The Antescofo Language in ReactiveML

Guillaume Baudart, ENS
Florent Jacquemard, IRCAM
Louis Mandel, Collège de France
Marc Pouzet, ENS

Collège de France
Mixed Music and Antescofo

[Cont 2008]
Mixed Music and Antescofo

[Cont 2008]
Antescofo Architecture

[Cont 2008]

Antescofo

Score

Listening machine

Position

Tempo

Feedback

Sequencer
Antescofo Architecture

[Cont 2008]
Antescofo Architecture

[Cont 2008]

Real-Time Environment

DSP Interface

Discrete Controller

Antescofo

Listening machine

Sequencer

Position

Tempo

Feedback

Focus
Motivations

• Link with the synchronous model
  ▪ An executable semantics for Antescofo
  ▪ Embedding in a synchronous reactive language

• Benefits
  ▪ Live coding
  ▪ Prototyping new features: new attributes, reactive behaviors, ...
I. The Antescofo Language
   - Description
   - Synchronization and error handling strategies

II. Semantics
   - Formalization
   - The three predicates

III. Implementation Architecture
   - Architecture
   - Embedding in ReactiveML

IV. Applications
   - Live Coding
   - New reactive behaviors
The Antescofo Language
The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

Anthèmes II (1994)

New version using antescofo (2008)
The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]
The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]

```
NOTE   65  1.0
0.25   GROUP    tight partial
  {   1.0   'a_11'
      1.0   'a_12' }

CHORD   (68 54)  0.5
1.0    'a_21'
0.5   GROUP    loose causal
  {   1.0    'a_22'
      0.0   GROUP    loose causal
        {   0.25   'a_23'
      0.25    'a_24' }
      1.0    'a_25' }

NOTE   52  2.0
0.5    'a_31'
2.5    'a_32'
```
The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]
The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]
The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]

```
NOTE 65 1.0
0.25 GROUP tight partial
  { 1.0 'a_11'
      1.0 'a_12' }

CHORD (68 54) 0.5
1.0 'a_21'
0.5 GROUP loose causal
  { 1.0 'a_22'
      0.0 GROUP loose causal
       { 0.25 'a_23'
       0.25 'a_24' }
      1.0 'a_25' }

NOTE 52 2.0
0.5 'a_31'
2.5 'a_32'
```

Delay relative to the tempo
The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]
Synchronization Strategies
Loose: Synchronization with the tempo stream.
Loose: Synchronization with the tempo stream.
Tight: Synchronization with tempo and events stream.
Tight: Synchronization with tempo and events stream.
**Tight**: Synchronization with tempo and events stream.
**Tight**: Synchronization with tempo and events stream.

![Illustration of musical notation with actions and events]

- **Group 1**
  - Action 1
  - Action 2
  - Action 3
  - Action 4

Triggering Events:
- Event 1
- Event 4
- Event 5
- Event 6
- Event 7
Tight: Synchronization with tempo and events stream.
**Tight:** Synchronization with tempo and events stream.
Error Handling Strategies
Causal: Actions should be launched immediately when the system recognizes the absence of the triggering event.
**Causal**: Actions should be launched immediately when the system recognizes the absence of the triggering event.
Causal: Actions should be launched immediately when the system recognizes the absence of the triggering event.
Partial: Actions should be dismissed in the absence of the triggering event
Partial: Actions should be dismissed in the absence of the triggering event
Partial: Actions should be dismissed in the absence of the triggering event.
Language Characteristics

• A global logical time relative to the tempo
• Specify electronic actions with:
  o synchronization strategies
  o error handling strategies
• Composer friendly
Semantics
Detected and Missed Event

\[ D: \text{the set of detected instrumental events} \]

For each missed event \( i \) we associate the next detected event

\[ M(i) = \min\{j \in D \mid j > i\} \]
Detected and Missed Event

$D$: the set of detected instrumental events

For each missed event $i$ we associate the next detected event

$$M(i) = \min\{j \in D \mid j > i\}$$
Detected and Missed Event

\( D \): the set of detected instrumental events

For each missed event \( i \) we associate the next detected event

\[ M(i) = \min\{j \in D \mid j > i\} \]
Detected and Missed Event

$D$: the set of detected instrumental events

For each missed event $i$ we associate the next detected event

$$M(i) = \min\{j \in D \mid j > i\}$$
Detected and Missed Event

\[ D: \text{the set of detected instrumental events} \]

For each missed event \( i \) we associate the next detected event

\[ M(i) = \min\{ j \in D \mid j > i \} \]
Detected and Missed Event

\[ D: \text{the set of detected instrumental events} \]

For each missed event \( i \) we associate the next detected event

\[ M(i) = \min\{j \in D \mid j > i\} \]
Detected and Missed Event

$D$: the set of detected instrumental events

For each missed event $i$ we associate the next detected event

$$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$$
Detected and Missed Event

\[ D: \] the set of detected instrumental events

For each missed event \( i \) we associate the next detected event

\[ M(i) = \min\{j \in D \mid j > i\} \]
Detected and Missed Event

\[ D: \text{the set of detected instrumental events} \]

For each missed event \( i \) we associate the next detected event

\[ \mathcal{M}(i) = \min\{j \in D \mid j > i\} \]
Detected and Missed Event

\[ D \]: the set of detected instrumental events

For each missed event \( i \) we associate the next detected event

\[ M(i) = \min\{j \in D \mid j > i\} \]
Formalization

\[
\begin{align*}
\text{score} & ::= \varepsilon \mid (\text{event : seq}) \text{ score} \\
\text{event} & ::= \text{event } i \ t \\
\text{seq} & ::= \varepsilon \mid (\delta \text{ ae}) \text{ seq} \\
\text{ae} & ::= \text{action } \mid \text{ group} \\
\text{group} & ::= \text{group synchro error seq} \\
\text{synchro} & ::= \text{tight } \mid \text{ loose} \\
\text{error} & ::= \text{local } \mid \text{ global } \mid \text{ partial } \mid \text{ causal}
\end{align*}
\]

A performance \( \text{perf} \) is a set of triplets \( (i, \delta, a) \)

\(D\) is the set of detected instrumental events
The Three Predicates

\[
D \quad \frac{\text{exec}}{\text{score} \Rightarrow \text{perf}}
\]
Execute a score

\[
D, i, \delta \quad \frac{\text{detected}}{\text{seq} \Rightarrow \text{perf}}
\]
Execute a sequence of actions bound to a detected event \(i\) with a delay \(\delta\)

\[
D, i, \delta \quad \frac{\text{missed}}{\text{seq} \Rightarrow \text{perf}}
\]
Execute a sequence of actions bound to a missed event \(i\) with a delay \(\delta\)
Execution of a score

(Empty Score) \[ D \xrightarrow{\text{exec}} \varepsilon \Rightarrow \emptyset \]

(Exec Score) \[ D \xrightarrow{\text{exec}} (\text{event } i t : \text{seq}) \Rightarrow p_1 \quad D \xrightarrow{\text{exec}} sc \Rightarrow p_2 \]

\[ D \xrightarrow{\text{exec}} (\text{event } i t : \text{seq}) \ sc \Rightarrow p_1 \cup p_2 \]
(Detect) \[ \begin{align*}
  i \in D & \quad D, i, 0.0 \quad \frac{\text{detected}}{\text{seq} \Rightarrow p} \\
  & \quad D \quad \frac{\text{exec}}{(\text{event } i \ t : \text{seq})} \to p
\end{align*} \]

(Miss) \[ \begin{align*}
  i \notin D & \quad D, i, 0.0 \quad \frac{\text{missed}}{\text{seq} \Rightarrow p} \\
  & \quad D \quad \frac{\text{exec}}{(\text{event } i \ t : \text{seq})} \to p
\end{align*} \]
Execution: Atomic Actions

(Detected Action) \[ D, i, \delta \xrightarrow{\text{detected}} a \rightarrow (i, \delta, a) \]

(Missed Action) \[ M(i) = j \quad \delta' = \max(0.0, \mathcal{E}(i) + \delta - \mathcal{E}(j)) \]
\[ D, i, \delta \xrightarrow{\text{missed}} a \rightarrow (j, \delta', a) \]

\( \mathcal{E}(i) \): date of event \( i \)

\( M(i) = \min\{j \in D \mid j > i\} \)

Error detection: \( i \) is missed
\( j \) is the first detection after \( i \)
**Execution: Atomic Actions**

(Detected Action) \[ D, i, \delta \xrightarrow{\text{detected}} a \rightarrow (i, \delta, a) \]

(Missed Action) \[ D, i, \delta \xrightarrow{\text{missed}} a \rightarrow (i, \delta, a) \]

\[ M(i) = j \quad \delta' = \max(0, E(i) + \delta - E(j)) \]

\[ E(i) : \text{date of event } i \]

\[ M(i) = \min\{j \in D \mid j > i\} \]

Error detection: \( i \) is missed \( j \) is the first detection after \( i \)
Execution: Atomic Actions

(Detected Action) \[ \begin{array}{c}
D, i, \delta \\
detected
\end{array} \quad a \rightarrow (i, \delta, a) \]

(Missed Action) \[ \begin{array}{c}
\mathcal{M}(i) = j \\
\delta' = \max(0, E(i) + \delta - E(j))
\end{array} \quad \begin{array}{c}
\text{missing} \\
D, i, \delta
\end{array} \]

\( \mathcal{E}(i) \): date of event \( i \)

\[ \mathcal{M}(i) = \min\{j \in D \mid j > i\} \]

Error detection: \( i \) is missed
\( j \) is the first detection after \( i \)
Execution: Atomic Actions

(Detected Action) \[ D, i, \delta \overset{\text{detected}}{\rightarrow} a \rightarrow (i, \delta, a) \]

(Missed Action) \[ \mathcal{M}(i) = j \quad \delta' = \max(0.0, \mathcal{E}(i) + \delta - \mathcal{E}(j)) \]
\[ D, i, \delta \overset{\text{missed}}{\rightarrow} a \rightarrow (j, \delta', a) \]

\( \mathcal{E}(i) \): date of event \( i \)

\( \mathcal{M}(i) = \min\{j \in D \mid j > i\} \)

Error detection: \( i \) is missed
\( j \) is the first detection after \( i \)
Implementation
Synchronous Embedding

- Several experiments
  Heptagon, Lucid Synchrone, ReactiveML

- Why ReactiveML?
  - Functional, typed language, on top of OCaml recursion and higher order processes
  - Dynamic features
difficult to get with Lustre/Esterel/…
new interactions, live coding
ReactiveML

OCaml extended with synchronous features à la Esterel
[Mandel-Pouzet 2005]

Process

let process <id> {<pattern>} = <expr>

State machines, executed through several instants.
Simple OCaml functions are considered to be instantaneous.

Basics

Synchronization: pause
Execution: run <expr>

Composition

Sequence: <expr> ; <expr>
Parallelism: <expr> | | <expr>

Signals

Definition: signal <id>
Emission: emit <id>
Waiting: await <id>

Broadcast communication between processes
Architecture

Input treatment

Motor

detected event

exec

missed events

Motor

detected

missed

Time

wait

Listening Machine (DSP)

tempo (bpm)

Audio (DSP)

control

position
Execution of a score

(Empty Score) \( D \overset{\text{exec}}{\Rightarrow} \emptyset \)

(Exec Score) \[
D \overset{\text{exec}}{\Rightarrow} (\text{event } i t : \text{seq}) \rightarrow p_1 \quad D \overset{\text{exec}}{\Rightarrow} sc \Rightarrow p_2
\]

\[
D \overset{\text{exec}}{\Rightarrow} (\text{event } i t : \text{seq}) \, sc \Rightarrow p_1 \cup p_2
\]
Execution of a score

(Empty Score) \[ D \vdash \text{exec} \; \varepsilon \Rightarrow \emptyset \]  
(Exec Score) \[ D \vdash \text{exec} \; (\text{event } it : \text{seq}) \Rightarrow p_1 \quad D \vdash \text{exec} \; sc \Rightarrow p_2 \]  
\[ D \vdash \text{exec} \; (\text{event } it : \text{seq}) \; sc \Rightarrow p_1 \cup p_2 \]

let rec process exec score =  
    match score with  
    | [] -> (* rule (Empty Score) *) ()  
    | se::sc ->  
      (* rule (Exec Score) *)  
      run (exec_score_event_event se) ||  
      run (exec sc)
Execution of a score

(Empty Score) \[ D \xrightarrow{\text{exec}} \varepsilon \Rightarrow \emptyset \]

(Exec Score)

\[
\begin{align*}
D & \xrightarrow{\text{exec}} (\text{event } it: \text{seq}) \rightarrow p_1 \\
D & \xrightarrow{\text{exec}} (\text{event } it: \text{seq}) \text{sc} \Rightarrow p_1 \cup p_2
\end{align*}
\]

let rec process exec score =
match score with
| [] -> (* rule (Empty Score) *) ()
| se::sc ->
  (* rule (Exec Score) *)
  run (exec_score_event_event se) ||
  run (exec sc)

parallel composition
let rec process exec_score_event se =
  let status = run (wait_event se.event) in
  match status with
  | Detected ->
    (* rule (Detect) *)
    run (exec_seq (detected i) 0.0 se.seq)
  | Missed(j) ->
    (* rule (Miss) *)
    run (exec_seq (missed i j) 0.0 se.seq)
Applications
Live Coding

Modify, correct and interact with the score during the performance
Automatic Accompaniment

The house of the rising sun

- Functional programming
  modular definition of the accompaniment

- Reactive programming
  interaction with the score during the performance
Definitions

1. Define the bass line

   \[
   \text{let } \text{bass} = [0.0, (A, \text{Min}); 2.0, (C, \text{Maj}); \ldots]
   \]\n
   \text{val } \text{bass}: \text{(delay } \ast \text{ chord) list}

2. Define the accompaniment style

   \[
   \text{let } \text{arpeggio chord} =
   \]

   \[
   \ldots
   \]\n
   \[
   \text{group Loose Local}
   \]

   \[
   [0.0, \text{action_note (fond)};
   1.0, \text{action_note (third)};
   2.0, \text{action_note (fifth)};]
   \]\n
   \text{val } \text{arpeggio: chord } \rightarrow \text{ asco_event}

3. Link with the performance

   \[
   \text{let process basic_accomp } =
   \]

   \[
   \text{run (link asco 2 roots)}
   \]\n
   \text{val } \text{basic_accomp: unit process}
Interactions

- Kill a process when a signal is emitted
  allow to modify the accompaniment

- Suspend the execution of a process
  pause and resume a process with a signal

- Dynamically change the behavior of a process
  switch between different kinds of accompaniment
Kill a Process

Example of a higher-order process

```ocaml
let process killable k p =
  do
    run p
    until k done
val killable:
  (unit, unit) event -> unit process ->
  unit process
```
Kill a Process

Example of a higher-order process

```
let process killable k p =
  do
    run p
  until k done
val killable:
  (unit, unit) event -> unit process ->
  unit process
```
Kill a Process

Example of a higher-order process

```ocaml
let process killable k p =
do
  run p
until k done
val killable: (unit, unit) event -> unit process -> unit process
```
Dynamic Changes

Example of a recursive higher-order process

```haskell
let process rec replaceable replace p =
  do
    run p
  until replace (q) ->
    run (replaceable replace q)
  done

val replaceable:
  (unit process, unit process) event ->
  unit process -> unit process
```
Dynamic Changes

Example of a recursive higher-order process

```ocaml
let process rec replaceable replace p =
  do
    run p
  until replace (q) ->
    run (replaceable replace q)
  done

val replaceable: (unit process, unit process) event ->
  unit process -> unit process
```

process
Dynamic Changes

Example of a recursive higher-order process

```
let process rec replaceable replace p =
do
  run p
until replace (q) ->
  run (replaceable replace q)
done
val replaceable:
  (unit process, unit process) event ->
  unit process -> unit process
```
Dynamic Changes

Example of a recursive higher-order process

let process rec replaceable replace p =
  do
    run p
  until replace (q) ->
    run (replaceable replace q)
  done

val replaceable:
  (unit process, unit process) event ->
  unit process -> unit process
New Reactive Behaviors

Example: Steve Reich's Piano Phase
Piano Phase ...

Piano Phase,
pour 2 pianos ou 2 marimbass

! = ca. 72

1 \( (x \cdot 8) \)
   r.h.
   \#\#\#\#\#\#\#\#

2 \( (x 12 \cdot 18) \)
   l.h.
   non legato
   \#\#\#\#\#\#\#\#

3 \( (x 16 \cdot 24) \)
   hold tempo 1
   \#\#\#\#\#\#\#\#

4 \( (x 16 \cdot 24) \)
   fade in
   non legato
   \#\#\#\#\#\#\#\#

5 \( (x 16 \cdot 24) \)
   accel very slightly
   hold tempo 1
   \#\#\#\#\#\#\#\#

6 \( (x 16 \cdot 24) \)
   \#\#\#\#\#\#\#\#
   hold tempo 1
   \#\#\#\#\#\#\#\#

Bob
Alice

Steve Reich

40
Piano Phase ...

Synchronization

Bob

Alice
Piano Phase ...

Piano Phase,
pour 2 pianos ou 2 marimbas

\[ \text{\(J = \text{ca. 72}\)} \]

Bob

Alice

Desynchronization
Piano Phase ...

Desynchronization

Bob

Alice
Piano Phase ...

Piano Phase,
pour 2 pianos ou 2 marimbas

\[ \text{\textbullet} = \text{ca. 72} \]

1 \((\times 4, 8)\)

2 \((\times 12, 18)\)

3 \((\times 16, 24)\)

Bob

Alice

Steve Reich
Piano Phase ...

Piano Phase,  
pour 2 pianos ou 2 marimbas

\begin{align*}
\mathcal{J} &= \text{ca. 72} \\
1 \hspace{1cm} (x \cdot 8) &\quad \text{r.h.} \\
2 \hspace{-1cm} (x \cdot 12 \cdot 18) &\quad \text{l.h.} \\
\text{fade in} &\quad \text{non legato} \\
&\quad \text{r.h.} \\
&\quad \text{l.h.} \\
&\quad \text{hold tempo 1} \\
&\quad \text{accel very slightly} \\
&\quad \text{hold tempo 1} \\
&\quad \text{non legato} \\
\end{align*}

\begin{align*}
4 \hspace{-1cm} (x \cdot 16 \cdot 24) &\quad \text{r.h.} \\
5 \hspace{-1cm} (x \cdot 16 \cdot 24) &\quad \text{l.h.} \\
&\quad \text{hold tempo 1} \\
&\quad \text{hold tempo 1} \\
\end{align*}

\begin{align*}
6 \hspace{-1cm} (x \cdot 16 \cdot 24) &\quad \text{r.h.} \\
&\quad \text{hold tempo 1} \\
&\quad \text{hold tempo 1} \\
&\quad \text{hold tempo 1} \\
\end{align*}

Steve Reich

Bob

Alice
Piano Phase ...

Bob

Alice

Piano Phase,
pour 2 pianos ou 2 marimbas

\( \text{\textcopyright ca. 72} \)

1 \((x \cdot 8)\)  r.h.
\(\text{mf, non legato}\)

2 \((x \cdot 12 \cdot 18)\)  r.h.
\(\text{hold tempo} 1\)

3 \((x \cdot 16 \cdot 24)\)  l.h.
\(\text{accel slightly, hold tempo} 1\)

4 \((x \cdot 16 \cdot 24)\)  l.h.
\(\text{hold tempo} 1\)

Steve Reich
Piano Phase ...

Piano Phase, pour 2 pianos ou 2 marimbas

\[ \boxed{\text{Bob}} \]

\[ \boxed{\text{Alice}} \]

\[ \text{Steve Reich} \]
Piano Phase ...

Piano Phase,
pour 2 pianos ou 2 marimbas

\( \text{\textgreek{j}} = \text{ca. 72} \)

1 \((x4 \cdot 8)\)
2 \((x12 \cdot 18)\)
3 \((x16 \cdot 24)\)

Bob
Alice

\( \text{mf non legato} \)
\( \text{r.h.} \)
\( \text{l.h.} \)

\( \text{accel very slightly} \)
\( \text{hold tempo 1} \)

\( \text{fade in} \)
\( \text{non legato} \)

\( \text{(tempo 1)} \)
\( \text{a.v.s} \)

\( \text{hold tempo 1} \)
\( \text{a.v.s} \)
Problem:
We do not want to compute a priori when resynchronizations will occur


... in Mixed Music

**Live musician**
Plays the constant speed part

**Electronic**
Handles the desynchronization

**Synchronization**

**Desynchronization**

*Bob*

*Alice*
... in Mixed Music

**Live musician**
Plays the constant speed part

**Electronic**
Handles the desynchronization

**Synchronization**
Play at the same speed

**Desynchronizatio**

Bob

Alice

*Listening Machine*

- Tempo
- Position
... in Mixed Music

Live musician
Plays the constant speed part

Electronic
Handles the desynchronization

Bob

Listening Machine
Tempo
Position

Synchronization
Play at the same speed

Desynchronization

- Play slightly faster
- Track the first note of Bob
- Resynchronize when the k-th note of Alice is close enough of the first note of Bob
Implementation

Two phases:

Synchronization

Desynchronization

```
let piano_phase sync desync first_note kth_note =
  let rec process piano_phase k =
    let ev = last_event asco in
    run (melody ev 4 0.25 first_note);
    emit desync;
    do
      let ev = last_event asco in
      run (melody (ev+1) 16 0.2458 first_note) ||
      run (track asco k kth_note) ||
      run (compare asco first_note kth_note sync 0.05)
    until sync done;
    run (piano_phase ((k + 1) mod 12))
  in
  piano_phase 1
```
Implementation

**Synchronization**

Play the melody four times and follow the tempo

Emit the signal `desync` after four iterations of the melody

```plaintext
let piano_phase sync desync first_note kth_note =
let rec process piano_phase k =
    let ev = last_event asco in
    run (melody ev 4 0.25 first_note);
    emit desync;
    do
        let ev = last_event asco in
        run (melody (ev+1) 16 0.2458 first_note) ||
        run (track asco k kth_note) ||
        run (compare asco first_note kth_note sync 0.05)
    until sync done;
    run (piano_phase ((k + 1) mod 12))
in
in piano_phase 1
```
Desynchronization

Play slightly faster and emit the signal `first_note` whenever the first note is played.

Track the k-th note of the musician.

Compare the emission of signals `kth_note` and `first_note` and emit `sync` when they are close enough.
Conclusion

• **Link with the synchronous model** [EMSOFT 2013]
  - An executable semantics for Antescofo
  - A sequencer efficient enough w.r.t. human ear
  - Embedding in a synchronous reactive language

• **Applications** [FARM 2013]
  - Live coding
  - Prototyping new features:
    new attributes, reactive behaviors, ...
To continue...

http://reactiveml.org/reactive_asco
References


