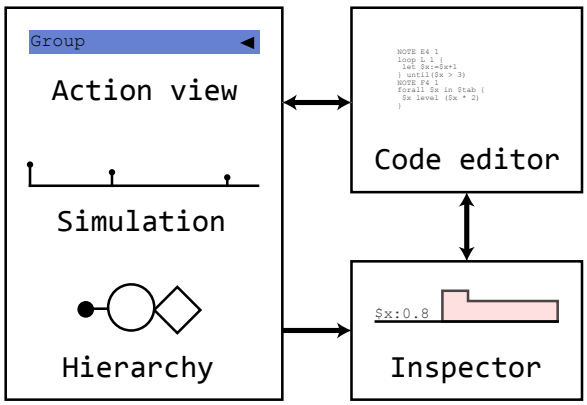



**Figure 17:** *Explicit linking in* AscoGraph*: the three main views are linked to the code editor, which is linked to the inspector. The main views also allow the user to select items to highlight in the inspector.*

# 5 Case studies

We present two use-case scenarios highlighting specific dynamic language constructs used in example *Antescofo* pieces. In each example, the instrumental score is minimal (a single event) and the focus is on the electronic actions produced by the machine in response to the instrumental event, and their visualisation using the four models we have described.

## 5.1 Reiteration and dynamic occurrence: loop, whenever

The first example is based on the basic rhythm and soundscape tutorials included in the *Antescofo* software package. Code listing 9 shows significant portions of an example algorithmic composition score[15] where a group contains two curve and two whenever blocks, whose states control the behaviour of the loop block. To all intents and purposes, the instrumental score is empty, except for "dummy" events for the start and end of the piece: everything happens on the electronic side where *Antescofo* acts as a sequencer. For an in-depth analysis of the score's operation, we refer the reader to the tutorial documentation; presently we shall concentrate on the visualisation solutions, as displayed in Figures 18a, b, c, d.

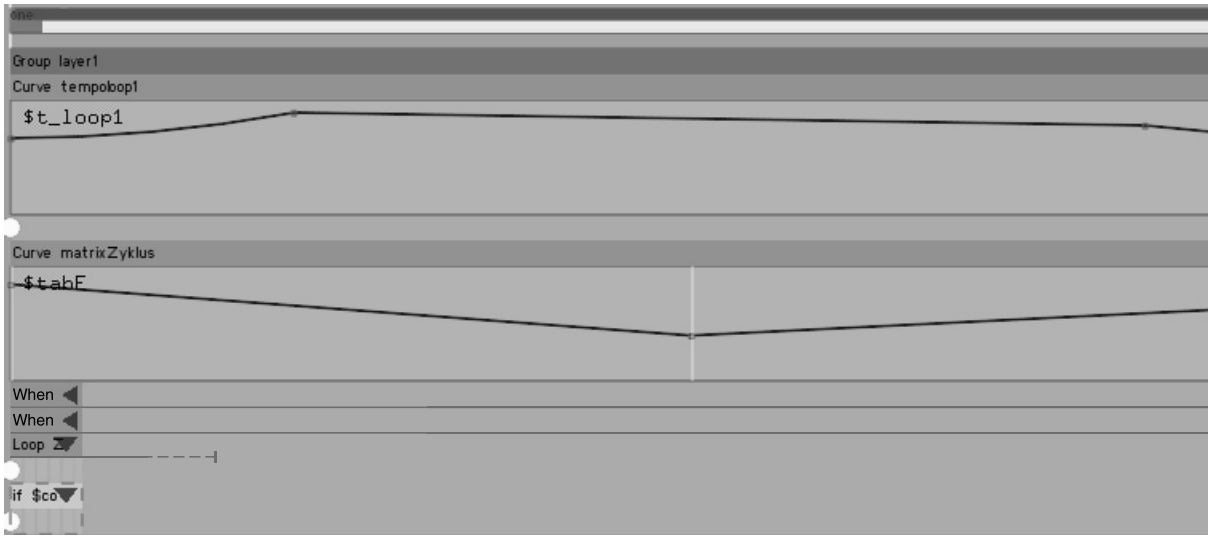
The action view follows the code structure, and includes the lifespans of the whenever and loop blocks, as introduced in section 3.1. Note how these lifespans are all constrained by the duration of the parent group.

The triggering frequency of loop construct is dictated by the evolution of the tempo (curve tempoloop1) and the beat division rate ($cycleloop). Their interdependence is reflected in the simulation view staff display. In the loop, receiver names are drawn by string concatenation via the @command instruction. The layout has been configured to draw a staff for each of the three receivers, containing pan and amplitude pairs.

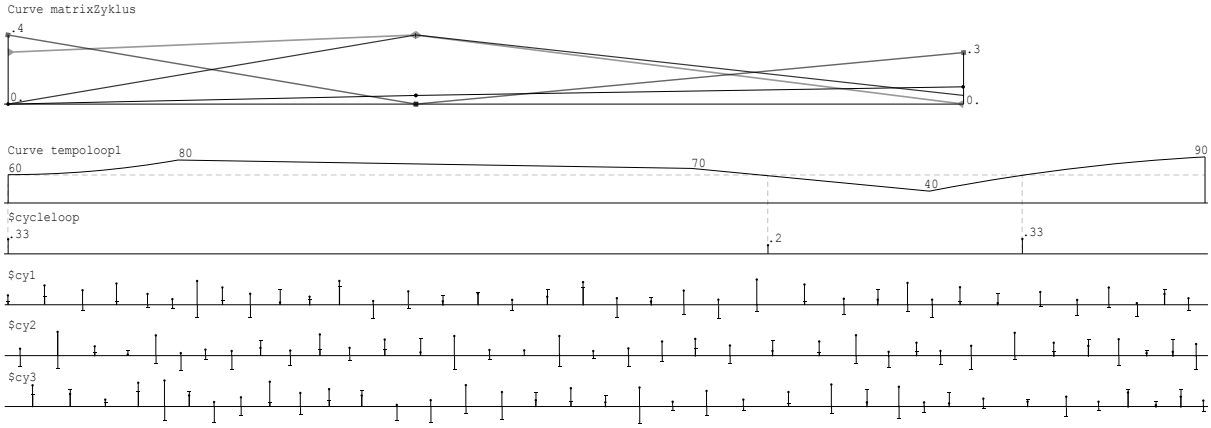
The hierarchy view uses EVENT objects one and three as primary nodes, and the layer1 group as the parent secondary node, from which the subgroups branch out. The existence of common variables between the final loop node and the previous constructs is pointed out through colour highlighting. The two whenever nodes are displayed next to the tree, while their parent node is in view.

Finally, in the inspector panel we track the variables $count, $t_loop1 and $cycleloop, assuming the current view position is after the end of the tempoloop1 curve, which ends on the value 90 for $t_loop1, thus fulfilling the loop's end condition. The user might choose to track a different configuration of receivers in the inspector, depending on their particular focus and the task at hand.

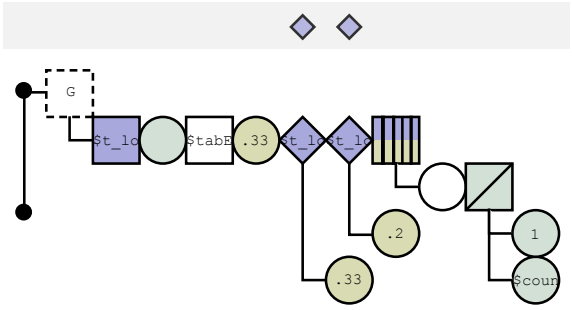
[15] as found in JBLO_loop_ex-StepTWO.asco.txt.



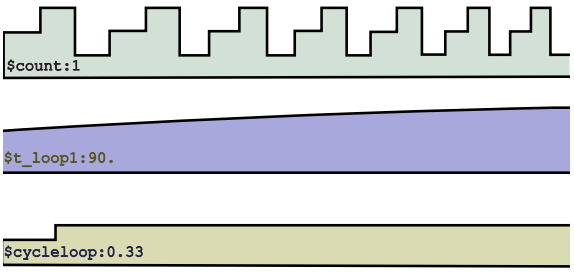

**(a)** *Action view. The right-side part has been cropped for space considerations.*




**(b)** *Simulation view*




**(c)** *Hierarchy view*

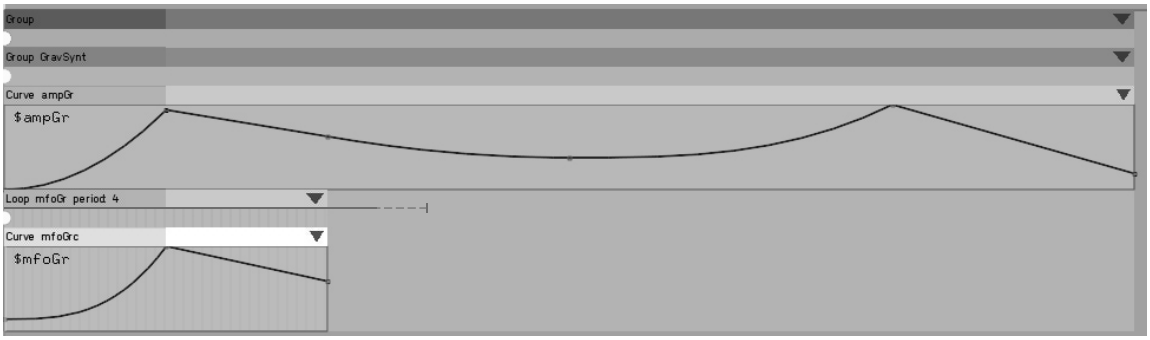



**(d)** *Inspector panel*

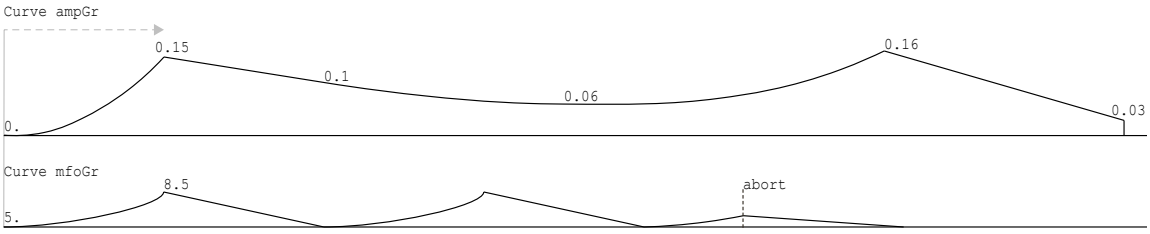
**Figure 18:** *Visualisations for code listing 9.*

```
EVENT 30 one                ; a dummy score event

GROUP layer1 {
        curve tempoloop1 @grain := 0.05s, @action := recvr_tempo $t_loop1 {
                $t_loop1 {
; [STATIC CURVE DEFINITION]
                }
        }

        $count := 1

        curve matrixZyklus @grain := 0.05s {
                $tabE {
; [4-DIMENSIONAL CURVE DEFINITION]
                }
        }

        $cycleloop := 1/3

        whenever ($t_loop1 > 60 )
                { let $cycleloop := 1/3 }

        whenever ($t_loop1 < 60 )
                { let $cycleloop := 1/5 }

        loop Zyklus $cycleloop @tempo := $t_loop1 {
                @command(("cy"+$count+"_freq")) ((@random()) + $freqZ)
                @ADSR($count,10,30,90,60)
                @command(("cy"+$count+"_pan")) ((@random()*2)-1)
                if ($count >= 3)
                        { let $count := 1 }
                else { let $count := $count + 1 }
        } until ($t_loop1 == 90)
}

EVENT 5 three               ; a dummy score event
```

**Code 9:** *Dynamic loop example: the tempo of the loop construct and the delay between iterations are computed at runtime.*




**(a)** *Action view. The parent group and all its members have a 2-beat sync target. The* loop *has a dynamic finite lifespan.*




**(b)** *Simulation view. The bottom curve receives an* abort *message during its third iteration. Both curves inherit the 2-beat synchronisation look-ahead.*

**Figure 19:** *Visualisations for code listing 10.*

## 5.2 Score excerpt: *Tesla*

Our final case study is excerpted from the score of "Tesla ou l'effet d'étrangeté" for viola and live electronics, by composer Julia Blondeau. Again we focus, in code listing 10, on the actions associated to a single event in the instrumental part. The visualisation models are proposed in Figures 19a and b.

A synthesizer is controlled by the components of GROUP GravSynt, which has a dynamic synchronisation target[16] of 2 beats. We indicate this lookahead interval as a shading of the group header in the action view, and a dotted line arrow in the simulation view.

The group contains the following: a static message to the SPAT1 receiver, a triggering of the ASCOtoCS_SYNTH_Ant process (previously defined through @proc_def), and 5ms later, the triggering of the SYNTH_Ant_curveBat process. After 5ms, curve ampGr is launched, which lasts 14 beats.

Simultaneously with the curve above, a loop is triggered, controlling an oscillation in the $mfoGr variable. Each iteration creates a 4-beat curve that starts where the previous one left off (after aborting the previous curve if necessary), and finally, when the loop is stopped, its @abort action ensures a smooth 2-beat reset of $mfoGr to the value 5. Thus, its lifespan is dynamic and finite: it has no set end condition, but is stopped by an abort message elsewhere in the score.

The inconvenience of an unknown endpoint is removed in the simulation view, which by definition is based on an execution of the score. This model is also able to unfold the

[16] as defined in section 2.4.

loop construct in time along its computed duration, attaching the @abort sequence at the end.

The hierarchical and inspector views provide no new insights in this case; we omit them for space considerations.

```
NOTE D3 4
; [STATIC ATOMIC MESSAGES]

GROUP GravSynt @target [2] {
   SPAT1 xy 0 -0.9
   ::ASCOtoCS_SYNTH_Ant(50,0.,50,0.09,0.,0.00001,0.001,5,1)
   0.005s ::SYNTH_Ant_curveBat([0.0001,2,0.15,2,0.0001,4,0.14,4,0.0001])

   0.005s curve ampGr @grain := 0.05s,
        @action := ASCOtoCS_SYNTH_L c kampAntes $ampGr {
        $ampGr {
; [STATIC CURVE DEFINITION]
        }
   }

   $mfoGr := 5.

   loop mfoGr 4
        @abort {
                curve mfoGr @grain := 0.05s,
                        @action := ASCOtoCS_SYNTH_L c mfoAntes $mfoGr {
                        $mfoGr {
                                        { $mfoGr }
                                2       { 5. }
                        }
                }
        } ; end abort (final) actions
        {
        abort mfoGrc
        curve mfoGrc @grain := 0.05s,
                @action := ASCOtoCS_SYNTH_L c mfoAntes $mfoGr {
                $mfoGr {
                                { $mfoGr } @type "cubic"
                        2       { 8.5 }
                        2       { 5. }
                }
        } ; end main loop
   }
}

NOTE 0 0
```

**Code 10:** *Score excerpt: Tesla by Julia Blondeau, lines 444 to 483.*

# 6 Conclusions and future work

We introduced four interlinked models for visualising processes in dynamic mixed music, building towards a framework that can support various compositional and performative approaches to the music. Throughout the paper, we have worked under the assumption that a standardised visual notation would be beneficial for current and future practitioners and theorists. Still, the question should be asked: is such a *lingua franca* a necessary, or even desirable goal to pursue? While this is not the time for a definitive assessment, we would like to contribute a point of view based on our experience with *AscoGraph*.

Visualisation models such as the ones we propose have proven useful in practice, at least for a subset of computer music. This includes, but is not limited to, works which employ real-time score following, algorithmic generation, digital sound processing and sampling, recurring electronic actions and/or dynamic feedback between machine and musician. One case where our models might not be as appropriate, would be for instance spectralist music, where the impetus is in the evolution of timbre over time, as opposed to discrete pitched events. Even so, the task of extending our models into this direction might prove worthwhile: *Antescofo* is on the way to incorporating audio processing in its engine, making integrated spectral operations a possibility. Other aspects of music-making that our present work is challenged by are spatialisation (or other heavily multi-dimensional processes), improvisation (which the *Antescofo* language approaches through whenever constructs and/or dynamic jumps) and complex synchronisation strategies. Our proposed methods are still far from comprehensively solving these problems.

Over the whole recorded history of music we have seen notation fostering and being in turn shaped by musical creativity. While there will always be artists working outside of general norms (this is as true today as it's ever been, with bespoke visualisations, audio-visual art, live coding and other practices that integrate the visual with the musical piece), we aim to outline a framework for creating and interpreting mixed music which will apply to, and in turn generate, a large set of approaches.

Today's democratisation and diversity of musical practices means we cannot hope to reach a new canonical notation standard. However, this fact also motivates us towards flexible alternatives to the traditional Western staff. The models we have proposed (action blocks, traced staff, code tree, inspector) capitalise on the state of the art in computer interface design, maintaining a generality of applicability while enabling specific perspectives into form and material. As the *Antescofo* reactive language evolves we intend to keep adapting our visual models to new abilities – like for instance, *patterns*, which help specify complex logical conditions for use in a whenever, or *objects*, which extend the language with aspects of object-oriented programming. Similarly to existing software coding environments, we are considering developing debugger-oriented features to further facilitate the testing of *Antescofo* programs.

Finally, we intend to further open up the system to user customisation through a powerful API and a robust import/export system that empowers musicians to integrate the visual notation into their personal authoring and performance workflows.

**Acknowledgments**

The authors would like to thank Julia Blondeau for contributions to section 5, and the Journal's editors for their highly constructive reviews.

# References

Agostini, A. and Ghisi, D., 2012. Bach: An Environment for Computer-Aided Composition in Max. In *International Computer Music Conference*. Ljubljana, Slovenia.

Allombert, A., Desainte-Catherine, M., and Assayag, G., 2008. Iscore: A System for Writing Interaction. In *International Conference on Digital Interactive Media in Entertainment and Arts*, DIMEA '08, pp. 360–367. ACM, New York, NY, USA. URL `http://doi.acm.org/10.1145/1413634.1413699`.

Arias, J., Desainte-Catherine, M., Salvati, S., and Rueda, C., 2014. Executing Hierarchical Interactive Scores in ReactiveML. In *Journées d'Informatique Musicale*. Bourges, France. URL `https://hal.archives-ouvertes.fr/hal-01095159`.

Assayag, G., Rueda, C., Laurson, M., Agon, C., and Delerue, O., 1999. Computer Assisted Composition at Ircam: From PatchWork to OpenMusic. *Computer Music Journal*, 23(3). URL `http://www.ircam.fr/equipes/repmus/RMPapers/CMJ98/index.html`.

Burloiu, G. and Cont, A., 2015. Visualizing Timed, Hierarchical Code Structures in AscoGraph. In *International Conference on Information Visualisation*. University of Barcelona, Barcelona, Spain. URL `https://hal.inria.fr/hal-01155618`.

Buxton, W., Patel, S., Reeves, W., and Baecker, R., 1979. The evolution of the SSSP score-editing tools. *Computer Music Journal*, 3(4):14–25.

Celerier, J.-M., Baltazar, P., Bossut, C., Vuaille, N., Couturier, J.-M., and Desainte-Catherine, M., 2015. OSSIA: towards a unified interface for scoring time and interaction. In M. Battier, J. Bresson, P. Couprie, C. Davy-Rigaux, D. Fober, Y. Geslin, H. Genevois, F. Picard, and A. Tacaille, eds., *International Conference on Technologies for Music Notation and Representation*, pp. 81–90. Institut de Recherche en Musicologie, Paris, France.

Chadabe, J., 1984. Interactive composing: An overview. *Computer Music Journal*, 8(1):22–27.

Clay, A. and Freeman, J., 2010. Preface: Virtual Scores and Real-Time Playing. *Contemporary Music Review*, 29(1):1–1. URL `http://dx.doi.org/10.1080/07494467.2010.509587`.

Coduys, T. and Ferry, G., 2004. Iannix Aesthetical/Symbolic Visualisations for Hypermedia Composition. In *Sound and Music Computing*.

Coffy, T., Giavitto, J.-L., and Cont, A., 2014. AscoGraph: A User Interface for Sequencing and Score Following for Interactive Music. In *International Computer Music Conference*. Athens, Greece. URL `https://hal.inria.fr/hal-01024865`.

Collins, K., Kapralos, B., and Tessler, H., 2014. *The Oxford Handbook of Interactive Audio*. Oxford Handbooks. Oxford University Press. URL `https://books.google.ro/books?id=kLsBBAAAQBAJ`.

Cont, A., 2008. ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music. In *International Computer Music Conference*. Belfast. URL `http://articles.ircam.fr/textes/Cont08a/`.

Cont, A., 2010. A coupled duration-focused architecture for realtime music to score alignment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(6):974–987.

Cont, A., 2011. On the creative use of score following and its impact on research. In *Sound and Music Computing*. Padova, Italy. URL `http://articles.ircam.fr/textes/Cont11a/`.

Cont, A., 2013. *Real-time Programming and Processing of Music Signals*. Habilitation à diriger des recherches, Université Pierre et Marie Curie - Paris VI. URL `https://tel.archives-ouvertes.fr/tel-00829771`.

Cont, A., Echeveste, J., Giavitto, J.-L., and Jacquemard, F., 2012. Correct Automatic Accompaniment Despite Machine Listening or Human Errors in Antescofo. In *International Computer Music Conference*. IRZU - the Institute for Sonic Arts Research, Ljubljana, Slovenia. URL `http://hal.inria.fr/hal-00718854`.

Cycling '74, 2016. Max is a visual programming language for media. `https://cycling74.com/products/max/`. URL `http://www.cycling74.com/`.

D. Fober, S. L., Y. Orlarey, 2012. INScore - An Environment for the Design of Live Music Scores. In *Proceedings of the Linux Audio Conference - LAC*.

Desainte-Catherine, M. and Allombert, A., 2005. Interactive scores : A model for specifying temporal relations between interactive and static events. *Journal of New Music Research*, 34:361–374. URL `https://hal.archives-ouvertes.fr/hal-00307925`.

Echeveste, J., Cont, A., Giavitto, J.-L., and Jacquemard, F., 2013a. Operational semantics of a domain specific language for real time musician-computer interaction. *Discrete Event Dynamic Systems*, 23(4):343–383. URL `https://hal.inria.fr/hal-00854719`.

Echeveste, J., Giavitto, J.-L., and Cont, A., 2013b. A Dynamic Timed-Language for Computer-Human Musical Interaction. Research Report RR-8422, INRIA. URL `http://hal.inria.fr/hal-00917469`.

Echeveste, J., Giavitto, J.-L., and Cont, A., 2015. Programming with Events and Durations in Multiple Times: The Antescofo DSL. *ACM Transactions on Programming Languages and Systems*. (submitted).

Freeman, J., 2010. Web-based collaboration, live musical performance and open-form scores. *International Journal of Performance Arts and Digital Media*, 6(2):149–170.

Giavitto, J.-L., Cont, A., Echeveste, J., and Members, M. T., 2015. *Antescofo: A Not-so-short Introduction to Version 0.x*. MuTant Team-Project, IRCAM, Paris, France. URL `http://support.ircam.fr/docs/Antescofo/AntescofoReference.pdf`.

Kurtz, M., 1992. *Stockhausen : a biography*. Faber and Faber.

Manoury, P., 1990. *La note et le son*. L'Hamartan.

Manoury, P., 2013. Compositional Procedures in Tensio. *Contemporary Music Review*, 32(1):61–97.

Manoury, P., 2016. List of works. `http://brahms.ircam.fr/philippe-manoury`.

McCartney, J., 1996. SuperCollider: a new real time synthesis language. In *International Computer Music Conference*. URL `http://www.audiosynth.com/icmc96paper.html`.

Nouno, G., Cont, A., Carpentier, G., and Harvey, J., 2009. Making an orchestra speak. In *Sound and Music Computing*. Porto, Portugal. URL `https://hal.inria.fr/hal-00839067`. SMC2009 Best Paper Award.

Poncelet, C. and Jacquemard, F., 2015. Model Based Testing of an Interactive Music System. In *ACM/SIGAPP Symposium On Applied Computing*. ACM, Salamanca, Spain. URL `https://hal.archives-ouvertes.fr/hal-01097345`.

Puckette, M., 1997. Pure data. In *International Computer Music Conference*. Thessaloniki, Greece.

Puckette, M., 2002. Using Pd as a score language. In *International Computer Music Conference*. Gothenburg, Sweden.

Puckette, M., 2004. A divide between 'compositional' and 'performative' aspects of Pd. In *First International Pd Convention*. Graz, Austria.

Risset, J.-C., 1999. Composing in real-time? *Contemporary Music Review*, 18(3):31–39. URL `http://dx.doi.org/10.1080/07494469900640331`.

Roads, C., 1977. Composing Grammars (2nd ed. 1978). In *International Computer Music Conference*.

Roberts, J. C., 2007. State of the art: Coordinated and multiple views in exploratory visualization. In *International Conference on Coordinated and Multiple Views in Exploratory Visualization*, pp. 61–71. IEEE.

Rowe, R., 1993. *Interactive Music Systems: Machine Listening and Composing*. MIT Press (MA). URL `https://books.google.ro/books?id=S9-9oQEACAAJ`.

Stroppa, M., 1999. Live electronics or...live music? Towards a critique of interaction. *Contemporary Music Review*, 18(3):41–77. URL http://dx.doi.org/10.1080/07494469900640341.

Toro-Bermúdez, M., Desainte-catherine, M., and Baltazar, P., 2010. A Model for Interactive Scores with Temporal Constraints and Conditional Branching. In *Journées de Informatique Musicale*.

Trapani, C. and Echeveste, J., 2014. Real Time Tempo Canons with Antescofo. In *International Computer Music Conference*, p. 207. Athens, Greece. URL https://hal.archives-ouvertes.fr/hal-01053836.

Wang, G., 2008. *The chuck audio programming language. a strongly-timed and on-the-fly environ/mentality*. Princeton University.

Wiggins, G., Miranda, E., Smaill, A., and Harris, M., 1993. Surveying musical representation systems: A framework for evaluation. *Computer Music Journal*, 17(3):31–42.

Winkler, T., 2001. *Composing Interactive Music: Techniques and Ideas Using Max*. MIT Press.

Xenakis, I., 1992. *Formalized Music (2nd ed.)*. Pendragon Press.