ACI Alidecs

Connexion entre ReactiveML et Lucky

Bilan de 15 jours à Grenoble

Louis Mandel

louis.mandel@lip6.fr

Laboratoire d'Informatique de Paris 6

23 mai 2005

Plan

- Importation de nœuds LUCKY en REACTIVEML
- Simulation d'un réseau de capteur

Motivations

- Avoir un langage pour la description de comportements indéterministes
- Utiliser ReactiveML comme langage de compostion de programmes réactifs

Actualisation

Les type des entrées/sorties peuvent être : `int`, `float`, `bool` et `event`

Exemple : nuage.luc

```
outputs { x_nuage: float ~init 350.0 ~max 700.0 ~min 0.0;
          y_nuage: float ~init 350.0 ~max 700.0 ~min 0.0;

locals { Wind_x : float ~min -0.3 ~max 0.3 ~init 0.0;
         Wind_y : float ~min -0.3 ~max 0.3 ~init 0.0; }

nodes { init : stable; }
start_node { init }
```

Exemple : nuage.luc

```
  abs (Wind_y - pre Wind_y) < 0.1 and
  abs (Wind_x - pre Wind_x) < 0.1 and

  (if pre Wind_y > 0.0
   then ((y_nuage - pre y_nuage) <= pre Wind_y
         and (y_nuage - pre y_nuage) >= 0.0)
   else ((y_nuage - pre y_nuage) <= 0.0
         and (y_nuage - pre y_nuage) >= pre Wind_y))

  and ...
}
```

Exemple : main.rml

```
  ["nuage.luc"]
#val nuage : Lucaml.step_mode ->
  unit -> (float , 'a) event * (float , 'b) event -> unit

let process main =
  signal x, y in
  run (nuage Lucaml.StepInside () (x,y))
  ||
  run (diplay_nuage x y)
#val main : unit process#
```

ReactiveML/Lucky

Les nœuds LUCKY sont des processus REACTIVEML comme les autres :

- Suspension
- Préemption
- Création dynamique

Exemple

```
  { Wind_x : float; Wind_y : float; }
  { x_nuage: float; y_nuage: float; } = ["nuage.luc"]

external.luc wind
  {}
  { Wind_x : float; Wind_y : float; } = ["wind.luc"]
```

Exemple

```
await new_nuage;
(run (add new_nuage wind nuages)
 ||
 signal x, y in
 run (nuage Lucaml.StepInside (wind) (x,y))
 ||
 loop
   let abs = await one x (v) in v
   and ord = await one y (v) in v
   in emit nuages (abs,ord)
 end)
```

⇒ Manque de types structurés en LUCKY

ReactiveML comme langage de composition

```
do
  run (env Lucaml.StepInside (t_out,on) t_in)
when lucky
||
do
  run (manual switch t_in)
when not_lucky
||
run (controller switch not_lucky lucky)
||
...
```

ReactiveML comme langage de composition

```
loop
  do
    loop emit manual; pause end
  until switch(n) ->
    for i = 1 to n do emit auto; pause done
  done
end
```

$\Rightarrow$ L'état interne du processus LUCKY n'est pas modifié lorqu'il est suspendu

Principe de la traduction

est traduit en

```
let step = Lucaml.make "a.luc" mode in
loop
  pause;
  let v_out = step (pre ?s_in) in
  emit s_out v_out
end
```

Simulation d'un réseau de capteurs

Pour simuler quoi ?

- Détection des feux de forêt
- Influence des protocoles de communication sur la consommation d'énergie
- Les couches 2, 3 et 4 du modèle OSI

Pourquoi utiliser ReactiveML ?

- Avoir un contrôle fin sur la simulation
- Avoir l'expressivité d'un langage généraliste
- Avoir de la composition parallèle et de la création dynamique
- Simulation de l'environnement avec Lucky

Démos

- Inondation
- Inondation avec nuage

Perspectives

- Pouvoir appeler du Lucky depuis Lucid Synchrone
- Pouvoir appeler du Lucid Synchrone, Lustre, Esterel, ... depuis ReactiveML