Switch to full screen

Playing with the idea of Aspect Programming for Reactive Systems

Verimag : Karine Altisen, Florence Maraninchi, David Stauch
Inria RA : Eric Rutten, Pascal Fradet
INRIA local coordinate research action "ctrl-a"

Contents

- Some words on "Aspect-Oriented Programming"
- Aspects for reactive systems
- A tentative formalization
  - The operational approach (FM)
  - The declarative approach (KA)
- Time for comments, questions, ...

Aspect Programming

--- Take a programming language L
with a well-defined structure

--- Take a program P written in L

--- Think of something more or different P should do, such that it is not possible while "respecting the structure of P"

Aspect Programming

Whenever you need a transverse modification of (almost) all components :

- This "new" functionality is an aspect (w.r.t. the structure of L)
- Or, you're a very bad programmer !

For any structuring mechanism, there will always be cases in which program evolutions need transverse modifications.

Questions

Very informal

- Defining structures ?
- Defining additional functionalities ?

Quite ambiguous

- How much does it depend on the language ?
- How much does it depend on the program ?

Examples ?

Implementations ?

Example of need: a trace mechanism

Add code such that every variable change is reported as an output.

AspectJ

A programming environment for aspects in Java.

--- insert code before, after or instead the original body code of methods

--- designating methods by lexical rules :
example : all the methods whose name is of the form
`set* (...)`

AspectJ

```
class C {
   private int a ;
   public void m (...) {
        //// place where aspect code can be inserted
        // some original code
        //// place where aspect code can be inserted
   }
   ...
}
```

Comments

Very powerful (or "expressive")
(can transform a compiler into a flight simulator)

Difficult to understand the effect of aspects, the order in which they should be applied, ...

How to find a set of such elementary transformations, starting from a more high-level idea of the aspect ?

Weaving tool = pre-processor (on the Java source, or on the bytecode)

Related Work

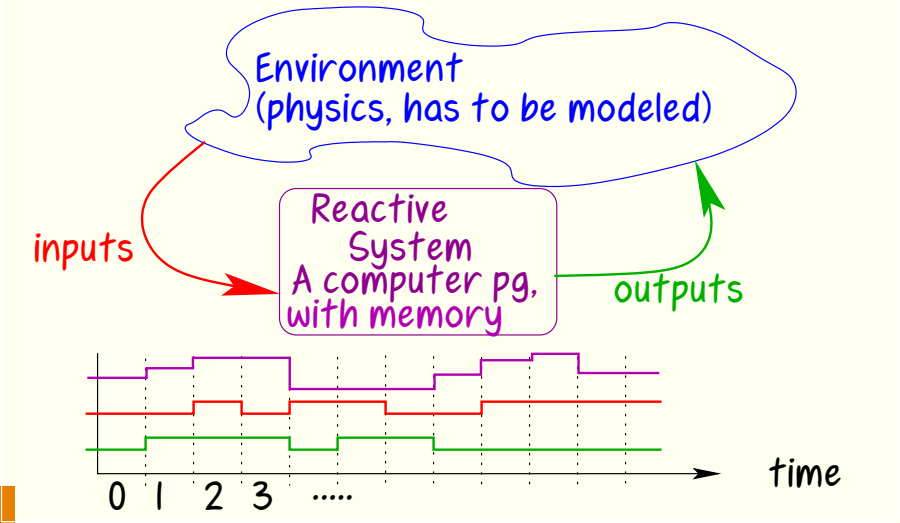
Enforcing safety properties of a (reactive) program.

1) by dynamic monitoring (F. Schneider)
2) by static analysis and program transformations (P. Fradet)

Contents

- Some words on "Aspect-Oriented Programming"
- Aspects for reactive systems
- A tentative formalization
  - The operational approach (FM)
  - The declarative approach (KA)
- Time for comments, questions, ...

Reactive Systems in the Synchronous Approach



Trace Semantics

The semantics of a program is a set of input/output traces.

Reactive Systems in the synchronous Approach

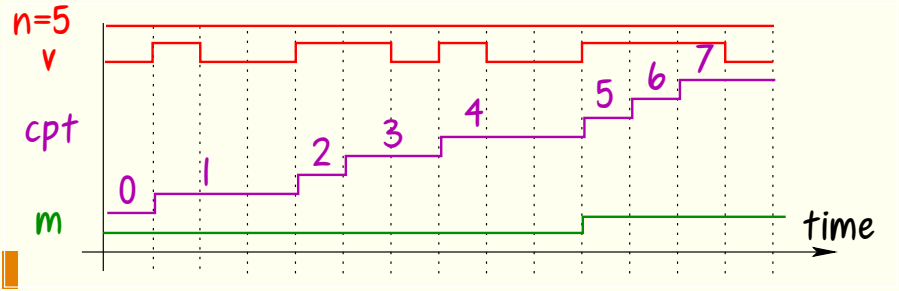
Languages:

- Dataflow style (Lustre, Signal, ...)
- Imperative style with control structures (Esterel)
- Imperative style with explicit automata (Argos, Sync-Charts, SSMs, ...)

Same structuring mechanism : parallel components communicating (and synchronizing) with the synchronous broadcast.

An Example Reactive Program

```
node maintain(n:  int; val:  bool) returns (m:  bool);
var cpt :  int ;
let   cpt = if val then (0 -> pre(cpt)) + 1 else 0 ;
      m = (cpt >= n) ;
tel
```



Aspects of reactive systems

- Reinitialization is an aspect in Lustre, not in Esterel
- Something more semantic : adding validity bits...

Re-initialization (Lustre)

```
node P (i : int)
returns (o : int) ;
let
   o = i -> pre(o)
            + 42 ;
tel.
```

```
node P  (i : int ;
         r : bool)
returns (o : int) ;
let
   o = i -> if r
            then i
            else pre(o)
                 + 42 ;
tel.
```

Transformation to be applied everywhere

Re-initialization (Esterel)

```
input  i : integer ;
output o : integer ;

P
```

```
input  i : integer ;
input  R : boolean ;
output o : integer ;

loop
   P
each R ;
```

The transformation is modular, because the language construct already exists !

Adding validity bits to inputs

Consider a program :

```
node P (i : integer) returns (o : integer) ...
```

We'd like to add a validity bit `v` to the input `i`.
The output `o` is the same as in P when `v` is true,
and copies its previous value when `v` is false.

Is this an aspect ?

```
node PP (i : integer ; v : boolean)
returns (o : integer)
   oo : integer ;
let
   oo = P (i) ;
   o  = if v then oo
              else ... -> pre(o) ;
tel.
```

Apparently not (P is not modified)...

but...

We want to copy the previous value of the output when the validity bit is false, but only if the output depends on the input !

Additional problem : define "`o` depends on `i` (now)".

Adding the validity bit will probably need a modification in P.

Example

```
node P (i : integer) returns (o : integer)
   toggle : boolean ;
let
   toggle = true -> not pre (toggle) ;
   o = 42  ->  if toggle then i+1
                         else 212 ;
tel.
```

Example

```
node P   (i : integer ;
          v : boolean)
returns  (o : integer)
   toggle : boolean ;
let
   toggle = true -> not pre (toggle) ;
   o = 42  ->  if toggle then
                              if v then  i+1
                                   else  pre(o)
                         else 212 ;
tel.
```

Summary

Aspects of reactive systems :

- May depend heavily on the syntactic constructs
- A lot of things can already be done by adding parallel components and a bit of synchronization.
  What remains?

Contents

- Some words on "Aspect-Oriented Programming"
- Aspects for reactive systems
- A tentative formalization
  - General setting
  - The declarative approach (KA)
  - The operational approach (FM)
- Time for comments, questions, ...

General setting

Program P :
inputs I ∪ I' and outputs O ∪ O' flows, trace semantics.

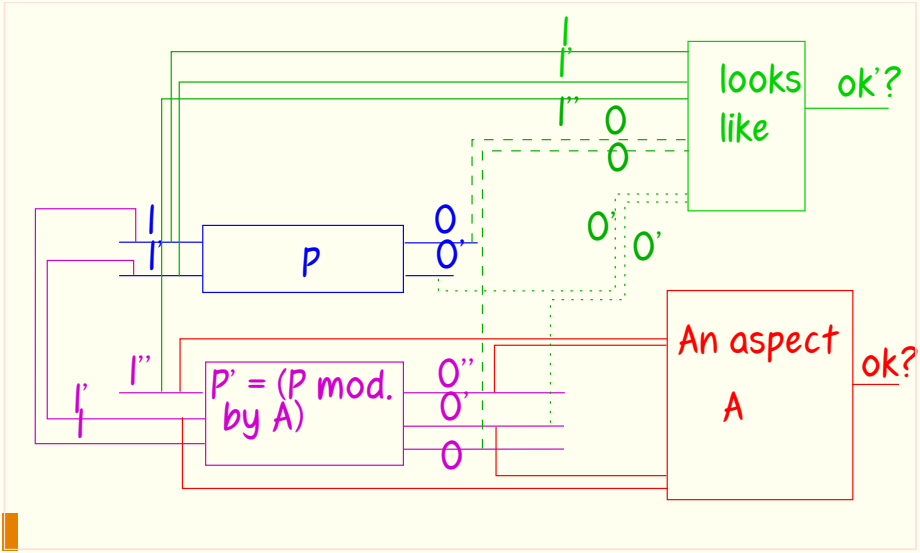
Aspect A :

- Add inputs I'' and outputs O'' to those of P
- safety property expressed as traces
  on I' ∪ I'' and O' ∪ O''

Objective :
build (P' = P modified by A) whose traces satisfy A

Additional constraint :
P and (P modified by A) should be comparable in some way.

The declarative approach



The declarative approach

- Given P, A, and a "generic" comparison criterion, how to compute (P modified by A), in such a way that `ok` and `ok'` are always true?
- How to define the comparison criterion ?

Reasonable comparison criteria ?

- Hiding some variables in the traces
  (a kind of projection on a set of names)
- Downsampling on a given condition `C`
  `ok' = if C then ... else true`
- More complex : `P'` is the same as `P`, but later
  `ok' = (o'~ = pre (o'))`
- A mixture of these three
- ...

Weaving mechanism ?

Looks very much like controller synthesis problems.

Cf. "Using Controller-Synthesis Techniques to Build Property-Enforcing Layers"
K. Altisen, A. Clodic, F. Maraninchi, E. Rutten --- ESOP 2003

Cf. "Interface Automata"
L. de Alfaro.

Contents

- Some words on "Aspect-Oriented Programming"
- Aspects for reactive systems
- A tentative formalization
  - General setting
  - The declarative approach based on trace semantics (KA)
  - The operational approach (FM)
- Time for comments, questions, ...

Start from a set of language constructs

Example (Argos without hierarchy) :

- Reactive and deterministic Mealy machines
- Parallel composition (synchronous product)
- Encapsulation (synchronous broadcast)

Invent elementary transformations

For each automaton of the program :

For each state `q` reachable by an input sequence satisfying ϕ :

for each transition `q -- m/o` $\longrightarrow$ `r` such that `m` satisfies `C`

-- replace it by `q -- m and m''/o,o1''` $\longrightarrow$ `r`

-- add transitions

`q -- m and not m''/o2''` $\longrightarrow$ `s`

to states s reachable by an input sequence satisfying ψ

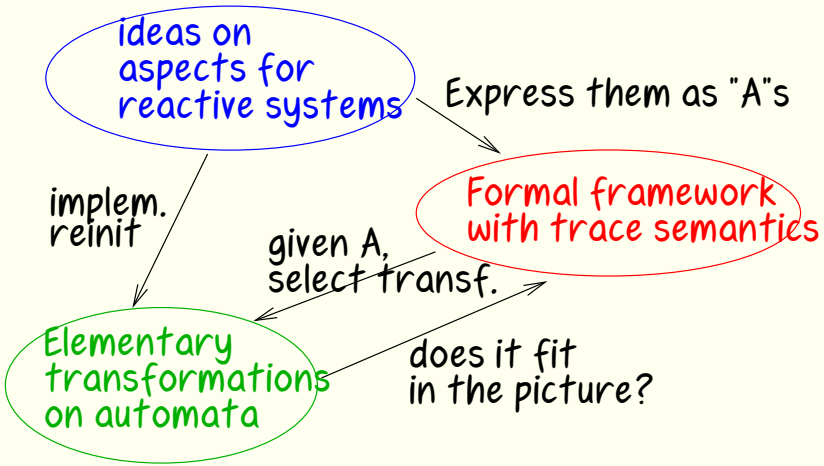
Parameters : ϕ, ψ, `C`, `o1''`, `o2''`, `m''`.

Invent elementary transformations

The re-initialization example, with additional input `R`:

ϕ = true (for all states),
ψ = ϵ (the initial state)
`C` = true (for all existing transitions)
`m'' = not R`
`o1'' = o2'' =` $\emptyset$

Conclusion


ideas on
aspects for
reactive systems
Express them as "A"s
implem.
reinit
Formal framework
with trace semantics
given A,
select transf.
Elementary
transformations
on automata
does it fit
in the picture?

More questions

- If P and Q are equivalent, what about (P modified by A) and (Q modified by A)?
- Aspect interferences :

  ```
  ((P mod.  by A1) mod.  by A2) ???
  ((P mod.  by A2) mod.  by A1)
  ```

- How much does the "looks like" box depend on P and A? (can it be completely generic ?)
- How much does the aspect specification (A) depend on P?
- How can we characterize things that cannot be done with a given set of language constructs ?