Programming Concurrency

Cédric Pasteur <cedric.pasteur@ens.fr>
PARKAS Team
Département d'Informatique, Ecole normale supérieure, Paris

27 févr. 2013
What is concurrency?

Demo
Language-based approach

Programming

- Programming Languages Design
  - Domain-specific Languages (DSL)

- New abstractions
  - Expressivity
  - Safety
  - Efficiency

Compilation

- Static analysis, Typing
- Efficient code generation
(Some) Concurrent Systems

Embedded Systems

Simulation of embedded systems

Hybrid (ODE) (Simulink, Modelica)

Discrete
Concurrency ≠ Parallelism

Concurrency = Model
- Doing several things at the same time
- Does not require hardware parallelism

Parallelism = Implementation
- Multi-core
- Going faster
- Implementation
Introduction

Synchronous languages

Beyond embedded systems: ReactiveML

My work: Time refinement
Introduction

Synchronous languages

Beyond embedded systems: ReactiveML

My work: Time refinement
Synchronous languages

Domain-specific languages for embedded systems

- Created in the 80's in France
  - Lustre (Caspi & Halbwachs, Grenoble)
  - Signal (Benveniste & Le Guernic, Rennes)
  - Esterel (Berry & Gonthier, Sophia)

Principles

- Time
- Deterministic Concurrency
- Mathematical foundations (= Semantics)
  - Formal verification
Synchronous languages

Programming embedded systems

- Reactive
- Real-time
- Critical

Interaction loop

- Read inputs (sensors)
- Compute
- Write outputs (actuators)
The synchronous hypothesis

Time

- Discrete
- Logical
- Global
The synchronous hypothesis

Computation and communication are instantaneous

- Computer is infinitely fast
The synchronous hypothesis

Computation and communication are instantaneous

- Computer is infinitely fast
- Check if hypothesis holds
Computer assisted specification (Airbus, 80's)
Synchronous data-flow language

- Equations on flows (= infinite sequence)

Example

- A counter
Synchronous data-flow language

- Equations on flows (= infinite sequence)

Example

- A counter

\[
\begin{align*}
{pcpt}_0 &= 0 \\
{pcpt}_{i+1} &= {cpt}_i \\
{cpt}_i &= \begin{cases} 
{pcpt}_i + 1 & \text{if } {tick}_i \\
{pcpt}_i & \text{otherwise}
\end{cases}
\end{align*}
\]
Synchronous data-flow language

- Equations on flows (= infinite sequence)

Example

- A counter

```plaintext
node event_counter
  (tick: bool) = (cpt: int)
var pcpt: int;
let
  pcpt = 0 fby cpt;
  cpt = if tick
    then pcpt + 1
    else pcpt;
tel
```

\[
\begin{align*}
pcpt_0 &= 0 \\
pcpt_{i+1} &= cpt_i \\
cpt_i &= \begin{cases} 
  pcpt_i + 1 & \text{if } tick_i \\
  pcpt_i & \text{otherwise}
\end{cases}
\end{align*}
\]
Code generation

- Sequential code
- Efficient
- Bounded time (WCET) and memory

Formal verification

- Model-checking
Safety-critical embedded systems

- Plane, trains, subways, nuclear plants
- Certified/Qualified (DO-178B, IEC 61508, etc.)
Introduction

Synchronous languages

Beyond embedded systems: ReactiveML

My work: Time refinement
Synchrony in a general-purpose language

- No real-time
- Dynamic creation

ReactiveML

- Functional language (ML, OCaml)
- Synchronous model of concurrency
  - Discrete logical time
  - Communication by broadcast
Application

- Discrete simulation

n-body problem

- Simulate the gravitational interactions of n bodies
- Numerical integration methods

\[ m_i \ddot{a}_i = \vec{f}_i = \sum_j \vec{F}_{i,j} \]
- One process per body
- A global signal
N-body in ReactiveML

1. Send force

- One process per body
- A global signal

\[
\text{env} \rightarrow f_1 \rightarrow \text{Body #1} \quad (x_t, v_t, w) \\
\rightarrow f_2 \rightarrow \text{Body #2} \quad (x_t, v_t, w) \\
\rightarrow f_{100} \rightarrow \text{Body #100} \quad (x_t, v_t, w)
\]
N-body in ReactiveML

- One process per body
- A global signal

1. Send force
2. Collect forces

\[ \text{env} = \sum f_i \]

Body #1
\((x_t, v_t, w)\)

Body #2
\((x_t, v_t, w)\)

Body #100
\((x_t, v_t, w)\)

...
N-body in ReactiveML

- One process per body
- A global signal

1. Send force
2. Collect forces
3. Compute new position

\[ \text{env} = \sum f_i \]

Body #1
\[(x_t, v_t, w) \rightarrow (x_{tp}, v_{tp}, w)\]

Body #2
\[(x_t, v_t, w) \rightarrow (x_{tp}, v_{tp}, w)\]

Body #100
\[(x_t, v_t, w) \rightarrow (x_{tp}, v_{tp}, w)\]

\[
\begin{align*}
&\text{env} = \sum f_i \\
&f_1 \\
&f_2 \\
&f_{100}
\end{align*}
\]
let dt = 0.01
signal env default (fun _ -> zero_vector)
  gather add_force

let rec process body (x_t, v_t, w) =
  emit env (force (x_t, w));
  await env(f) in
  (* euler semi-implicit method *)
  let v_tp = v_t +++ (dt **. (f x_t)) in
  let x_tp = x_t +++ (dt **. v_tp) in
  run body (x_tp, v_tp, w)

let process main =
  for i = 1 to 100 dopar
    run body (random_planet ())
done
Introduction
Synchronous languages
Beyond embedded systems: ReactiveML
My work: Time refinement
Motivation

Different time scales

- Exemple: network of sensors
  - Internals of each sensor (microseconds)
  - Communication on the network (seconds)

Refinement

- Replace a process with a more detailed version
- Do not change the observable behavior
Reactive domains

- Local instants
- Invisible from the outside of the domain
Multiple steps integration methods

- eg Runge-Kutta
- one computation step = one local instant
- Switch method on-the-fly
Adaptive integration

- Compute in several steps
  - New positions
  - Estimate of the error
- If error too big, take a smaller time step

Two reactive domains

- One for computation steps
- One for the different tries
Time and Events

Course by Gerard Berry, College de France

First lesson

- L’informatique du temps et des événements
- March 28th 2013 18:00

Courses and seminars

- On Tuesdays, at 10:00
Conclusion

Programming languages should deal with

- Time
- Concurrency

Links

- ReactiveML (http://rml.lri.fr)
- Try ReactiveML online (http://rml.lri.fr/tryrml)