ALIDECS

ACI Sécurité 2004–2007

Langages et Atelier Intégré pour le Développement de Composants Embarqués Sûrs

http://www-verimag.imag.fr/~synchron/alidecs/

Démons/LRI, Orsay(LIP6 en 04/05) Synchrone/VERIMAG, Grenoble PopArt/INRIA Rhône-Alpes Mimosa/INRIA Sophia-Antipolis CMoS/IBISC, Evry

Objectifs

- Outils et langages pour le développement de systèmes embarqués offrant des garanties fortes de correction.
- Chaîne de développement :
  - langages de haut niveau pour la spécification et la programmation du système et de son environnement,
  - mécanismes d'exécution précoce, de simulation du système et de son environnement,
  - outils de validation formelle et de test,
  - analyse statique et génération automatique de code.

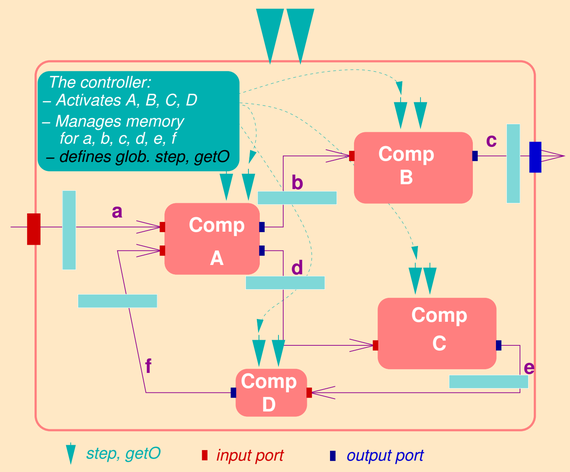
Modèle de composants pour l'embarqué

Caractéristiques :

- interface standardisée,
- séparation contrôle/données,
- la composition nécessite un *contrôleur*,
- le bloc est lui-même un composant.

Besoins :

- modèle dédié au prototypage virtuel,
- langage de description de l'architecture,
- langages de description fonctionnelle du système (comportements déterministes et non déterministes),
- plateforme d'exécution/simulation.



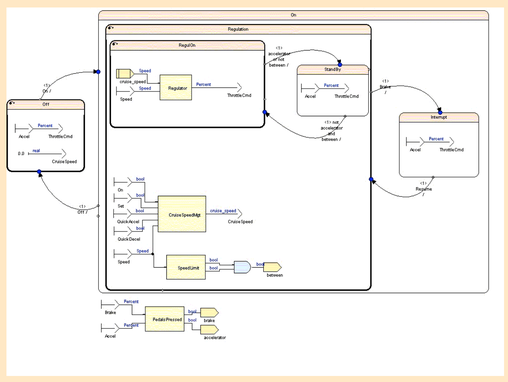
Programmation synchrone

La programmation synchrone est fondée sur un modèle de concurrence et de temps global commun à tous les processus.

Lucid Synchrone

Langage de programmation de systèmes synchrones déterministes. Il combine le modèle dataflow de **Lustre** et des traits propres aux langages fonctionnels de la famille **ML**.

- synthèse automatique des types et des horloges, analyses par typage (e.g., causalité, initialisation),
- les types servent à spécifier les contraintes fonctionnelles et temporelles,
- combinaison des paradigmes dataflow et automates,
- ordre supérieur pour décrire des systèmes reconfigurables dynamiquement,
- système de types pour spécifier et vérifier les contraintes d'architecture,
- modèle de communication par buffers pour les systèmes périodiques (réseaux de Kahn N-synchrones)



```
(* a PI regulation *)
let node regulator (cruise_speed, speed) = throttle where
  rec
      delta = cruise_speed -. speed
  and
      throttle = delta *. kp +. aux *. ki
  and
      aux = delta +. (0.0 -> pre aux)

let node cruiseControl (onn, off, resume, speed, set, quickAccel, quickDecel, accel, brake) =
  (cruiseSpeed, throttleCmd) where
  rec
    last cruiseSpeed = 0.0
  and
    braking = brake > 0.0
  and
    accelerator = accel > 0.0
  and
      automaton
        Off ->
          do throttleCmd = accel until onn then On
      | On ->
          do automaton
                Regulation ->
                  let
                      cruise_speed =
                        cruiseSpeedMgt(onn, set, quickAccel, quickDecel, speed)
                  and
                      between = (speed >= speedMin) & (speed <= speedMax) in
                  do
                    cruiseSpeed = cruise_speed
                  and
                    automaton
                        RegulOn ->
                          do
                            throttleCmd = regulator(cruise_speed, speed)
                          until (accelerator or not between) then StandBy
                      | StandBy ->
                          do
                            throttleCmd = accel
                          until
                            (not accelerator & between) then RegulOn
                  end
                until braking then Interrupt
              | Interrupt ->
                  do throttleCmd = accel
                  until resume then Regulation
            end
          until off then Off
      end
--:--  cruise.ls      17% L49   CVS:1.6  (Tuareg Abbrev)----------
val regulator : float * float => float
val regulator :: 'a * 'a -> 'a
val cruiseControl :
  bool * bool * bool * float * bool * bool * bool * float * float =>[] float * float
val cruiseControl :: 'a * 'a * 'a * 'a * 'a * 'a * 'a * 'a * 'a -> 'a * 'a
-1:**  *shell*        44% L24    (Shell:run)----------
```

Lucky/Lutin

Modélisation et simulation des environnements et autres composants indéterministes.

- **Lucky** : machine abstraite basée sur des automates stochastiques concurrents.
- **Lutin** : langage impératif de haut niveau pour programmer des machines Lucky.

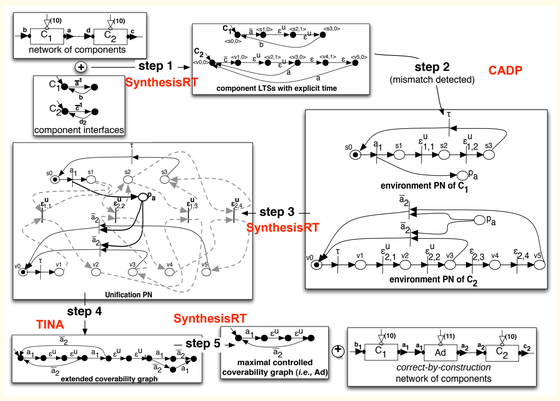
```
rabbit.lut + (~/other_tools/LUTIN-2/devel) - VIM
let between(x, min, max : real) : bool = ((min < x) and (x < max))

let up_and_down(speed, pspeed : real ref; min, max, delta : real) =
   if (pre speed < min) or (
      (pre speed < max) and (pre pspeed <= pre speed)
   ) then  (
      between(speed, pre speed, pre speed + delta)
   ) else (
      between(speed, pre speed - delta, pre speed)
   )

system rs () returns (
   speed : real 0.0;
   min, max, delta : real 0.0;
   pspeed : real 0.0
) =
   (speed = 0.0) fby (speed = 0.0) fby
   assert pspeed = pre speed in
   let init_params () : bool = (
      between(min, 0.0, 10.0) and
      between(max, 20.0, 50.0) and
      between(delta, 0.0, 1.0)
   ) in
   loop {
      init_params () and (speed = pre speed)
      fby
      assert (min = pre min) in
      assert (max = pre max) in
      assert (delta = pre delta) in
      loop~100:10 {
         up_and_down(speed, pspeed, min, max, delta)
      }
   }
```

Larissa

Programmation par aspects dans le cadre synchrone :

- basée sur un modèle général des programmes synchrones (machines de Mealy concurrentes),
- permet la description et le *tissage* de modifications globales transversales aux structures du langage de programmation,
- approche intéressante pour la production des interfaces des composants.

Composants et synthèse d'adaptateurs



- composants boîte noire (spécification du comportement, de l'horloge, de la contrôlabilité, latence et durée des actions),
- traduction (compilation) en LTS,
- composition parallèle (communication par FIFO) et analyse de blocage,
- si blocage détecté, modélisation de l'environnement attendu par chaque composant par un réseau de Petri (RdP),
- extraction d'un adaptateur (borné et non bloquant) du RdP composé
- systèmes communicants par FIFOs bornés + calcul statique des tailles

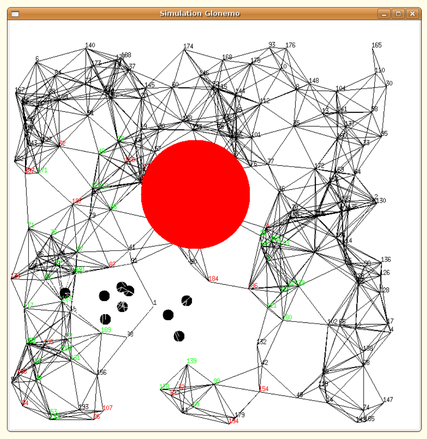
Programmation réactive

Le modèle réactif est une variante du modèle synchrone permettant de garantir par construction l'absence d'incohérences temporelles (problèmes de causalité). Il permet de traiter l'ajout et la suppression dynamique de composants.

FunLoft

- FairThreads = Programmation réactive à base de threads coopératifs regroupés en zones synchrones (schedulers)
- FunLoft = FairThreads + analyse statique pour assurer la réactivité et l'absence d'interférences dans l'accès à la mémoire
- En FunLoft, les threads deviennent aussi efficaces que des threads standards mais aussi sûrs que des processus

ReactiveML est une extension de **OCaml** basée sur le modèle réactif.



- extension conservative de OCaml et de son système de type
- interface avec **Lucky** pour décrire des comportements indéterministes
- implantation efficace de l'ordonnancement dynamique
- Étude de cas : simulation de protocoles de routage dans les réseaux mobiles ad hoc (en collaboration avec le LIP6) et dans les réseaux de capteurs (en collaboration avec VERIMAG/France Télécom) pour la conception de protocoles basse consommation.

Publications

- J-L. Colaço, G. Hamon et M. Pouzet. Mixing Signals and Modes in Synchronous Data-Flow Systems. Emsoft'2006.
- J-L. Colaço, B. Pagano et M. Pouzet. A Conservative Extension of Synchronous Data-flow with State Machines. Emsoft'2005.
- A. Cohen, M. Duranton, Ch. Eisenbeis, C. Pagetti, F. Plateau et M. Pouzet. N-Synchronous Kahn Networks. POPL'2006.
- L. Mandel et M. Pouzet. ReactiveML, a Reactive extension of ML. PPDP'2005.
- F. Dabrowski, F. Boussinot. Cooperative Threads and Preemptive Computations. TV'06
- L. Samper, F. Maraninchi, L. Mounier, L. Mandel. GLONEMO: Global and Accurate Formal Models for the Analysis of Ad-Hoc Sensor Networks. InterSense'06
- L. Mandel, F. Benbadis. Simulation of Mobile Ad-Hoc Networks in ReactiveML. SLAP 2005.
- K. Altisen, F. Maraninchi et David Stauch. Larissa: Modular design of man-machine interfaces with aspects. ISSC 2006.
- K. Altisen, F. Maraninchi et David Stauch. Interference of Larissa aspects. FOAL 2006.
- P. Raymond, E. Jahier, and Y. Roux. Describing and executing random reactive systems. SEFM 2006.
- T. Ayav, P. Fradet et A. Girault. Implementing Fault-Tolerance in Real-Time Systems by Automatic Program Transformations. Emsoft'2006.
- M. Tivoli, P. Fradet, A. Girault et G. Göessler. Adaptor Synthesis for Real-Time Components. Rapport INRIA
- N. Benaissa, B. Djafri, G. Hutzler et H. Klaudel. Towards modelling and verification of mobile agent systems. Workshop on Industrial Applications of Distributed Intelligent Systems (INADIS 2006).