A Dynamic Timed-Language for Computer-Human Musical Interaction

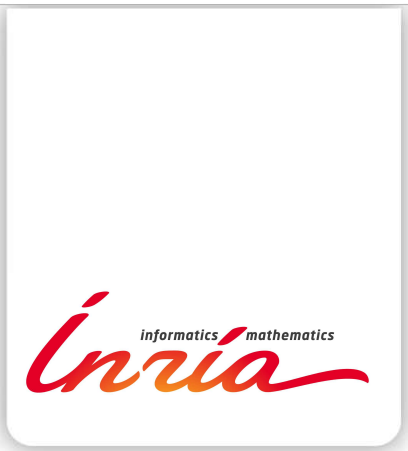
José Echeveste, Jean-Louis Giavitto, Arshia Cont

A Dynamic Timed-Language for Computer-Human Musical Interaction

José Echeveste, Jean-Louis Giavitto, Arshia Cont

RESEARCH REPORT
N° 8422
December 2013
Project-Teams Mutant

ISSN 0249-6399 ISRN INRIA/RR--8422--FR+ENG



A Dynamic Timed-Language for Computer-Human Musical Interaction

José Echeveste, Jean-Louis Giavitto, Arshia Cont

Project-Teams Mutant

Research Report n° 8422 — December 2013 — 22 pages

Abstract: In this paper, we present the programing of time and interaction in *Antescofo*, a real-time system for performance coordination between musicians and computer processes during live music performance. To this end, *Antescofo* relies on artificial machine listening and a domain specific real-time programing language. It extends each paradigm through strong coupling of the two and strong emphasis on temporal semantics and behavior of the system.
The challenge in bringing human actions in the loop of computing is strongly related to temporal semantics of the language, and timeliness of live execution despite heterogeneous nature of time in the two mediums. Interaction scenarii are expressed at a symbolic level through the management of musical time (*i.e.* events like notes or beats in relative tempi) and of the 'physical' time (with relationships like succession, delay, duration, speed).
Antescofo unique features are presented through a series of paradigmatic program samples which illustrate how to manage execution of different audio processes through time and their interactions with an external environment.
The *Antescofo* approach has been validated through numerous uses of the system in live electronic performances in contemporary music repertoire by various international music ensembles.

Key-words: Realtime Programming, Computer Music, Domain Specific Language, Score Following, Timed System, *Antescofo*

RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Un langage dynamique temporisé pour l'interaction musicien-machine

Résumé : Ce papier présente le système temps-réel Antescofo et son langage dédié. Celui-ci permet de décrire des scénarios temporels où des processus de musique électronique sont calculés et ordonnancés en fonction du jeu d'un musicien. Pour ce faire, *Antescofo* couple un système de suivi de partition avec un module réactif. L'originalité du système réside dans la sémantique temporelle du langage adaptée aux caractéristiques critiques de l'interaction musicale. Le temps et les événements peuvent s'exprimés de façon symbolique dans une échelle absolue (en seconde) ou des échelles relatives à des tempos. Nous aborderons les caractéristiques du langage et de la partie réactive d'*Antescofo* illustrées à travers une série d'exemples paradigmatiques.

Mots-clés : Programmation temps réel, Inforamtique musicale, langage dédié, suivi de partition ,système temporisé, *Antescofo*

1 Introduction

1.1 Musical Interpretation and Mixed Music

Reading the score of a musical piece, a perfomer is able to understand the temporal organizations of musical objects such as notes, chords, phrases or movements, designed by the composer, involving specific timing, duration, sequencing and dynamics. The timing relationship in the score between objects and their durations are *virtual i.e.*, partially specified relatively to other objects' locations and to a musical clock undetermined until real-time performance. During live performance, the performer interprets such information, execute the score with precise timing and in a personal way where durations and delays are evaluated to physical time (measurable in second). This interpretation is dynamic and a function of musicians' parameters, leading to different values for the same duration depending on the context. These values depend highly on the performance, individual performers and musical interpretation strategies such as stylistic features that are neither determined nor easily formalizable. Nevertheless, when several musicians play together the sound result remains coherent despite indeterminism of the score. Coherency is achieved through dynamics of attention, anticipation and adapted synchronization strategies between musicians.

In this paper, we focus on *mixed music*, defined as live association of computers and human musicians on stage, that weaves together acoustic instruments and real-time electronics. Since the middle of 20th century, composers have included electronic sounds and effects in their creation and the role of electronic processes has been broadened as a result of technological progress. The electronic of a mixed piece consists nowadays of parallel programs executed in real-time whose computational nature vary from signal processing (sound transformations or spatialization) and control (trigger of a process, change of a parameter) to transactional processing (concatenative synthesis on sound databases) and more.

1.2 The Antescofo System

The work presented here is about how to achieve the musical interaction, as described above, between human musicians and computer, in a context of mixed music where the temporal development of musical processes depends on an active listening and complex synchronization strategies [9].

We have proposed in [6] a novel architecture relying on the strong coupling of artificial machine listening and a domain specific real-time programming language for compositional and performative purposes. This way, the programmer composes a program as an *augmented score* where musical objects stand next to computer programs, following specific temporal organizations. The augmented score defines both instrumental and electronic parts and the instructions for their real-time coordination during a performance, that is to say, the specification of the events to recognize in real-time and of the actions triggered by these. During the performance, the language runtime evaluates the augmented score and controls processes synchronously with the musical environment thanks to artificial machine listening. This approach has been implemented in the *Antescofo* system and validated in many live performances.

1.3 Programming Musical Interactions

While the real-time detection of score events out of live musicians' performance has been widely addressed in the literature, the compositional and reactive aspects associated to machine listening has been poorly investigated. The performative dimension of the computer accompaniment that

must be explicitly introduced in the score raises new challenges with respect to other real-time languages:

- the handling of multiple notions of time (event and duration) and clocks (tempi and physical time);
- the specification of subtle "musical synchronizations" between the parts performed by the computer and the human performance;
- the management of errors w.r.t. the augmented score, both from the musicians (wrong or missed note) and from the listening machine (recognition error).

We discuss these challenges and present our solution, the *Antescofo* domain specific real-time language (DSL). The rationales and the design decisions behind the expressive temporal constructions it offers are examined and validated through paradigmatic examples in the field of mixed music.

The remainder of this paper is organized as follows. The next section gives some background and reviews related works. Section 3 presents the main constructions and the various notions of time and clock offered by the system. The rationale behind these constructs are discussed. Synchronization strategies and error handling strategies are exposed in Section 4. Section 5 sketches the architecture of the runtime and some implementation details. In section 6, a series of score examples illustrate how to manage the execution of different musical processes through time and how to interact with the external environment. These "temporal design patterns" are partly inspired by the extensive use in live electronic performances of *Antescofo* in contemporary music repertoire of composers such as P. Boulez, P. Manoury, M. Stroppa, E. Nunes... adopted by various music ensembles such as Los Angeles and Berlin Philharmonics, just to name a few.

2 Background and Related Work

The *Antescofo* system is an attempt to address both the compositional (programming) and the performative (real-time evaluation and synchronization) dimensions in mixed music. *Antescofo* achieves this goal relying on two subsystems (see figure 1): a *listening machine* that tracks in real-time the position of the musicians in the augmented score and a *reactive engine* relying on a *dedicated synchronous and timed programming language*. *Antescofo* is unique in the sense that it attempts to couple two literatures, real-time recognition and reactive systems, usually considered separately but whose integration makes complete sense in interactive musical practices. To this respect, existing systems address either of the two issues.

2.1 Score Following

Traditionally score following is defined as the real-time automatic alignment of an audio stream played by one or more musicians, into a symbolic musical score. It was first invented in the 80s [12, 34] employing symbolic inputs and extended a decade later to audio inputs, following advances in real-time signal processing. Late 90s saw the integration of probabilistic methods for score following improving the robustness of such systems.

An example of such systems is *Music-Plus-One* [31] which undertakes automatic accompaniment for classical music repertoire. This system is similar to *Antescofo* in its machine listening features. However it does not allow musicians to arrange and to compose the accompaniment scores; moreover synchrony is a result of the machine listening and does fault tolerance is not explicitly addressed [9].

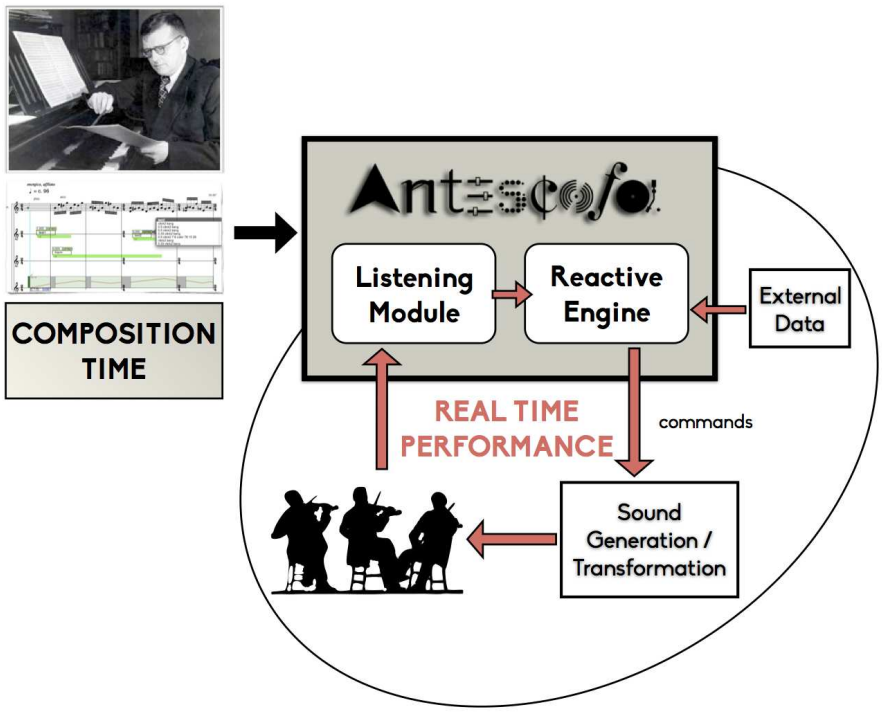


Figure 1: General architecture of *Antescofo*. The score is an input of both the listening machine and the reactive engine. The events recognized by the listening machine are signaled to the reactive engine which schedule the actions at the right time.

Comparing solely the score following capabilities, the *Antescofo* listening machine is polyphonic [7] and constantly decodes the tempo of the live performer. This is achieved by explicit time models inspired by cognitive models of musical synchrony in the brain [22] which provide both the tempo of the musician in real-time and also the *anticipated* position of future events (used for real-time scheduling). To this respect, *Antescofo* imitates human musicians who constantly base their actions on current events and also future belief of incoming events. Readers curious on the algorithmic details of the listening machine can refer to [7].

Recently in the domain of score following, it should also be noted that a certain number of new tablet computer applications have arisen like Tonara (for automatic page turning) or MusicScore [25] (for score editing and automatic accompaniment). These latter applications are intended towards content creation or education and do not address interactivity for authoring and performing mediums.

2.2 Computer Music Languages and Sequencers

FORMULA is a computer music performance system, developed by Anderson and Kuivila, in 1990 [3]. It is similar to the reactive part of *Antescofo*. A human performer controlls the inputs of the system, triggering computations and outputs actions as accordance with scheduling strategies. The management of virtual time and asynchronous event is similar to that of *Antescofo*. Furthermore users can specify a temporal window for the evaluation of processes so

that the system knows the priorities and optimizes the scheduling. *Antescofo* differs from this system in its strong coupling with the listening machine, the management of errors, the dynamic synchronization strategies and the stream nature of variables.

The *Iscore* project [14] is a computer-assisted composition tool. It allows composers to build musical parts and structures their pieces, binding them with temporal logical relations inspired by Allen relations [2]. However, even if *Iscore* and similar approaches in the field of computer-assisted composition may act as a kind of sequencer, they do not address the problem of dynamic binding and of real-time interactivity with an outside environment.

Real-time computer music languages should allow the specification of temporal developments of musical processes amongst many other things. In Max/MSP or PureData [30] which are dataflow graphical languages and where control treatments and signal processing are explicitly determined, it is easy to construct permanent behaviors, but much harder to organize and control complex changing behaviors. In Csound [5] successor of MusicN languages family, an `Orchestra` part defines complex audio generators and a `Score` part organizes them in time. However, static nature of CSound programs evade real-time interaction and interpretation.

Chuck [35], SuperCollider [28], Impromptu [32] and other musical Live Coding languages propose a distinct approach since the time of composition (programming) coincides with the time of performance. The programmer is an integral part of the musical environment. Thus, there is a much weaker link between instrumentalists and electronic accompaniment. Interaction is enabled between the programmer and processes, but interactivity with continuous media such as audio from live musicians is impossible.

Dannenberg has developed several functional languages for composition and sound synthesis ([11], [13]). An important notion developed in this work is the concept of behavioral abstraction; user can specify a behavior (as a transposition) that will adapt according to the context in which it is realized. We will study this expressive feature in the context of *Antescofo* through high-order constructions and reification in future works. These languages propose a declarative programming style for temporal specification of processes and synthesis parameters but do not integrate a listening machine and favor a programming style with respect to the traditional score, whereas *Antescofo* tries to stick with classical music notation.

Compared with classical sequencers such as LogicPro, CuBase or ProTools, *Antescofo* is far more dynamic. Tasks can be launched with delays computed by arbitrary expressions. These expressions evaluates to a duration expressed in physical time or in relative time provided by the integrated listening machine (through the real-time tempo estimation). Ableton Live adds more possibilities of interaction compared to usual sequencers but the tools proposed for time organization are restricted to factory preset time grids.

2.3 Real-Time Programming Languages for Embedded Systems

The language developed in *Antescofo* can be seen as a domain specific synchronous and timed reactive language in which the accompaniment actions of a mixed score are specified. As a matter of fact, an analogy can be made between interactive music composition using *Antescofo* and synchronous programming or others timed languages. In a score, a composer predicts and specifies the temporal evolution of processes and the events of the musical performance (inputs) on which they depend. In the same way that an engineer would use a synchronous language to specify commands to be sent to actuators depending on data acquired through sensors, the composer describes the musical responses of the system (the electronic actions) relative to its environment (expected events). As for reactive systems [17], an augmented score assumes the *strong synchrony hypothesis*: actions hooked directly to an event should occur in zero-time and

in the right order. This hypothesis is unrealistic, however it is an excellent abstraction of musical behavior. In practice, the system needs to be quick enough to preserve the auditory perception of simultaneity, which is on the order of 20 milliseconds. And similarly to classical reactive systems, the result is expected to be deterministic (as described in the score) despite real-time variations in the environment (human musicians' interpretations).

As in Lustre [18], Signal [24] and Lucid Synchrone [29], the programmer can access to past values of variables, achieving the idea of a stream of values. However, the programming style of *Antescofo* is not declarative and the handling of variables is similar to the imperative style of Esterell [4]. However, contrary to these languages, *Antescofo* is dynamic and allows the dynamic creation of parallel processes on the reception of an event. And contrary to ReactiveML [26], there is no need to explicitly manage the notion of current instant.

The above mentioned synchronous languages support an event-triggered programming style: in an event triggered system, a processing activity is initiated as a consequence of the occurrence of a significant event [20]. In addition, *Antescofo* supports a time-triggered style: activities can also be initiated at predetermined point in real-time using arbitrary delays. From this point of view, the augmented score can be compared with other timed languages like Giotto [19] and XGiotto [16] inspired by the time-triggered architecture [21]. This kind of languages provides an abstract model for the implementation of embedded control systems with hard real-time constraints. They allow to specify precise timing of each tasks but focus mainly on periodic tasks. While Giotto is purely time-triggered, xGiotto is also an event-triggered language which accommodates asynchronous events. However, in *Antescofo*, the specification of the scores raises new issues and requires special extensional notations. The handling of multiple related tempi and the handling of errors are also specific to the musical application domain.

3 The Antescofo Language

An *Antescofo* score is a specification of both an instrumental part and the accompaniment actions. As explained above, this specification is given as an input to the system within one augmented score (see Figure 1). During live performance, *Antescofo* is a system reacting to data from the listening machine and from the external environment. The reactive system dynamically considers the tempo fluctuations and the values of external variables for the interpretation of accompaniment actions. The possibility of *dating the events and the actions relatively to the tempo*, as in a classical score, is one of the main strengths of *Antescofo*. Within the augmented score language, the composer can thus decide to associate actions to certain events with delays (physical time or relative to the tempo), to group actions together, to define timing behaviors, to structure groups hierarchically and to allow groups act in parallel.

This section briefly describes Antescofo primitive constructs that are show-cased later in section 6.

Instrumental Events An augmented score is a sequence of instrumental events and actions. The syntax for the instrumental part allows the description (pitches + duration) of events such as notes, chords, trills, glissandi and improvisation boxes ([6]).

Actions Actions are divided into *atomic action* performing an elementary computation and *compound actions*. Compound actions group others actions. Actions are triggered by an event.

Atomic Actions. An atomic action can corresponds to a variable assignment (*let* $\$v := expr$), or another specific internal command (like killing an action referred by its name) or an external command sent to the audio environment, for instance to a sound synthesis module.

```
let <variable> := <expression>
kill <name>
<command_name> <expression_list>
```

Group. The `group` construction describes several actions logically within a same block that share common properties of tempo, synchronization and errors handling strategies in order to create polyphonic phrases.

```
group <name> { <actions_list> }
```

Loop. The `loop` construction is similar to group where actions in the loop body are iterated depending on a period specification. Each iteration takes the same amount of time, a period. A optional `until` statement evaluated at each iteration stops the `loop` if the condition is true.

```
loop <name> <period>
{ <actions_list> }
until <expression>
```

Curve. The curve constructions convert group atomic actions to continuously evolving actions through specified interpolation and time grain. The sampling rate *step* can be as small as needed to achieve perceptual continuity.

```
curve <command> <grain> <interpol>
{ ( <delay> [<expression_list>] )* }
```

Actions are sampled following the time grain and through interpolation between the reference values given by a timed sequence $(d_1\ \vec{v}_1)\ldots(d_k\ \vec{v}_k)$ defined by the expressions in the body of the `curve`. The delay d_i specifies the duration of the interpolation between $\vec{v}_i$ and $\vec{v}_{i+1}$. They can be given in absolute or relative time. The interpolation is either linear or exponential.

If the continuous group starts at time t_0, the kth parameter goes from $\vec{v}^k_{j-1}$ to $\vec{v}^k_j$ during the time interval $[t_0 + \sum_i^{j-1} d_i, t_0 + \sum_i^j d_i]$ where $1 < j$ and $\vec{v}^k_j$ denotes the kth component of the vector $\vec{v}_j$.

Conditions Each action ends by the optional specification of a guard. The guard statement allows to launch an action depending on a logical condition:

```
<action> guard (<expression>)
```

Delays Each action can be preceded by an optional specification of a *delay*. It defines the amount of time between the previous event or the previous action in the score. Thus, the following sequence

```
NOTE C 2.0
  d1 <action1>
  d2 <action2>
NOTE D 1.0
```

specifies that, in an ideal performance that adhere strictly to the temporal constraint specified in the score, `<action1>` will be triggered d_1 after the recognition of the C note, and `<action2>` will be triggered d_2 after the launching of `<action1>`.

There are several ways to implement these temporal relationships at runtime. For instance, assuming that `<action2>` actually occurs *after* the occurrence of `NOTE D`, one may count a delay of $d_1 + d_2 - 2.0$ starting from `NOTE D` before launching `<action2>`. This approach will be for instance

more tightly coupled with the stream of musical events. Synchronization strategies are discussed in section 4.

The absence of a delay is equivalent to a zero delay. A zero-delayed action is launched synchronously with the preceding action or with the recognition of its associated event. Synchronous actions are performed in the same logical instant and last zero time, cf. paragraph 5.1.

Relative Time and Tempo Delays are arbitrary expressions and can be expressed in *relative time* $\mathcal{T}$ (in beats) or in physical time $\mathcal{P}$ (in milliseconds). Physical time $\mathcal{P}$ or relative time $\mathcal{T}$ are frames of reference used to interpret delays and to give a date to the occurrence of an event or to the launching of an action.

A frame of reference is defined by a *tempo* $T_{\mathcal{T}}$ which specifies the "passing of time" in the frame of reference relatively to the physical time [27]. In short, a tempo is expressed as a number of beats per minutes. The tempo $T_{\mathcal{T}}$ associated to a relative time $\mathcal{T}$ can be any *Antescofo* variable. The date $t_{\mathcal{P}}$ of the occurrence of an event in the physical time and the date $t_{\mathcal{T}}$ of the same event in the relative time $\mathcal{T}$ are linked by the equation:

$$t_{\mathcal{T}} = \int_0^{t_{\mathcal{P}}} T_{\mathcal{T}} \tag{1}$$

Variable updates are discrete in *Antescofo*; so, in this equation, $T_{\mathcal{T}}$ is interpreted as a piecewise constant function.

Antescofo provides a predefined dynamic tempo variable through the system variable `$RT_TEMPO`. This tempo is refered as "*the* tempo" and has a tremendous importance because it is the time frame naturally associated with the musician part of the score. This variable is extracted from the audio stream by the listening machine, relying on cognitive model of musician behavior [8]. The corresponding frame of reference is used when we speak of "relative time" without additional precision.

Programmers may introduce their own frames of reference by specifying a tempo local to a group using a dedicated attribute. This frame of reference is used for all relative delays and datation used in the actions within this group. As for other attributes, a *local tempo* is inherited by the nested groups. A local tempo is an arbitrary expression involving any variables. This expression is evaluated continuously in time for computing dynamically the relationships specified by equation 1.

Attributes and Compound Actions Local tempi, synchronization and error strategies are specified for each action using *attributes*. Attributes are also used to name an action (for later reference), to trace its execution, etc. If they are not explicitly defined, the attributes of an action are *inherited* from the enclosing action. Thus, using compound actions, the composer can create easily nested hierarchies (groups inside groups) sharing an homogeneous behavior.

Expressions Expressions can be used to specify delays, loops periods groups local tempi, guards, and parameters values sent to the environment. Expression are evaluated into values: atomic values include booleans, ints, floats, strings, etc.; non-atomic values are data structures like maps or interpolated functions. A large set of predefined functions can be used to combine values to build new values. The programmer can defines its own functions and macros.

User variables *Antescofo* variables are global and visible everywhere in the augmented score or they are declared local to a group which limits its scope and its life. For instance, as common

in scoped programming language, the life of variable declared local in a `loop` is restricted to one instance of the loop body.

Variable are managed in a imperative manner. The assignment of a variable is seen as an internal event that occurs at some date. Such event is associated to a logical instant. Each *Antescofo* variable has a time-stamped history. So, the value of a variable at a given date can be recovered from the history, achieving the notion of *stream of values*. Thus, `$v` corresponds to the last value of the stream and

```
[<date>]:$v
```

corresponds to the value of variable `$v` at date `date`. The date can be expressed in three ways; as an update count: expression `[2#]:$v` returns then antepenultimate value of the stream; as an absolute date: expression `[3s]:$v` returns the value of `$v` three seconds ago; and as a relative date: expression `[2.5]:$v` returns the value of `$v` 2.5 beats ago.

For each variable, the programmer may specify the size n of its history. So, only the n "last values" of the variable are recorded. Accessing the value of a variable beyond the recorded values returns an undefined value.

User variables are assigned within an augmented score using the `let` construct. However, they can also be assigned by the external environment, using dedicated API.

Dates functions Two functions let the composer know the date of a logical instant associated to the assignment of a variable `$v`: `@date([n#]:$v)` returns the date in the absolute time frame of the nth to last assignement of `$v` and `@rdate([n#]:$v)` returns the date in the relative time frame.

System variables There are several variables which are updated by the system in addition to `$RT_TEMPO`. Composers have read-only access to this variables. Variable `$NOW` corresponds to the absolute date of the "current instant" in seconds. The "current instant" is the instant at which the value of `$NOW` is required. Variable `$RNOW` is the date in relative time (in beats) of the "current instant". Variables `$PITCH`, `$BEATPOS` and `$DUR` are respectively the pitch, the position in the score and the duration of the value of the last detected event.

`whenever` statement The `whenever` statement allows the launching of actions conditionally on the occurrence of a signal.

```
whenever (<predicate>)
{
  <actions_list>
} until (<expression>)
```

The condition `predicate` is an arbitrary predicate. Each time the variables of the `predicate` are updated, the predicate is re-evaluated. The `whenever` statement is a way to reduce and simplify the specification of the score particularly when actions have to be executed each time an event is detected. It also escapes the sequential nature of traditional scores. Resulting actions of a whenever statement are not statically associated to an event of the performer but dynamically satisfying some predicate, triggered as a result of a complex calculation, launched by external events, or any combinations of the above.

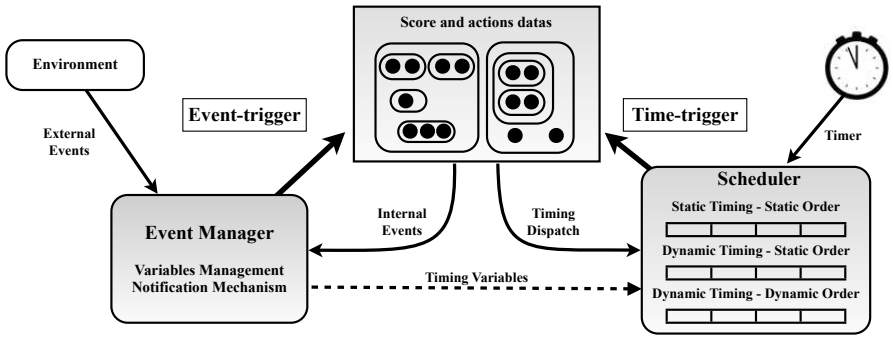


Figure 2: Antescofo Runtime

4 Strategies for Dynamic Synchronization and Fault Tolerance

The musician's performance is subject to many variations w.r.t. the score specification. There are several ways to adapt computer processes to this musical indeterminacy based on specific musical context. To this, one must add required real-time fault tolerance to errors introduced in the environment (human errors, noisy acoustics, etc.) or machine listening errors. The musical context that determines the correct synchronization and error handling strategies is at the composer or arranger's discretion. We propose several explicit synchronization strategies taking into account performance variations, programmable by composers and specified as attributes for compound actions. In this section, we only introduce those strategies that lie in the (temporal) context of this paper and refer the curious reader to [10].

The *Antescofo* scheduler (described in section 5) is responsible for scheduling actions at later instants during runtime using their associated delays. Event scheduling is interrupted by default if the delays are relative to a changing environmental variable (including musicians' tempo), and rescheduled accordingly. This behavior is *Antescofo*'s default strategy and qualified as `loose`. On top of synchronizing with (dynamic) time, one might expect events to be triggered not only according to their timing-positions, but also with respect to their relative event-positions. This second strategy is qualified as `tight` and provides a mixed time-trigger and event-trigger strategies of actions with regard to the external environment.

Similar to critical embedded systems, *Antescofo* can not stop in case of any error as "the show must go on"! Error handling strategies are also provided as action attributes in the *Antescofo* language and deal with the scope of an action, if their associated expected (musician) event is missing for any reason. To this end, an action (or compound action) is either `local` or `global`.

The overall behavior of actions during real-time execution depend on the combination of their associated synchronization and error handling strategies as discussed in [10]. It is also shown in [10] how these strategies correspond to various common forms of authoring in music writing.

5 Runtime

The Reactive Engine of *Antescofo* is based on a hybrid event-driven and timed architecture. The figure 2 illustrates this notion through its main components as described hereafter:

5.1 Logical Instant

From the Reactive Engine point of view, a *logical instant* is initiated for three possible reasons:

- the recognition of a musical event;
- the assignment by the external environment of a variable;
- the expiration of a delay.

The notion of logical instant is instrumental to maintain the synchronous abstraction and to reduce temporal approximation. Whenever a logical instant is started, the internal variables `$NOW` and `$RNOW` are updated. These variables are implicitly used in the management of variable history and in calculating dynamic delays. Within the same logical instant, synchronous actions are performed sequentially in the same order as in the score. This behavior may seem curious at first, but used in most synchronous programming languages, and it was proved very useful for controling side-effects among synchronous (parallel) actions [17].

Atomic actions are performed inside one logical instant. Compound actions can take several logical instants to complete. Between two logical instant, an action is either in the scheduler queue or is waiting for an event.

5.2 Event Notification

Notification of events from the machine listening module drops down to the more general case of variable-change notification from an external environment. The Reactive Engine maintains a list of actions to be notified upon the update of a given variable.

Actions associated to a musical event are notified through the `$BEATPOS` variable. This is also the case for the `group`, `loop` and `curve` constructions which need the current position in the score to launch their actions with `loose` synchronization strategy. The `whenever` construction, however, is notified by all the variables that appear in its condition.

The scheduler must also be globally notified upon any update of the tempo computed by the listening module and on the update of variables appearing in the local tempi expressions.

The notification of a variable change may trigger a computation that may end, directly or indirectly, in the assignment of the same variable. This is known as a "temporal shortcut" or a "non causal" computation in synchronous languages. The Event Manager takes care of stopping the propagation when a cycle is detected. Program resulting in temporal shortcuts are usually considered as bad practice and we are developing a static analysis of augmented scores to avoid such situations.

5.3 Timing Dispatch and Scheduling

At any instant, several parallel compound actions might be awaiting the expiration of the delay preceding the next action to be executed. Notice that multiple occurrences of the same compound action can be pending: this is the case for `whenever` when its temporal condition becomes true again, before the achievement of the previous instance.

Antescofo relies on a "timer service" provided by the host environment (Max, PureData, POSIX or IOS) to implement delays. Using a dedicated timer for each pending action is not acceptable because of overhead.

All pending actions can be completely ordered according to their delays, even if they are waiting in parallel. Theoretically, only one timer associated to a waiting queue is needed. However, this waiting queue should be reordered completely and the timer should be reset each time

a delay must be rescheduled. This occurs when the tempo or the local tempi are updated. To minimize the management overhead, we propose three waiting queues instead:

Static Timing-Static Order Queue: The first queue stores actions associated to delays expressed in absolute time. No delay conversion is needed and it suffices to insert new pending actions at the right place. For optimization, this queue also stores actions with constant local tempi: in this case, the conversion between the relative delay and the physical time is done once at insertion.

Dynamic Timing-Static Order Queue: The second queue corresponds to delays expressed relative to the musician tempo. The ordering of this queue is directly based on these delays expressed in beats: the order will be preserved, even if the tempo changes. Only the delay of the first action in this queue is converted in absolute time at runtime.

Dynamic Timing-Dynamic Order Queue: The third queue corresponds to delays which depend on local tempo expressions that are dynamic. When a variable appearing in a local tempo expression is updated, the subset of impacted delays are recomputed and their associated actions reinserted at the right place in the queue. The ordering is based on the delay converted in physical time.

Following this timing dispatch, *Antescofo* relies only on one timer corresponding to the waiting of the minimal delay between the heads of the three waiting queue. If the head of one waiting queue changes before the expiration of the timer for a new delay less than the remaining time, the timer is reset and the delay of the two other heads is updated (delays in queue are stored relatively to the previous element, so updating the head suffices).

The approach proposed here minimizes both the scheduling overhead and the interaction with the host environment. Notice that it preserves a deterministic order and hierarchical structures in the score between actions and events with the same frame of reference.

6 Examples

We illustrate the features provided by the *Antescofo* language for time and interaction management through some actual examples directly inspired by its use in several real-world pieces of mixed music. For the sake of this presentation, these examples have been simplified compared to their initial context but still reveal *temporal design patterns* enabled through the language. Here we refer to the notion of design pattern developed by Alexander [1] where "each pattern describes a problem which occurs over and over again (...)". The notion of design pattern is well known in the field of software development but a catalog of useful temporal organization has not yet been formalized in computer music. This lack of a descriptive catalog is certainly due to the recent emergence of computer music programming language handling time as a first class entity [33] far beyond the usual approach of "time as a resource" or "time as a non-functional property" or "real-time as a quality of service problem" [23].

6.1 Evolving Durations

The first example implements *Piano Phase*, a piece initially composed for two pianists by composer Steve Reich. The concept of the piece is based on a repeating sequence of 12 notes, played by musician M_1 and M_2 in parallel, where the second undergoes a systematic phase difference with regards to the first. In this example, the delays of the second process evolve as a function of iterations of the first; resulting to an evolutionary phase shift of the timing of the process with

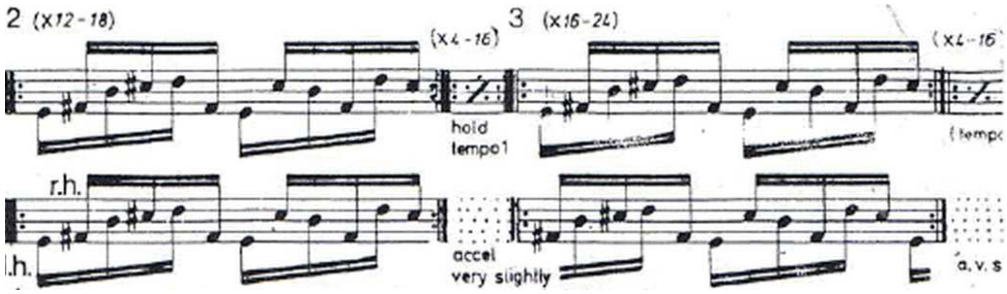


Figure 3: Fragment of the score of *Piano Phase* by Steve Reich.

respect to the first. This process is best described as follows and according to the original score in figure 3: During the first n_p periods ($12 \leq n_p \leq 18$) M_1 and M_2 play the same note at the same time. Then, a new stage begins: M_2 plays a little bit faster such that a phase shift of a quarter note is achieved after n_d periods ($4 \leq n_d \leq 16$). At this point, the two voices become synchronous again with the same phase, but with a logical event-shift during n_p periods. These two stages are iterated until the 12 possible shifts have been carried out.

In the *Antescofo* implementation shown below, a synthesizer simulates M_2 whereas M_1 is delegated to a human musician. The 12 statements `NOTE X Y` correspond to the notes played by the musician M_1 which will be recognized by the listening machine. The instruction `jump` defines a cyclic score. The actions played by the synthesizer are defined by the body of the `loop` with a period `$period` initialized to the duration of M_1' bar (3 beats). The synchronization strategy is `tight` in order to synchronize both with timing and position of the human musician as described in section 4. The delays between each action of the `loop` are specified by the variable `$delay` (initialized to 1/4 beats). Variable `$cpt` is employed to differentiate between the two stages of performance discussed above. The counter is incremented in the loop. When it reaches n_p, the period and the delays are updated. When it reaches zero modulo $(n_p + n_d)$, the period and the delay are reset to their initial values.

This *Antescofo* program shows how to author a complex algorithmic piece involving live musicians and computer processes. The behavior of computer actions stay intact despite any variations in musician's performance (tempo, etc.).

```
 let $period := 3.0
 let $delai := 0.25
 let $cpt := 0

NOTE E4 1/4 starting_note
  loop $period antetight {
    let $cpt:=($cpt+1) % (n_p + n_d)
    0.0 synth E4
    $delay synth F#4
    $delay synth B4
      ...seven additional notes...
    $delay synth D5
    $delay synth C5
  }

NOTE F#4 1/4
NOTE B4  1/4
 ...seven additional notes...
NOTE D5  1/4
NOTE C5  1/4 jump starting_note

  whenever($cpt = n_p) {
    let $periode:=$period-(1/4)/n_d
    let $delay:=$period/12
  }
  whenever($cpt = 0) {
    let $period:=3.0
    let $delay:=1/4
  }
```

6.2 Dynamic Tempi Management

Musicians have extreme abilities in dealing with various time-grids simultaneously during music performances. A significant example of this correponds to patterns investigated by composer Colon Nancarrow as shown in Figure **??**. This figure shows the original diagram representing the temporal organization of the two voices in "Study 21, Cannon X" for mechanical piano. The first voice accelerates while the second decelerates.

The specification of such processes with different tempi is straightforward in *Antescofo*. The main idea can be expressed as follows: The *Antescofo* score describes two main processes whose tempo evolutions are defined using the continuous `curve` construct controlling the ratio between the two tempi. The evolution is done every 0.1 second. Notice that here we mix physical time (sampling rate in seconds) and relative time in the same statement. The expressions of the loops' tempi depend on the musician's tempo estimation and on the $ratio variable. In this way, the first process will accelerate starting with a slower tempo than that of the musician and finishing with four times its tempo in 40 seconds.

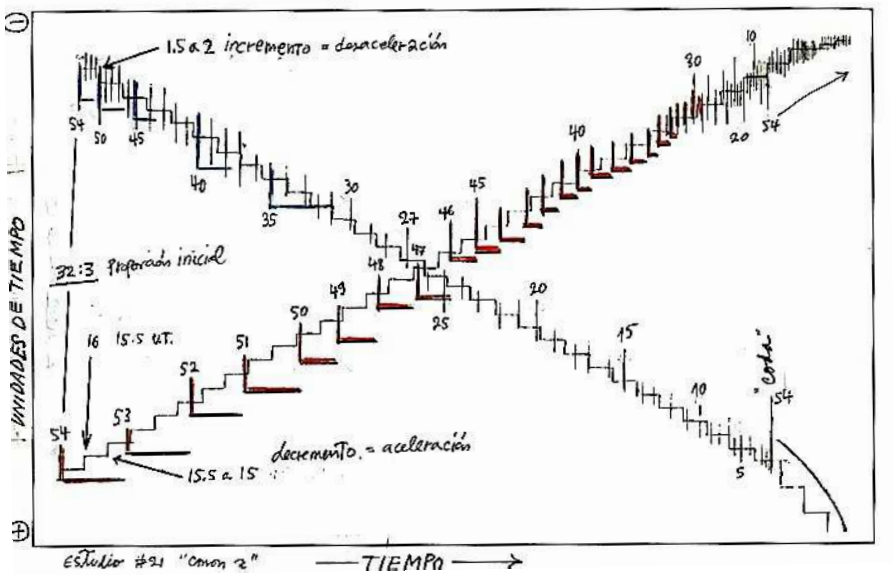


Figure 4: Graphical description by C. Nancarrow of its piece Study 21, *Canon X*, representing the tempi relations between 2 periodic voices. Abscissa axes corresponds to time, ordinate axis to tempo.

```
curve $r grain=0.1s {
  0.0   [1/4]
  40s   [4]
}
loop acc_loop 2.0 @tempo=$RT_TEMPO*$r {
  <actions...>
}
loop dec_loop 2.0 @tempo=$RT_TEMPO/$r {
  <actions...>
}
```

6.3 Context Switching and Watchdog

External controllers can be easily integrated in the *Antescofo* control flow, through an internal variable representing the state of the controller. This variable will be updated by the external environment (*e.g.*, using message-passing from a host environment).

The following example shows how to switch between two loops at the reception of a signal coming from a controller or when no signal has been received during the past 8 beats. Counter `$cpt` indicates which loop to play. The `abort` command is used to kill the loop currently played. Each time a signal from the controller is received, the current *watchdog* is aborted and a new one is set. The watchdog is implemented as a group which sets a restart signal after an 8 beat delay.

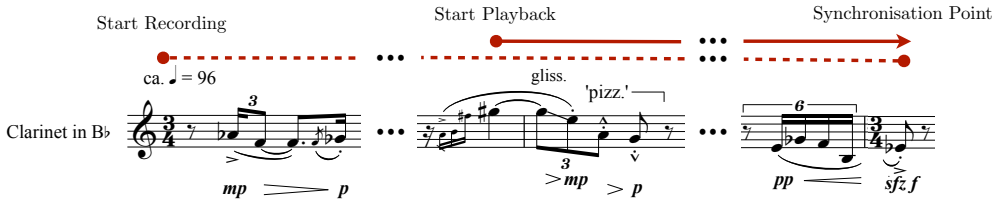


Figure 5: Excerpt from a sketch composed by Christopher Trapani. The dotted line corresponds to audio chunk recorded into a buffer. The arrowhead corresponds to its playback. The phase vocoder speed is dynamically computed and updated to synchronize the buffer to achieve the desired synchronization point.

```
    let $cpt := 0

whenever($control || $restart) {
    group guard($cpt=0) {
      abort loop1
      loop loop0 2 {
        <actions...>
      }
    }
    group guard($cpt=1) {
      abort loop0
      loop loop1 2 {
        <actions...>
      }
    }
    let $cpt := ($cpt+1) % 2
    let $restart := false
}

whenever($control) {
    abort watchdog
  8 group watchdog {
      let $restart := true
  }
}
```

6.4 Proactive Synchronous Audio Processing

This example is inspired by a musical idea by composer Christopher Trapani, cf. Fig. 5. The goal is to simulate a second musician performing a playback of the live performer but on a different time-frame than the original. The constraints require that the audio playback be synchronous at a predefined point `EndNote`. This procedure, hence, simulates a non-reactive but proactive synchrony where synchronization point is at a later time-position, leading to a dynamic process clock which should be constantly reevaluated to assure synchrony. The audio processing can be achieved by real-time phase vocoding, where the playback speed can be dynamically changed while preserving original audio quality. The theoretical speed between the original phrase and the playback can be computed statically but must be adjusted in real-time to compensate the

fluctuation of the tempo from the musician in order to ensure a strict synchronization on the end point. Moreover, the playback position can not bypass that of the live recording for obvious causal reasons.

```
NOTE 0 1/2
  let $startRec := $NOW
NOTE Ab4 1/5
    ...
NOTE G#5 4/3 ;; starting canon
  let $played := 0
  let $r_rest := EndNote - $RNOW
  let $a_rest := $r_rest*60/$RT_TEMPO
  let $lag := $NOW - $startRec
  let $speed := ($a_rest+$lag)/$a_rest

  superVP speed $speed
  superVP start bang

  whenever($RT_TEMPO) {
    let $played := $played
           + ((@date($speed)-$NOW)*$speed)
    let $lag := ($NOW-$startRec)-$played
    let $r_rest := (EndNote-$RNOW)
    let $a_rest := $r_rest*60/$RT_TEMPO
    let $speed := ($a_rest+$lag)/$a_rest

    superVP speed $speed

  } until ($RNOW ≥ EndNote)

NOTE E5 1/3
  ...
NOTE Eb4 1/2 EndNote ;; end canon
```

This complex musical idea is accomplished by maintaining several variables associated to the current state of the playback process. After the start of the playback and their initialization, these variables are updated at each tempo change where: `$played` represents the audio time-chunk in the buffer already played; `$r_rest` represents the remaining chunk to be recorded in beats; `$a_rest` represent the same quantity but in absolute time which by itself requires an estimated arrival time to `EndNote`. This is estimated based on the current position (in the score) and the current tempo. `$speed` represents the speed of the playback and is used as the control parameter of the phase vocoder superVP. Variable `$lag` is the difference between the progression of the recording and the progression in the reading of the buffer. This quantity represents the interval between the writing head and the reading head in the buffer: the former must always be in front of the latter and they must join together (`$lag` $= 0$) only at the synchronization point.

7 Conclusions

Whereas the importance of *time* is indisputable in both music authoring and performance and real-time critical systems, very few attempts have been made to leverage time as first-class citizen in computer languages bringing humans in the loop of computing and in real-time. In this

paper, we attempted to present the timed-language of *Antescofo* that allows human musicians and computers to perform synchronously together, and allow composers to program sophisticated temporal design patterns common in musical authoring. The *Antescofo* system has been validated in numerous world-class music events involving live electronics and music ensembles such as Berliner Philharmoniker, Los Angeles Philharmonic to name a few; and through collaborations with composers such as Philippe Manoury and Marco Stroppa and Jonathan Harvey.

The *Antescofo* system is the result of a strong coupling between a real-time machine listening module and a reactive engine. This paper focuses on the dynamic temporal semantics of *Antescofo* and its runtime behavior. The formal semantics of the language has been exposed elsewhere in [15] in terms of static Parametric Timed Automata. An *Antescofo* score thus defines both environmental dynamics (for machine listening) and reactive programs. The language allows hierarchical, concurrent and heterogeneous organization of computer processes over time and next to expected events from an external environment. This temporal organization is further enhanced by various notions of time access and triggering in the language constructs for process initiation, synchronization strategies and dynamic variable handling over time. The runtime system allows timely behavior of such processes despite variations of the external environment and provides hybrid event and time triggering of processes in real-time.

We presented several examples of musical *Temporal Design Patterns* using the language, that reflect the diversity of temporal problems addressed by the language, and interactive scenarios where musical time plays an important role. In this paper, we showed how *Antescofo* departs from timed reactive synchronous language families but differentiates in its dynamic and timely behavior. We believe that musical authorship and performance provides an exceptional playground for studies in dynamic temporal semantics and dynamic time systems.

References

[1] C. Alexander. *A pattern language: towns, buildings, construction*, volume 2. Oxford University Press, USA, 1977.

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

[3] D. P. Anderson and R. Kuivila. A system for computer music performance. *ACM Trans. Comput. Syst.*, 8(1):56–82, Feb. 1990.

[4] G. Berry and G. Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Sci. Comput. Program.*, 19(2):87–152, 1992.

[5] R. Boulanger, editor. *The Csound book: perspectives in software synthesis, sound design, signal processing, and programming*. MIT Press, Cambridge, MA, USA, 2000.

[6] A. Cont. Antescofo: Anticipatory synchronization and control of interactive parameters in computer music. In *Proceedings of International Computer Music Conference (ICMC)*. Belfast, August 2008.

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

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

[9] A. Cont. On the creative use of score following and its impact on research. In *Sound and Music Computing*, Padova, Italy, July 2011.

[10] A. Cont, J. Echeveste, J.-L. Giavitto, and F. Jacquemard. Correct Automatic Accompaniment Despite Machine Listening or Human Errors in Antescofo. In *ICMC 2012 - International Computer Music Conference*, Ljubljana, Slovenia, Sept. 2012. IRZU - the Institute for Sonic Arts Research.

[11] R. B. Dannenberg. Arctic: A functional language for real-time control. In *In 1984 ACM Symposium on LISP and Functional Programming*, pages 96–103, 1984.

[12] R. B. Dannenberg. An on-line algorithm for real-time accompaniment. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 193–198, 1984.

[13] R. B. Dannenberg. The implementation of nyquist, a sound synthesis language. *Computer Music J*, 1997.

[14] M. Desainte-Catherine and A. Allombert. Interactive scores: A model for specifying temporal relations between interactive and static events. *Journal of New Music Research*, 34(4):361–374, 2005.

[15] J. Echeveste, F. Jacquemard, A. Cont, and J.-L. Giavitto. Operational semantics of a domain specific language for real time musician-computer interaction. *Discrete Event Dynamic Systems*, 2013. to appear.

[16] A. Ghosal, T. A. Henzinger, C. M. Kirsch, and M. A. A. Sanvido. Event-driven programming with logical execution times. In *Proc. of HSCC 2004, Lecture Notes in Computer Science*, pages 357–371, 2004.

[17] N. Halbwachs. *Synchronous Programming of Reactive Systems.*, volume 1427 of *Lecture Notes in Computer Science*. Springer, 1998.

[18] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.

[19] T. Henzinger, B. Horowitz, and C. Kirsch. Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84 – 99, jan 2003.

[20] H. Kopetz. Event-triggered versus time-triggered real-time systems. In *Proceedings of the International Workshop on Operating Systems of the 90s and Beyond*, volume 563, pages 87–101, Dagstuhl Castle, Germany, July 8-12 1991. Springer.

[21] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112 – 126, jan 2003.

[22] E. Large and M. Jones. The dynamics of attending: How people track time-varying events. *Psychological review*, 106(1):119, 1999.

[23] E. A. Lee. Computing needs time. *Communications of the ACM*, 52(5):70–79, 2009.

[24] P. LeGuernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real-time applications with signal. *Proceedings of the IEEE*, 79(9):1321–1336, 1991.

[25] Z. Liu, Y. Feng, and B. Li. Musicscore: mobile music composition for practice and fun. In *Proceedings of the 20th ACM international conference on Multimedia*, MM '12, pages 109–118, New York, NY, USA, 2012. ACM.

[26] L. Mandel and F. Plateau. Interactive programming of reactive systems. In *Proceedings of Model-driven High-level Programming of Embedded Systems (SLA++P'08)*, Electronic Notes in Computer Science, pages 44–59, Budapest, Hungary, Apr. 2008. Elsevier Science Publishers.

[27] G. Mazzola and O. Zahorka. Tempo curves revisited: Hierarchies of performance fields. *Computer Music Journal*, 18(1):40–52, 1994.

[28] J. McCartney. A new real-time synthesis language. In *International Computer Music Conference*, 1996.

[29] M. Pouzet. Lucid synchrone, version 3. *Tutorial and reference manual. Université Paris-Sud, LRI*, 2006.

[30] M. Puckette. Combining event and signal processing in the max graphical programming environment. *Computer Music Journal*, 15:68–77, 1991.

[31] C. Raphael. The informatics philharmonic. *Commun. ACM*, 54:87–93, March 2011.

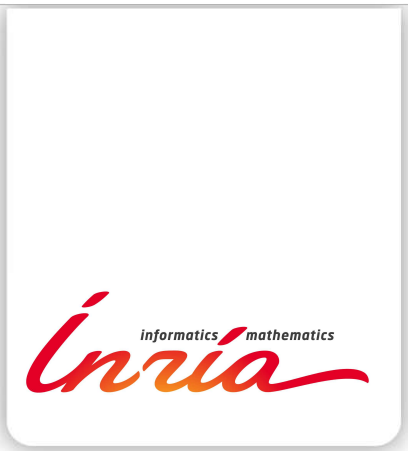
[32] A. Sorensen. Impromptu: An interactive programming environment for composition and performance. In *Proceedings of the Australasian Computer Music Conference 2009*, 2005.

[33] A. Sorensen and H. Gardner. Programming with time: cyber-physical programming with impromptu. In *ACM Sigplan Notices*, volume 45, pages 822–834. ACM, 2010.

[34] B. Vercoe. The synthetic performer in the context of live performance. In *Proceedings of the ICMC*, pages 199–200, 1984.

[35] G. Wang. *The ChucK audio programming language." A strongly-timed and on-the-fly environ/mentality"*. PhD thesis, Princeton University, 2009.

Contents



RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399