ALIDECS / Programmation réactive

Frédéric Boussinot
projet MIMOSA, Inria-Sophia

http://www.inria.fr/mimosa/rp

octobre 2004

Plan

- Programmation réactive
- FairThreads
- Objectifs dans ALIDECS

Programmation réactive

- Concurrence + Instants + Création dynamique
- Événements diffusés (*broadcast*) - Réaction retardée à l'absence
- Pas de problème de causalité : modularité, dynamicité
- 3 variantes :
  - Instructions réactives + machines réactives (parallélisme imbriqué)
  - Programmation graphique : *Icobjs*
  - Multi-threading : *FairThreads* (parallélisme au plus haut niveau uniquement)

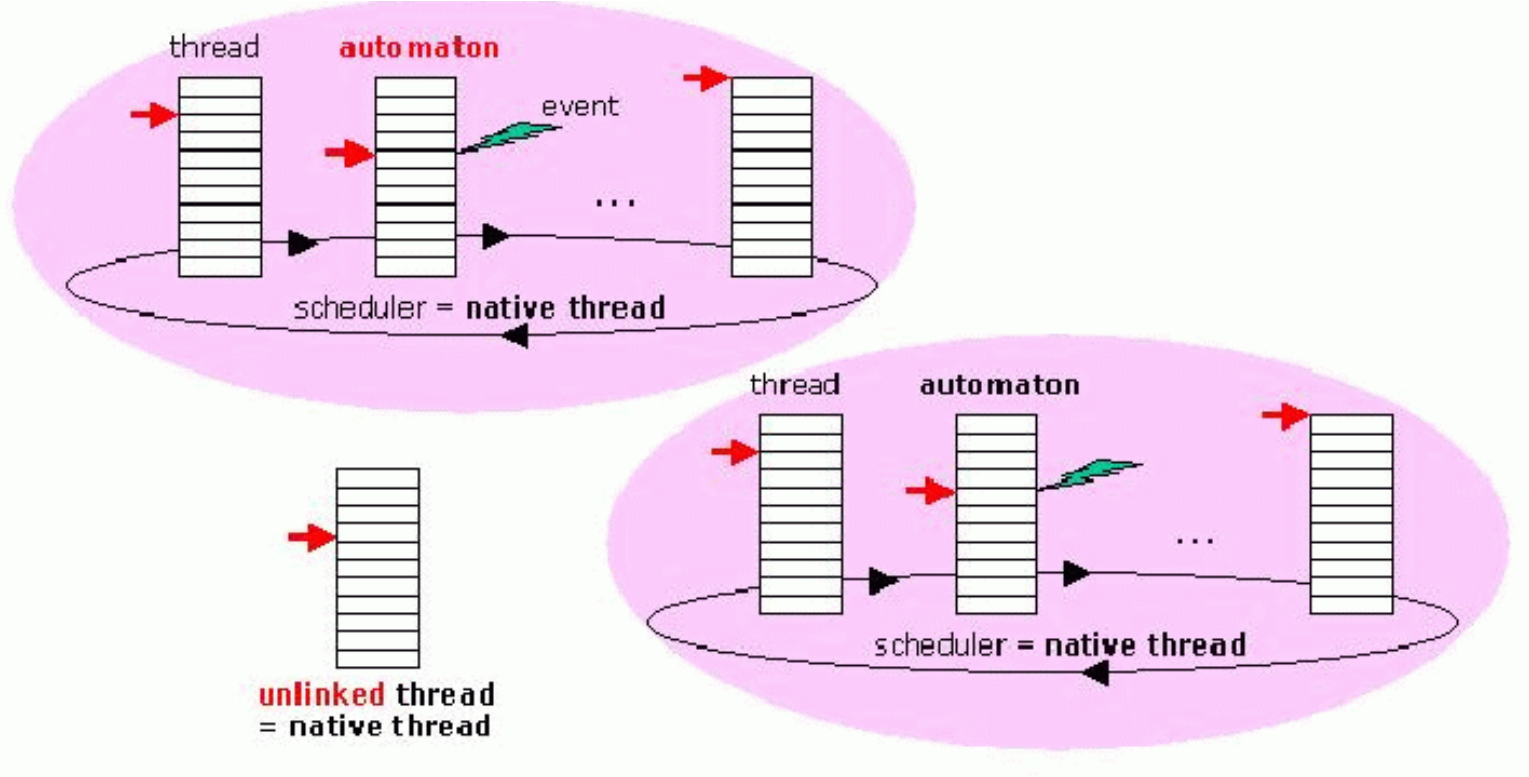
Langages & implémentations

- C
  - Reactive-C (1988, F. Boussinot)
  - FairThreads (2003, F. Boussinot)
- Java
  - SugarCubes : classes Java pour la prog. react. (J-F. Susini)
  - Junior : noyau + sémantique opérationnelle + optimisations (L. Hazard)
  - Rejo : language + primitives pour la migration (R. Acosta)
  - Icobjs : framework (C. Brunette)
- Langages fonctionnels
  - Scheme : FairThreads (M. Serrano)
  - ML : librairie réactive sur Standard ML (R. Pucella), RML (L. Mandel)

- ULM : en cours de développement (G. Boudol)

FairThreads en C

- Threads coopératifs + Instants + Événements diffusés
- Scheduling avec un ordre fixe d'exécution



- LOFT : *Language over Fair Threads*

Exemple de code LOFT

```
module cell (int x, int y, event_t activation, int status, int state)
local neighborhood_t neighborhood, info_t info, image_t image, int counter, int more, int fired;
  initialize_cell (self ());
  while (1) do
      if (local(status) == QUIESCENT) then await (local(activation));
      else activate_neighborhood (self ()); end
      {
        local(more) = 1; local(counter) = 0; local(fired) = 0;
        clear_neighborhood (local(neighborhood));
      }
      while (local(more)) do
        get_value (local(activation),local(counter),(void**)&local(info));
        {
          if (return_code () == OK) {
            set_neighbor (local(neighborhood),local(info));
            local(fired) += local(info)->fired;
            local(counter)++;
          } else local(more) = 0;
        }
      end
      cell_behavior (self ());
  end
end module
```

Axes de travail

- Programmation concurrente sur des systèmes embarqués avec de faibles ressources
  - *GameBoy Advance*
  - *THINK* (FT R&D)
- Liaisons synchrone/asynchrone
  - interfacage avec les couches basses des systèmes (interruptions)
  - GALS
  - multiprocessing (SMP)
- Sémantique formelle et implémentation efficace (vitesse, grand nombre de composants)
- Automates cellulaires

Objectifs dans ALIDECS

- Aspect langage
  - Prog. concurrente par threads

    besoin de sémantique
  - FairThreads = C + librairie

    API insuffisante = besoin de syntaxe
  - Langage LOFT

    besoin de sécurité
  - Assurer la coopération (absence de bouclage instantané)
    - Atomes (restriction de C + typage ? F. Dabrowski, CRISS ; Contrôle dynamique ?)
    - Passage des instants = automates (?)

Objectifs dans ALIDECS

- Aspect réutilisation de code : threads POSIX implémentés en réactif
- Aspect implémentation : machine virtuelle réactive ?

et pourquoi pas aussi :

- FairThreads en ML ?
- ??