Antescofo
Synchronous Interpretations of a Language for Mixed Music

Supervisors
F. Jacquemard : MuSync, INRIA
M. Pouzet : PARKAS, ENS

Guillaume BAUDART
SYNCHRON’12
Objectives

- Study and formalization of a language for mixed music: the language of Antescofo
- Link with synchronous languages
Antescofo
[Cont 2010]

- Score Follower: Position, Tempo
- Reactive Sequencer: Automatic accompaniment
Mixed Music

Performance/Real Time

Composition/Program

OTEMO (1st section)
Antescofo Architecture

Listening Machine

Max/MSP

Control Messages

Sequencer

Partition antescofo

signal

bpm

event

Pre-treatment

Real Time Communication
Antescofo Architecture

Partition antescofo

Listening Machine

Max/MSP

Sequencer

Control Messages

signal

bpm

event

Reactive System!

Partition antescofo

Pre-treatment

Real Time Communication
The Language of Antescofo

Goal: Jointly specify electronic and instrumental parts

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'
0.5 'a_31'
2.5 'a_32'
The Language of Antescofo

Goal: Jointly specify electronic and instrumental parts

Instrumental Score

```
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 Language of Antescofo

Goal: Jointly specify electronic and instrumental parts

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 Language of Antescofo

Goal: Jointly specify electronic and instrumental parts

```
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 Language of Antescofo

Goal: Jointly specify electronic and instrumental parts

```
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'
```
Language Objectives

• Specify a ‘critical’ reactive system
• Time programming
• Real time synchronization
• Error handling strategies
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.
Error Handling Strategies
Causal: Actions should be launch immediately when the system recognize 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 launch immediately when the system recognize 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.
Model

\[ a ::= A(\delta, m) \]
\[ g ::= G(\delta, \text{sync}, \text{err}, \text{ae}^*) \]
\[ \text{ae} ::= a|g \]
\[ \text{sync} ::= T|L \]
\[ \text{err} ::= P|C \]
\[ \delta ::= \text{delay} \]
\[ m ::= \text{message} \]
\[ i ::= \text{int} \]
\[ \text{se} ::= (i : \text{ae}) \]
\[ \text{sc} ::= \text{se}^* \]
\[ p ::= (i, \delta, m)^* \]
Model

\[ a ::= A(\delta, m) \]
\[ g ::= G(\delta, sync, err, ae^*) \]
\[ ae ::= a|g \]
\[ sync ::= T|L \]
\[ err ::= P|C \]
\[ \delta ::= delay \]
\[ m ::= message \]
\[ i ::= int \]
\[ se ::= (i : ae) \]
\[ sc ::= se^* \]
\[ p ::= (i, \delta, m)^* \]
Model

\[ a ::= A(\delta, m) \]
\[ g ::= G(\delta, sync, err, ae^*) \]
\[ ae ::= a | g \]
\[ sync ::= T | L \]
\[ err ::= P | C \]
\[ \delta ::= delay \]
\[ m ::= message \]
\[ i ::= int \]
\[ se ::= (i : ae) \]
\[ sc ::= se^* \]
\[ p ::= (i, \delta, m)^* \]

**Objective**

\[ D, M, sc \mid_{\text{exec}} p \]

**Dating** \( \mathcal{E} : \text{int} \rightarrow \text{delay} \)

**Detectors** \( D : \text{int}^* \)

**Errors** \( M : \text{int} \rightarrow \text{int} \)
Execution Rules

\[ D, M, sc \vdash_{\text{exec}} p \quad \text{ Execute a score} \]

\[ D, M, sc \vdash_{\text{detect}}^{\delta, i} a \rightarrow p \quad \text{ Execute an action } a \text{ attached to a detected event } i \text{ with a delay } \delta \]

\[ D, M, sc \vdash_{\text{missed}}^{\delta, i} a \rightarrow p \quad \text{ Execute an action } a \text{ attached to a missed event } i \text{ with a delay } \delta \]
Execution

**Empty Score**

\[ D, M, \emptyset \mid^{\text{exec}} \emptyset \]

**Detected**

\[ i \in D \]
\[ D, M, sc \mid^{\text{detect}}_{0, i} ae \rightarrow p_1 \]
\[ D, M, sc \mid p_2 \]
\[ D, M, sc \cup (i : ae) \mid^{\text{exec}} p_1 \cup p_2 \]

**Missed**

\[ i \not\in D \]
\[ D, M, sc \mid^{\text{missed}}_{0, i} ae \rightarrow p_1 \]
\[ D, M, sc \mid p_2 \]
\[ D, M, sc \cup (i : ae) \mid^{\text{exec}} p_1 \cup p_2 \]
**Execution**

**Detected Action**

\[
D, M, sc \vdash_{\text{detect}_{\delta, i}} A(\delta_a, m) \rightarrow \{(i, \delta + \delta_a, m)\}
\]

**Missed Action**

\[
M(i) = j
\]

\[
D, M, sc \vdash_{\text{missed}_{\delta, i}} A(\delta_a, m) \rightarrow \{j, \max(0, \delta + \delta_a + \varepsilon(i) - \varepsilon(j)), m\}
\]
An Interpret in Reactive ML
Reactive ML

[Mandel 2005]

• A Reactive extension to OCaml
• Synchronous constructs (|| ;)
• Communication between processes via signals
Logical Time

let process wait_abs dur =
    let d = int_of_float (dur /. clock_step) in
    for i=d downto 1 do pause;

• Wait a duration in millisecond

• Compilation in sampling mode
  (clock_step : argument for the compiler)
let process wait_abs dur =
    let d = int_of_float (dur /. clock_step) in
    for i=d downto 1 do pause;

let process ab d1 d2 =
    (run (wait_abs d1); print a)
 ||
    (run (wait_abs d2); print b)
 Signals

```plaintext
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;
```
Signals

signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;

let process follow ext_event event missed_event =
  loop
  await ext_event (ev) in
  let pre_event = last ?event in
  emit event ev;
  emit missed_event (nat (p_event+1) (event-1));
  end
Signals

```plaintext
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;

let process follow ext_event event missed_event =
  loop
    await ext_event (ev) in
    let pre_event = last ?event in
    emit event ev;
    emit missed_event (nat (p_event+1) (event-1));
  end
```
Signals

```plaintext
signals

signal ext_event default 0 gather fun x y -> x ;;
signal event default 0 gather fun x y -> x ;;
signal missed_event default [] gather fun x y -> x ;;

let process follow ext_event event missed_event =
    loop
        await ext_event (ev) in
        let pre_event = last ?event in
        emit event ev;
        emit missed_event (nat (p_event+1) (event-1));
    end
```
Signals

```plaintext
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;

let process follow ext_event event missed_event =
  loop
  await ext_event (ev) in
  let pre_event = last ?event in
  emit event ev;
  emit missed_event (nat (p_event+1) (event-1));
  end
```

Await a reception
Signals

```plaintext
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;

let process follow ext_event event missed_event =
  loop
    await ext_event (ev) in
    let pre_event = last ?event in
    emit event ev;
    emit missed_event (nat (p_event+1) (event-1));
  end
```

Emit on a signal
Signals

```plaintext
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;

let process follow ext_event event missed_event =
  loop
    await ext_event (ev) in
    let pre_event = last ?event in
    emit event ev;
    emit missed_event (nat (p_event+1) (event-1));
  end
```

Last received value
Signals

```plaintext
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;

let process follow ext_event event missed_event =
  loop
    await ext_event (ev) in
    let pre_event = last ?event in
    emit event ev;
    emit missed_event (nat (p_event+1) (event-1));
```
Why Reactive ML?

- Time programming expressivity
- High level programming
- Recursivity: Hierarchical structure
- Lazy waiting: tempo changes
Architecture

UDP Interface \rightarrow \text{Follow} \rightarrow \text{Motor} \rightarrow \text{UDP Interface}

- ext\_event
- missed\_event
- event
- detected
- missed
- control
- bpm
Types

(** Basic action *)

```haskell
(type action =
   { action_delay : delay;
     action_body : message; }
)

(** Asco group *)

```haskell
(type group =
   { group_delay : delay;
     group_synchro : sync;
     group_error : err;
     group_body : asco_event list; }
)

(** Generic asco event *)

```haskell
and asco_event =
   Group of group
   | Action of action
```
Execution

let rec process exec ae ev =
  (await event (e) when (ev = e) in
   run detect ae)
 ||
  (await missed_event (me) when (List.mem ev me) in
   run missed ae)

and process detect ae = ...

and process missed ae = ...

• Three mutually recursive processus
• Dynamical construction of the processus tree
Score example
Score example
Score example
Score example

exec_score

exec g1 e1 exec a21 e2 exec g2 e2 exec a41 e4

exec_detected g1 exec_missed a21 exec_missed g2 exec_detected a41

exec a11 e1 exec a12 e2 exec_detected a22 exec_detected a23

exec_detected a11 exec_missed a12
Summary

- Direct translation of the model
- Implementation close to the specifications
- Powerful tool for prototyping
An Interpret in Heptagon
Heptagon

- A Lustre/Scade like language
- Clock inference
- With advanced constructions (automata...)
Why Heptagon?

- Explore another point of view
- Generate C: Efficiency
- No dynamic allocation
- More guarantees on the runtime behavior (concert is a ‘critical situation’)

concert is a 'critical situation'
Normalization

- Propagation of error handling attributes
- Actions repartition: triggering events
- Flattening: no more groups
Example
Example

Propagation
Example

Repartition
Example

Flattening
Example

Track

a11 partial

a12 partial

a21 partial

a22 causal

a23 causal

a41 partial
Score: Normalized and stored in a static structure.
Execution

```plaintext
node exec(tr: track) returns (o: bool)
var td, tm: bool;
let
  automaton
  state Wait
    do o = false;
    unless td then Detect
    unless tm then Missed
  state Detect
    do o = send(bpm, tr);
  state Missed
    do o = send(bpm, update(tr));
end;
  td = testDetect(getEvent(tr), event);
  tm = testMissed(getEvent(tr), missed_event);
tel
```
Conclusion

- A Semantics for Antescofo
- Two synchronous points of view
- Two Antescofo Interprets
Perspectives

• Prototyping new features for Antescofo
  • New synchronization and error handling strategies
  • Variables, loops, etc...

• The Heptagon experiment
  • Prove some properties on the score
  • Performance could enable audio synthesis