Observability and Controllability of Wireless Software Components

Fabien Romeo

Liuppa, Université de Pau

fabien.romeo@fromeo.fr www.fromeo.fr

Background

- Component-Based Software Engineering (CBSE)
  - «*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.* » — Szyperski *et al* (2002).
- CBSE applied to Wireless Systems (WS)
  - WS are heterogeneous systems and software componentization allows adaptation
  - WS are dynamic and open systems, and as such, they often use Service-Oriented Architecture, which relies on software components (*eg*. OSGi bundles)

The need for management in WS

- Toward Open-World Software: Issues and Challenges

(IEEE Computer Magazine, October 2006):

« *A highly dynamic and open system necessitates* ***runtime monitoring*** *to watch for situations that might require suitable reactions to assure the desired level of global quality. […] As a result of monitoring, it should be possible to handle deviations from* ***expected behaviors*** *and plan for a* ***reconfiguration****.* »

—Luciano Baresi, Elisabetta Di Nitto, and Carlo Ghezzi

- I. Crnkovic, "Component-based Software Engineering for Embedded Systems," in ICSE'05.
- A. Möller, J. Fröberg, and M. Nolin, "Industrial Requirements on Component Technologies for Embedded Systems," in CBSE'04.

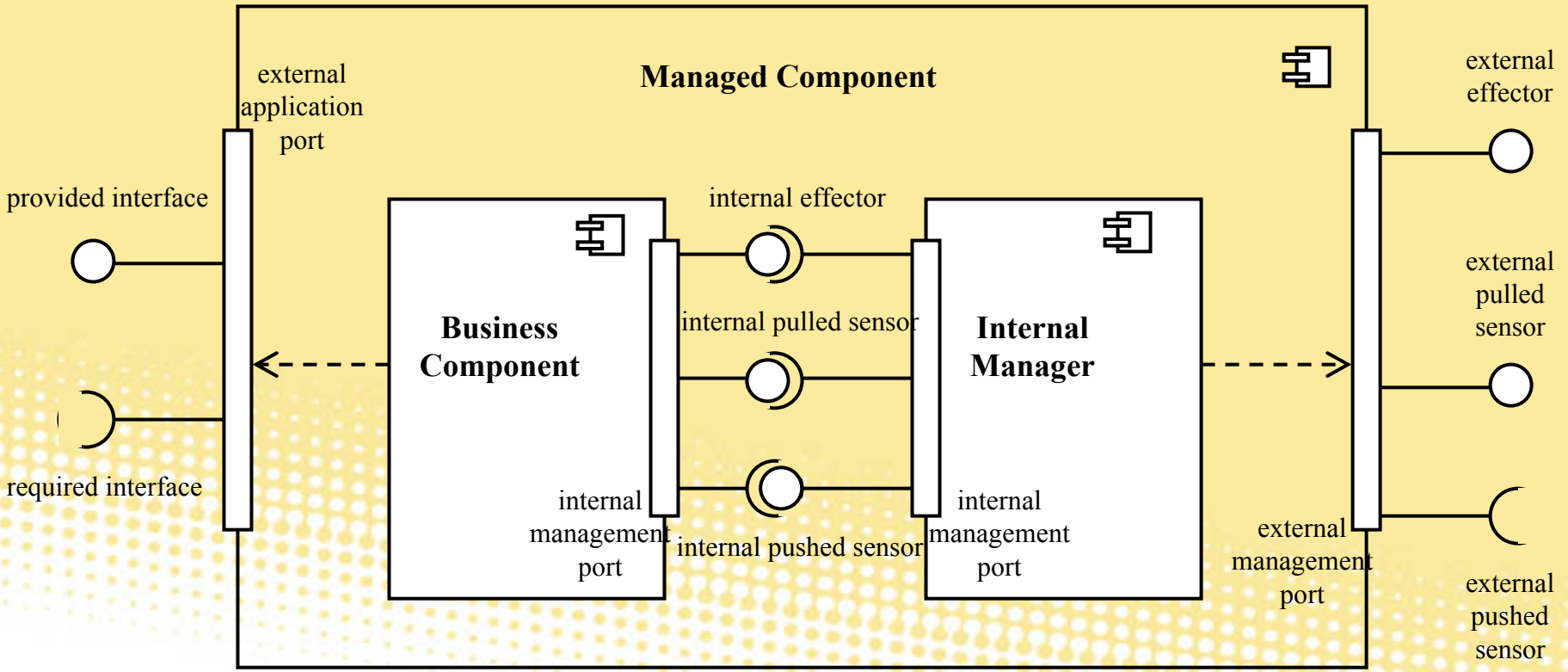
Management in CBSE

- Maintaining component-based systems is a difficult problem – Voas (1998)
  - Black-boxes, in particular COTS components
  - No communication between providers and assemblers
  - Management is often an after-thought
- Three kinds of approaches in the literature:
  - based on the components' infrastructure
    - Lifecycle management, low-level states (attributes) access
  - based on the application's architecture
    - Components reify the architecture, control by replacement
  - based on the components' behavior
    - External observations are confronted against behavior models

Our proposal

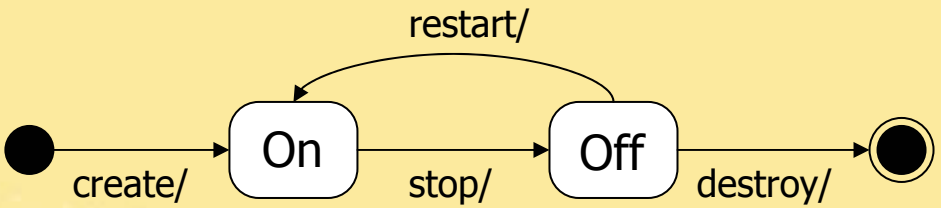
- A runtime management infrastructure
  - Address the internal behavior of software components
  - Wireless remote access
  - Deployment on constrained devices (PDA, mobile phones, …)
- Model-Driven Engineering philosophy
  - The components' behavior is designed with UML State Machines
  - Direct execution and control of components' behavior by their internal managers which embed a PauWare Model Execution Engine
  - The same models designed at development time are retrieved at runtime for management purpose

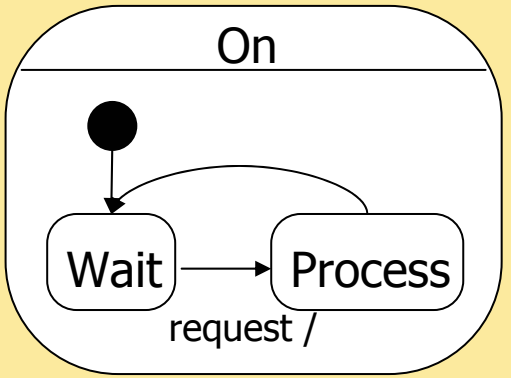
Wireless Software Component's Architecture

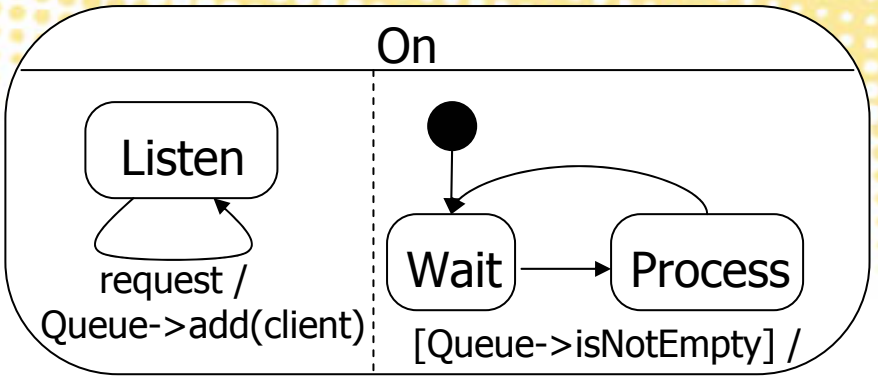


Statecharts

- [Harel 1987] : *Statecharts* =
  - *state-diagrams* : XOR
  - + *depth* : hierarchie
  - + *orthogonality* : AND
  - + *broadcast-communication :*
    - Asynchronous Communication
    - Notation: ^signal

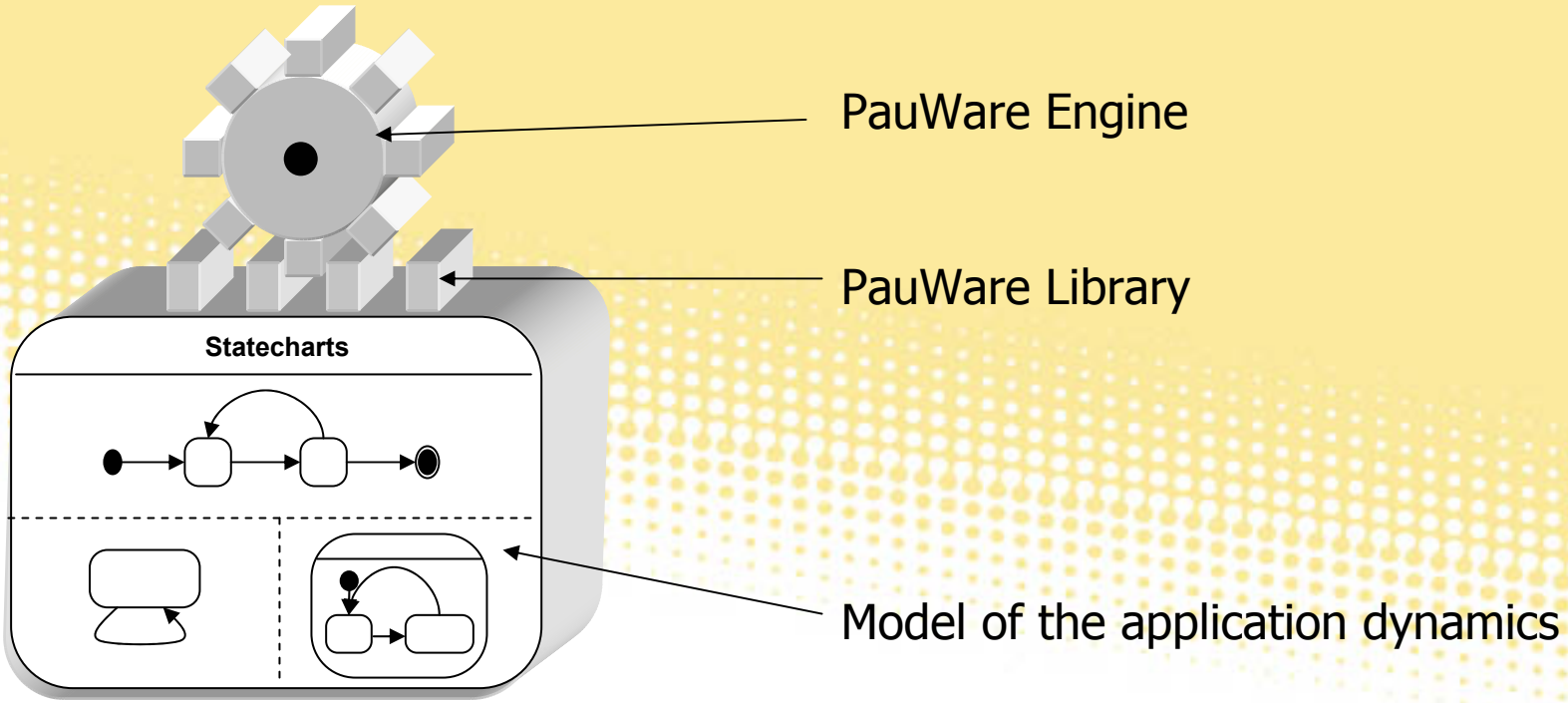






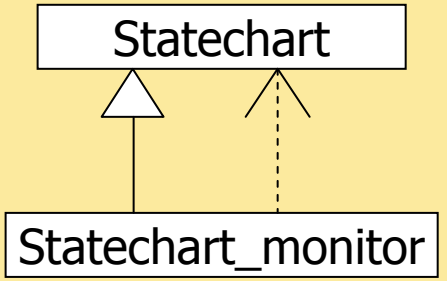
The PauWare Engine

- Power the execution of Statecharts models (simulation, verification, implementation)
  - All-terrain Java Implementation (J2EE, J2SE et J2ME)



The PauWare Library

- Declaration of states :
  - Classes Statechart and Statechart_monitor



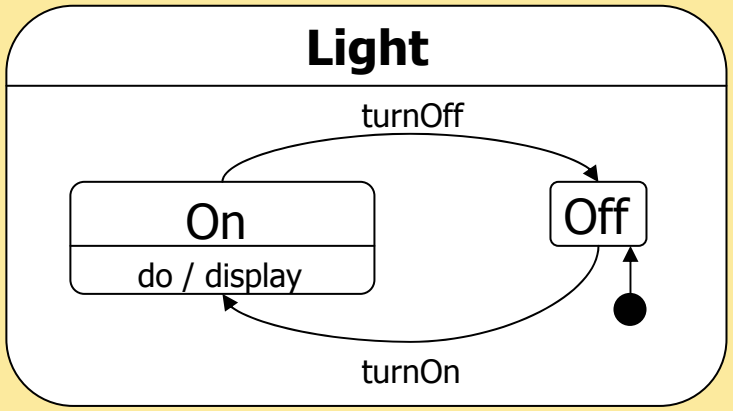
- Composition of states :
  - XOR and AND operators
- Declaration of transitions :

```
fires(String event, Statechart from, Statechart to,
      boolean guard, Object target, String action, Object[] args);
```

- Generation of events :

```
_statechart_monitor.run_to_completion("event");
```

Example : Light




```
/* UML statecharts */
Statechart _On = new Statechart("On");
_On.doActivity(this,"display");

Statechart _Off = new Statechart("Off");
_Off.inputState();

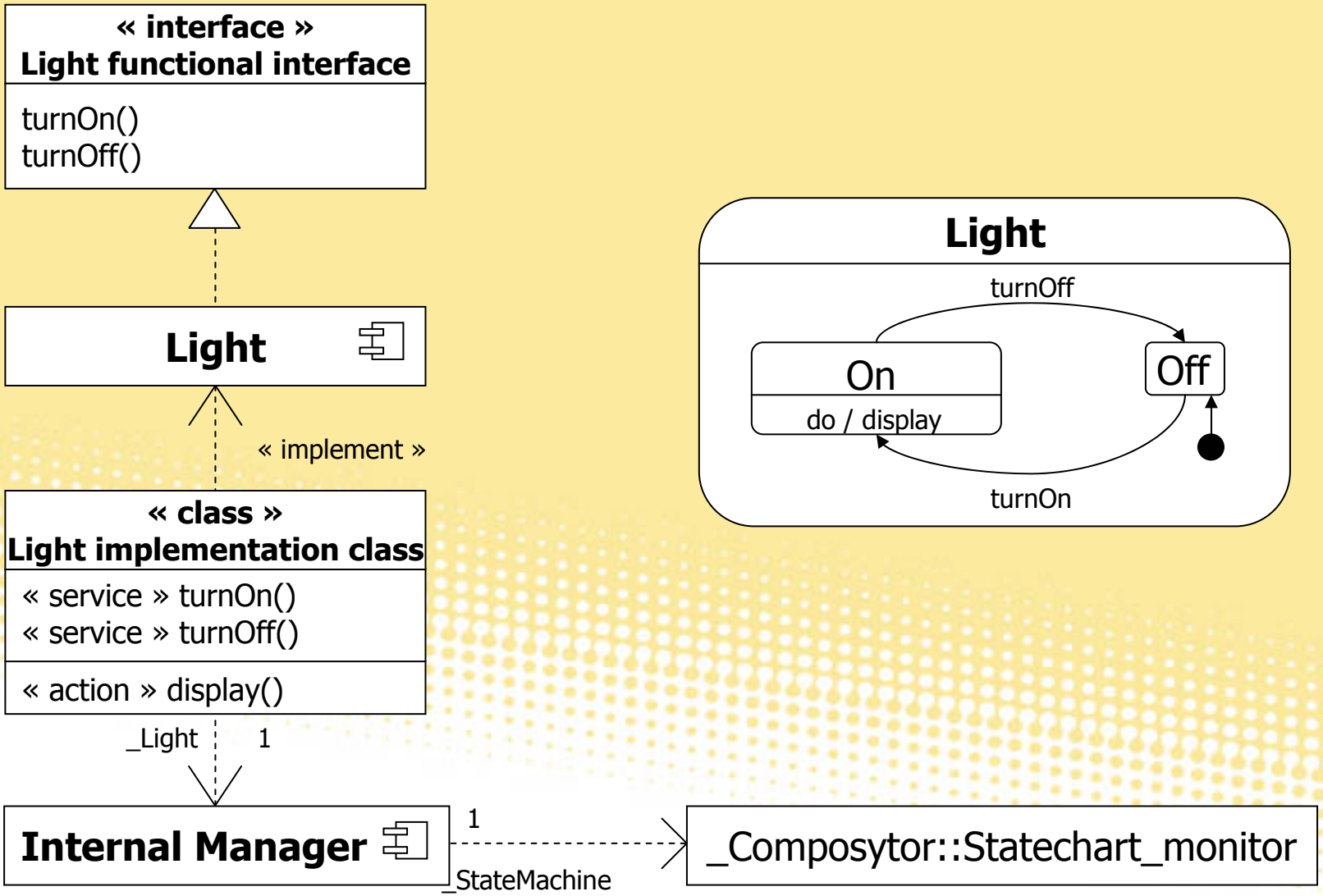
/* PauWare engine */
Statechart_monitor _Light = new Statechart_monitor(_On.xor(_Off),"Light");

/* UML transitions */
_Light.fires("turnOn",_Off,_On);
_Light.fires("turnOff",_On,_Off);
```

/* the code is incomplete */

```
/* UML events */
public void turnOn() throws Statechart_exception {
   _Light.run_to_completion("turnOn");
}
public void turnOff() throws Statechart_exception {
    _Light.run_to_completion("turnOff");
}
```

PauWare Component Model



PauWare → WMX

- PauWare with JMX provides a management framework for non-wireless components
- Too much for constrained devices

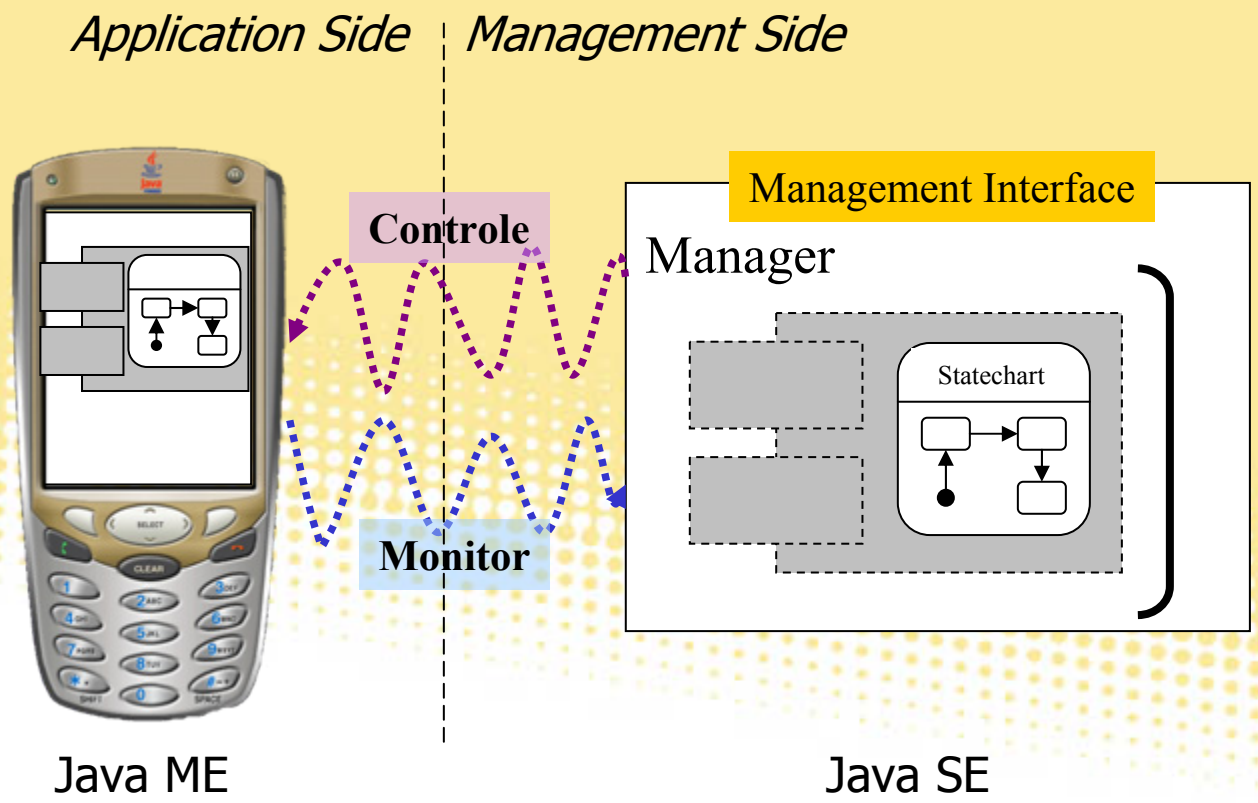
Java ME

Not feasible

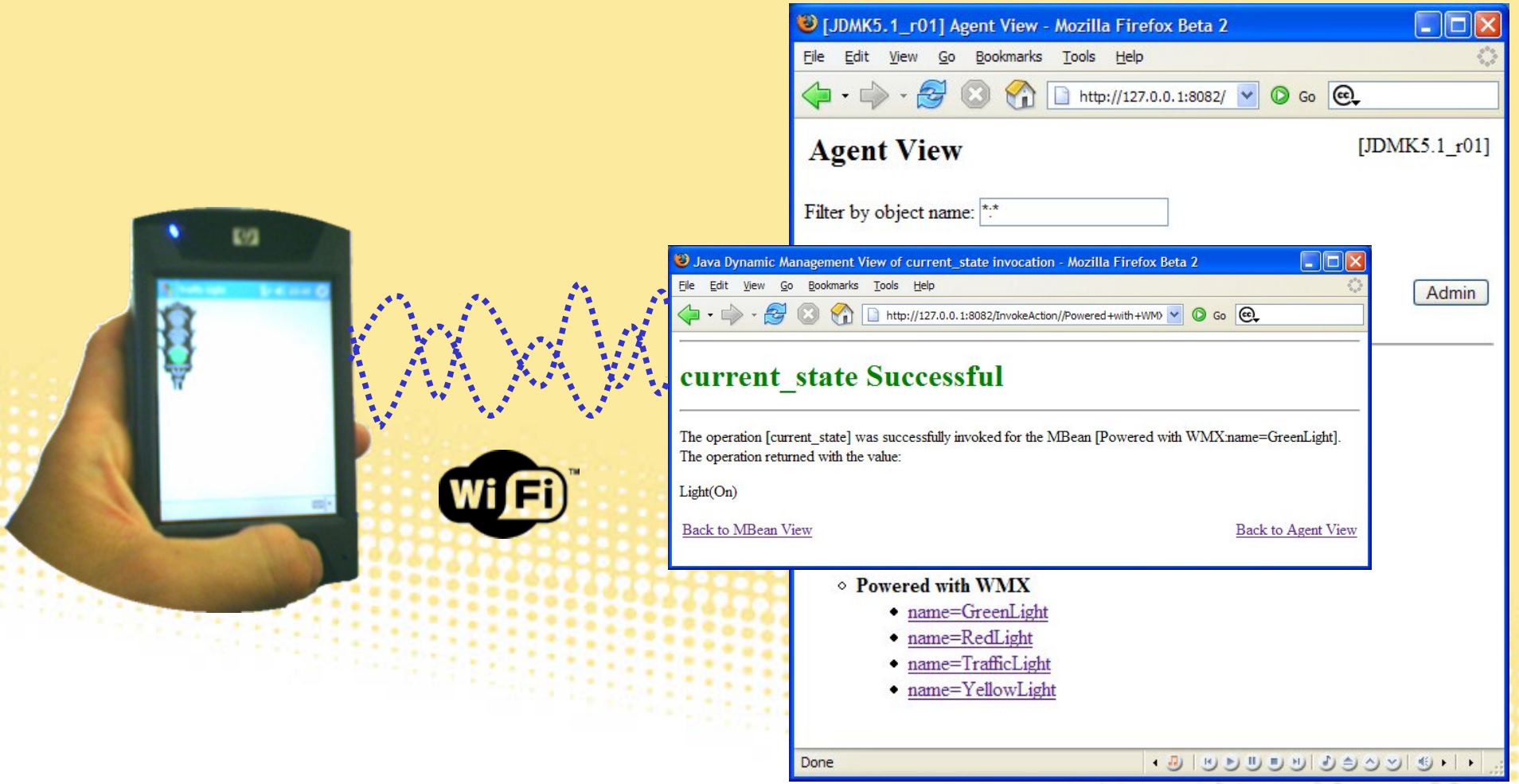
- PauWare
- Java Reflexion Mechanisms
- JMX
- Web Server

WMX

- Wireless Management eXtensions
  - http://wmx.fromeo.fr

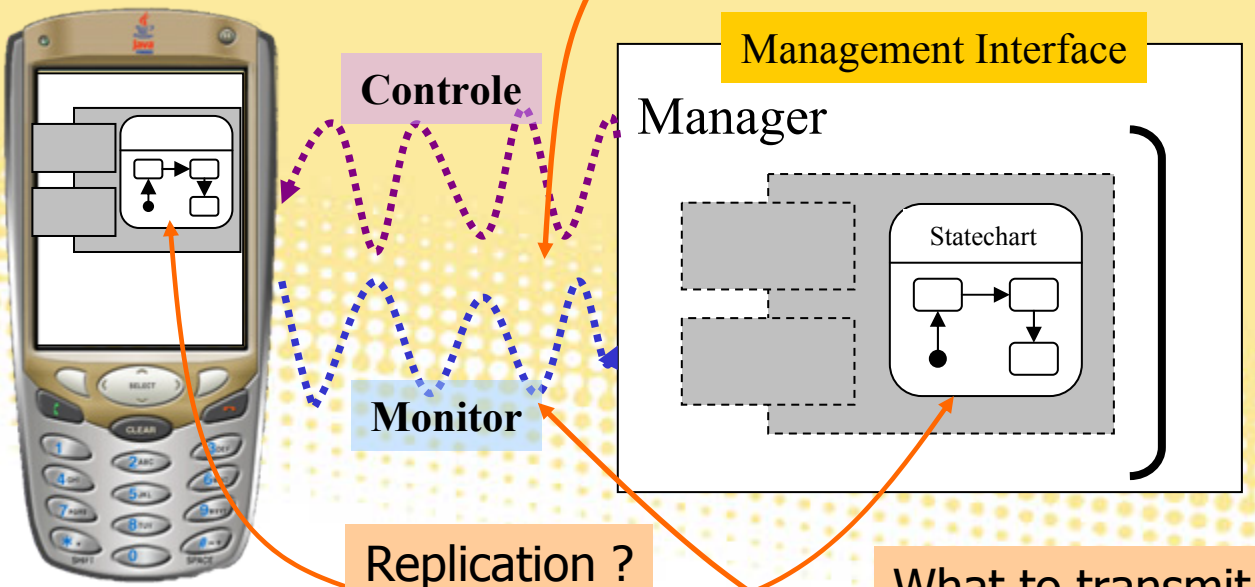


Demo : Traffic Light

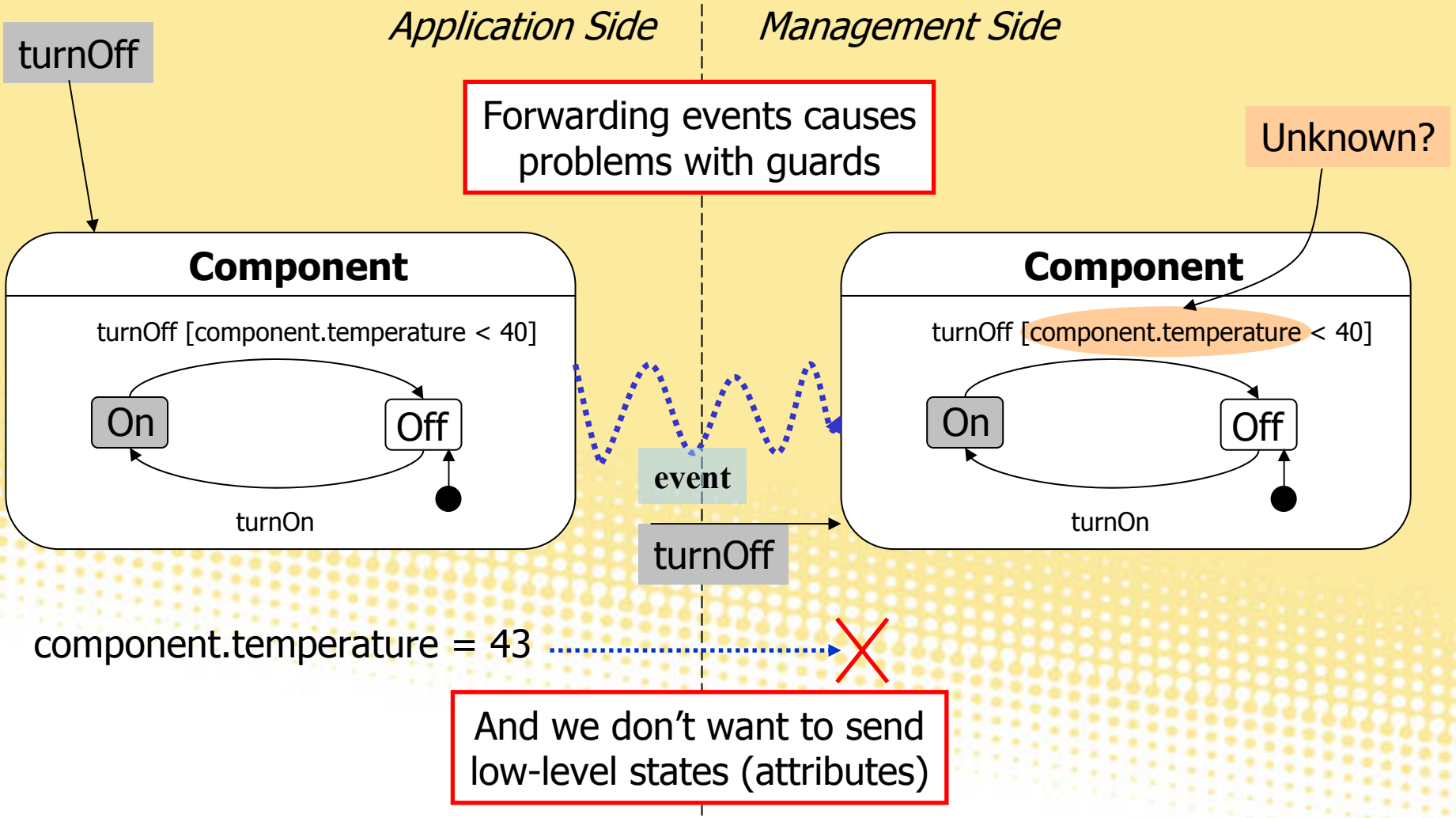


WMX: new problems

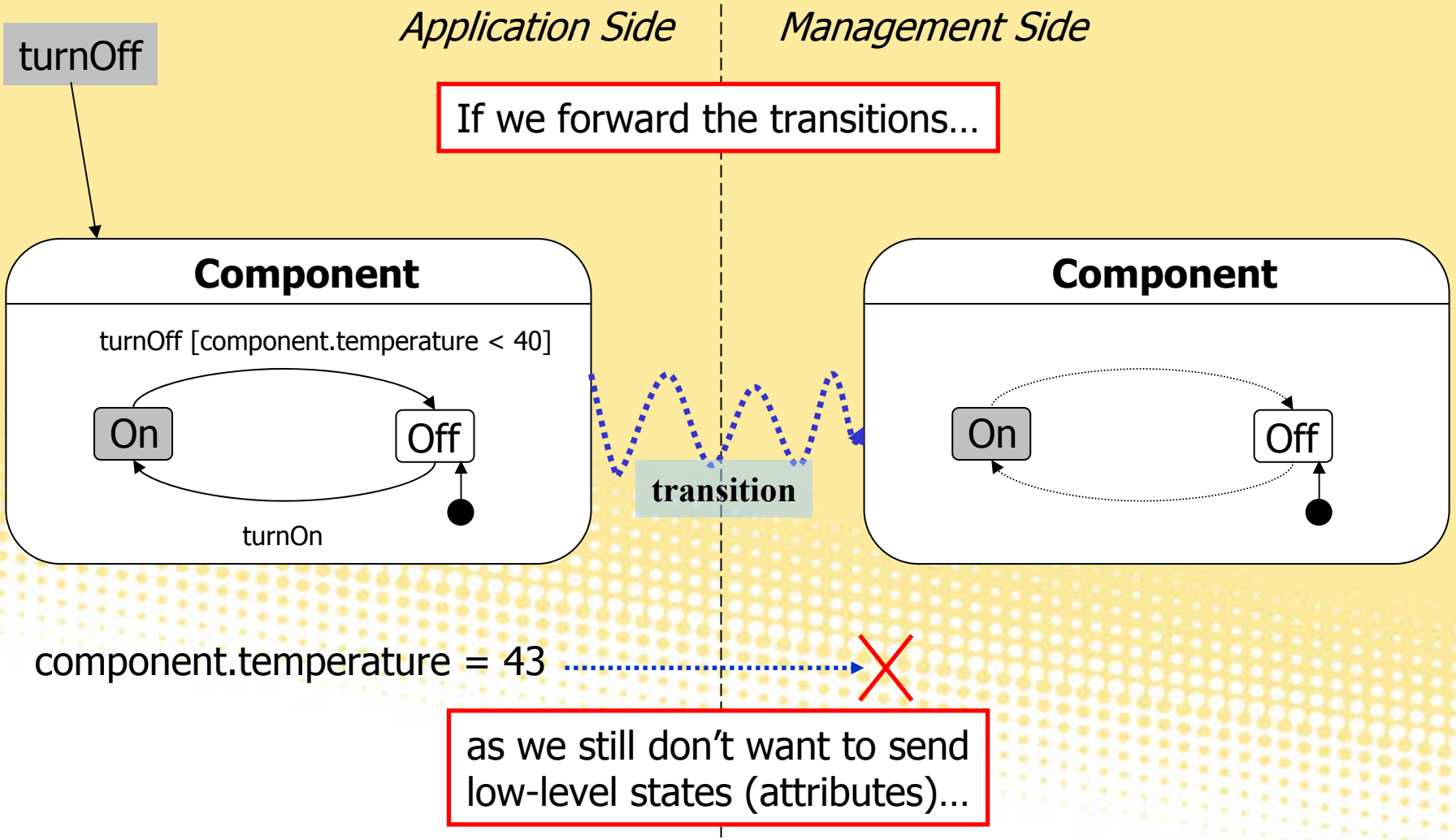
- Wireless Management eXtensions
  - http://wmx.fromeo.fr



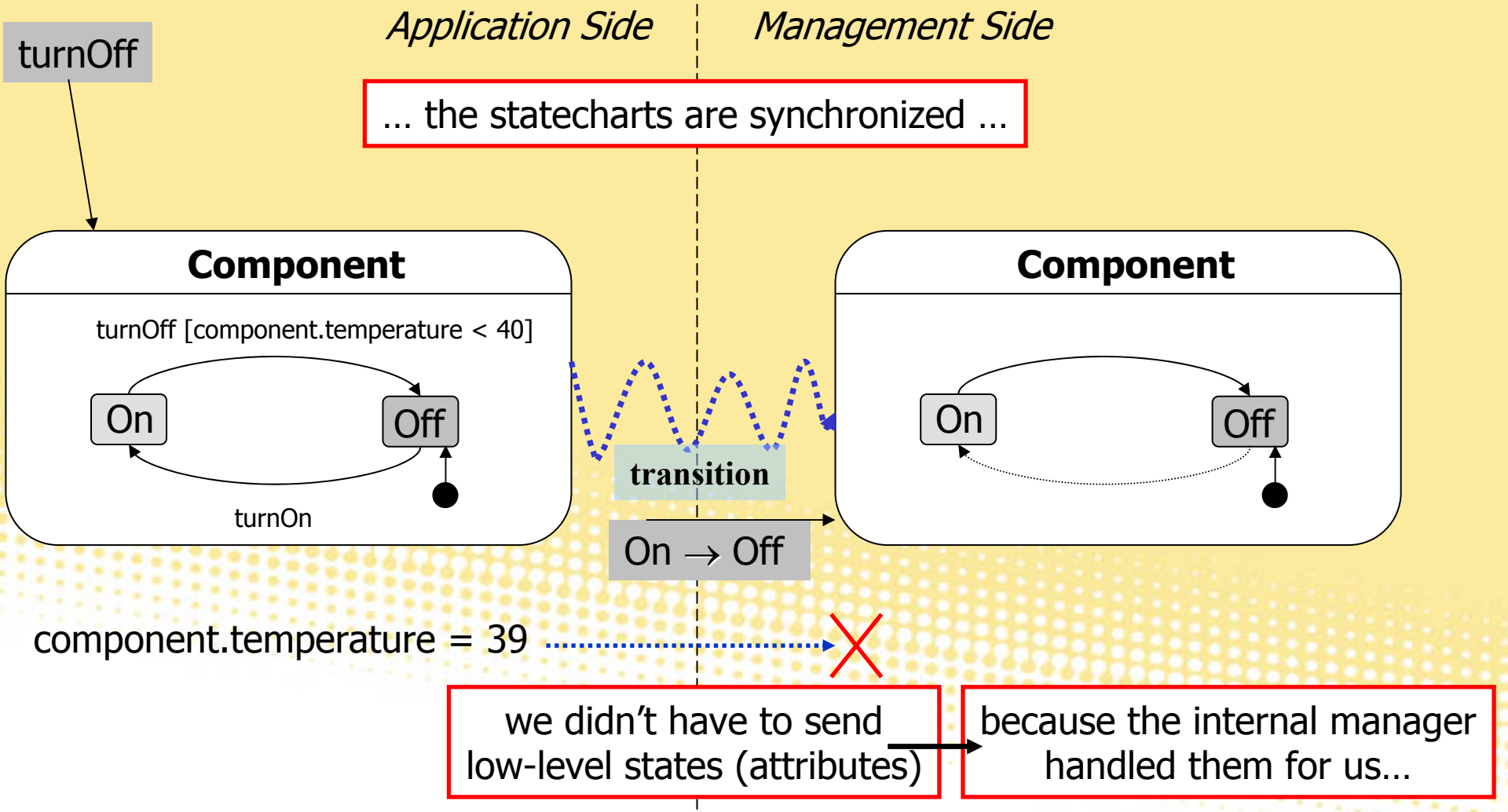
Coherence of statecharts (1)



Coherence of statecharts (2)



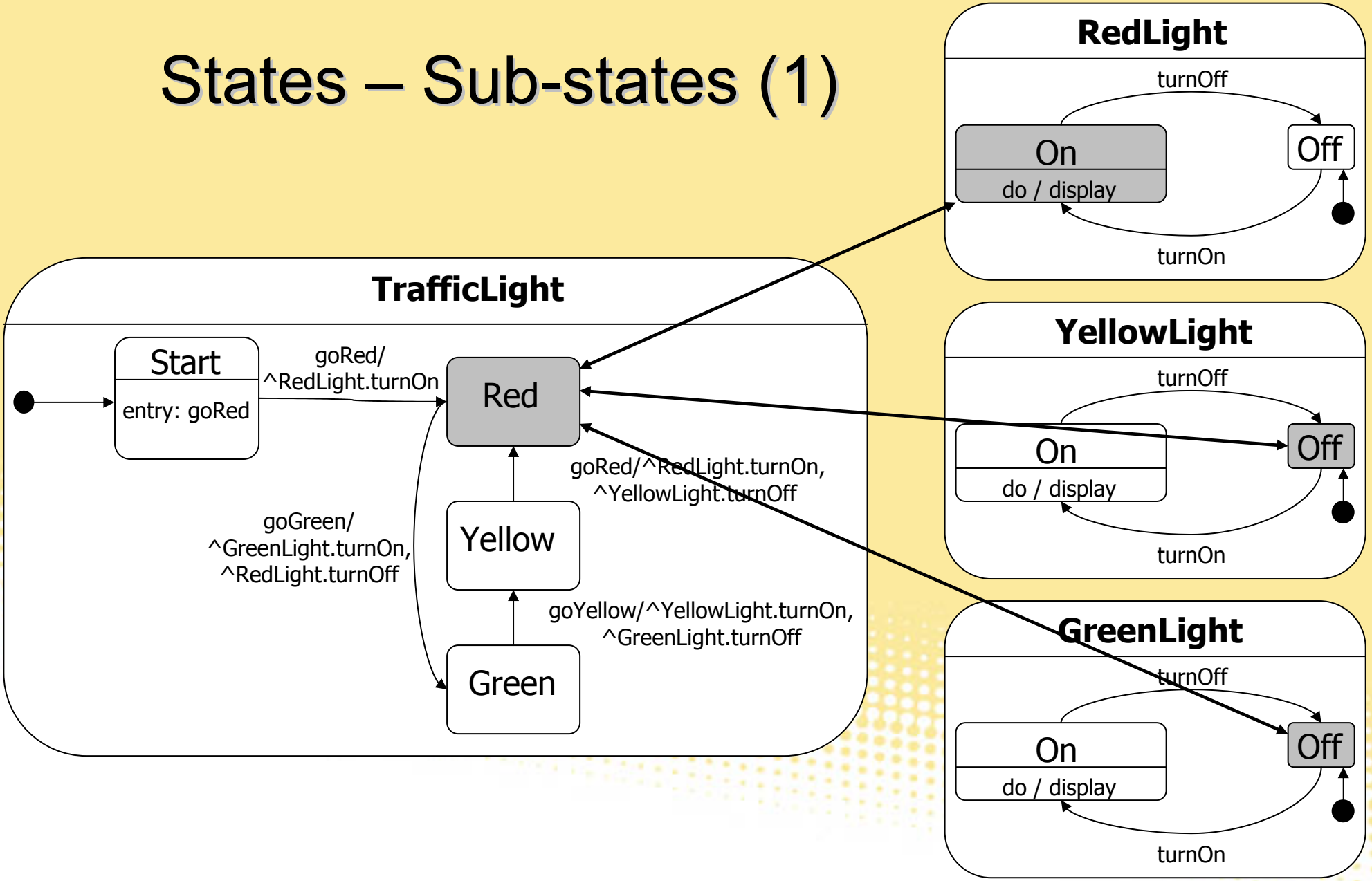
Coherence of statecharts (3)



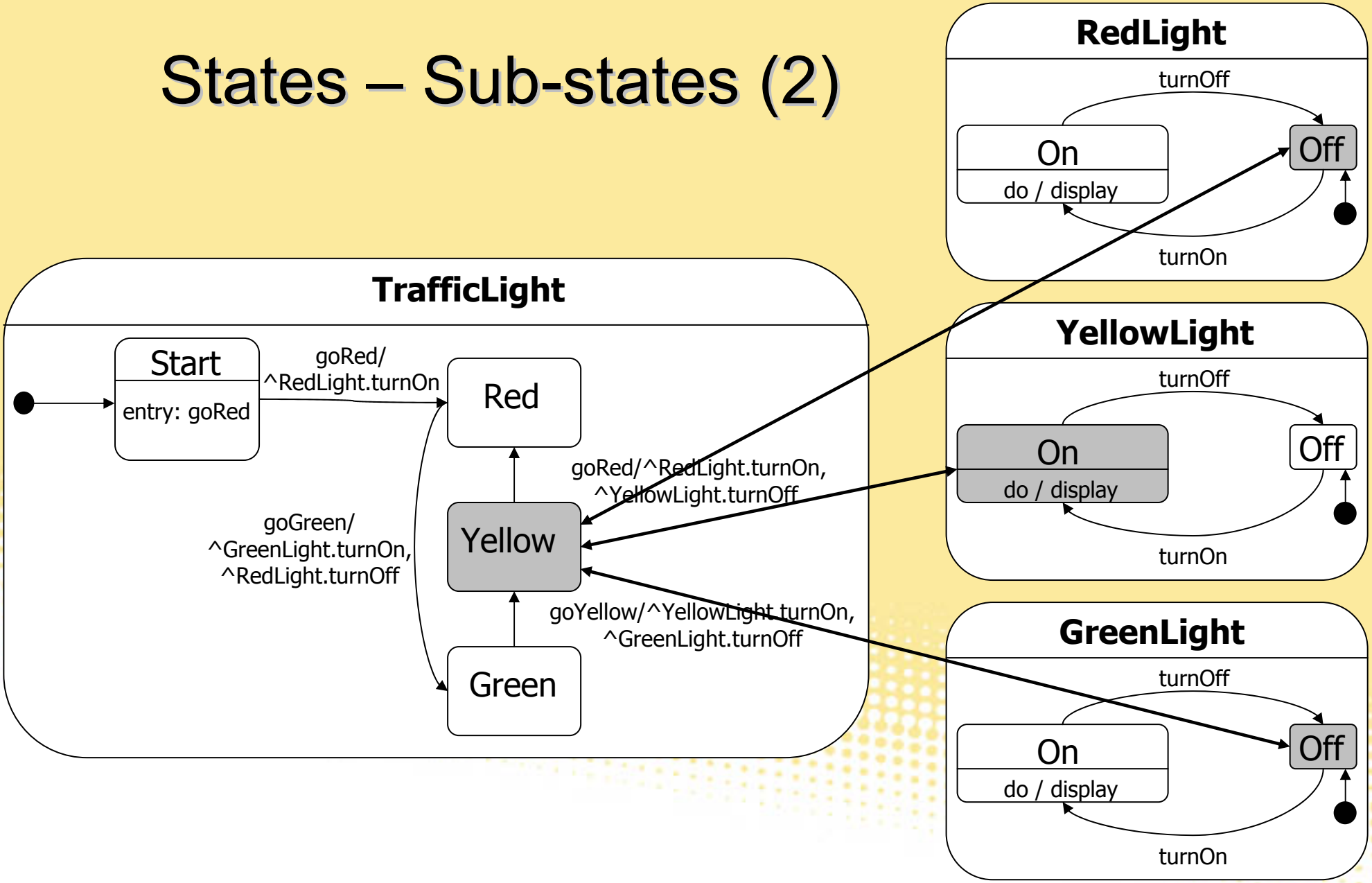
Composition Management

- Desynchronization of automates (example : Traffic Light)
- Undetectable Error by Model Checking
  - It is supposed to work
- Shows the possibility to define management policies based on the replicated statechart

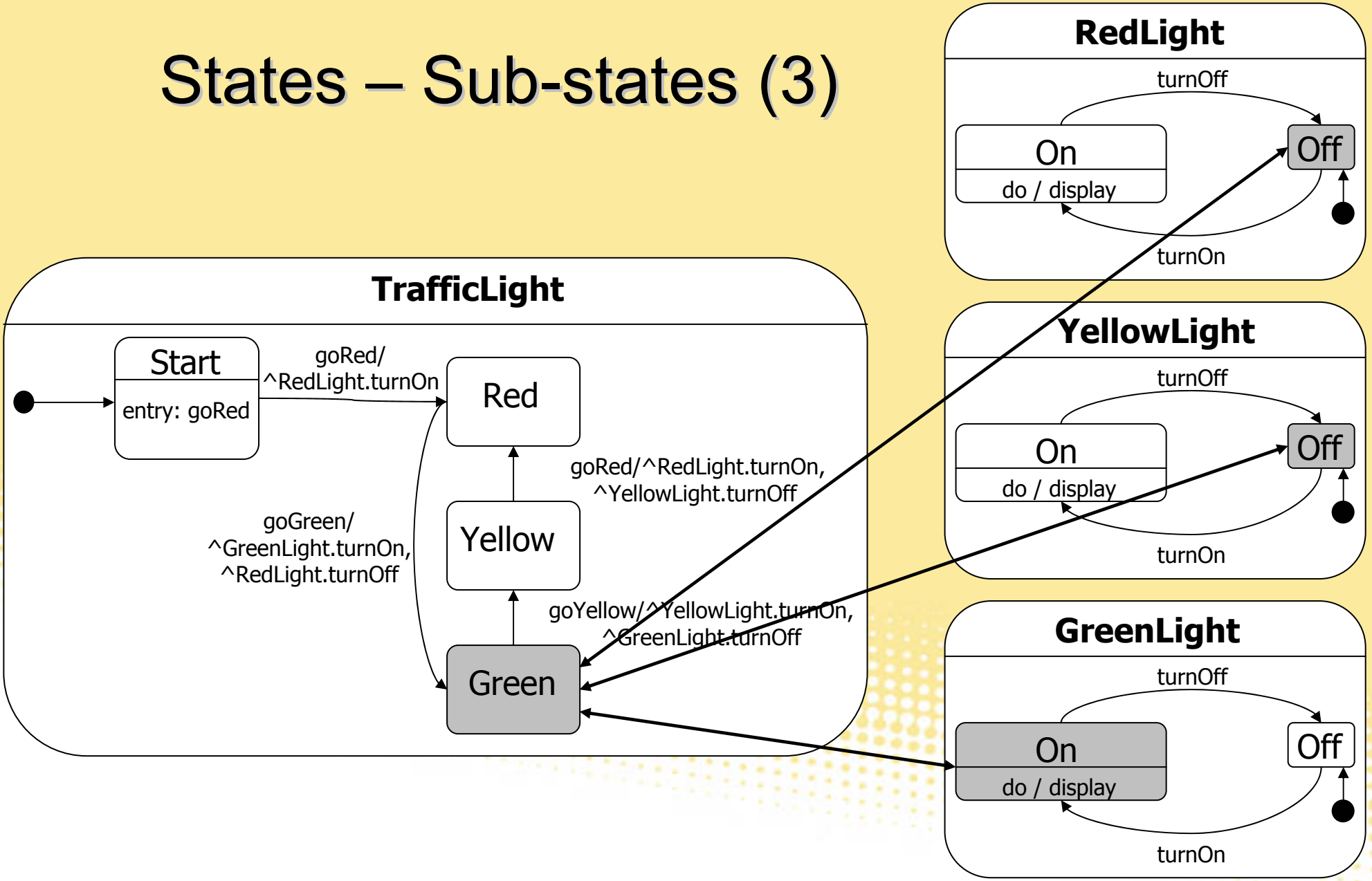
States – Sub-states (1)



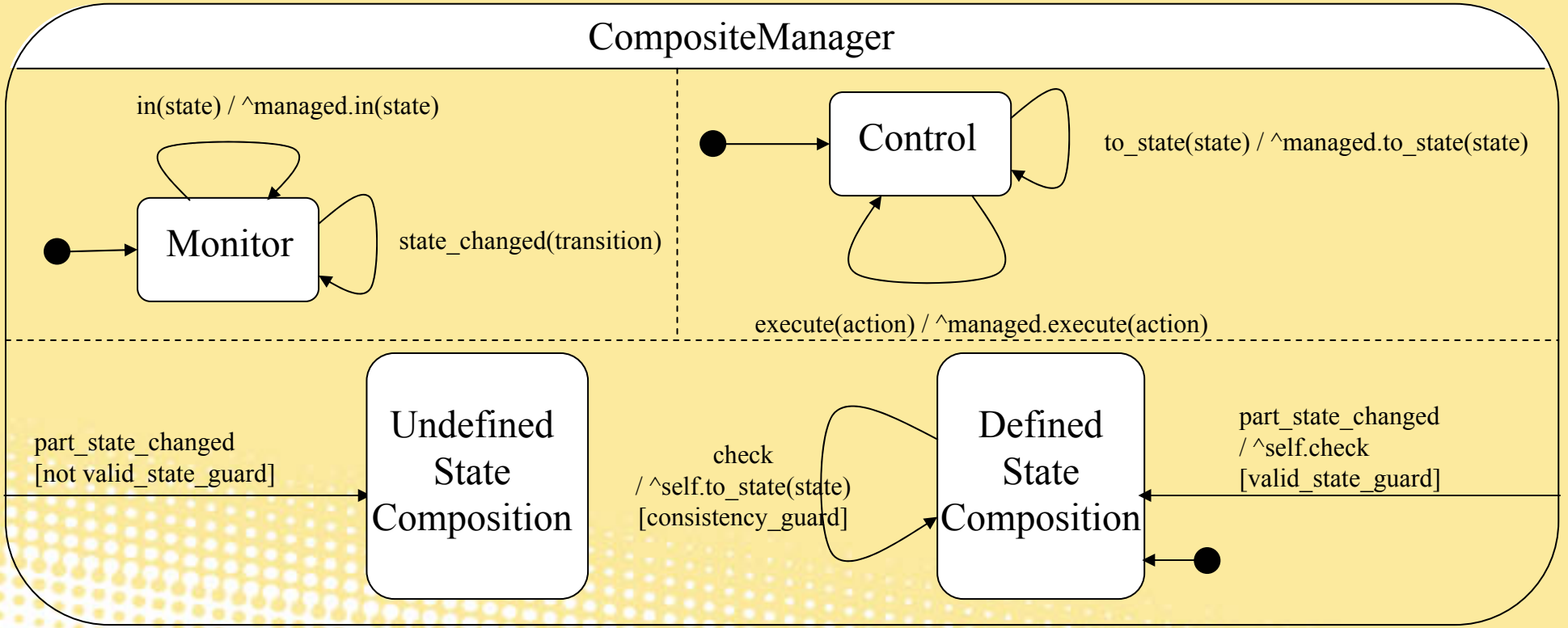
States – Sub-states (2)



States – Sub-states (3)



Composite Manager

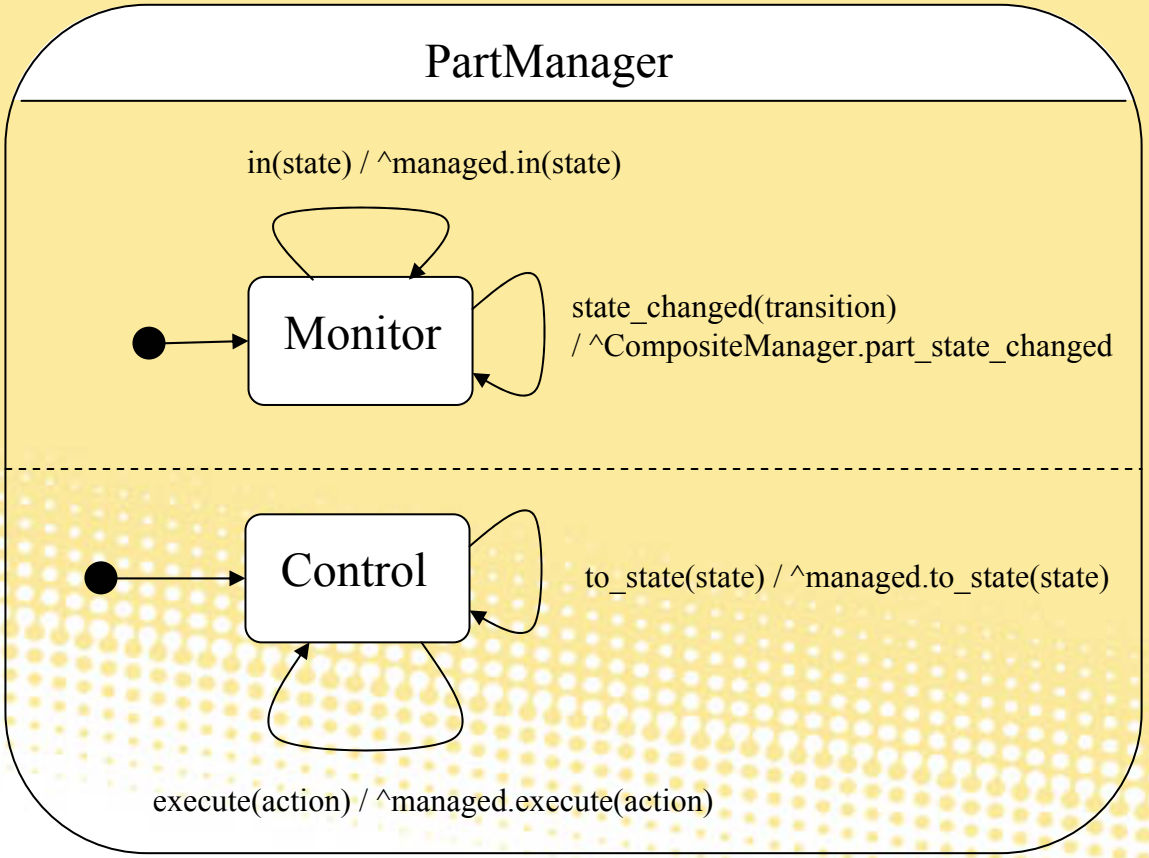


valid_state_guard = [(part1Manager.managedIn(part1ComposedState1)
&& … && partNManager.managedIn(partNComposedState1))
|| … || (part1Manager.managedIn(part1ComposedStateN)
&& … && partNManager.managedIn(partNComposedStateN))]

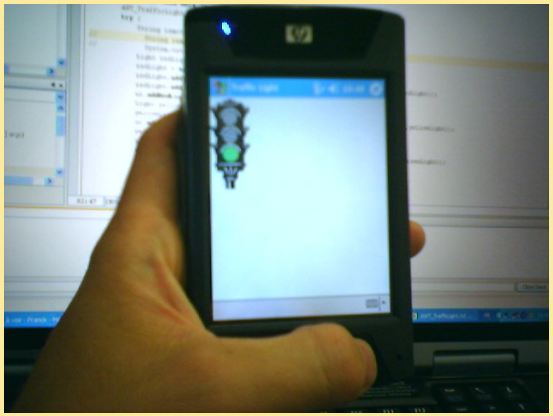
consistency_guard = [(!(state = compositeState1) &&
!(self.managedIn(compositeState1))
&& part1Manager.managedIn(part1ComposedState1)
&& … && partNManager.managedIn(partNComposedState1))
|| … || (!(state = compositeStateN) && !(self.managedIn(compositeStateN))
&& part1Manager.managedIn(part1ComposedStateN)
&& … && partNManager.managedIn(partNComposedStateN))]

fabien.romeo@fromeo.fr www.fromeo.fr

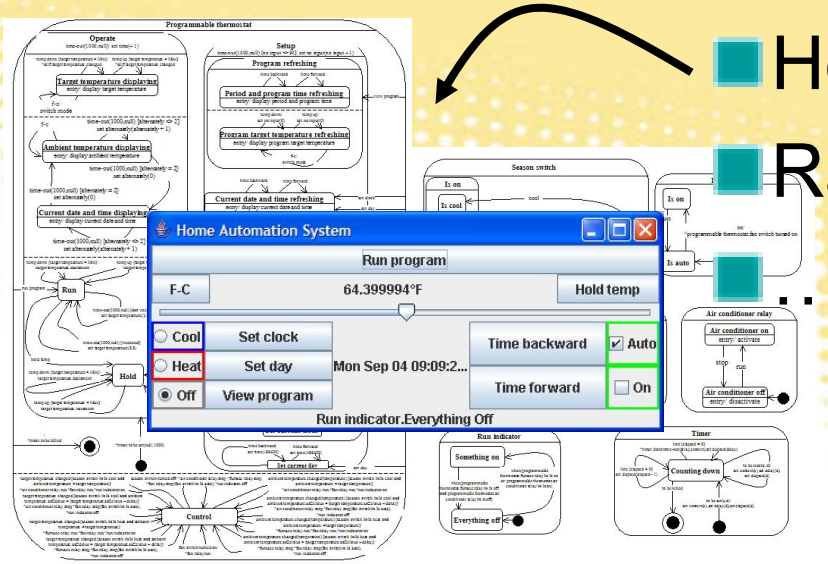
Part Manager

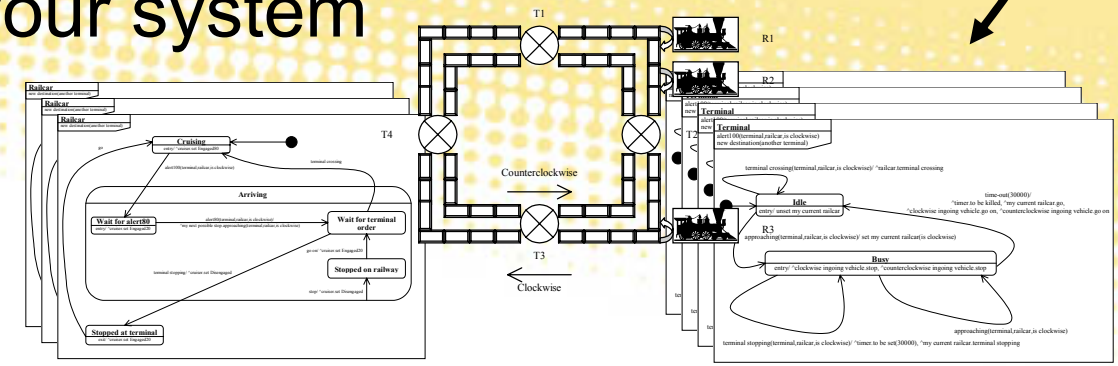


Performances?



- Deployed on a HP iPAQ hx4700
  - JVM IBM J9
- Complex Case Studies :
  - Home automation system
  - Railcar System (J2EE + JMS)
  - …, your system





fabien.romeo@fromeo.fr www.fromeo.fr

Performances – Quantitative Study

- Benchmark
  - 100.000 loops of state changes (adapted from JAC Benchmark)
  - Intel Centrino 1600MHz, 512 Mo RAM, WinXP, JVM 1.5 SUN

Implementation	Benchmark	Overhead per state change
Pure Java	2 ms	0 µs
Java + reflect API	14 ms	0,12 µs
JMX (internal access)	721ms	7,19 µs
PauWare (w/o cache)	1491 ms	14,89 µs
PauWare (w cache)	1027 ms	10,25 µs
Velcro (w/o cache)	1529 ms	15,27 µs
Velcro (w cache)	1038 ms	10,36 µs
Following implementations include I/O or networking		
Pure Java + System.out.print()	2584 ms	25,82 µs
WMX (velcro + sockets)	3893 ms	38,91 µs
JMX + RMI connector	22077ms	220,75 µs

Conclusion

- PauWare
  - Components driven by executable statecharts
  - But also a *framework* for experimenting research results in MDA, CBSE, …
- Velcro + WMX
  - Management of the software components' behavior in the context of wireless systems
- Perspectives
  - Integration into other component models (OSGi, Fractal, …)
  - Persistence at runtime of other types of models, (other views for management)
  - Autonomic Computing: a solution to put the human **on** the loop…

[PauWare]

« ~~Power~~ *is not a means, it is an end.* »

– *George Orwell, Nineteen Eighty-Four*

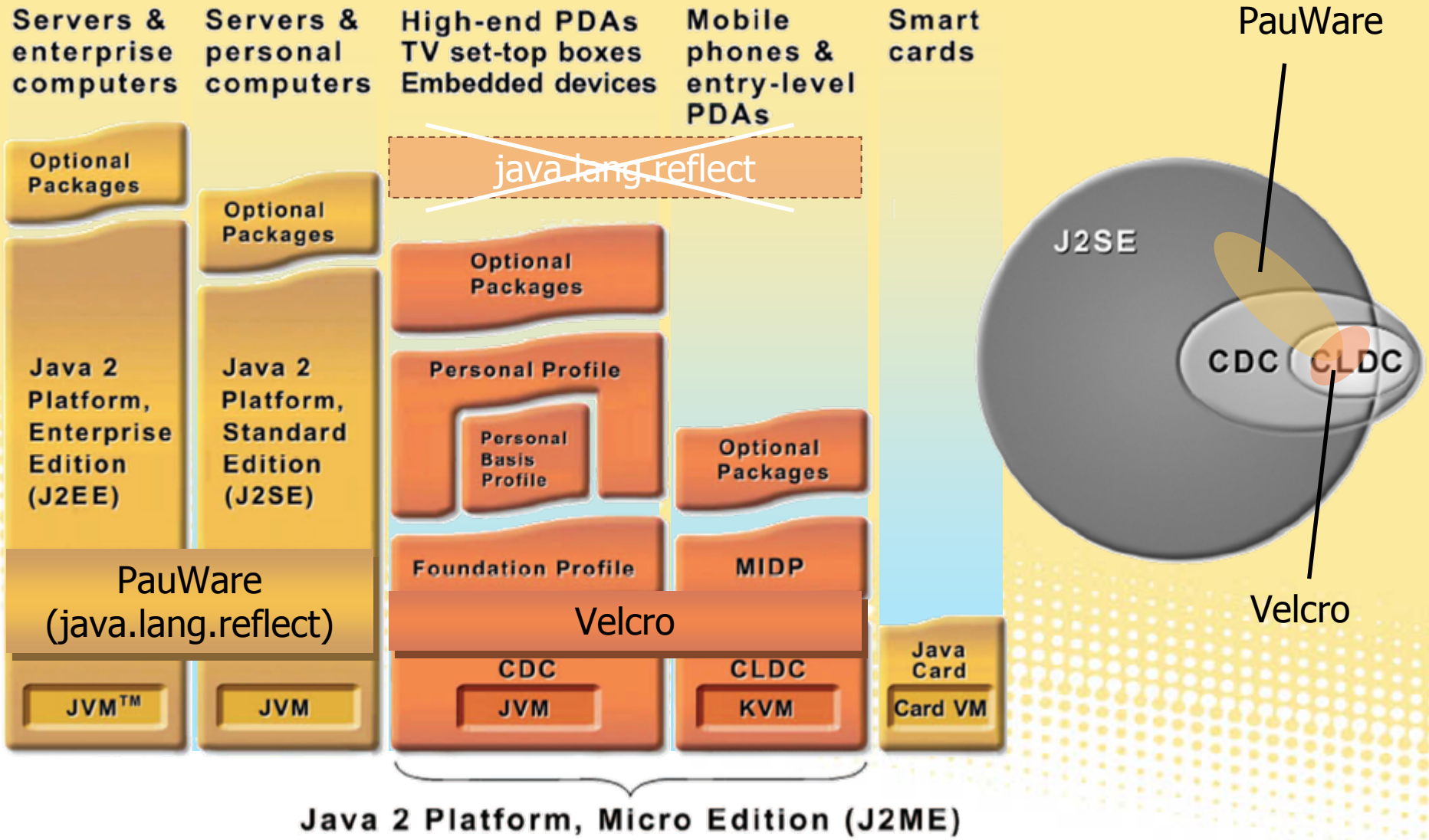
Thank you for listening!

- Fabien Romeo, Franck Barbier, Jean-Michel Bruel, *Observability and Controllability of Wireless Software Components*, 7th IFIP Conference on Distributed Applications and Interoperable Systems, Paphos, Cyprus, 5-8 june, 2007.

- Code available at http://www.pauware.com
  http://wmx.fromeo.fr

Bonus Track…

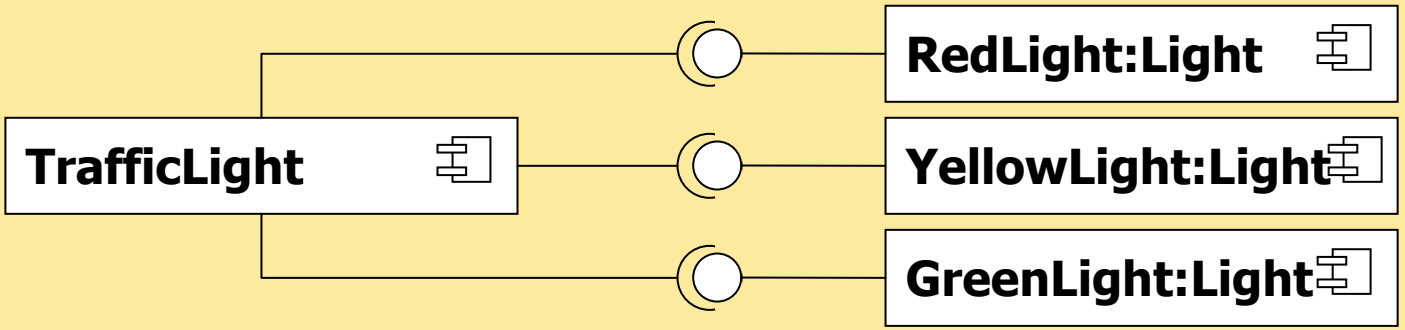
Velcro

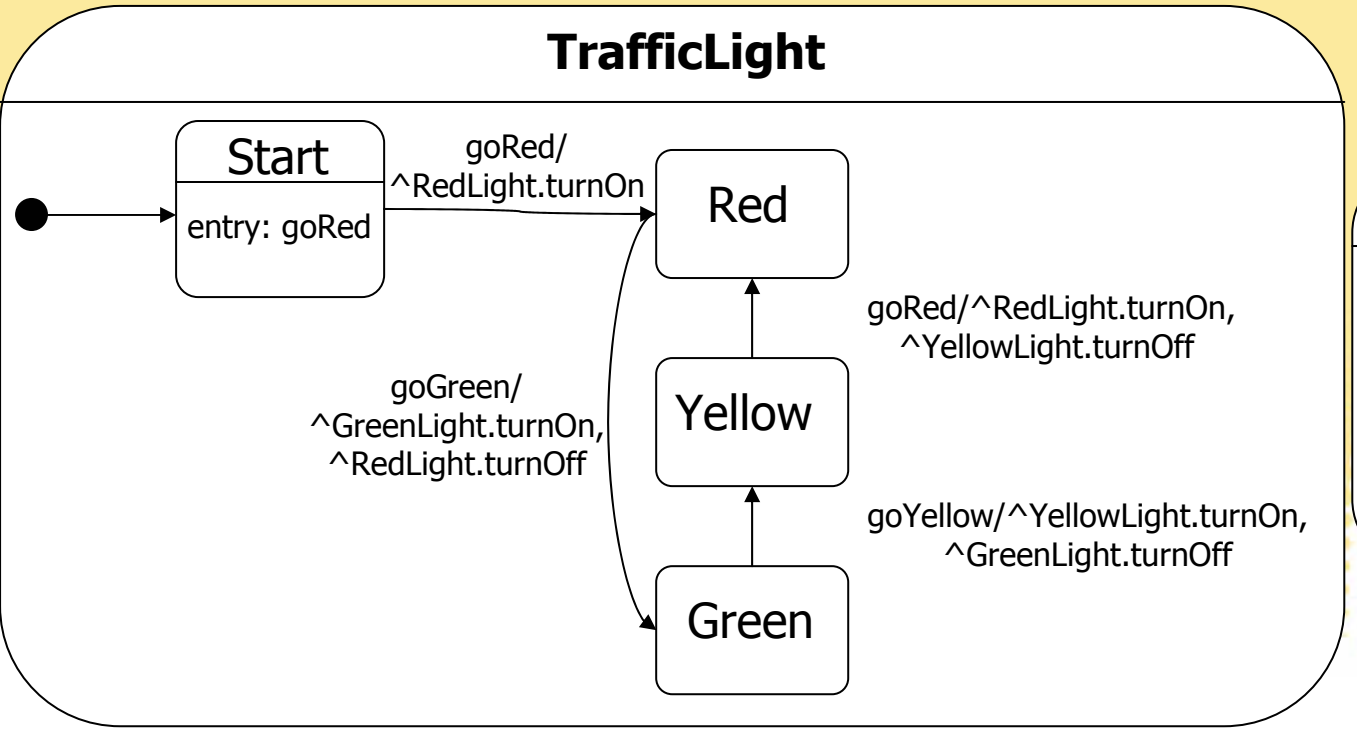

Servers & enterprise computers
Servers & personal computers
High-end PDAs TV set-top boxes Embedded devices
Mobile phones & entry-level PDAs
Smart cards
java.lang.reflect
Optional Packages
Optional Packages
Optional Packages
Optional Packages
Java 2 Platform, Enterprise Edition (J2EE)
Java 2 Platform, Standard Edition (J2SE)
Personal Profile
Personal Basis Profile
Foundation Profile
MIDP
PauWare (java.lang.reflect)
Velcro
JVM™
JVM
CDC
JVM
CLDC
KVM
Java Card
Card VM
Java 2 Platform, Micro Edition (J2ME)
PauWare
J2SE
CDC
CLDC
Velcro

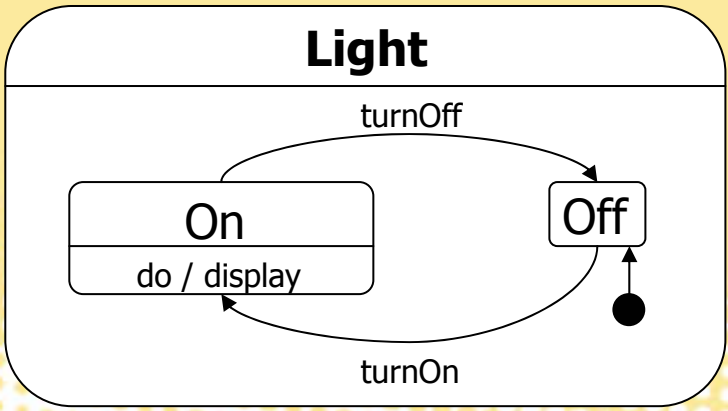
Horizontal Composition

- Components have the same granularity
  - Client / Server or Peer to Peer
- Provided Interfaces / Required Interfaces
- Low coupling
- Distributed Application
- Statecharts Communication by signal
  - JMS, Message-Driven Bean

Example : Traffic Light



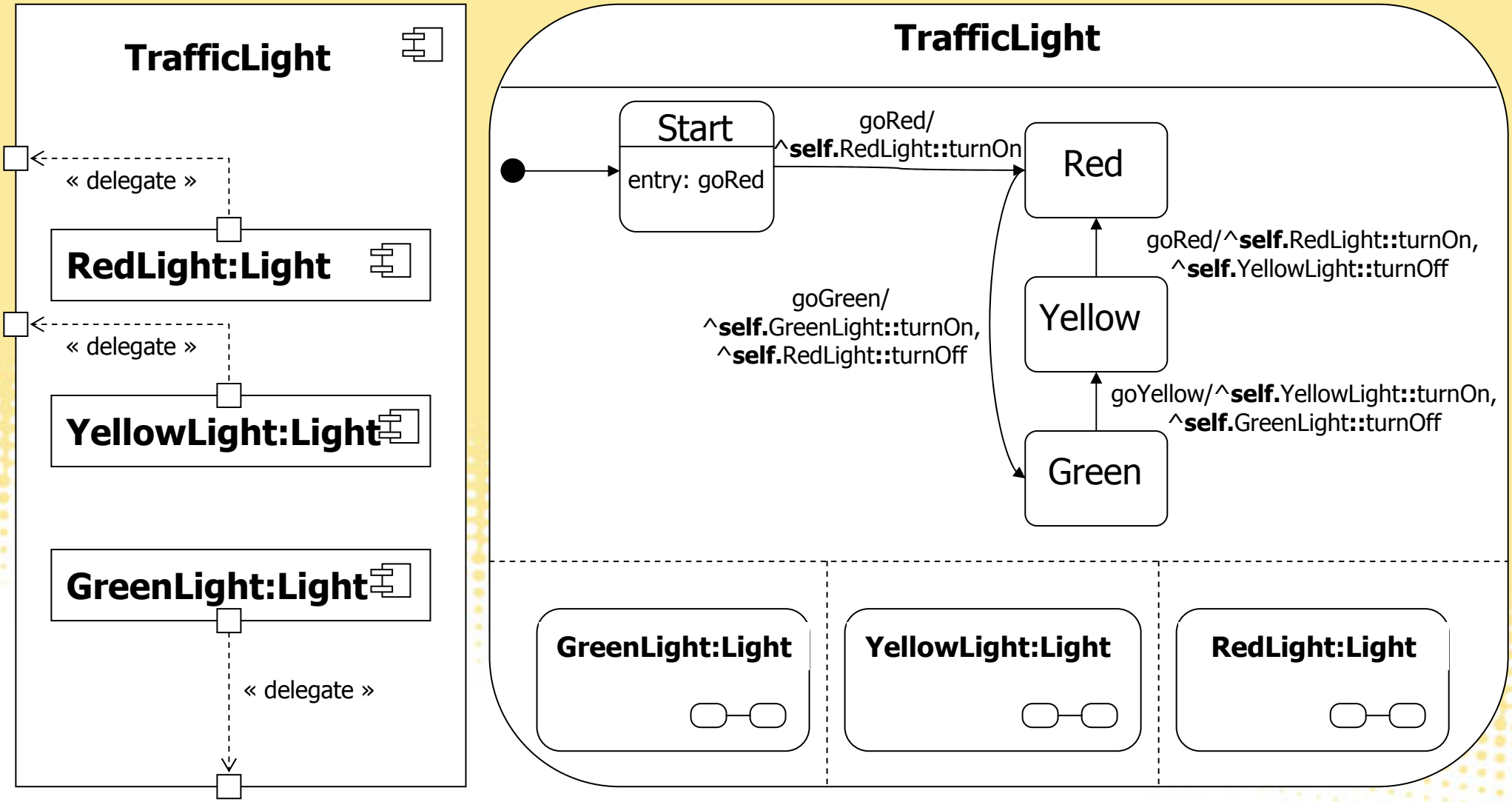




Vertical Composition

- (De)composition composite/compound
  - Hierarchie (cf. Fractal)
- A higher coupling
  - Life-cycle Dependances, encapsulation, …
- Theoretical Foundations
  - Revised formalization of *aggregation* and *composition* in UML (IEEE TSE 29(5) et 29(11) - 2003)
- State Machines of compound components integrated into the State Machine of the composite as concurrent macro states :
  - `public class Light extends _PauWare.Composable {/* ... */}`
  - `Light  redLight, yellowLight, greenLight;`
  - ```
    Statechart_monitor trafficLight =
       new Statechart_monitor(redLight.state_machine()
                              .and(yellowLight.state_machine())
                              .and(greenLight.state_machine())
                              .and(start.xor(red).xor(yellow).xor(green)));
    ```
    ```

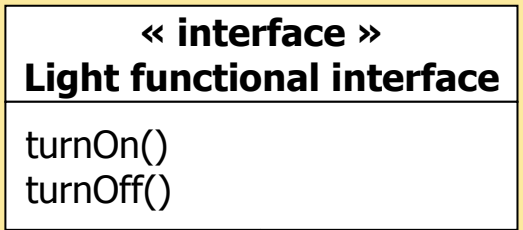
Example : Traffic Light



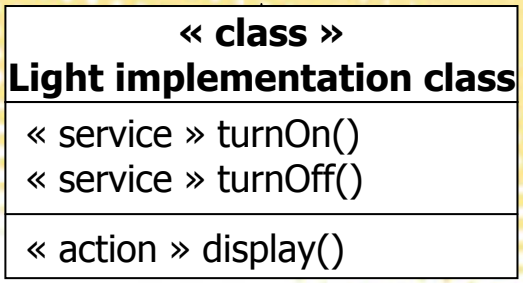
PauWare Component Model

- A State Machine Execution Engine (diagrammes UML 2)
  - Model verification at development time
  - Support for implementation
  - Models persist at runtime
- State Machine → Components ?
  - Structure-Behavior Binding (*i.e.* component-statechart)
  - Support for horizontal composition and vertical composition (hierarchical)

PauWare Component Model

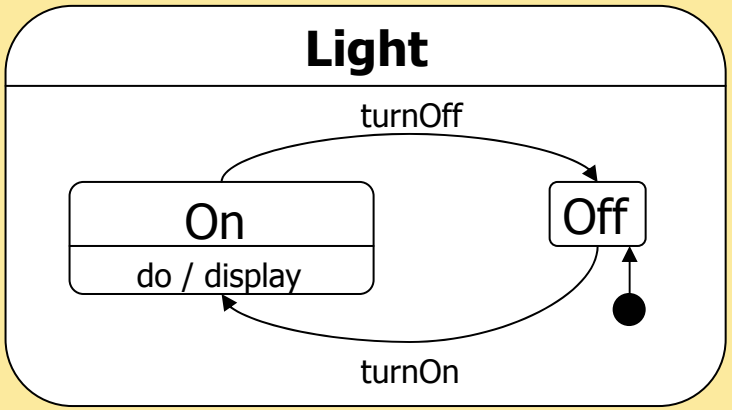


« implement »

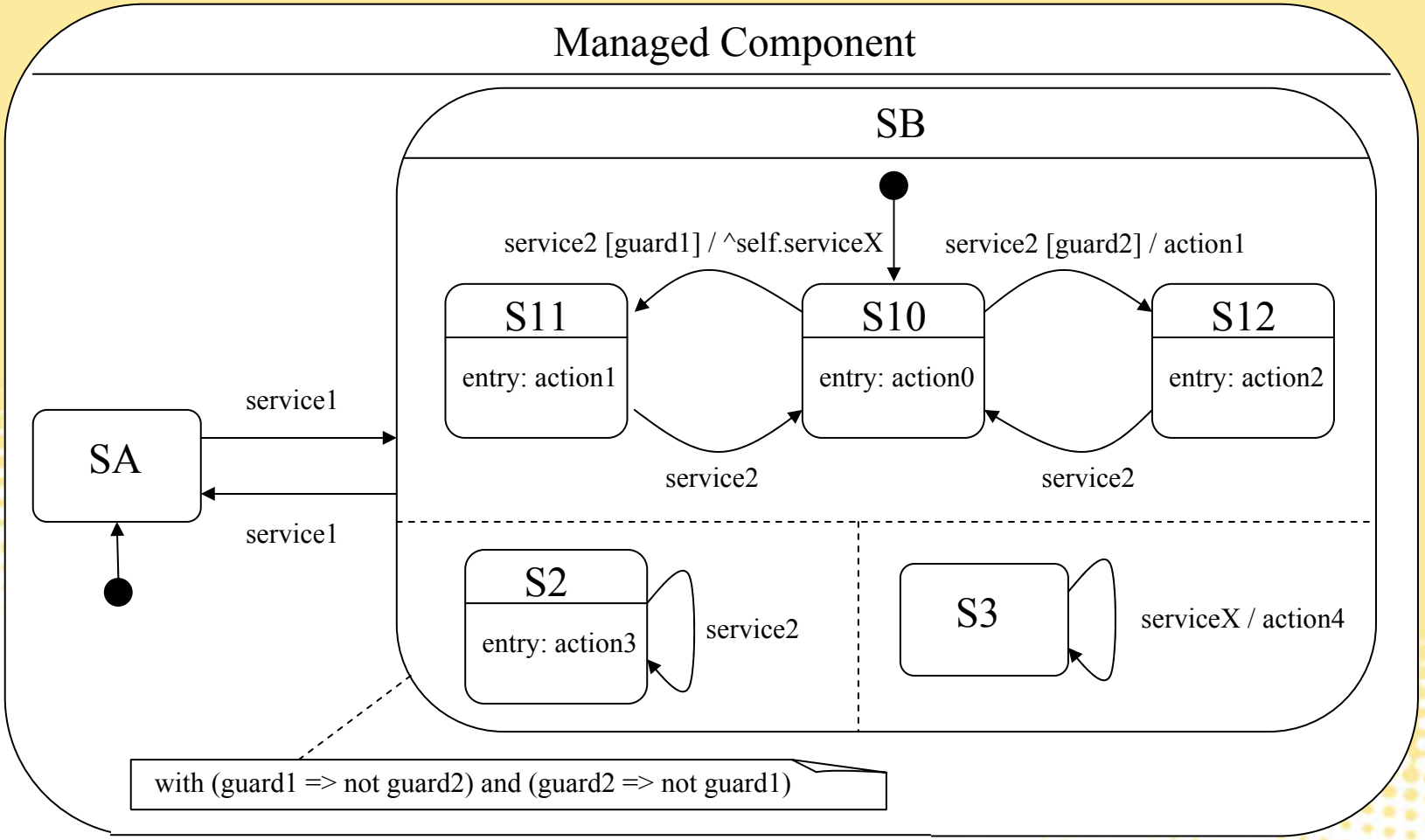


_Light 1

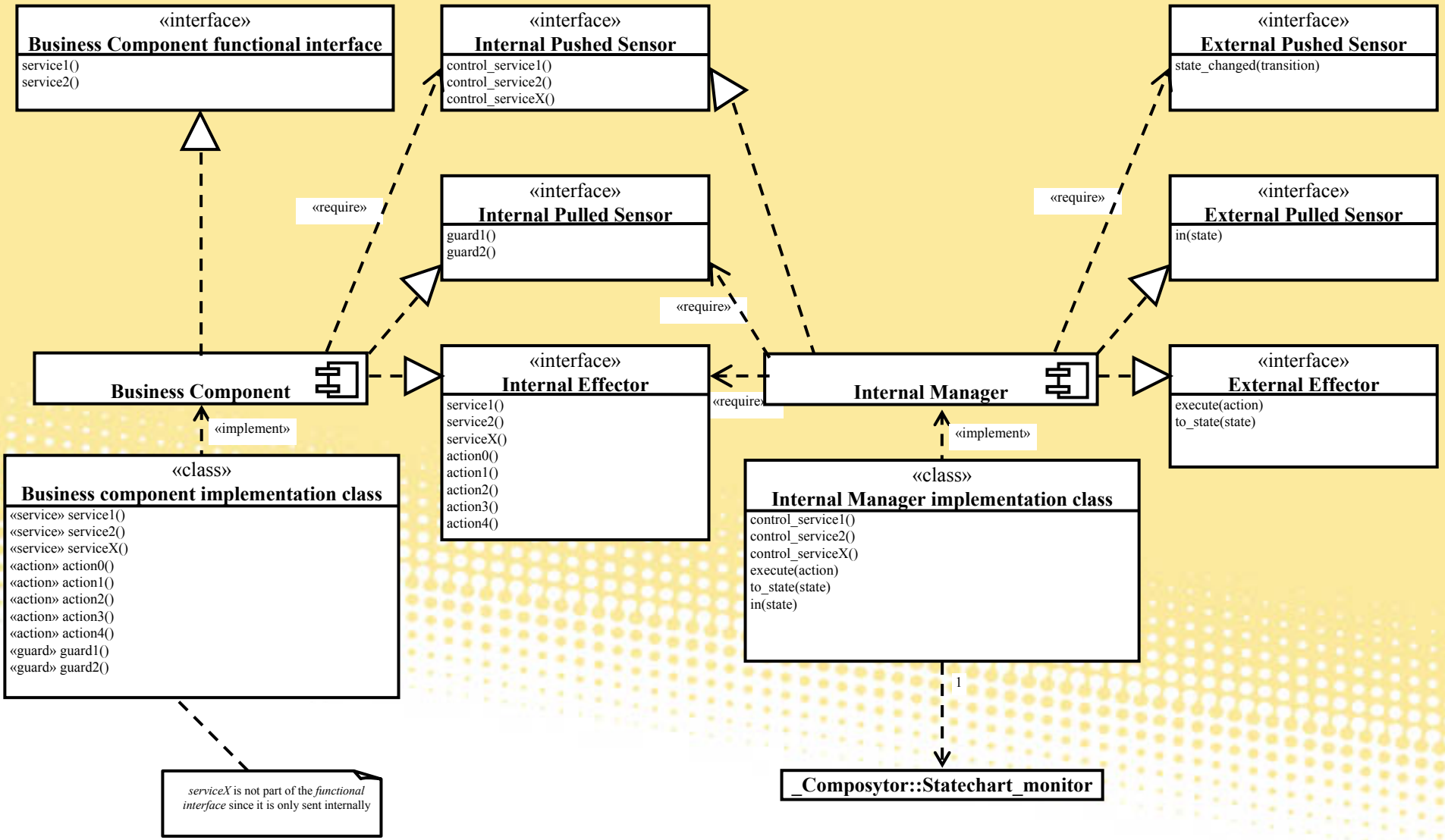
_Composytor::Statechart_monitor



Behavior modeled with Statecharts

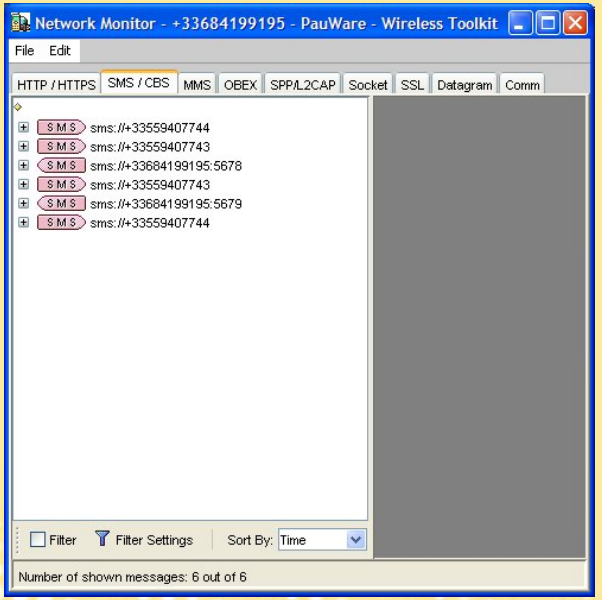


Detailed Architecture

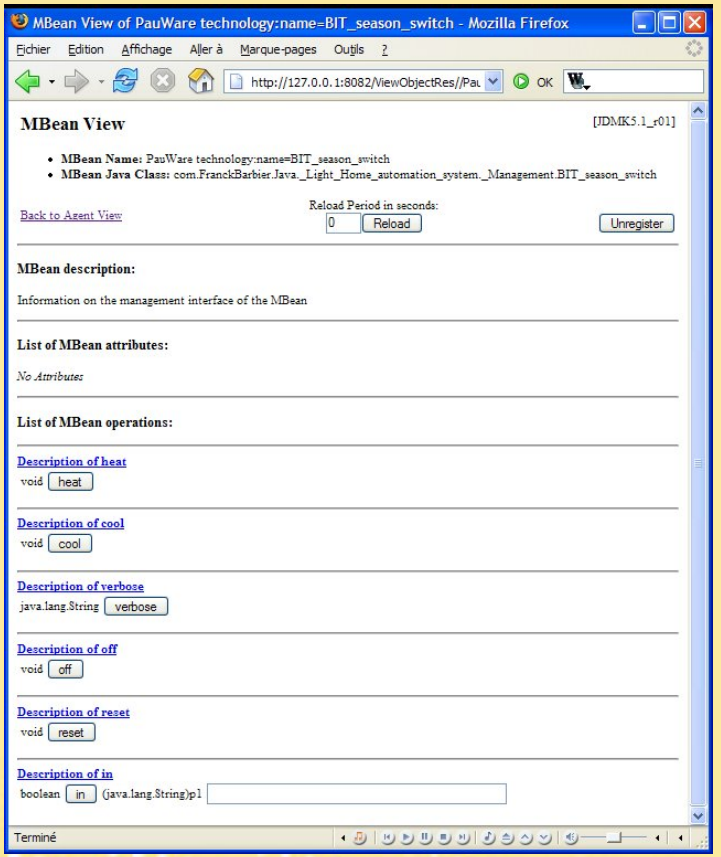


Demo

Wireless Components (J2ME)



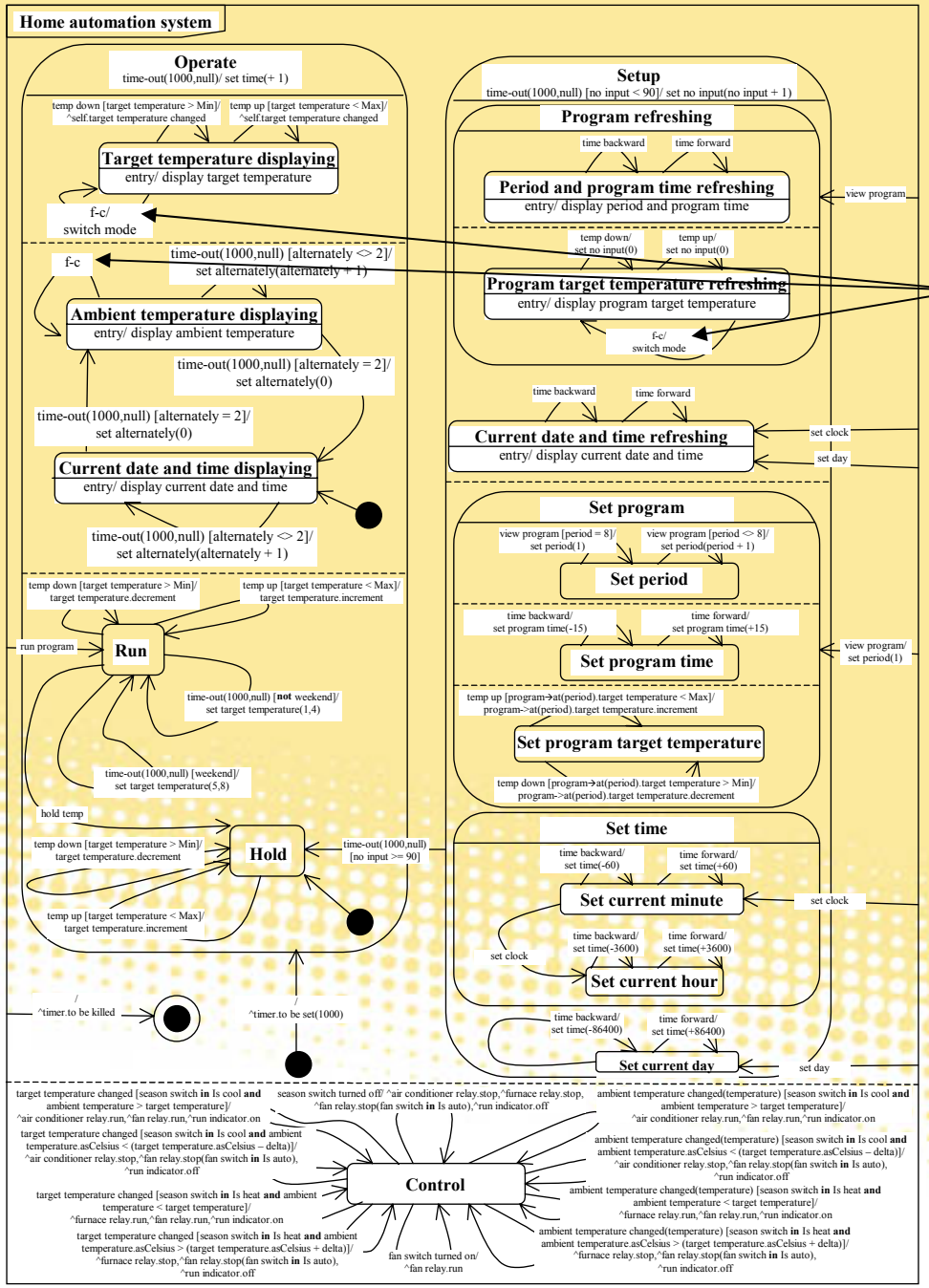
Management Communication (WMA)



Management Console (JMX)

fabien.romeo@fromeo.fr www.fromeo.fr

PauWare Library



Event

State-based programming support

```
public void f_c() throws Statechart_exception {

_Programmable_thermostat.fires(_Ambient_temperature_displaying,_Ambient_temperature_displaying);

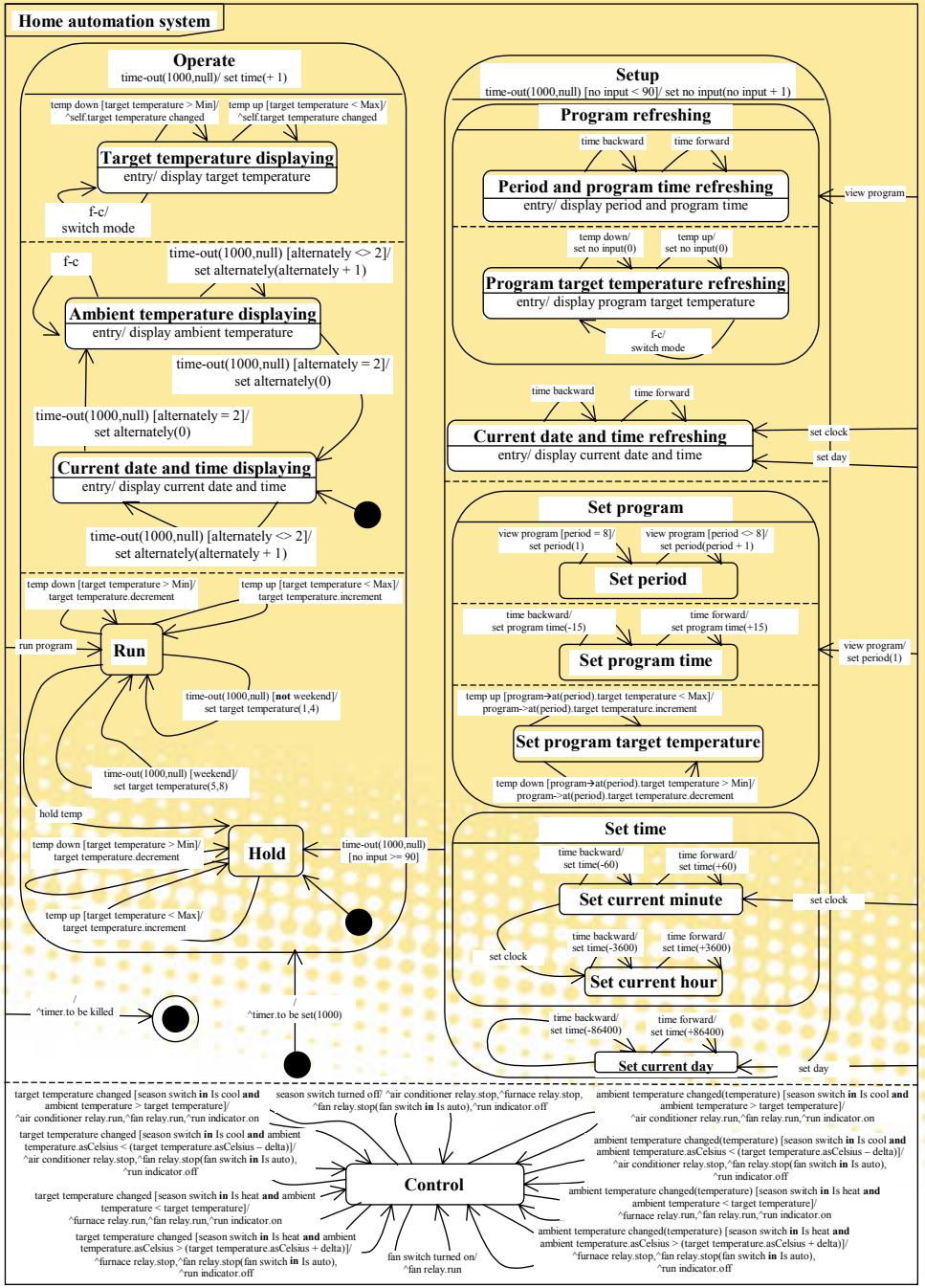
_Programmable_thermostat.fires(_Target_temperature_displaying,_Target_temperature_displaying,true,this,"switch_mode");

_Programmable_thermostat.fires(_Program_target_temperature_refreshing,_Program_target_temperature_refreshing,true,this,"switch_mode");

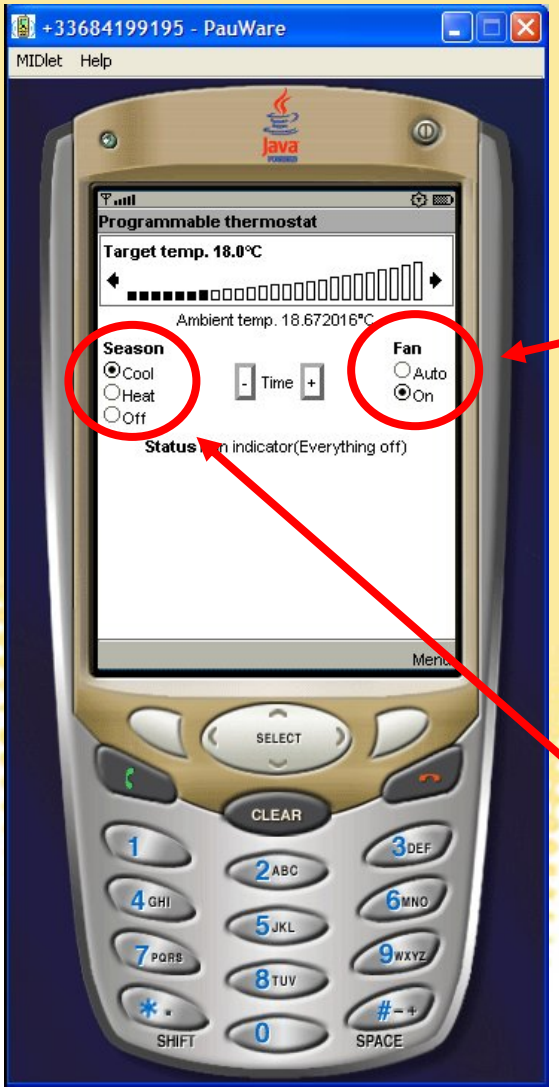
    _Programmable_thermostat.run_to_completion();
}
```

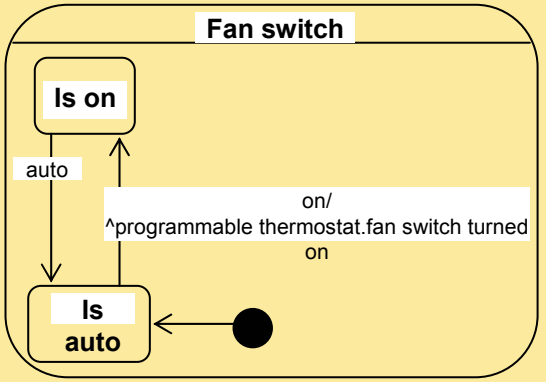
UML-2 *run-to-completion* model execution mode

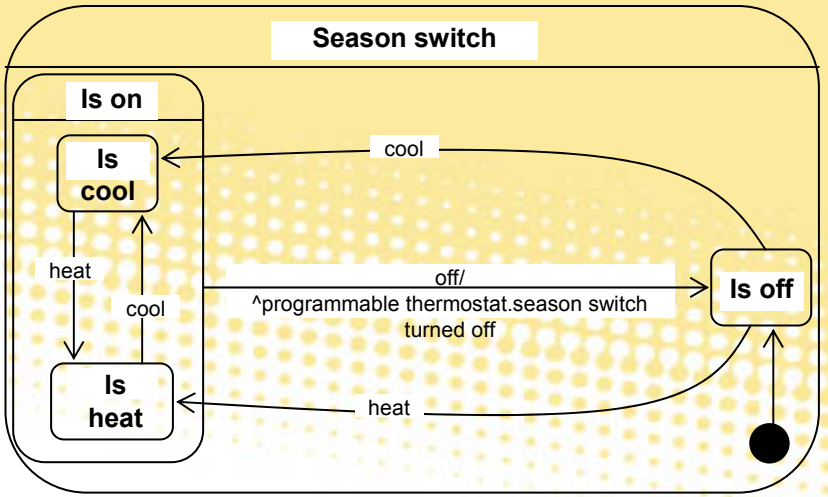
Model Execution



Demo

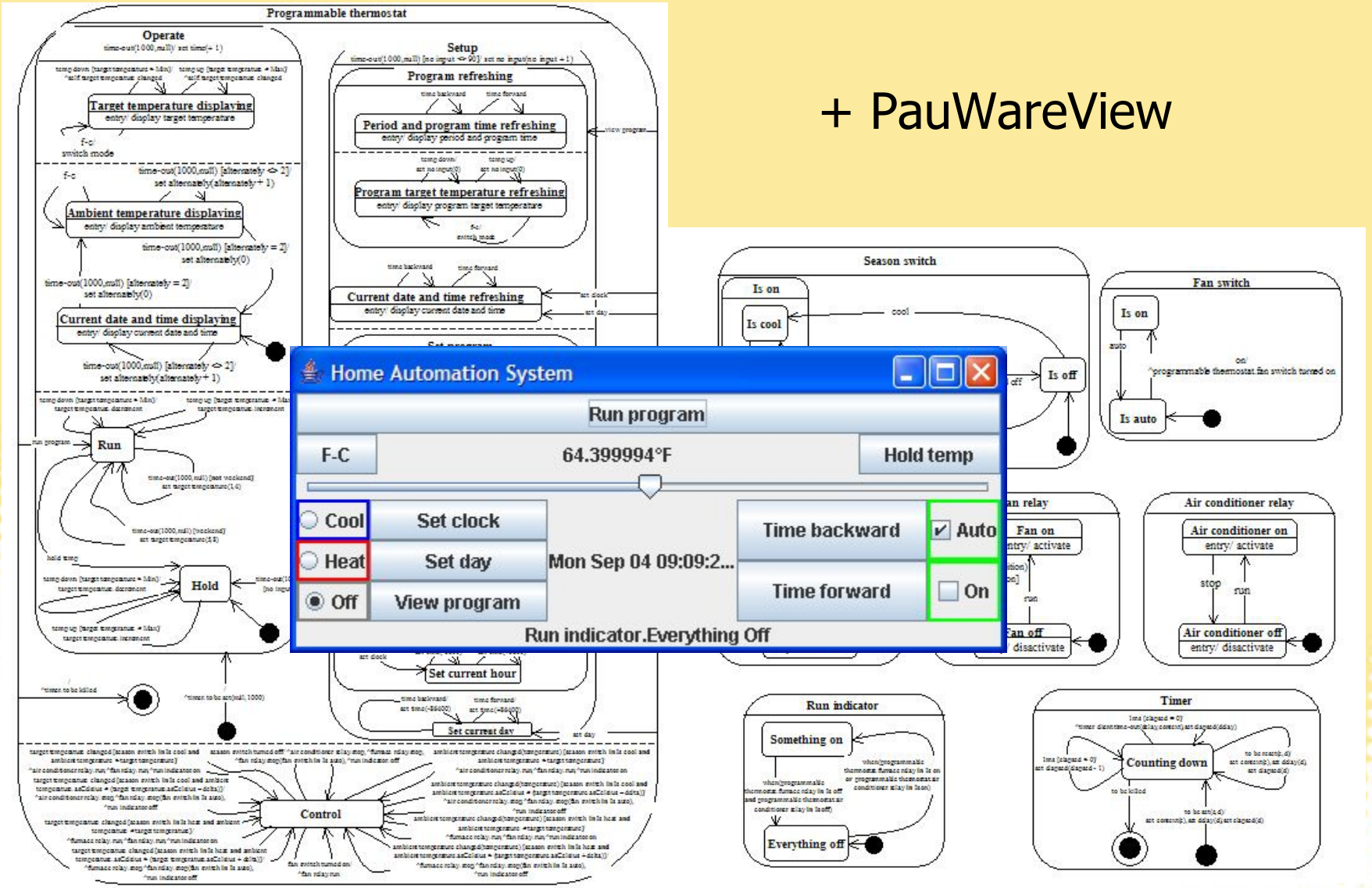

+33684199195 - PauWare
MIDlet Help
Programmable thermostat
Target temp. 18.0°C
Ambient temp. 18.672016°C
Season
Cool
Heat
Off
Time
Fan
Auto
On
Status
SELECT
CLEAR


Fan switch
Is on
auto
on/
^programmable thermostat.fan switch turned on
Is auto


Season switch
Is on
Is cool
cool
heat
cool
off/
^programmable thermostat.season switch turned off
Is off
Is heat
heat

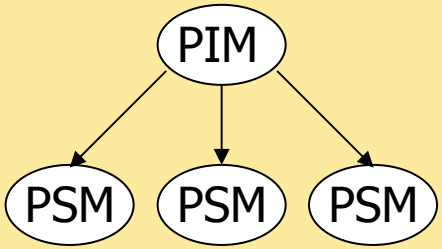
Demo : Home Automation System

+ PauWareView

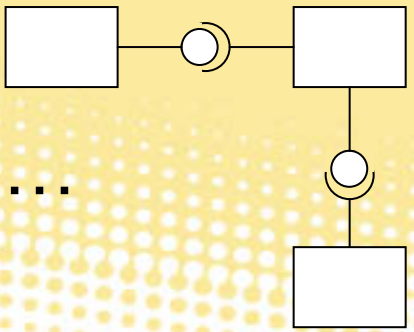


MDA + CBSE

- MDA (Model-Driven Architecture)
  - Model Centric Developpements
  - Model Transformation
  - UML : structural models and behavioral models

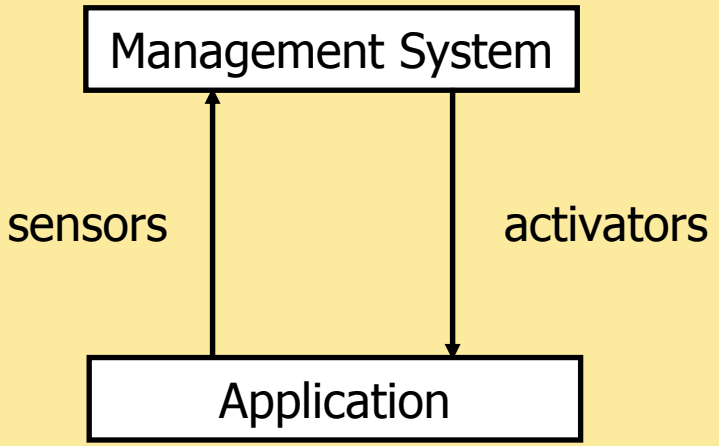


- CBSE
  - Structural Models : architecture, composition, interfaces…
  - Technological Component Model (model = format)



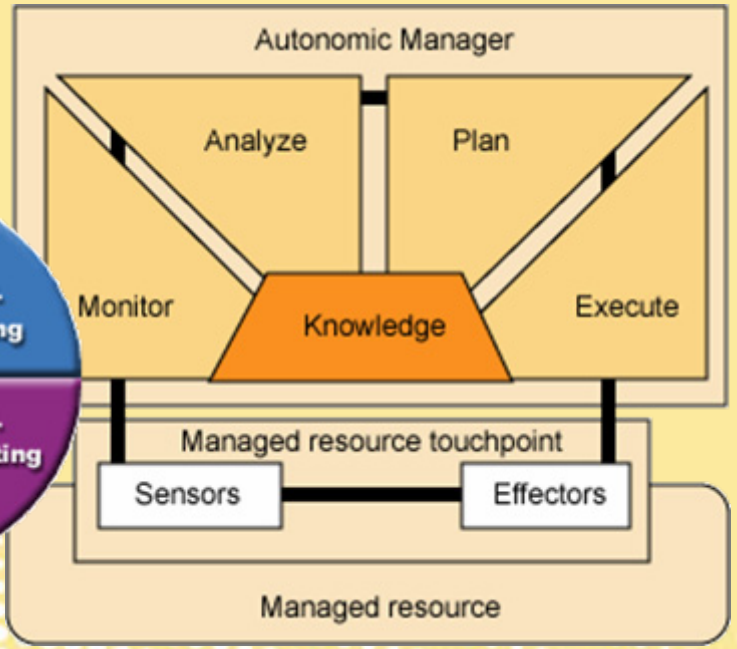
Application Management

- Monitoring
  - Sensors
- Controlling
  - Activators



- What to monitor?
- What to control?
- Where is the line between the application and the management system?

Towards Autonomic Computing



Cf. Cyril Ballagny's on-going thesis...