Mixed Music and Antescofo

[Cont 2008]
Mixed Music and Antescofo

[Cont 2008]
Antescofo Architecture

[Cont 2008]
Antescofo Architecture

[Cont 2008]
The score is a specification of a musical reactive system
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]

- Time is relative to the tempo
- Electronic actions are characterized by a delay
- Hierarchical structure: groups and nested groups
- Synchronization with the musician: tight, loose
- Error handling strategies: partial, causal
The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Écheveste et al. 2012]

- Time is relative to the tempo
- Electronic actions are characterized by a delay
- Hierarchical structure: groups and nested groups
- Synchronization with the musician: tight, loose
- Error handling strategies: partial, causal

Link with synchronous languages?
ReactiveML

The temporal expressiveness of synchronous languages with the power of functional programming
ReactiveML
[Mandel-Pouzet 2005]

OCaml
- Data structures
- Control structures

Synchronous model of concurrency
- A global logical time
- Parallel composition
- Communication between processes
The Synchronous Hypothesis

- Inputs
- Treatment
- Outputs

Real-time

Logical Time

0 1 2 3
The Language

Process

\[ \text{let process } \langle \text{id} \rangle \ \{ \langle \text{pattern} \rangle \} = \langle \text{expr} \rangle \]

*State machines, executed through several instants.*

*Simple OCaml functions are considered to be instantaneous.*

Basics

Synchronization: \texttt{pause}

Execution: \texttt{run } \langle \text{expr} \rangle

Composition

Sequence: \langle \text{expr} \rangle \ ; \ \langle \text{expr} \rangle

Parallelism: \langle \text{expr} \rangle \ | \ | \ \langle \text{expr} \rangle

Signals

Definition: \texttt{signal } \langle \text{id} \rangle

Emission: \texttt{emit } \langle \text{id} \rangle

Waiting: \texttt{await } \langle \text{id} \rangle

Broadcast communication between processes
First Example

Wait in parallel for the emission of two signals

```
let process simple a b =
  (await a; print "a")
||
  (await b; print "b")
val simple:
  (unit, unit) event -> (unit, unit) event ->
  unit process
```
First Example

Wait in parallel for the emission of two signals

```
let process simple a b =
    (await a; print "a") ||
    (await b; print "b")
val simple : (unit, unit) event -> (unit, unit) event -> unit process
```
First Example

Wait in parallel for the emission of two signals

```ocaml
let process simple (a, b) =
  (await a; print "a") ||
  (await b; print "b")
val simple : (unit, unit) event -> (unit, unit) event ->
  unit process
```
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

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

2. Define the accompaniment style

   \[
   \text{let} \quad \text{arpeggio} \text{ chord} = \\
   \ldots \\
   \text{group} \quad \text{Loose Local} \\
   [0.0, \text{action_note} (\text{fond}); \\
    1.0, \text{action_note} (\text{third}); \\
    2.0, \text{action_note} (\text{fifth});\ldots] \\
   \text{val} \quad \text{arpeggio: chord -> asco_event}
   \]

3. Link with the performance

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

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

- **Suspend a 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

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

let process killable \( k \) \( p \) =
  do
    run \( p \)
  until \( k \) done

val killable :
  (unit, unit) event \to\ unit process \to\ 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
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
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 marimbas

\[ \begin{align*}
    \text{1 (x 4 \times 8)} & & \text{2 (x 12 \times 18)} & & \text{3 (x 16 \times 24)} \\
    \text{r.h.} & & \text{r.h.} & & \text{hold tempo 1} \\
    \text{mf non legato} & & \text{accel very slightly} & & \text{hold tempo 1} \\
    \text{fade in} & & \text{non legato} & & \text{a.v.s} \\
\end{align*} \]

\[ \begin{align*}
    \text{4 (x 16 \times 24)} & & \text{5 (x 16 \times 24)} & & \text{6 (x 16 \times 24)} \\
    \text{(tempo 1)} & & \text{(tempo 1)} & & \text{(tempo 1)} \\
    \text{hold tempo 1} & & \text{hold tempo 1} & & \text{hold tempo 1} \\
    \text{a.v.s} & & \text{a.v.s} & & \text{a.v.s} \\
\end{align*} \]
Piano Phase ...

Bob

Alice

Synchronization
Piano Phase ...

Desynchronization
Piano Phase ...

Desynchronization
Piano Phase ...

Bob

Alice
Piano Phase ...

Piano Phase,
pour 2 pianos ou 2 marimbas

\[ \text{\textit{\textbf{Alice}}} \]
\[ \text{\textit{\textbf{Bob}}} \]

Steve Reich

Bob

Alice
Piano Phase ...

Alice

Bob
Piano Phase ...

Piano Phase,
pour 2 pianos ou 2 marimbas

\( \text{\( \mathbf{\text{\( x 4 \)}}\ \mathbf{\text{\( x 4 \)}}\) ca. 72} \)

1 \( (x 4 \cdot 8) \)

\( \text{\r.h.} \)

\( \text{\l.h.} \)

mf non legato

2 \( (x 12 \cdot 18) \)

\( \text{\r.h.} \)

\( \text{\l.h.} \)

fade in

\( \text{\r.h.} \)

\( \text{\l.h.} \)

non legato

3 \( (x 16 \cdot 24) \)

\( \text{\r.h.} \)

\( \text{\l.h.} \)

hold tempo 1

accel very slightly

4 \( (x 16 \cdot 24) \)

\( \text{\r.h.} \)

\( \text{\l.h.} \)

hold tempo 1

5 \( (x 4 \cdot 16) \)

\( \text{\r.h.} \)

\( \text{\l.h.} \)

hold tempo 1

6 \( (x 16 \cdot 24) \)

\( \text{\r.h.} \)

\( \text{\l.h.} \)

hold tempo 1

Steve Reich

Bob

Alice
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

*Bob*

*Alice*

---

**Listening Machine**

- Tempo
- Position

---

**Synchronization**

**Desynchronization**
... in Mixed Music

**Live musician**
Plays the constant speed part

**Electronic**
Handles the desynchronization

![Diagram](image)

**Bob**

**Listening Machine**
- Tempo
- Position

**Synchronization**
Play at the same speed

**Desynchronization**

**Alice**
... in Mixed Music

**Live musician**
Plays the constant speed part

**Electronic**
Handles the desynchronization

- Play at the same speed

**Listening Machine**
- Tempo
- Position

**Synchronization**

**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

Bob
Implementation

Two phases:
Synchronization
Desynchronization

```haskell
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
in
```
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
        piano_phase 1
    in
```
Implementation

**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.
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
  dynamical creation of processes
In Practice

• Embedding the Antescofo language
  new implementation of the sequencer
  using the actual antescofo listening machine

• Extend the Antescofo language
  functional and reactive programming

• A tool for prototyping new features
  reactive behaviors, live coding, new attributes

• Link with other media
  graphical interface, top-level, ...
References


