Dynamic Reconfiguration and Collaborative Synchronous Programming

Gwenaël Delaval

LIG — Université Joseph Fourier (Grenoble)

Synchron’2011
Motivations

Dynamicity everywhere:
- cellphones
- internet boxes

How to express **dynamic reconfiguration** with a synchronous language?

Proposal: combination of **synchronous** (for the time model) and **higher-order** (for “computations” seen as “values”) features
A first example: software radio

Reception channels in a software radio

Reception channel:
- composed of a filter and a demodulator
- dynamic reconfiguration depending of the protocol used
Dynamic reconfiguration using higher-order

- only one “reception channel” component, parameterized with the nodes filter and demod

```plaintext
let node channel (reconfigure, x) = y where
  rec automaton
  | Init ->
  do y = x
  until reconfigure(filter, demod)
  then Configure(filter, demod)
  | Configure(filter, demod) ->
  let f = run filter x in
  do y = run demod f
  until reconfigure(filter’, demod’)
  then Configure(filter’, demod’)
end
```
Dynamic reconfiguration using higher-order

- only one “reception channel” component, parameterized with the nodes filter and demod
- one reconfiguration component, sending dynamically the values for filter et demod (stream of nodes)

let node multichannel_sdr (x, umts, gsm) = y where
rec y = channel (switch_channel, x)
and automaton
    | Reconfigure_GSM ->
        do emit switch_channel = (filter_1800, demod_gmsk)
        then GSM
    | GSM -> do until umts then Reconfigure_UMTS
    | Reconfigure_UMTS ->
        do emit switch_channel = (filter_2000, demod_qpsk)
        then UMTS
    | UMTS -> do until gsm then Reconfigure_GSM
end
- Everything known at compilation time

- What about programming only the “server”?
Demo
**Server architecture**

- **Browser** (http server)
- **Lucid synchrone module** (dyn.ls)
- **HTTP server** (dyn.cmo)
- **Lucid synchrone compiler**
- **Vehicles** (dyn.ls, dyn.cmo)
let node server (static yinit) reconfigure = (x,y,cap) where

rec automaton

Init ->
do

acc = 0.
and brake = false
and left = false
and right = false
until reconfigure(g) then Configure((fun x => run (g f_init) x))

| Configure(f) ->
do

(acc,brake,left,right) = run f(last x, last y, last cap)
until reconfigure(g) then Configure((fun x => run (g f) x))

end

...
Collaborative programming
Conclusion

- Higher-order allows the (quite) easy expression of dynamic behaviors

- Dynamic reconfiguration can be useful:
  - easy experiments with the language
  - for live prototyping
  - for teaching?
Perspectives

Limits of this approach:

- **Lost of some usual properties** of synchronous languages: e.g., bounded memory and reaction time

- **Programs not reductibles to finite models** (finite state automata, symbolic transition systems, ...) : usual analysis/verification/synthesis tools non appliables

- **Expression of mobility, use of actual marshalling**
Thanks!

... a last demo?