Synchronous Languages for the Modeling of: Logical Behavior, Functional/Non-Functional Time, Energy Consumption, etc.

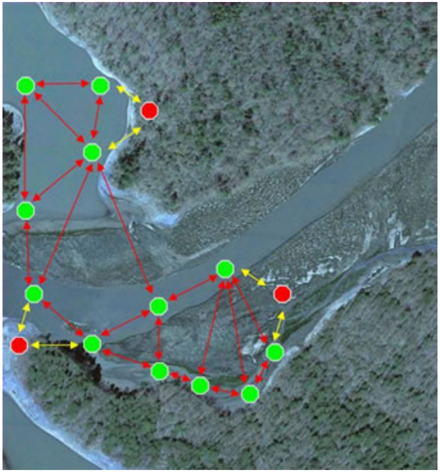
with an application to wireless sensor networks

Florence Maraninchi and Catherine Parent

SYNCHRON meeting, Fréjus

November 30th, 2010

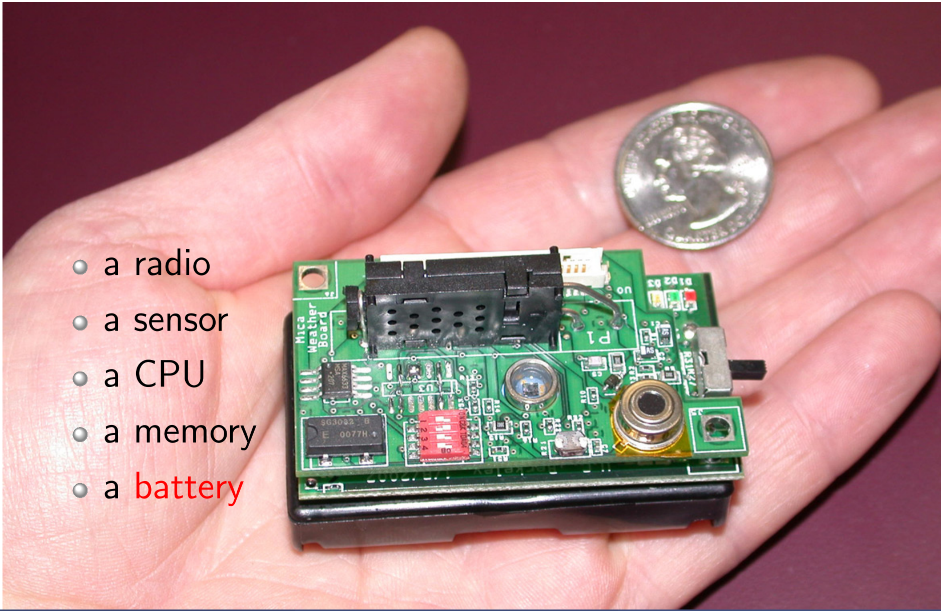
Sensor Networks



Several thousands of sensors communicating by radio + a special node connected to the network (the sink).

Examples: detection of a radioactive cloud, ...

The node itself



- a radio
- a sensor
- a CPU
- a memory
- a battery

Current Topics at Verimag

- Synchronous programming of device drivers (talk by N. Berthier, LCTES'11)
- Designing protocols (mainly at the routing level) for finding an acceptable trade-off between energy consumption and security.
- Applying a systematic "*distributed system* + *fault tolerance*" approach to the design of protocols for WSNs (talk by Yvan Rivierre on self-stabilization)
- Using probabilities... (in the design of protocols, in the modeling, next year talk)
- Fine-grain Modeling of energy consumption (this talk)

What do we Model?

- Functional behavior +
- Energy consumption +
- Several Time Scales...

Why do we Model?

For simulations before deployment:

- To find functional bugs (messages are lost because a node is not listening when it hould be listening),
- and non-functional potential gains (the radio is not turned off while it could be)

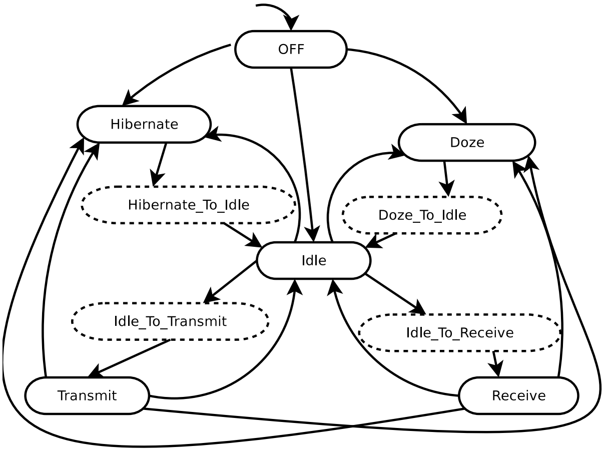
Direct vs. Indirect Models of Non-Functional properties

- Indirect models of energy consumption: decide once and for all that *sending a message to the sink costs N units of energy*, then forget about energy and count messages;

Direct vs. Indirect Models of Non-Functional properties

- Indirect models of energy consumption: decide once and for all that *sending a message to the sink costs N units of energy*, then forget about energy and count messages;
- Direct models of energy consumption: describe each hardware device by a *power-state model*; then add the model of the rest of the system, that drives this power-state model.

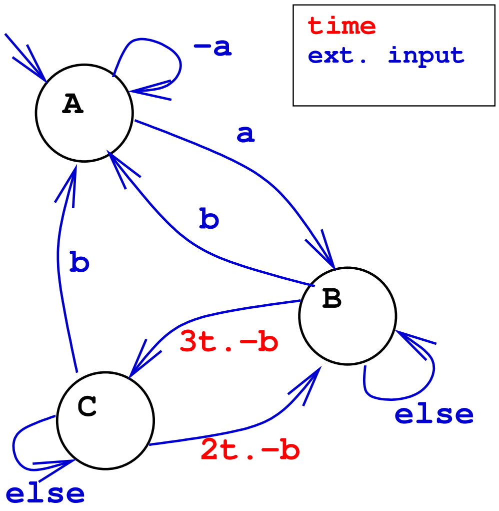
Power-State Models from Device Documentation



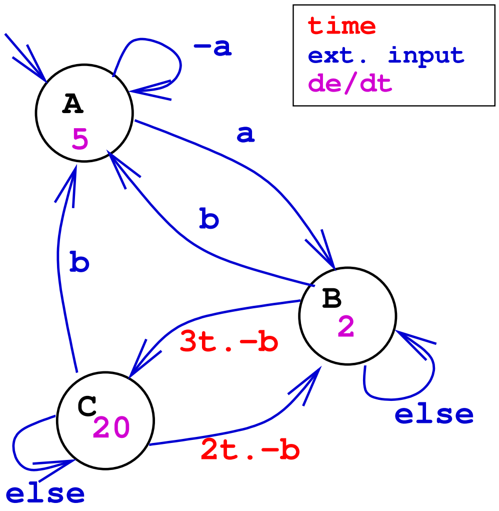
- Equations $de/dt = k$ on the states
- Delays T and energy penalties E on transitions

This can be encoded into a simple hybrid automaton (with additional states); see also LPTA (Linear-Priced Timed Automata).

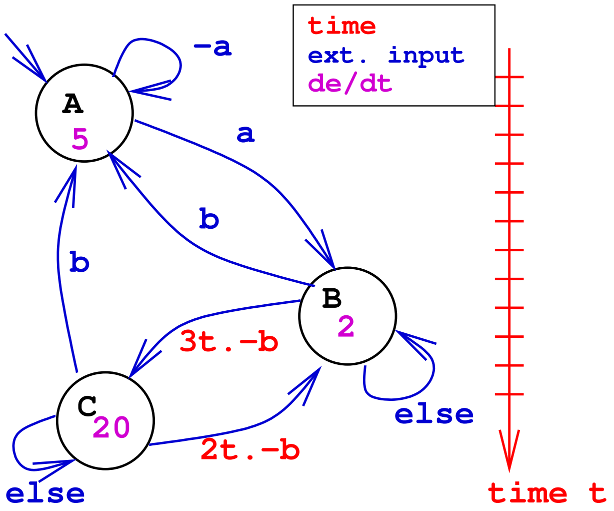
Energy Automata in a Synchronous Setting



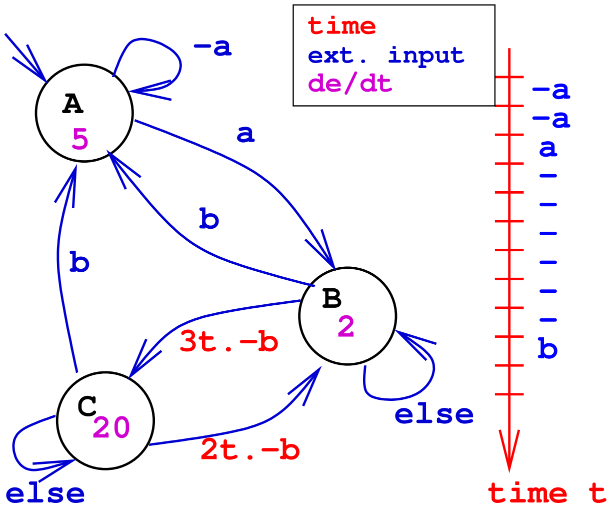
Energy Automata in a Synchronous Setting



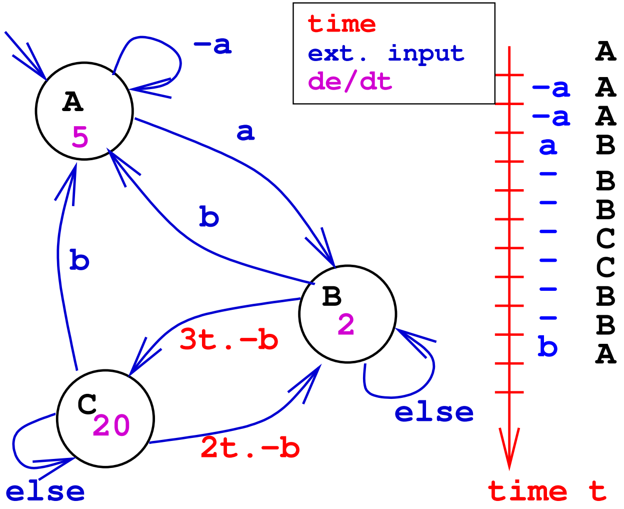
Energy Automata in a Synchronous Setting



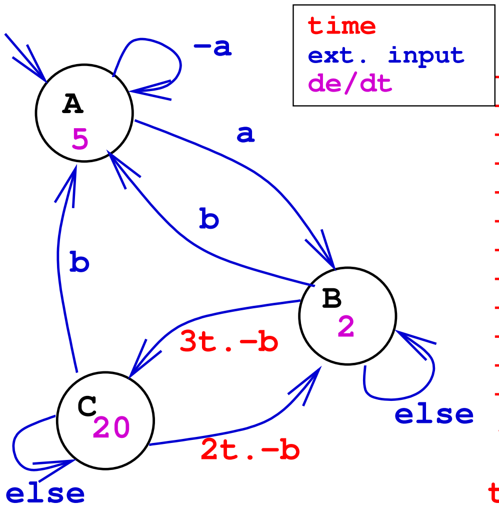
Energy Automata in a Synchronous Setting

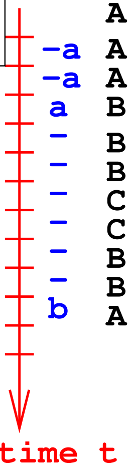


Energy Automata in a Synchronous Setting



Energy Automata in a Synchronous Setting

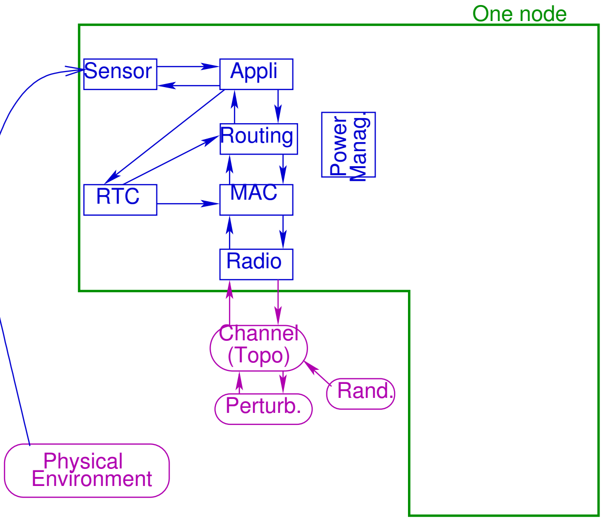




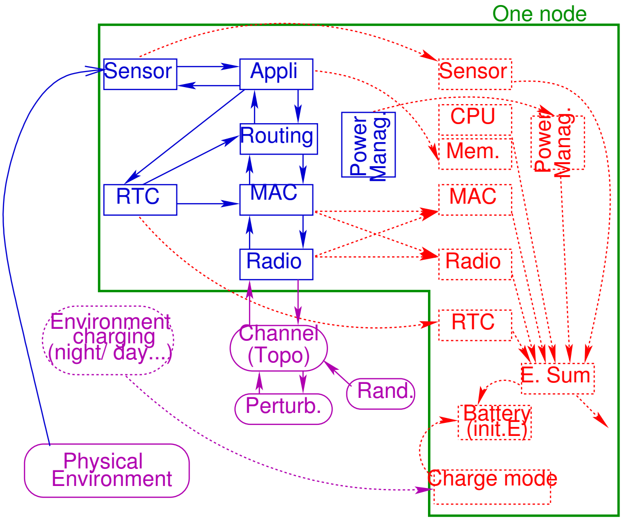
	A	5	5
-a	A	5	10
-a	A	5	15
a	B	2	17
-	B	2	19
-	B	2	21
-	C	20	41
-	C	20	61
-	B	2	63
-	B	2	65
b	A	5	70

Cumulated energy

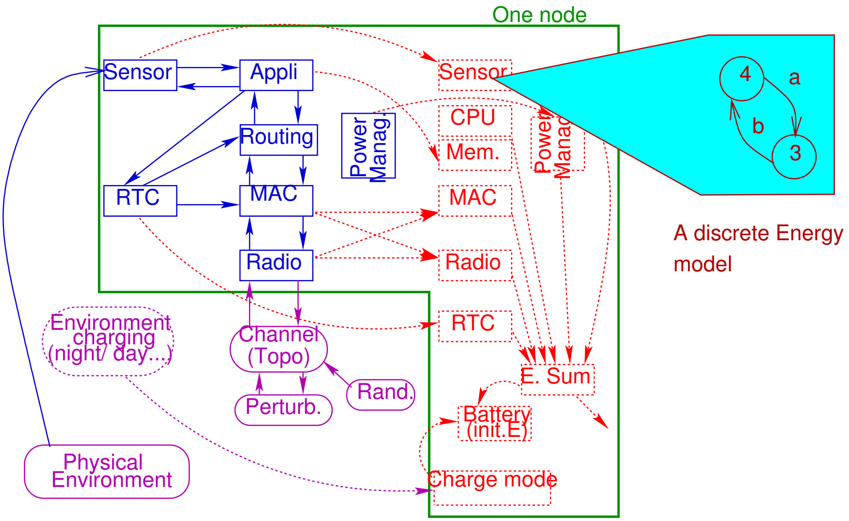
Component-Based System-Wide Models



Component-Based System-Wide Models

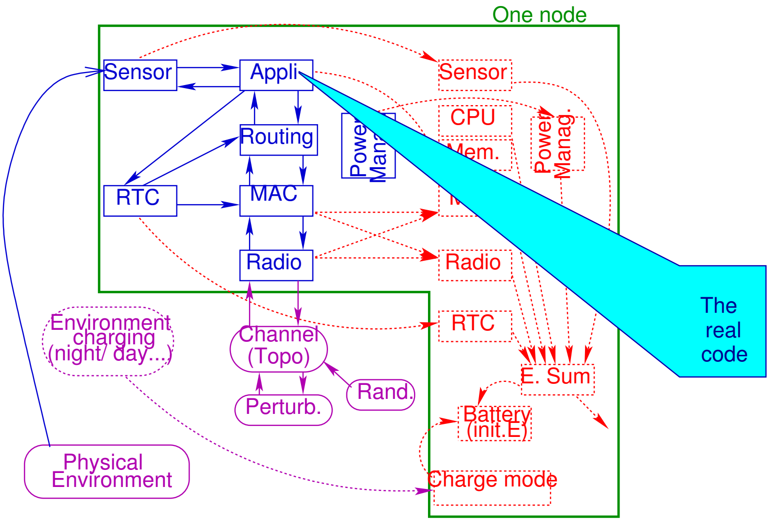


Component-Based System-Wide Models

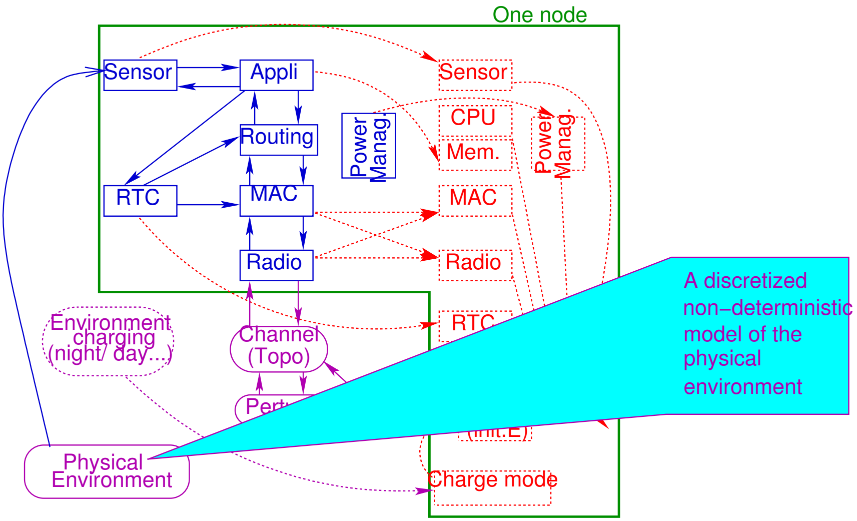


A discrete Energy model

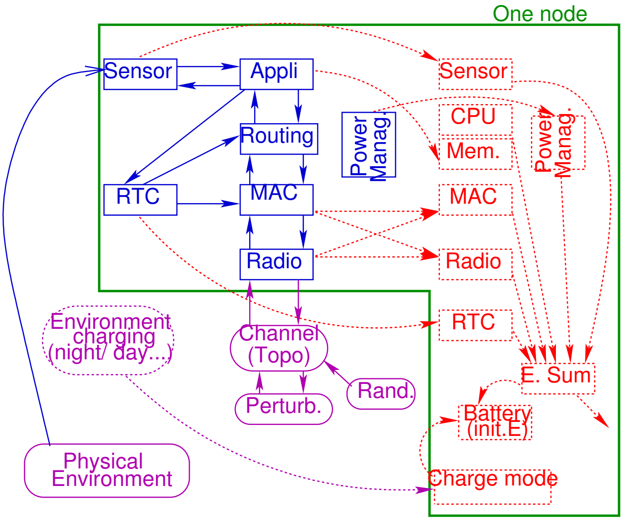
Component-Based System-Wide Models



Component-Based System-Wide Models



Component-Based System-Wide Models



Several Approaches

- Models in ReactiveML, Lustre
- *Comparison with SystemC and other event-driven simulation engines (Synchron'08)*
- Current: Using WSNET (an existing simulator for sensor networks, Lyon) to be able to use the protocols as they are written (without re-encoding them in RML); WSNET alone is not sufficient: designing a MAC protocol that limits energy consumption (in particular overhearing) is quite tricky; WSNet does not show the benefits because the radio is not modeled!
- Current: Using Lutin/Lucky (Verimag) or CCSL to build more abstract models, relying on "time" constraints only

Common Point: Avoid Synchronous Models!

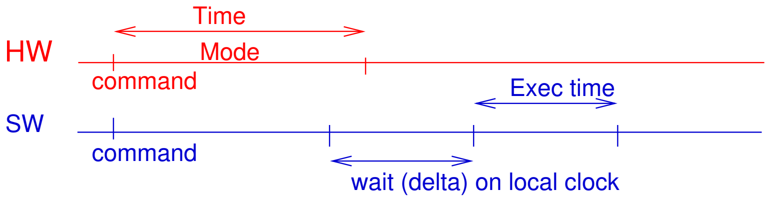
A sensor network is a typical GALS: inside a node it's synchronous, between nodes it's asynchronous.

The physical clocks of the nodes are not synchronized, and they can derive a lot.

Avoid *"unique-time-scale"* simulation models that describe synchronized clocks!

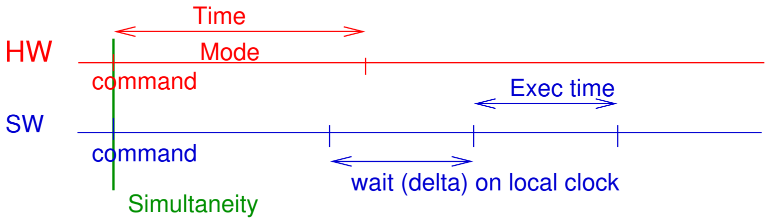
But you can use a synchronous language to describe a constrained asynchronous system (e.g., a quasi-synchronous system as defined by P. Caspi)

Overview



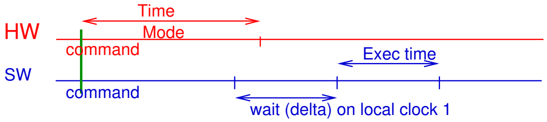
— HW has (energy) modes and transitions between modes that take some time
— HW receive mode changing commands from the software
— SW has execution time, and explicit wait instructions (counting on a local clock)

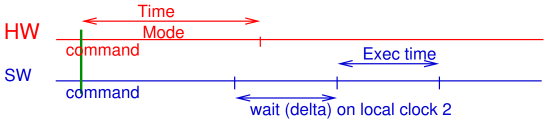
Overview



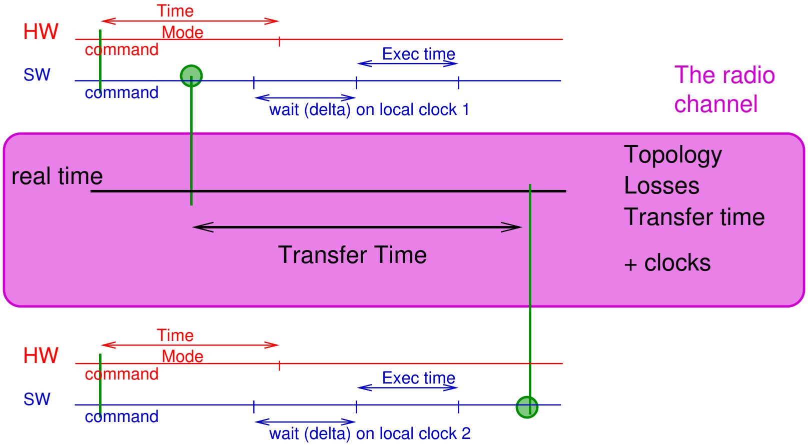
— HW has (energy) modes and transitions between modes that take some time
— HW receive mode changing commands from the software
— SW has execution time, and explicit wait instructions (counting on a local clock)

Overview: several nodes

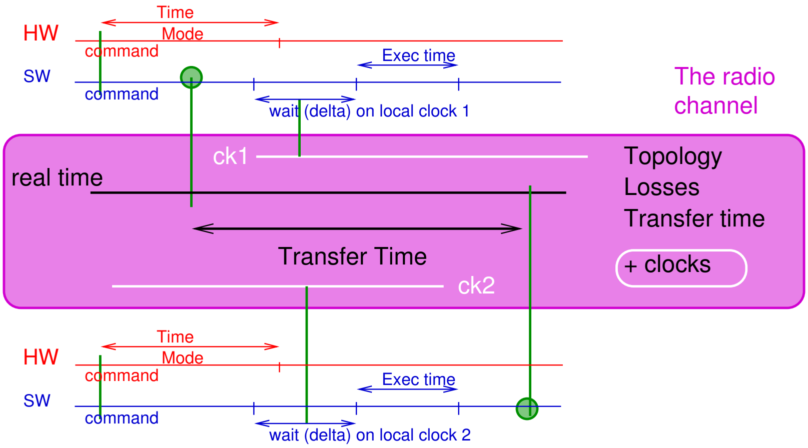




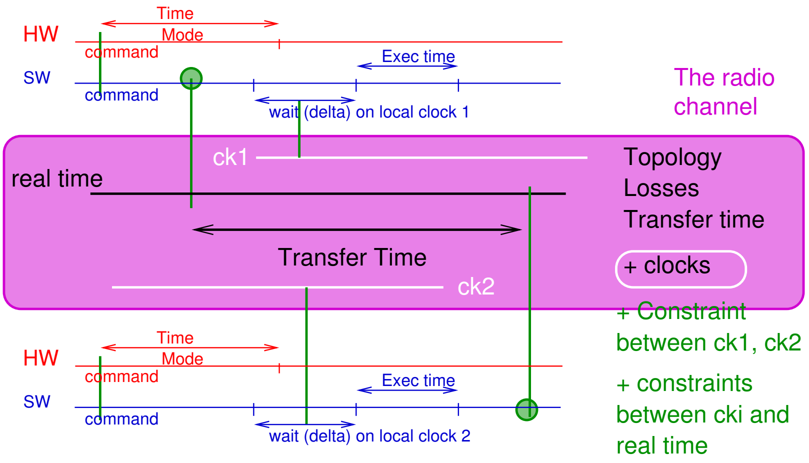
Overview: several nodes



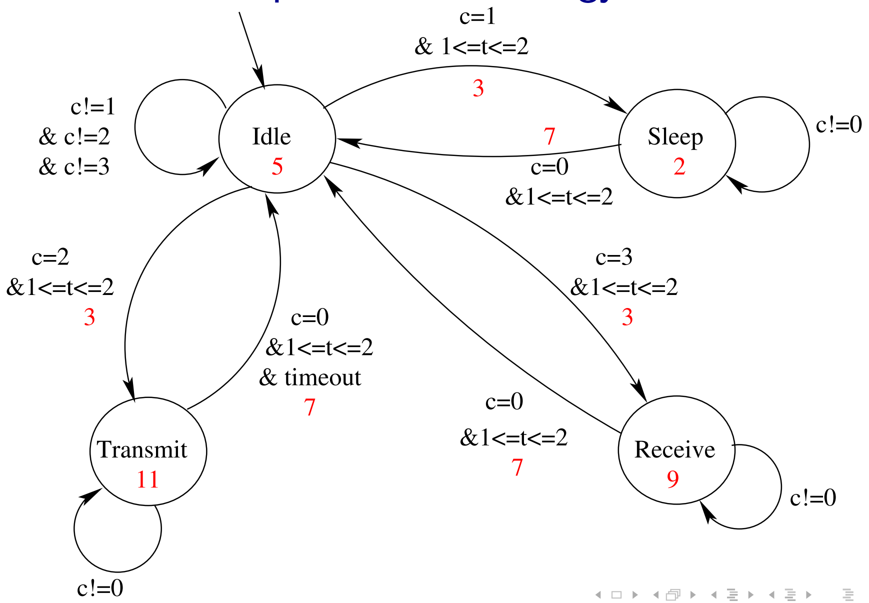
Overview: several nodes



Overview: several nodes



The Radio component with energy modes



The software (e.g., a MAC protocol)

```
while true
{
  // send
  emit command to put the radio in Transmit mode
  wait max time for the radio to go Idle->Transmit->Idle

  // receive
  emit command to put the radio in Receive mode
  wait max time of a reception
  emit command to put the radio in Idle mode again
}
```

The same in Lutin (input: local clock)

```
node mac (clk : bool) returns (comm : int)  =
let wait_clk =
  loop { not clk and comm = 4 (* 4 means "no command" *)} in

  let manage_time (t : int) =
   exist time : int in
         time = t and comm = 4
         fby
            assert (pre time > 1 and comm = 4) in
            loop {
                        clk and (time = pre time - 1)
                      | not clk and time = pre time
                    } in
```

The same in Lutin, cont'd

```
loop {
  wait_clk
(* sending *)
  fby comm = 2 (* 2 means "go to Transmit" *)
  fby wait_clk
(* wait the time to do Idle-Transmit-Idle *)
  fby manage_time (9)
  fby wait_clk
(* receiving *)
  fby comm = 3 (* 3 means "go to Receive" *)
  fby wait_clk
(* waiting to arrive in Receive *)
  fby manage_time (2)
  fby wait_clk
(* waiting the time of a complete receiving *)
  fby manage_time (5)
```

Generalization for SW

- Encoding of the control structure: easy
- Wait statements: count occurrences of the clock
- send events: a constraint on the occurrences of the event
- + a bit of non-determinism for data-dependent choices
- *Execution times: for the moment, we ignore them. Otherwise: WCET on basic blocks.*

An imperative C program can be translated systematically into this kind of formalism.

The Radio Device with Energy Modes

Easy to encode.
Time is counted on the “real-time” scale, different from all the software clocks.

Demo with Verimag Tools (Catherine)

Testing tool, language for non-deterministic behaviors, execution engine: Lurette/Lutin/Lucky... (P. Raymond et al.)

- Describing the system with Lutin
- Using Lutin/Lucky
- Connection to sim2chro for the timing diagrams

Exercice for Charles: the same in CCSL + t^2.
Report is due tomorrow!

Work in Progress

- With Lutin/Lucky (or CCSL): model an existing stack (choice of a routing and MAC protocols for sensor networks from WSNET or SensLab)
- With WSNET: trick to add the radio model and the constraints between the MAC layer and the radio modes

Same goal: find functional bugs (messages are lost because a node is not listening when it should be listening), and non-functional potential gains (the radio is not turned off while it could be)

No Conclusion Yet... However:

When debugging a MAC protocol that cares about the radio states, people try and produce pathological orders of events (sending, being in receive mode, ...).

You don't need the whole detailed code to do that.
The type of models we build could be used in formal verification.