A Synchronous Embedding of Antescofo, a Domain-Specific Language for Interactive Mixed Music

Guillaume Baudart, ENS
Florent Jacquemard, IRCAM
Louis Mandel, ENS
Marc Pouzet, ENS

EMSOFT’13
Montreal 09.30.2013
Mixed Music and Antescofo

[Cont 2008]
Mixed Music and Antescofo

[Cont 2008]
Antescofo Architecture

[Cont 2008]
Antescofo Architecture

Real-Time Environment -> DSP Interface

Antescofo

Listening machine

Position

Feedback

Sequencer

Discrete Controller
Antescofo Architecture

[Cont 2008]

This paper
I. The Antescofo Language

○ Description

○ Synchronization and error handling strategies

II. Semantics

○ Formalization

○ The three predicates

III. Implementation

○ Architecture

○ Embedding in ReactiveML
The Antescofo Language
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'
```
The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]
The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Note: Antescofo score notation]

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'

Electronic Score
The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Chavez 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

[Note: 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'
```

Group Attributes
Language Characteristics

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

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

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

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

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

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

\( \mathcal{E}(i) \): date of event \( i \)
Detected and Missed Event

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

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

For each missed event \( i \) we associate the next detected event.
Detected and Missed Event

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

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

$\mathcal{E}(i)$: date of event $i$
For each missed event $i$ we associate the next detected event

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

$\mathcal{E}(i)$: date of event $i$
Detected and Missed Event

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

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

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

**Diagram:**
- Listening machine to Input Treatment
- \( \mathcal{E}(i) \) as the date of event \( i \)
- \( M(i) \) as the next detected event
Detected and Missed Event

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

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

\( \mathcal{E}(i) \): date of event \( i \)
Detected and Missed Event

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

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

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

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

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

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

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

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

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

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

\[ t \in \mathbb{Q} \quad \text{Duration} \]
\[ \delta \in \mathbb{Q} \quad \text{Delay} \]
\[ i \in \mathbb{N} \quad \text{Label} \]
\[ a \in \mathcal{A} \quad \text{Action} \]

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

\[ D \text{ is the set of detected instrumental event} \]

**Semantics**

\[ D \xrightarrow{\text{exec}} \text{score} \Rightarrow \text{perf} \]
The Three Predicates

\[ D \xrightarrow{exec} sc \Rightarrow p \]

Execute a score

\[ D, i, \delta \xrightarrow{detected} seq \Rightarrow p \]

Execute a sequence of actions bound to a detected event \( i \) with a delay \( \delta \)

\[ D, i, \delta \xrightarrow{missed} seq \Rightarrow p \]

Execute a sequence of actions bound to a missed event \( i \) with a delay \( \delta \)
Execution of a score

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

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

\[ D \xrightarrow{\text{exec}} (\text{event } i t : \text{seq}) \Rightarrow p_1 \cup p_2 \]
Triggering

(Detect) \[ i \in D \quad D, i, 0.0 \quad \frac{\text{detected}}{\text{seq} \Rightarrow p} \]

\[ D \quad \frac{\text{exec}}{(\text{event } i \ t : \text{seq})} \rightarrow p \]

(Miss) \[ i \notin D \quad D, i, 0.0 \quad \frac{\text{missed}}{\text{seq} \Rightarrow p} \]

\[ D \quad \frac{\text{exec}}{(\text{event } i \ t : \text{seq})} \rightarrow p \]
**Execution: Atomic Actions**

(Detected Action)

\[ D, i, \delta \quad \frac{\text{detected}}{\Rightarrow} \quad a \rightarrow (i, \delta, a) \]

(Missed Action)

\[ D, i, \delta \quad \frac{\text{missed}}{\Rightarrow} \quad a \rightarrow (j, \max(0.0, \mathcal{E}(i) + \delta - \mathcal{E}(j)), a) \]

\[ \mathcal{M}(i) = j \]

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

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

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

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

(Missed Action) \[ D, i, \delta \overset{\text{missed}}{\longrightarrow} a \rightarrow (j, \max(0.0, \mathcal{E}(i))) \]

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

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

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) \[ D, i, \delta \xrightarrow{\text{missed}} a \rightarrow (j, \max(0.0, \mathcal{E}(i))) \]

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

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

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

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

(Missed Action) \[ D, i, \delta \quad \text{missed} \quad a \rightarrow (j, \max(0.0, \mathcal{E}(i))) \]

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

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

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)

\[
D, i, \delta \xrightarrow{\text{missed}} a \rightarrow (j, \max(0.0, E(i) + \delta - E(j)), a)
\]

\(E(i)\): Position of event \(i\)

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

Error detection: \(i\) is missed
\(j\) is the first detection after \(i\)
Implementation
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
Why ReactiveML?

- A synchronous language expressiveness for time and events
- Functional, typed language, on top of OCaml recursion and higher order processes
- Efficient implementation no busy waiting
- Dynamical features new interactions, live coding
Architecture

Listening Machine (DSP) → Motor
  - detected event → exec
  - missed events → exec

Motor
  - exec → detected
  - exec → missed

Motor
  - detected → exec
  - missed → exec

Time
  - tempo (bpm) → wait

Audio (DSP) → exec
  - control
Execution of a score

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

let rec process exec score =
  match score with
  | [] -> (* rule (Empty Score) *) ()
  | se::sc ->
    (* rule (Exec Score) *)
    run (exec_score_event se) ||
    run (exec sc)
(Detect) \[ \begin{align*} i \in D & \quad D, i, 0.0 \quad |_{\text{detected}} \quad \text{seq} \Rightarrow p \\ D & \quad |_{\text{exec}} \quad (\text{event } i \ t : \text{seq}) \rightarrow p \end{align*} \]

(Miss) \[ \begin{align*} i \notin D & \quad D, i, 0.0 \quad |_{\text{missed}} \quad \text{seq} \Rightarrow p \\ D & \quad |_{\text{exec}} \quad (\text{event } i \ t : \text{seq}) \ \text{sc} \rightarrow p \end{align*} \]
Triggering

\[
\begin{align*}
(Detect) & \quad \text{if } i \in D, \text{ then } D, i, 0.0 \quad \text{detected} \quad \text{seq} \Rightarrow p \quad \text{exec} \quad (\text{event } i \text{ t : seq}) \rightarrow p \\
(Miss) & \quad \text{if } i \notin D, \text{ then } D, i, 0.0 \quad \text{missed} \quad \text{seq} \Rightarrow p \quad \text{exec} \quad (\text{event } i \text{ t : seq}) \text{ sc} \rightarrow p
\end{align*}
\]

let rec process exec_score_event se =
    let i = se.event in
    await events.(i)(status) in
    match status with
    | Detected ->
      (* rule (Detect) *)
      run (exec_seq (detect i) 0.0 se.seq)
    | Missed(j) ->
      (* rule (Miss) *)
      run (exec_seq (missed i j) 0.0 se.seq)
• Contributions
  
  ○ a new semantics for Antescofo

  ○ a sequencer efficient enough to compare well with the actual one

  ○ prototyping new features:
    new attributes, reactive behaviors, live coding, ...

• Next?

  ○ interaction with other system: gesture follower, voice recognition, ...

  ○ link with synthesis tool or other media
To continue...

www.reactiveml.org/emsoft13
References


From Logical Time to Physical Time
From Logical Time to Physical Time
From Logical Time to Physical Time

let process tick period clock =
  let next = ref (Unix.gettimeofday () +. period) in
  loop
    let current = Unix.gettimeofday () in
    if (current >= !next)
      then (emit clock (); next := !next +. period);
      pause;
    end

val tick : float -> (unit, 'a) event -> unit process
Wait!

let process wait dur period clock =
  let d = int_of_float (dur /. period) in
  do
    for i=1 to d do pause done
  when clock done

val wait : float -> float -> ('a, 'b) event -> unit process
and process detected i delta ae =
    match ae with
     | Action(a) ->
     (* rule (Detected Action) *)
     run (wait date delta);
     emit perf (i,delta,a)
     | Group(g) ->
     begin match g.group_synchro with
     | Loose ->
     (* rule (Detected Loose Group) *)
     let bg = g.group_seq in
     run (exec_seq (detected i) delta bg)
     | Tight ->
     (* rule (Detected Tight Group) *)
     let gs = slice i delta g in
     run (exec gs)
     end