

# Ponder<sup>2</sup>

## Policies in Pervasive Systems

---

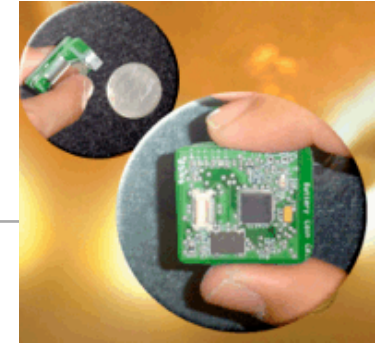
Kevin Twidle, Emil Lupu  
Department of Computing  
Imperial College London  
<http://www-dse.doc.ic.ac.uk/policies>

# Overview

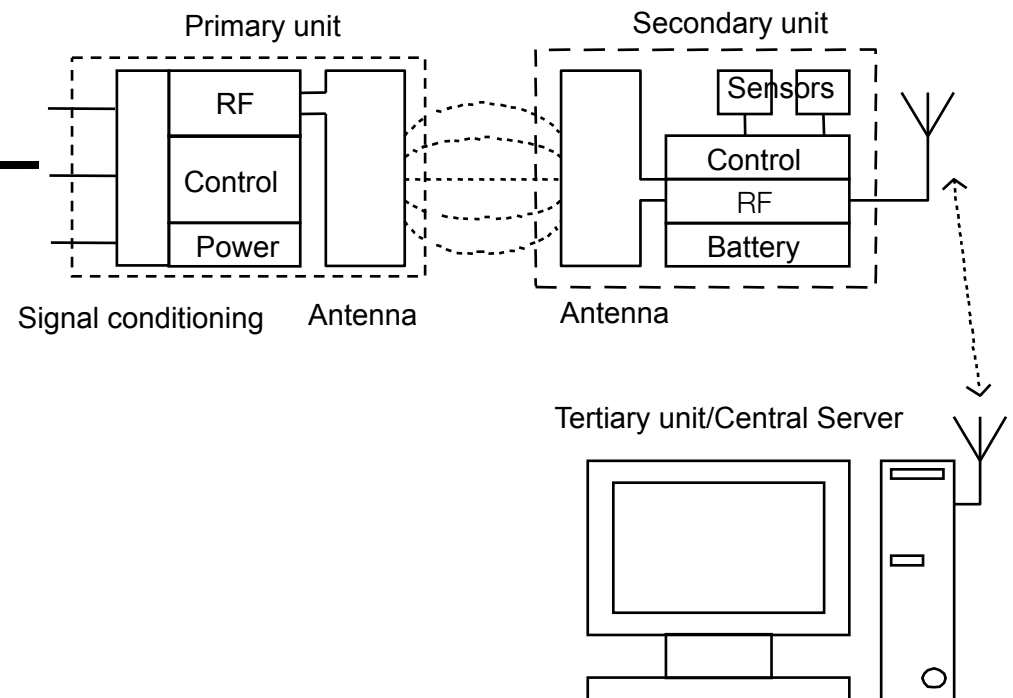
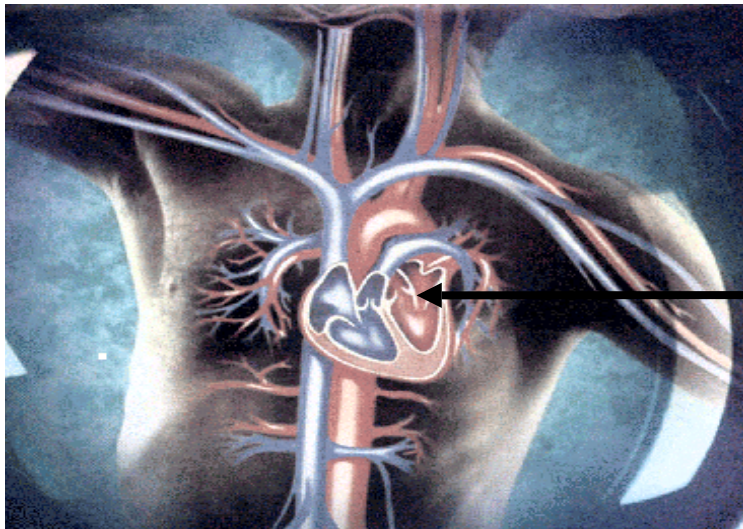
---

- Policies in Pervasive Systems
  - Policy-based adaptation
  - the Self-Managed Cells Architecture
  - Cross-SMC Interactions
- Implementation and Demonstration Overview
  - Implementation of the SMC
  - Demonstration programs
- Practical Exercises

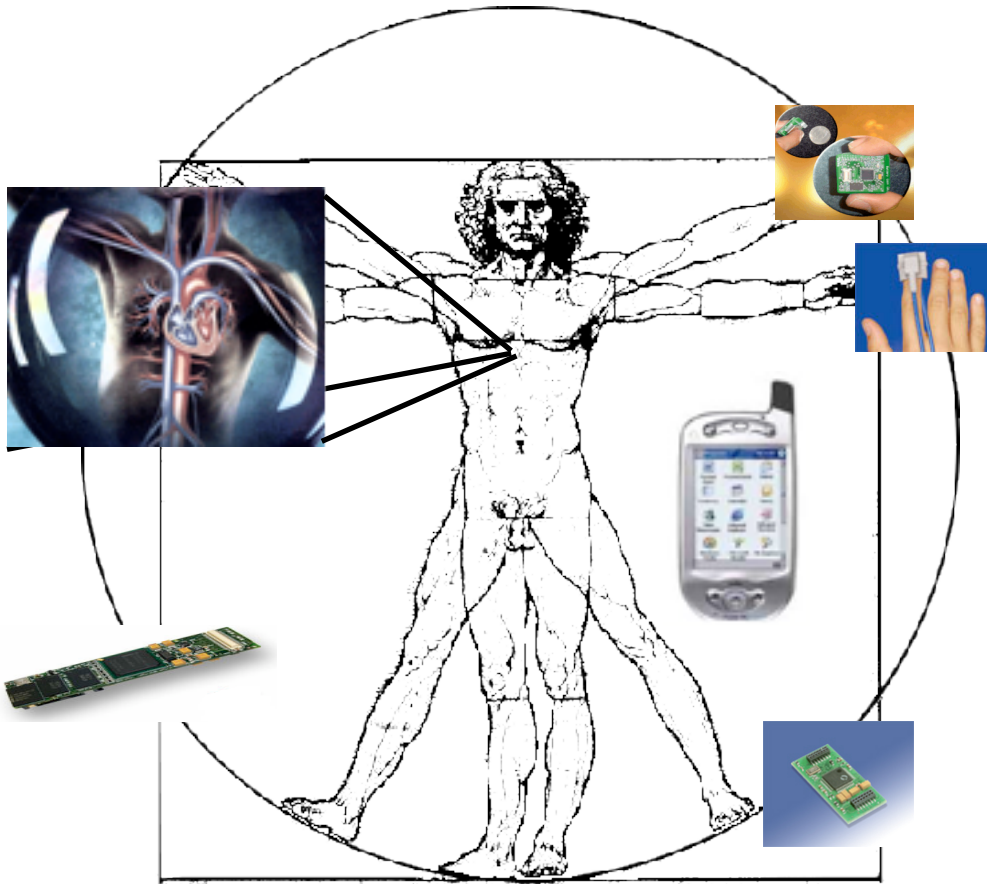
# Cardiac Monitoring



UbiMon Body Sensor Node



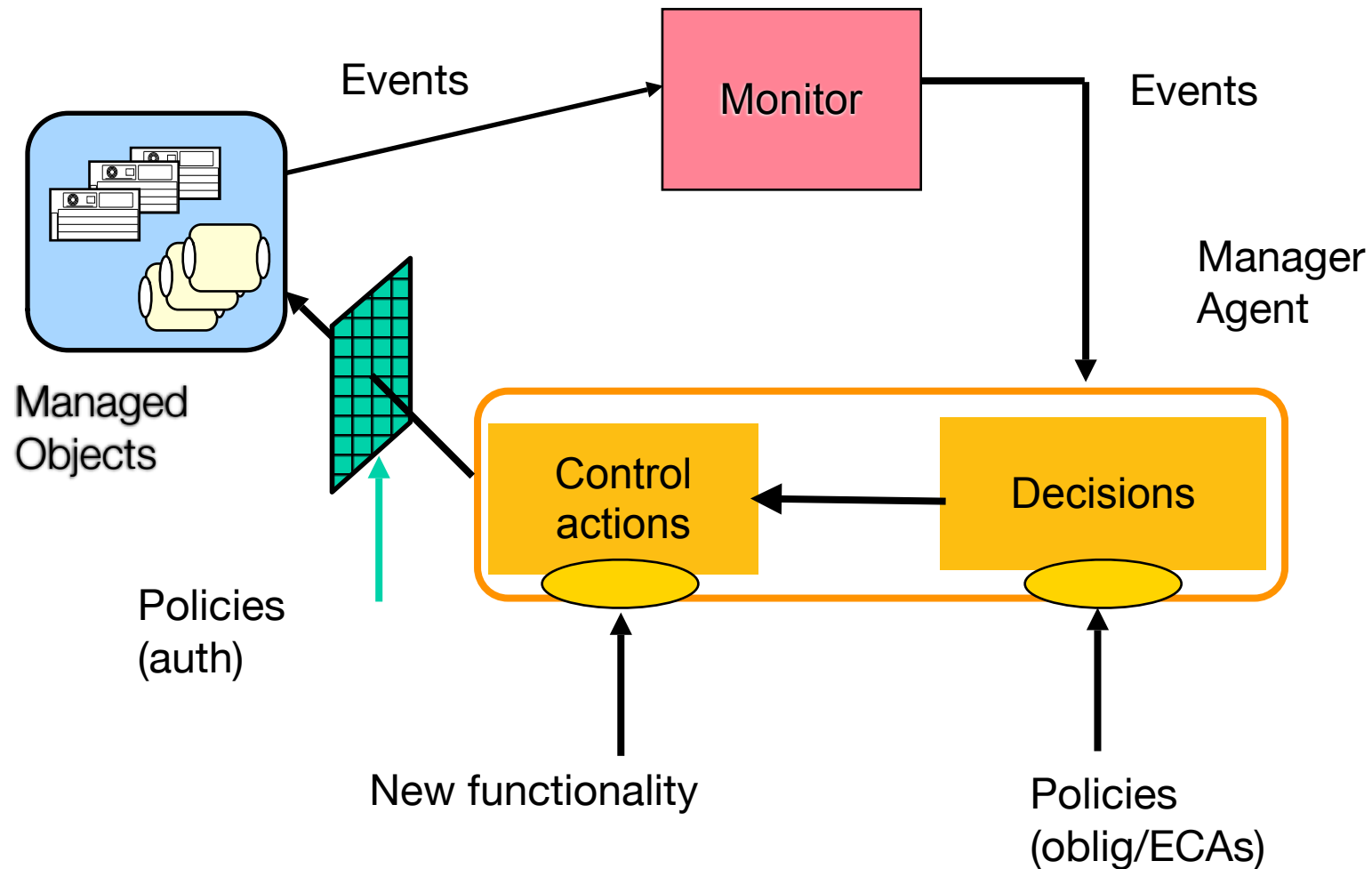
# On-body Networks for eHealth



Personal Area Networks

- Heart monitoring, blood-pressure, oxygen saturation, etc.
- Implanted and wearable sensors.
- Need for continuous adaptation:
  - sensor failures, new sensors and diagnostic units
  - changes in user activity and context
  - changes in the patient's medical condition
  - interactions with other medical and non medical equipment e.g. nurse visits at home etc.

# Policy-based closed adaptation loop



# Policies

---

## Rules governing choices in the behaviour of systems

- Derive from the need to separate strategy for adaptation from the implementation of functional aspects.
- Can be dynamically changed: loaded, enabled, disabled without interrupting the system.
- Are specified for groups of objects, often before objects are instantiated.

# Different Policy Types

---

- **Obligations** define which operations need to be performed when certain events occur. **Event-Condition-Action Rules**
- **Authorisations** define which operations are permitted and under which circumstances.
- Other policy types: Membership management, Information Filtering, Trust Management, Delegation, Negotiation, etc.

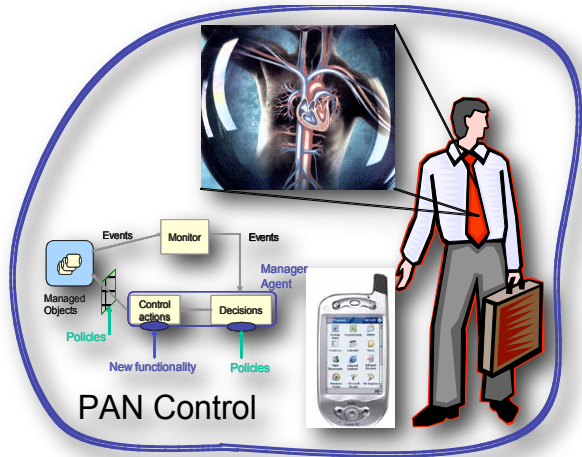
# Policies for Different Functional Areas

---

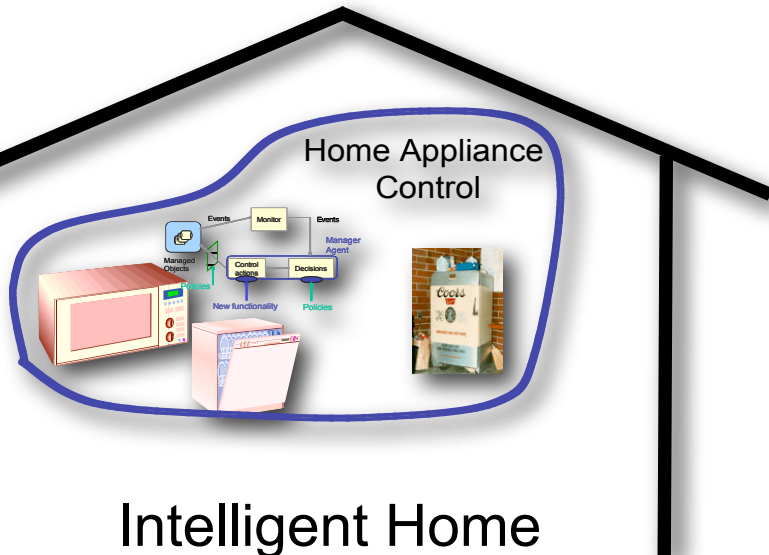
- **Device and Service Discovery.** How to react to new devices and services and their disappearance.
- **Membership Management.**
- **Context Management.** How to react to changes in location, activities of the user, surrounding environment.
- **Clinical Management.** How to react to changes in the clinical condition.
- **Security Management.**
- **Policy Management.** Enable, disable, unload policies.



# Pervasive Spaces



## Personal Area Networks



## Intelligent Home Networks



## Autonomous Vehicles



## Pervasive Environments



# A common pattern

---

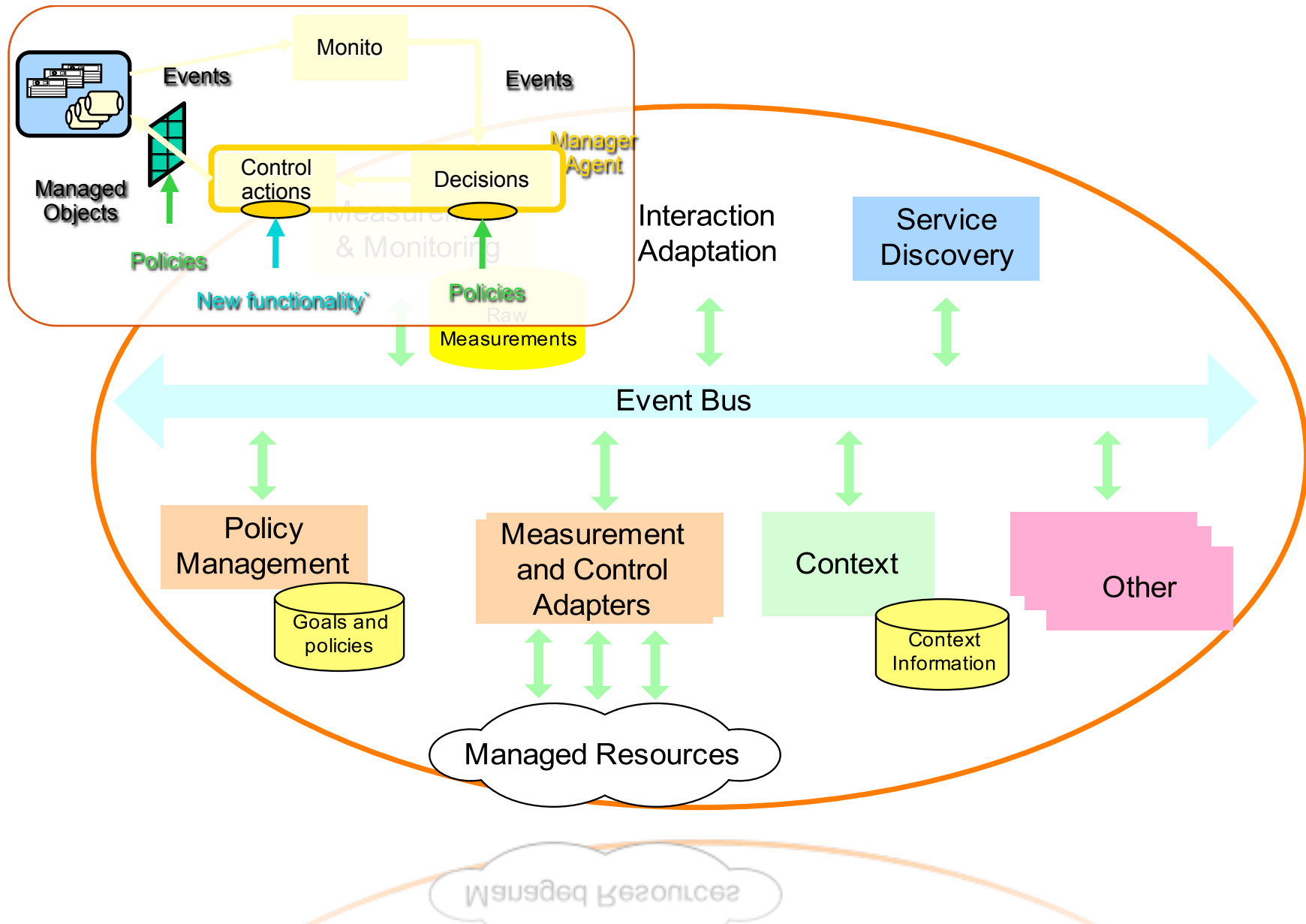
- That can be used at **different levels of scale**: from personal area networks, to unmanned vehicles, intelligent homes, and large distributed systems and networks.
- That can provide **self-management** and closed-loop adaptation at the local level.
- That can provide different levels of functionality.
- That is **architectural** as well as functional.
- Provides **low-coupling** between the different services.

# What is a Self Managed Cell?

---

- A set of hardware and software components forming an administrative domain that is able to function autonomously and thus capable of self-management.
- Management services interact with each other through asynchronous events propagated through a content-based event bus.
- Policies provide local closed-loop adaptation.
- Able to interact with other SMCs and able to compose in larger scales SMCs.

# Self-Managed Cell (SMC)



# SMC Pattern

---

- Provides low-coupling between the different services.
- Permits the use of different service implementations when used at different levels of scale.
- Permits to add services to SMCs in order to add functionality:
  - Context service(s) for mobile users and gathering information from the environment.
  - Authentication, Access Control and other security services.
  - Provisioning and Optimisation services for control of resources

# SMC Core Services

---

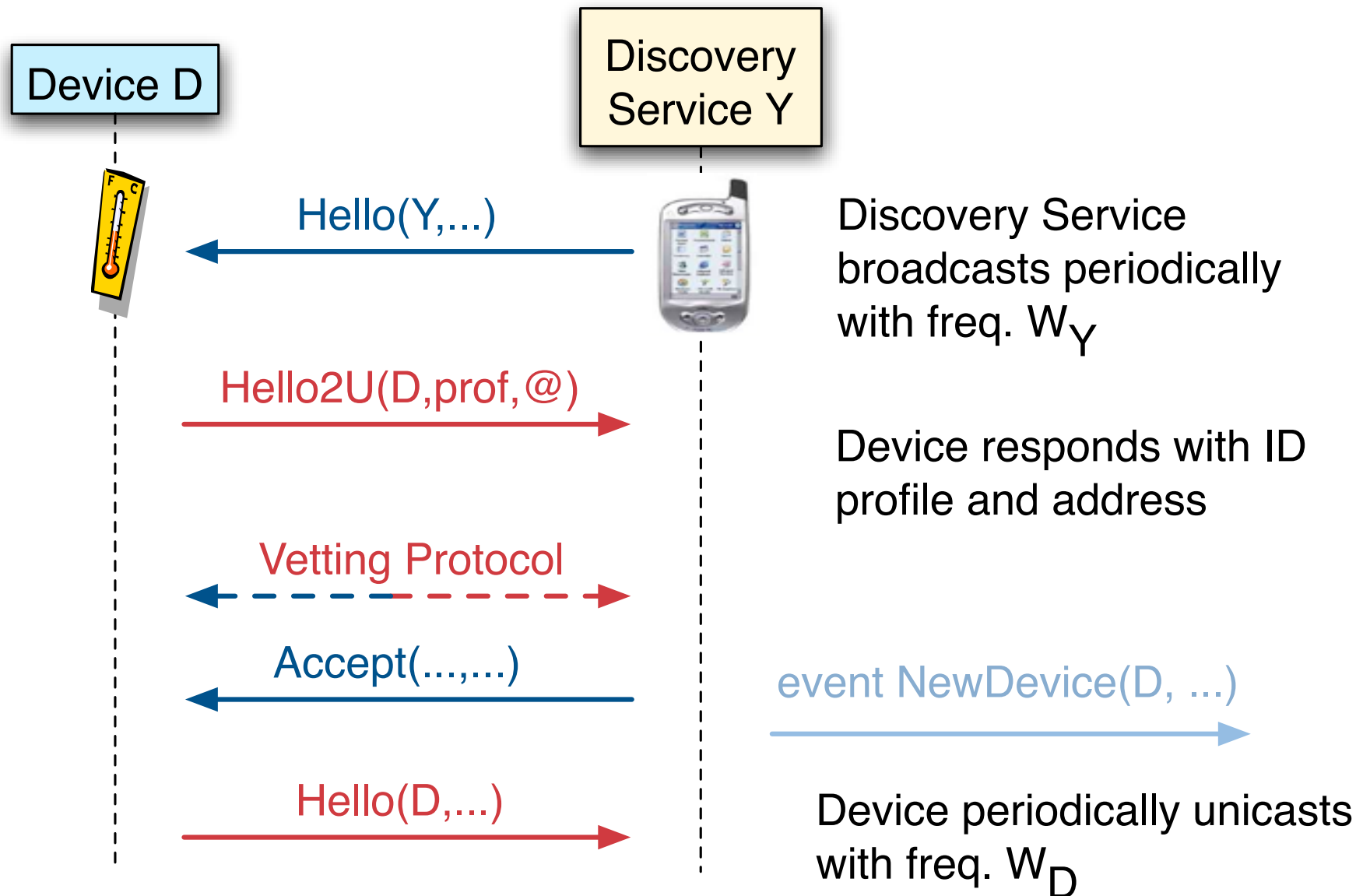
- Discovery Service (including membership management)
- Event Service
- Policy Service

# Cell discovery service

---

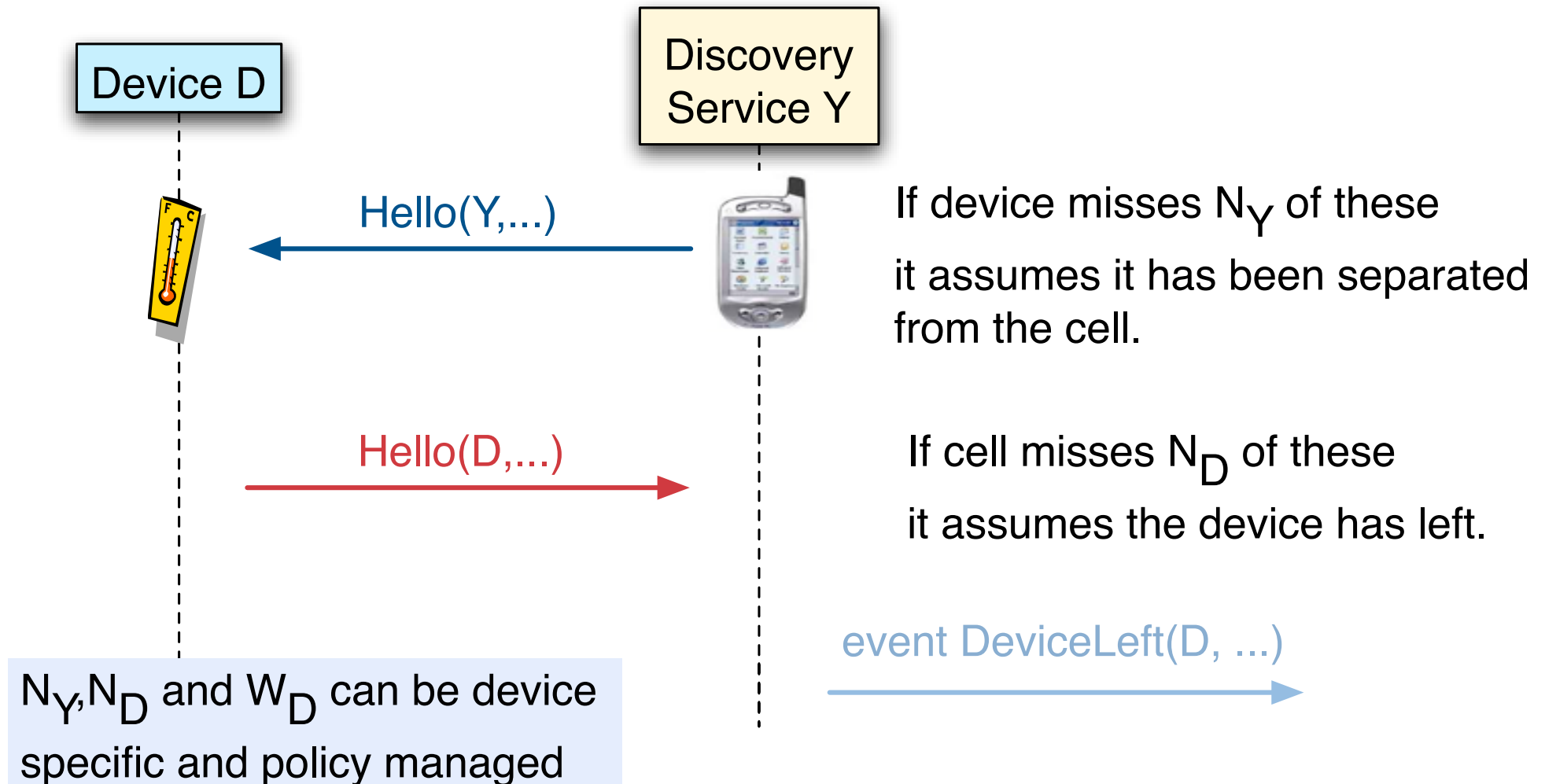
- Discovers new devices and maintains membership.
- Queries device for its profile and services;
- Performs any vetting functions e.g. authentication, admission control.
- Listens for new service offers and service removals from the devices
- Generates events to signal new/disconnected devices or software components. Any interested services can react to these events.

# Discovery Service I



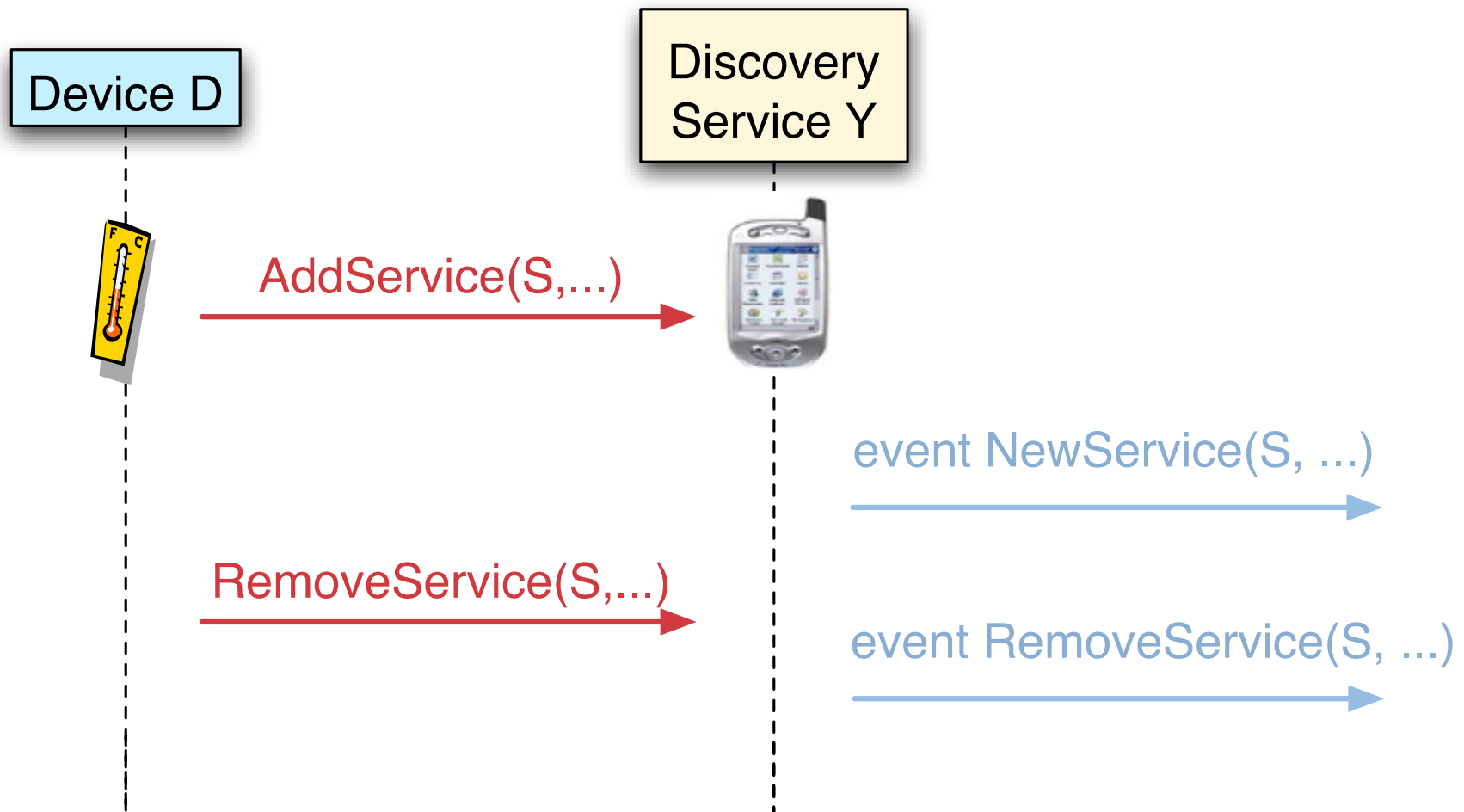


# Device Discovery - Separation



# Service/Component Discovery

---

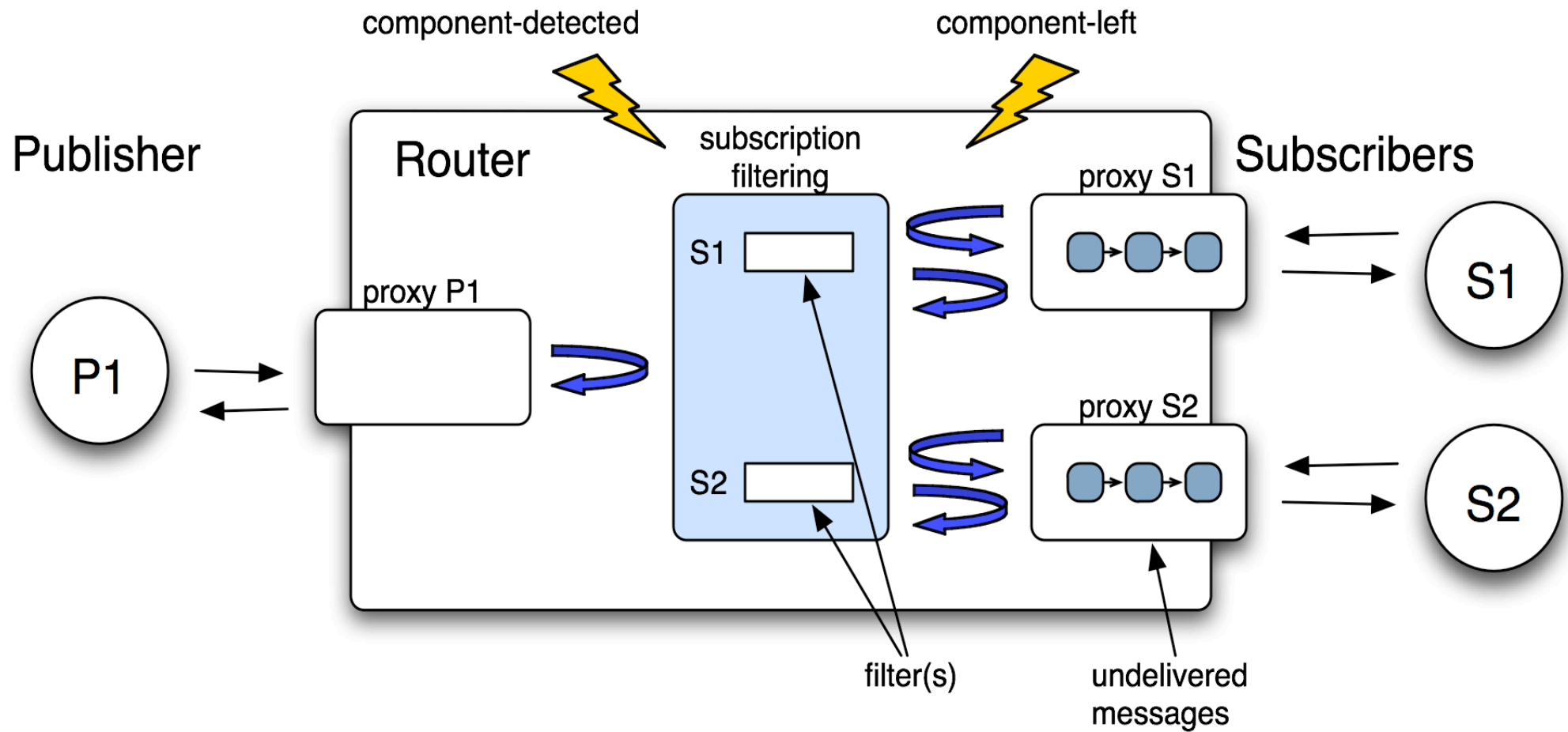


# Cell Event Service

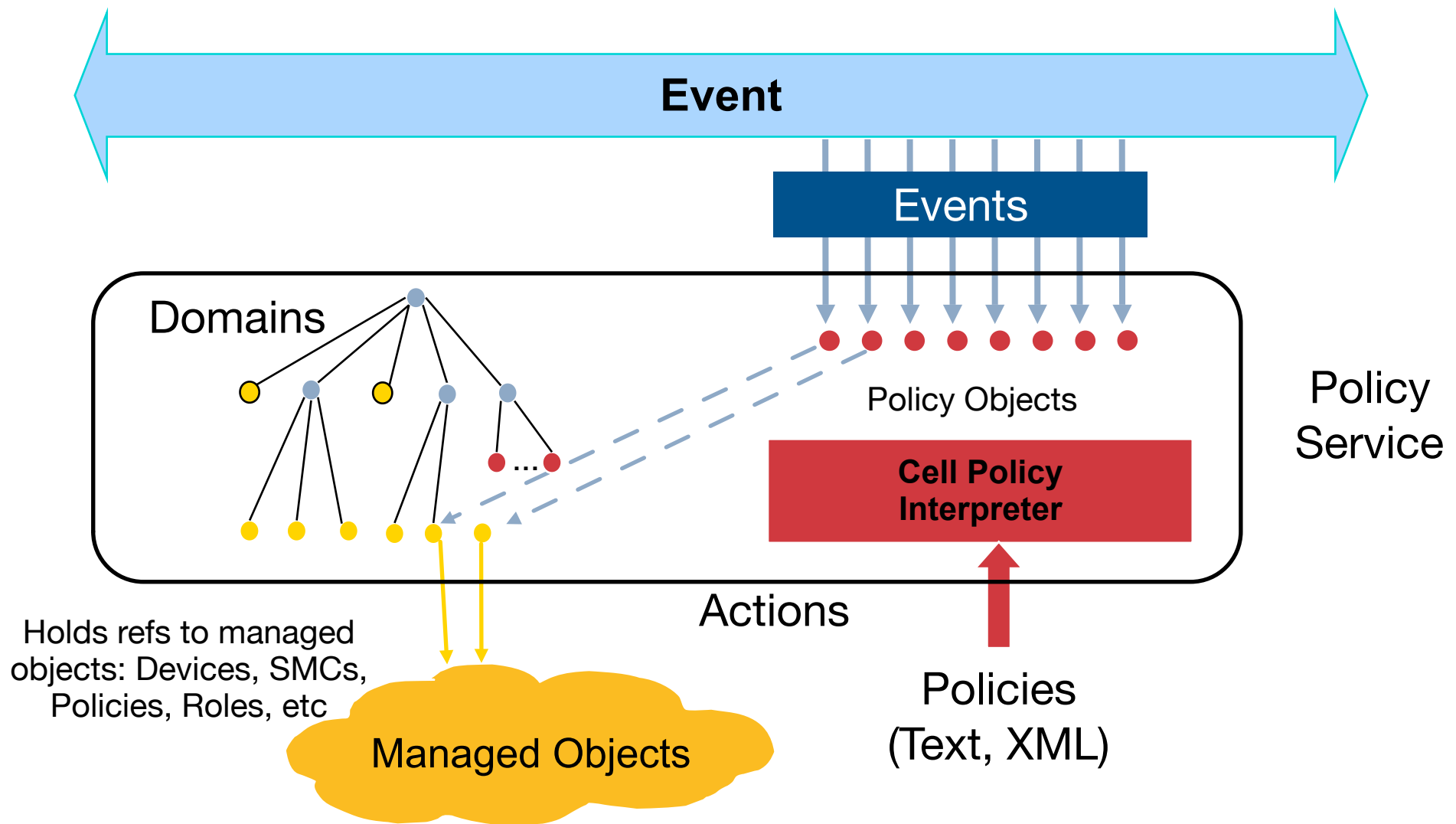
---

- Publish/Subscribe with content based router.
- At-most-once, reliable event delivery.
- To an individual recipient events are delivered in the same order as received by the router.
- Quenchable publishers to minimise number of messages and power consumption.
- Supports heterogeneous communication.

# Event Service Architecture

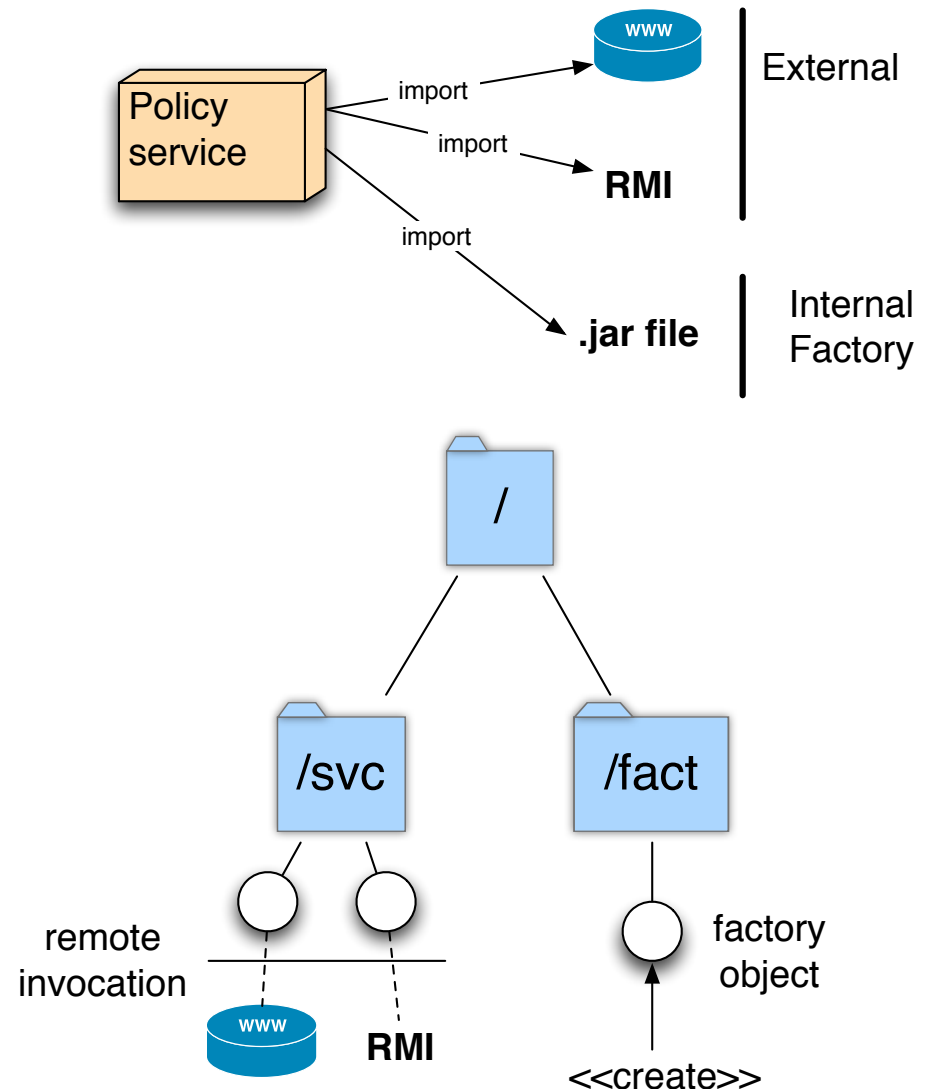


# Cell Policy Service



# Managed Objects

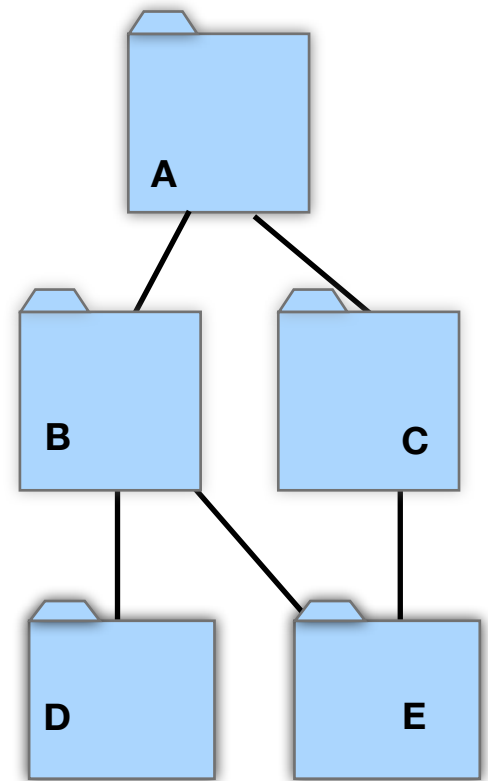
- General purpose object management environment.
- A managed object is anything that conforms to a set of interface rules.
- Managed objects can accept commands
- Four pre-defined types of managed objects: domains, policies, factories, external



# Domains for grouping objects

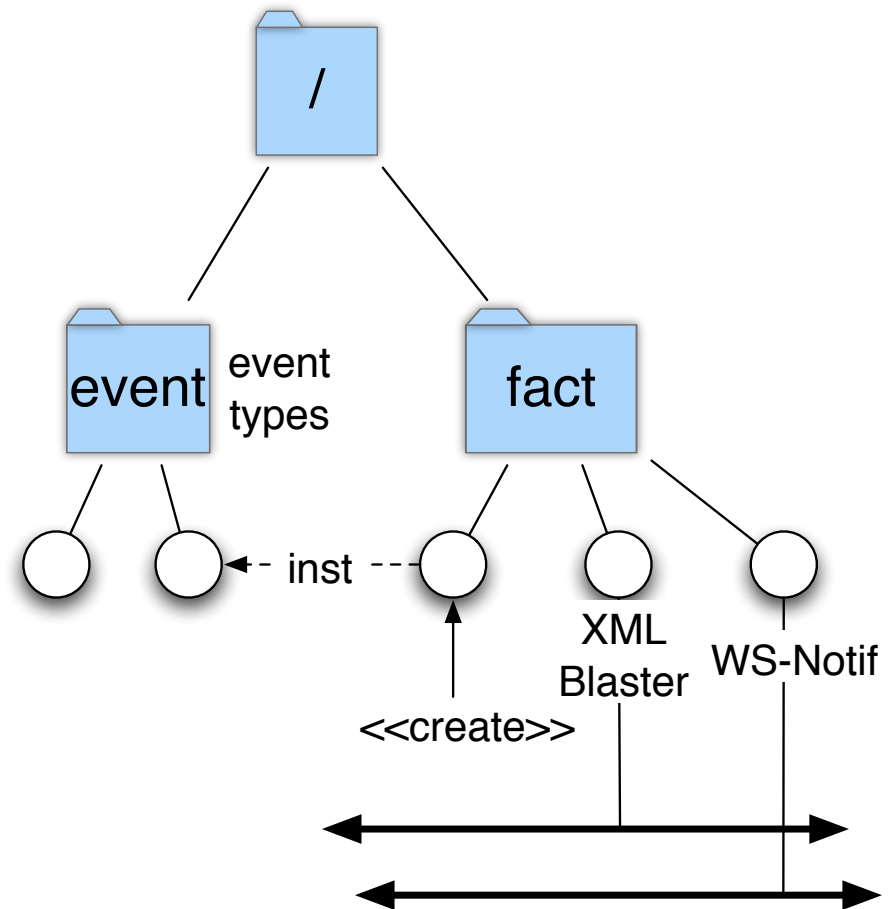
---

- A domain is a collection of objects which have been explicitly grouped together for management purposes e.g. to apply a common policy
- Domains can be nested.
- Domains can overlap.
- Policies specified in terms of domains.
- Can change domain membership without changing policy.



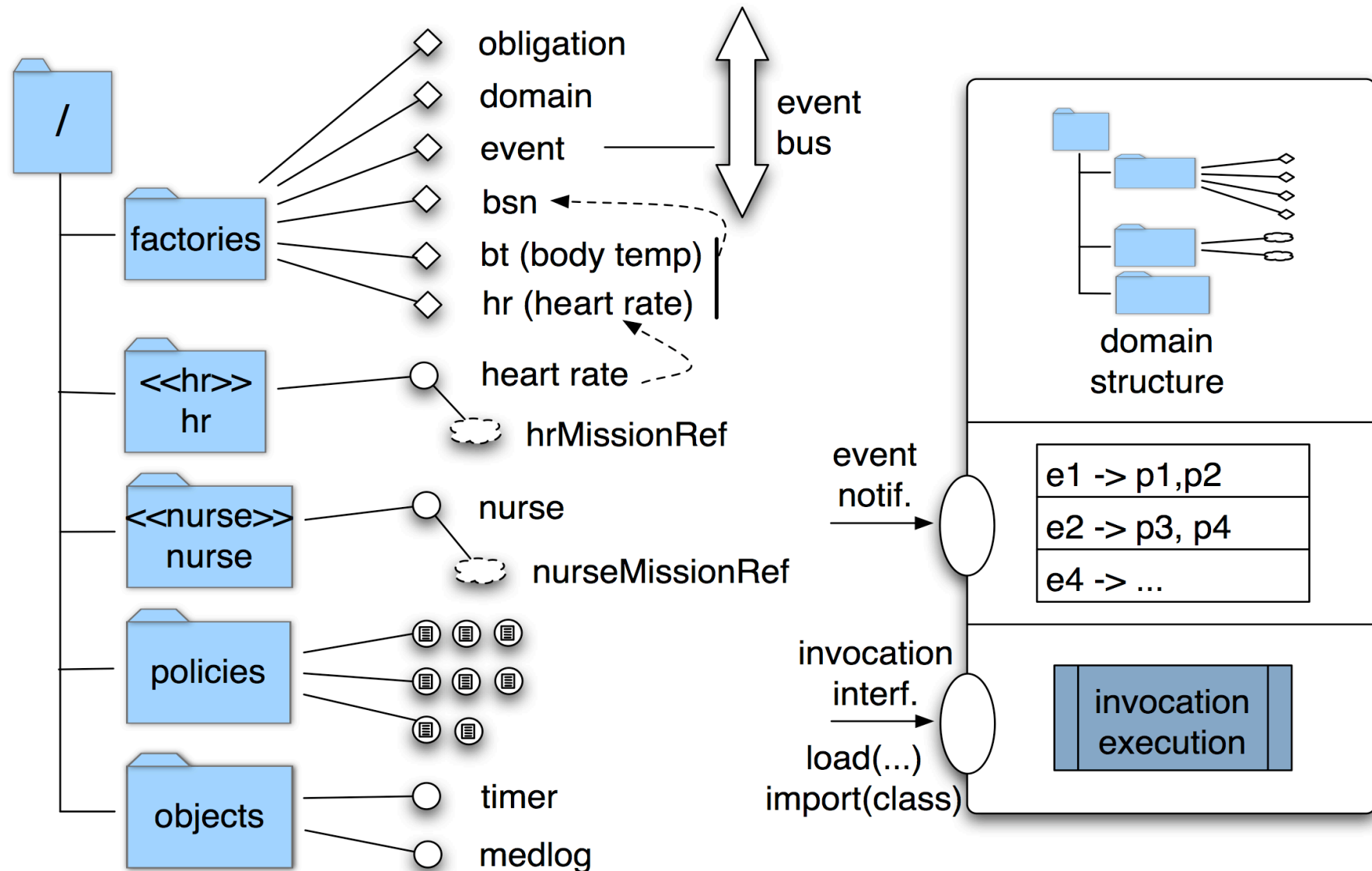
# Events

- Event = notification with named attributes.
- Created by Managed Objects.
- Trigger policies.
- Can integrate with one or several external event buses through adapter objects.





# Cell Policy Service II



# SMC Policies

---

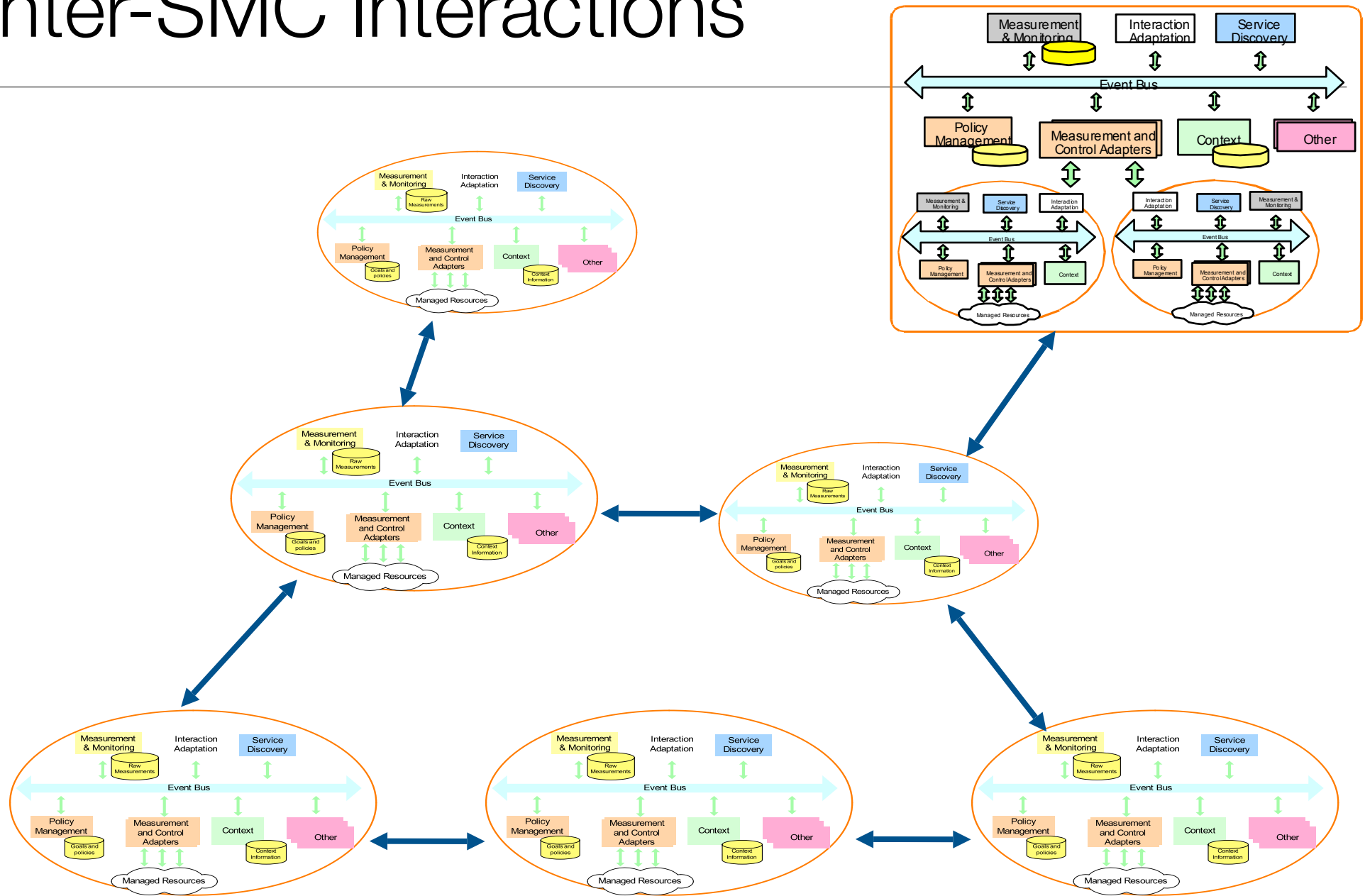
```
on new_component(id, profile, addr) do
  if profile == "heart rate" then
    r = /fact/hr.create(profile, addr); /sensors.add(r)

on hr(level) do
  if level > 100 then /sensors/os.setfreq(10min);
    /sensors/os.setMinVal(80)

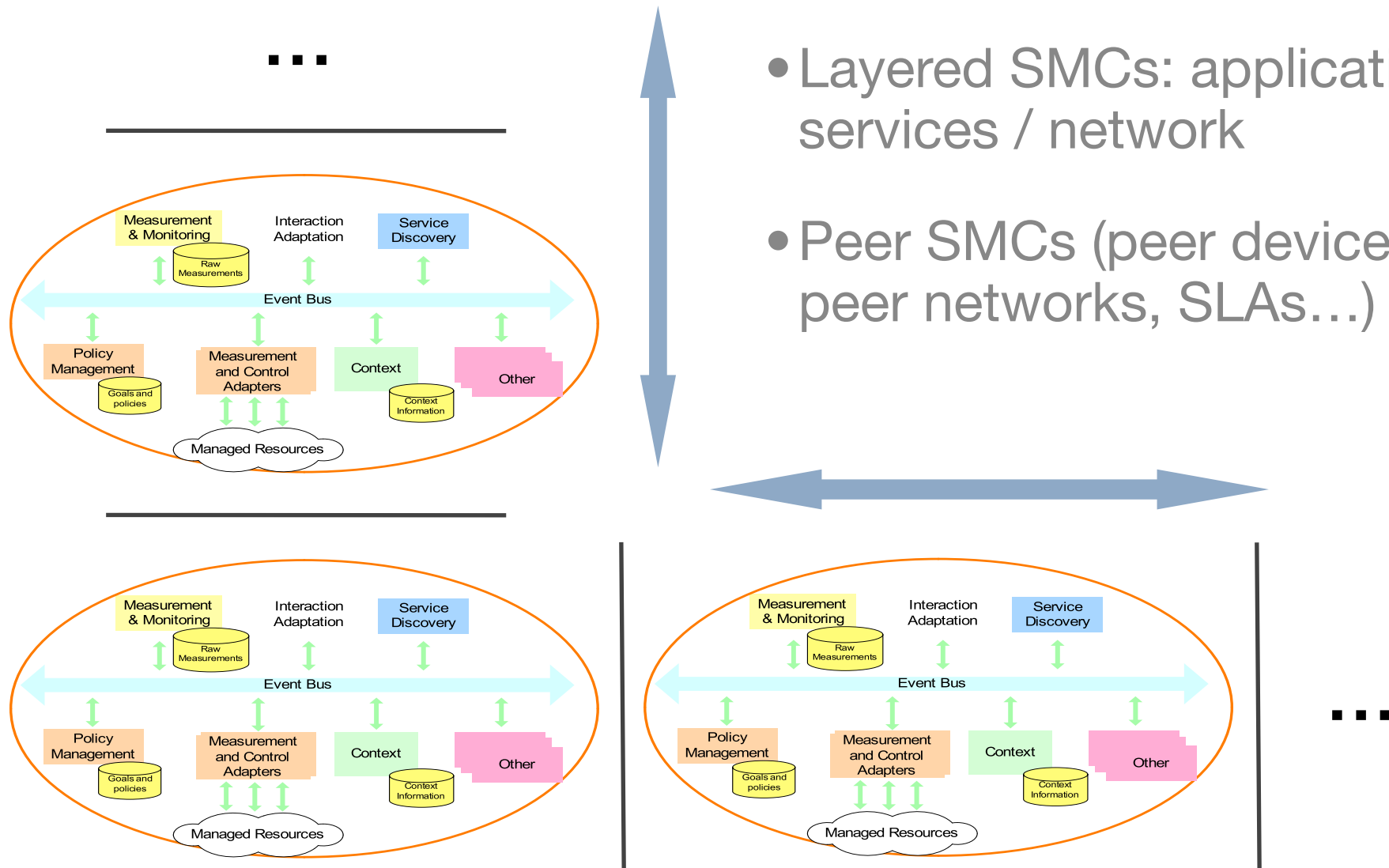
on context(activity) do
  if activity == "running" then
    /policies/normal.disable(); /policies/active.enable()

auth+ /patient → /os.{setfreq, setMinVal, stop, start}
auth+ /patient → /policies.{load, delete, enable, disable}
```

# Inter-SMC Interactions



# Peer-to-Peer Interactions

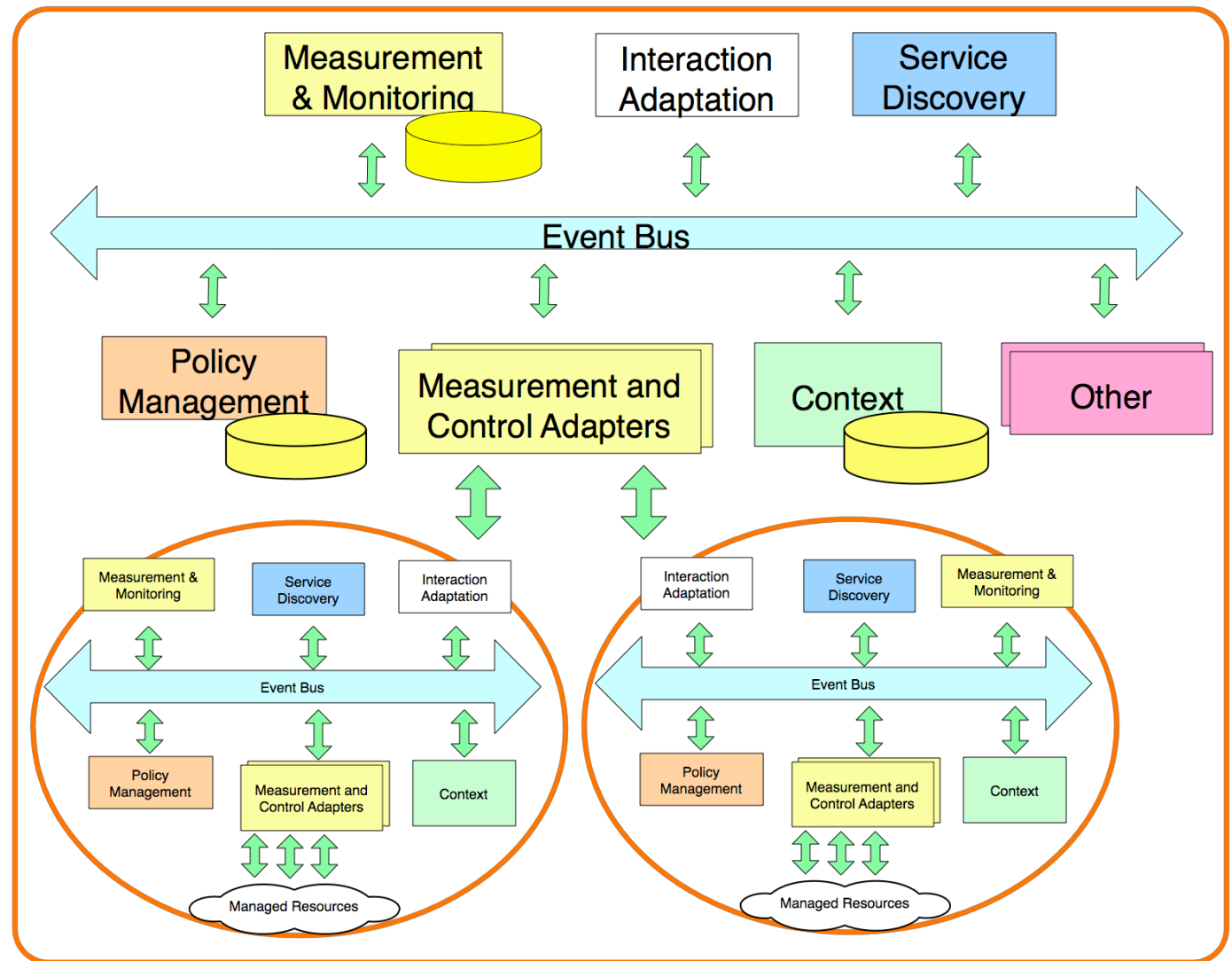


- Layered SMCs: application / services / network
- Peer SMCs (peer devices, peer networks, SLAs...)

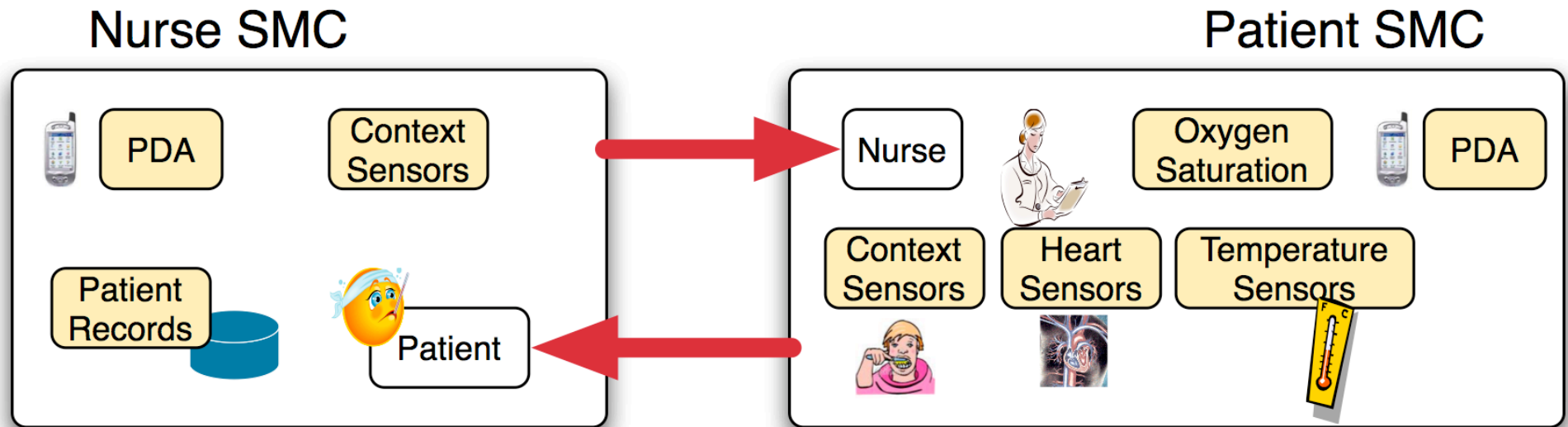
# SMC Composition

The internal SMCs cease to advertise themselves externally.

The enclosing SMC programs the nested SMCs

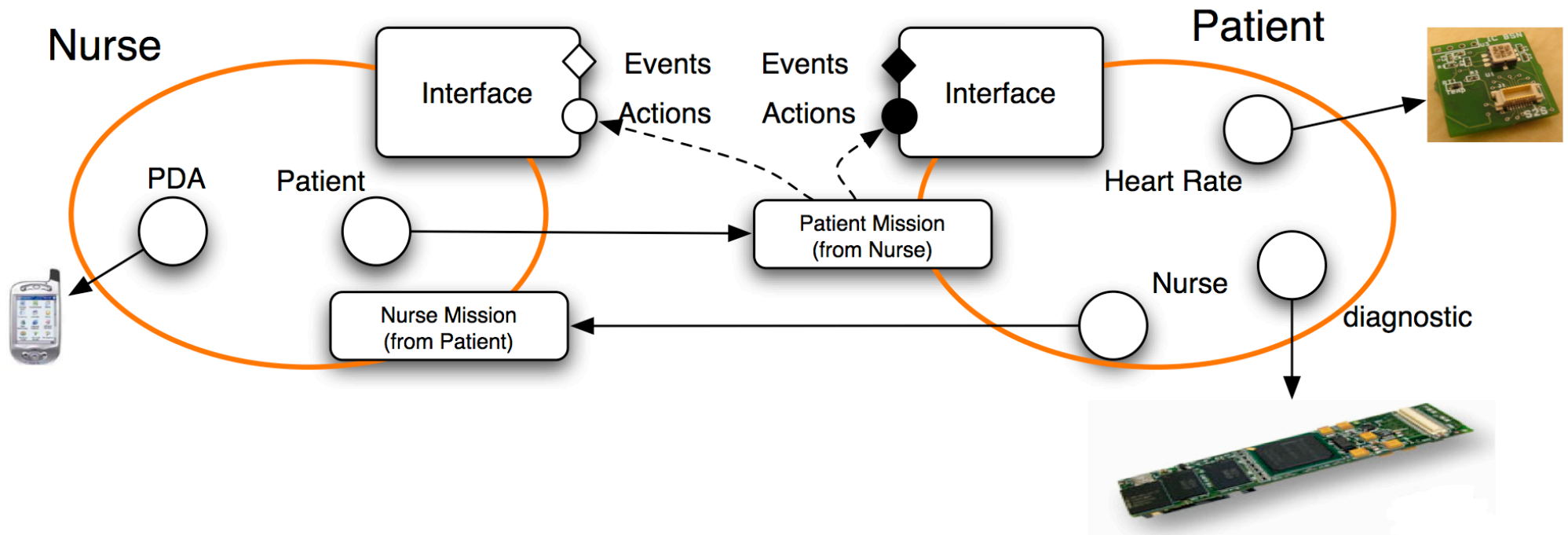


# SMCs discovery



- On SMC discovery, each SMC assigns discovered SMC to pre-defined domains.
- Policies for domain apply to assigned SMC.
- SMC Discovery can also result in policy-exchange and sharing of events and services.

# SMC Missions: Policy Exchange



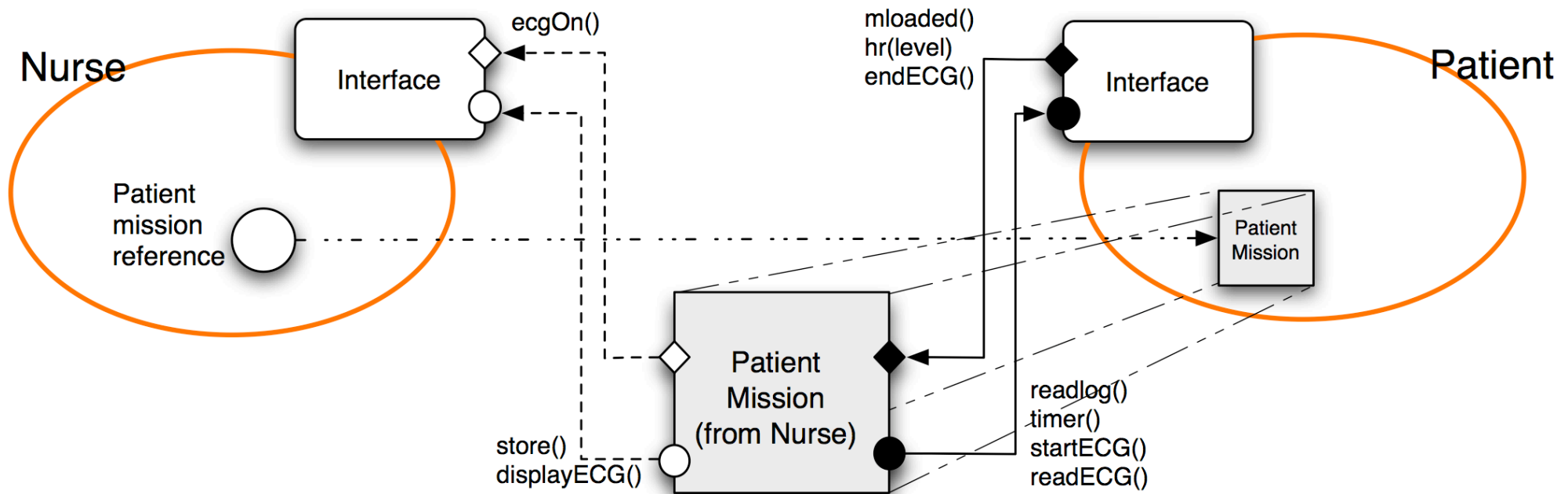
# Policy Exchange II

---

```
mission patientT(nurse, patient, ECGlevel, ECGTime) do
  on patient.mloaded() do
    nurse.store(patient.readlog())
  on patient.hr(level) do
    if level > ECGlevel then
      patient.startECG()
      patient.timer(ECGTime, endECG())
      nurse.ecgOn()
    on patient.endECG() do
      nurse.display(patient.readECG())
```



# SMC Missions: Policy Exchange



auth+ /nurse → /patient.loadMission // at the Patient

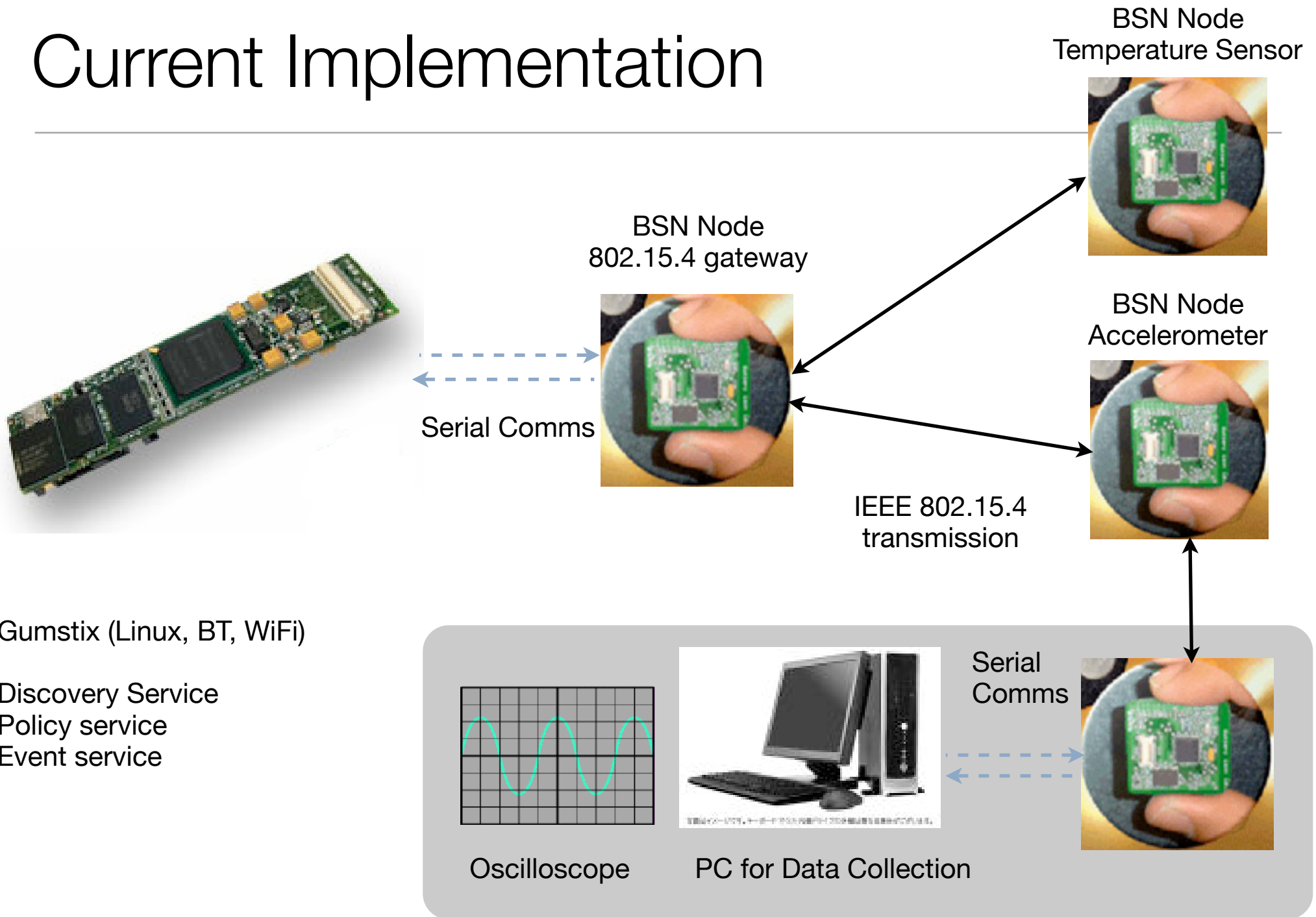
auth+ /patient → /nurse.store // at the Nurse

auth+ /patient → /nurse.displayECG

on newPatient(p) do

```
ref = p.loadMission(/patients.interface, p.interface, 82, 40); /
roles[p].add(ref)
```

# Current Implementation



# Summary

---

- Common Architectural Pattern applied at different levels of scale.
- Content-based filtering event bus provides flexibility and de-coupling between services.
- Policies for Adaptation and Access Control.
- Composition and P2P interactions across Cells
- Implementation Status
  - Event, discovery and policy service - Gumstix and PDAs
  - Event and discovery clients + basic policy interpreter on BSN nodes.

# Acknowledgements

---

Imperial College  
London

Sye-Loong Keoh



Naranker Dulay



Morris Sloman



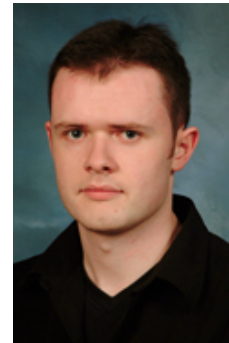
Alberto Schaeffer



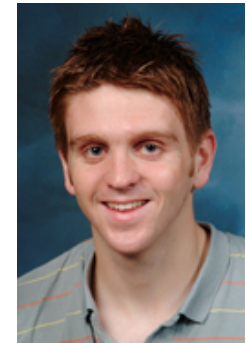
UNIVERSITY  
of  
GLASGOW



Joe Sventek



Stephen Strowes



Steven Heeps

# Ponder<sup>2</sup> - The Self Managed Cell

---

Implementation aspects and practical exercises

# Managed Objects

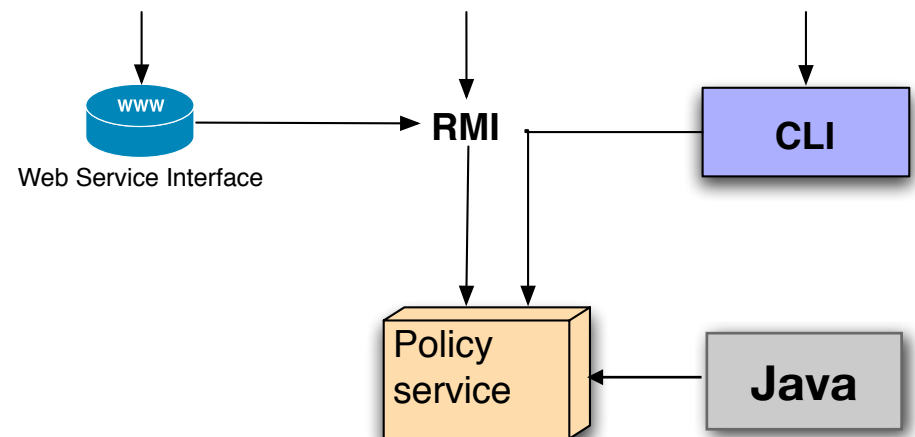
---

- A Managed Object is anything that conforms to the SMC interface rules
- Four “built-in” types of Managed Objects
  - Domains, Policies, Templates, External
- Managed Objects can generate Events
- Managed Objects can give commands to other Managed Objects

# Policy Service

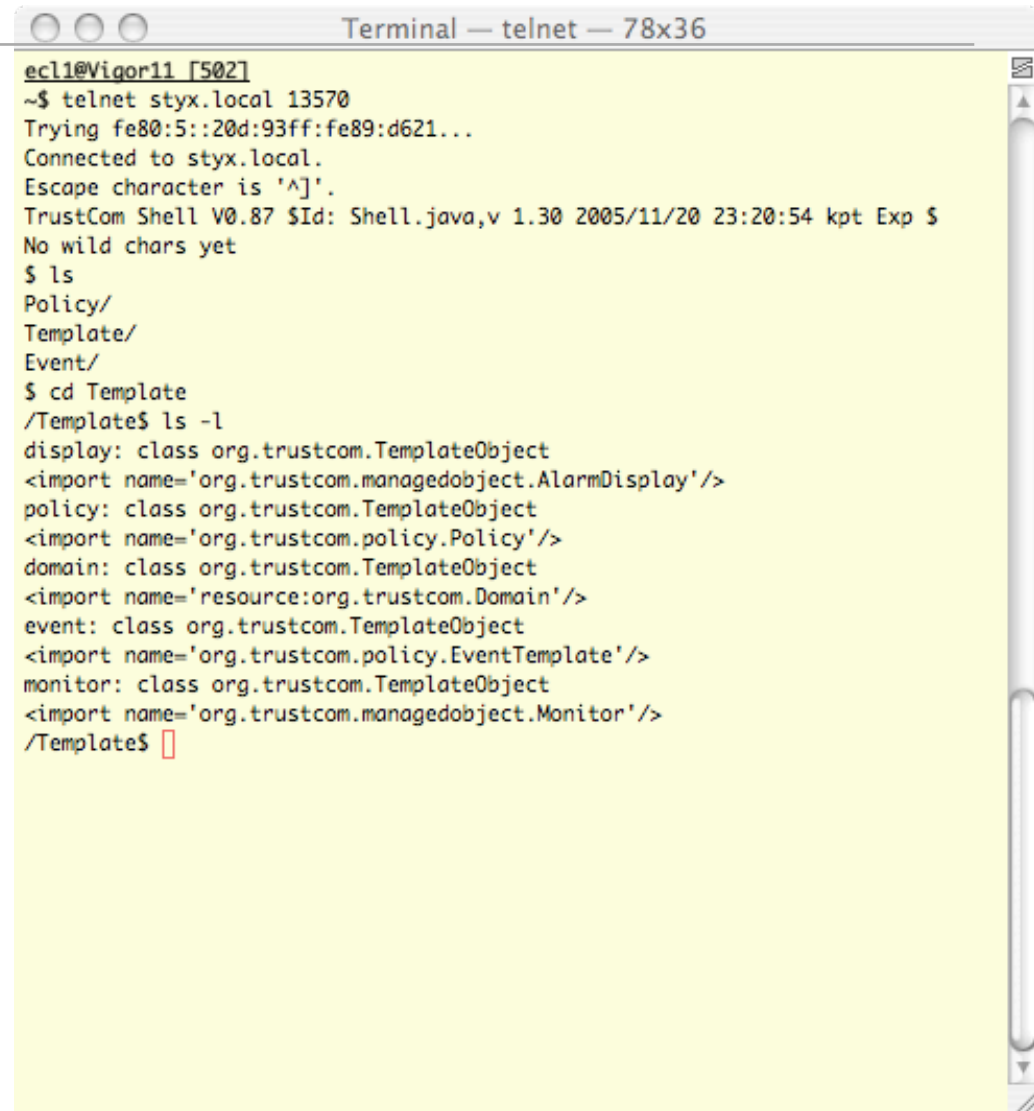
- Implements domain structure.
- Responsible for managing the “managed objects”
- Triggers evaluation of policies when events occur.

```
Terminal — telnet — 77x25
Last login: Fri Feb 24 12:44:31 on ttty2
Welcome to Darwin!
Listening ...
ec11@charon [581]
~$ telnet charon.local 13570
Trying fe80:4::20a:95ff:fe8f:5518...
Connected to charon.local.
Escape character is '^]'.
TrustCom Shell V0.87 $Id: Shell.java,v 1.30 2005/11/20 23:20:54 kpt Exp $
No wild chars yet
$ ls
Policy/
Template/
Event/
$
```



# The Client Shell

- Unix-like client shell
- Multiple concurrent client shells
- Commands include: ls, cd, mkdom, rm, ln, read
- Commands generate the XML for evaluation.
- Can input XML. XML terminates with "."
- telnet localhost 13570

A terminal window titled "Terminal — telnet — 78x36" showing a telnet session. The user is at a prompt "ecl1@Vigor11 [502]" and enters "~\$ telnet styx.local 13570". The terminal shows the connection process, including the IP address "fe80:5::20d:93ff:fe89:d621...", the connection to "styx.local", and the escape character "^[". The shell version is "TrustCom Shell V0.87 \$Id: Shell.java,v 1.30 2005/11/20 23:20:54 kpt Exp \$". The user enters "\$ ls" and the output is "Policy/", "Template/", "Event/". The user enters "\$ cd Template" and the prompt changes to "/Template\$". The user enters "ls -l" and the output is a list of XML files: "display: class org.trustcom.TemplateObject", "<import name='org.trustcom.managedobject.AlarmDisplay'/>", "policy: class org.trustcom.TemplateObject", "<import name='org.trustcom.policy.Policy'/>", "domain: class org.trustcom.TemplateObject", "<import name='resource:org.trustcom.Domain'/>", "event: class org.trustcom.TemplateObject", "<import name='org.trustcom.policy.EventTemplate'/>", "monitor: class org.trustcom.TemplateObject", "<import name='org.trustcom.managedobject.Monitor'/>", and "/Template\$ ".

```
Terminal — telnet — 78x36
ecl1@Vigor11 [502]
~$ telnet styx.local 13570
Trying fe80:5::20d:93ff:fe89:d621...
Connected to styx.local.
Escape character is '^['.
TrustCom Shell V0.87 $Id: Shell.java,v 1.30 2005/11/20 23:20:54 kpt Exp $
No wild chars yet
$ ls
Policy/
Template/
Event/
$ cd Template
/Template$ ls -l
display: class org.trustcom.TemplateObject
<import name='org.trustcom.managedobject.AlarmDisplay'/>
policy: class org.trustcom.TemplateObject
<import name='org.trustcom.policy.Policy'/>
domain: class org.trustcom.TemplateObject
<import name='resource:org.trustcom.Domain'/>
event: class org.trustcom.TemplateObject
<import name='org.trustcom.policy.EventTemplate'/>
monitor: class org.trustcom.TemplateObject
<import name='org.trustcom.managedobject.Monitor'/>
/Template$
```



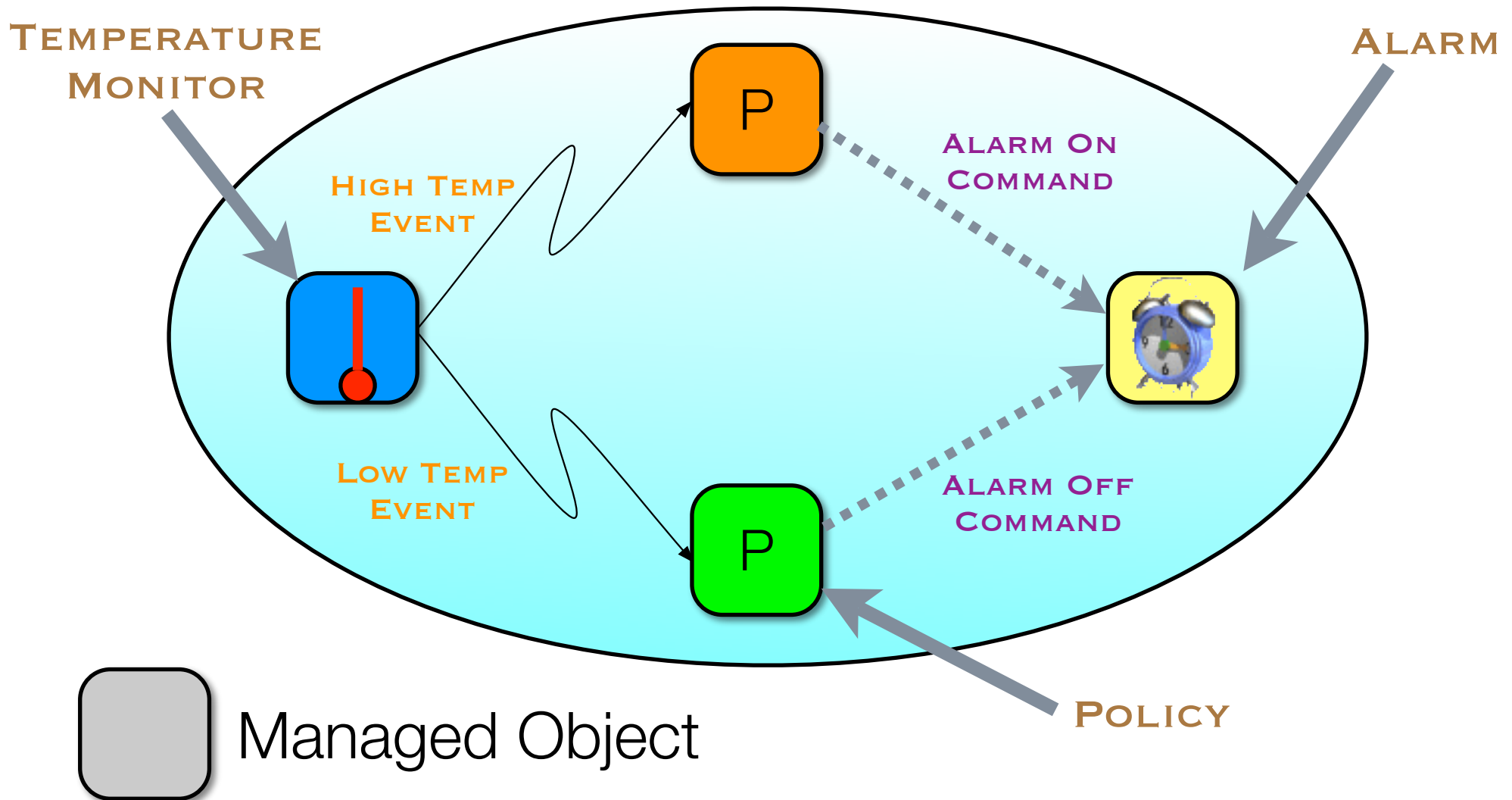
# Self Managed Cell

---

- Parses and executes XML
- One basic command **use** - to select a particular object to send XML commands to
- All other commands are implemented by managed objects
  - Domain - add, link, remove, list
  - Factory Object - create
  - Policy Object - activate, event, condition, action

```
<use name="/pathname/of/object/"  
      arg1="value1" arg2="value2">  
  <operation1 arg1="value1" ... >  
    <oparg1 arg1="value1" ... >  
      ...  
    </oparg1>  
    <oparg2 arg1="value1" ... >  
      ...  
    </oparg2>  
  </operation1>  
  <operation2 ...>  
    ...  
  </operation2>  
</use>
```

# Events and Policies



# Factory Managed Object

---

- Used to create a new Managed Object
- Accepts a “create” command and returns a new instance of a Managed Object.
- Importing new Managed Object code (currently only from a Jar or Java class file) produces a Factory Object.

# XML in Action

---

- use an object  
send it commands
- Domain - add, remove
- Root domain - import
- Factory - create
- Alarm - show, hide

/Template/alarm  
/alarm

```
<xml>
  <!-- Import the Alarm display -->
  <use name="/Template">
    <add name="alarm">
      <use name="/">
        <import name=
          "managedobject.AlarmDisplay"/>
      </use>
    </add>
  </use>
  <!-- Create an alarm instance -->
  <use name="/">
    <add name="alarm">
      <use name="/Template/alarm">
        <create/>
      </use>
    </add>
  </use>
</xml>
```

# Events

---

- Notification with named values
- Event types created from the event factory
- Event types hold a list of their named arguments
- Managed Objects can create events from the event types
- Events contain named arguments and their values
- Events trigger policies

```
<!-- Create /Event/toohigh -->
<use name="/Event">
  <add name="toohigh">

  <!-- Use the event factory -->
  <use name="/Template/event">

    <!-- Create the event type -->
    <create>
      <!--Name the arguments -->
      <arg name="msg"/>
      <arg name="value"/>
    </create>

  </use>

</add>
</use>
```

# Policy Types

---

- **Obligation Policies** define which actions need to be performed when events occur in the system. Use the form of event-condition-action rules.
  - Conditions: Use Event attributes
  - Actions: Give commands to one or more Managed Objects and/or generate new events
- **Authorisation Policies** define which actions a subject is permitted (prohibited) to perform on a target object. [not yet supported in the this version of the SMC]

# Policies

---

- Created through a policy factory.
- Can dynamically associate new or additional events with a policy.
- Must declare expected parameters.
- Can be activated and deactivated.
- Are managed objects. Can be moved, deleted, created, activated, deactivated by other policies.

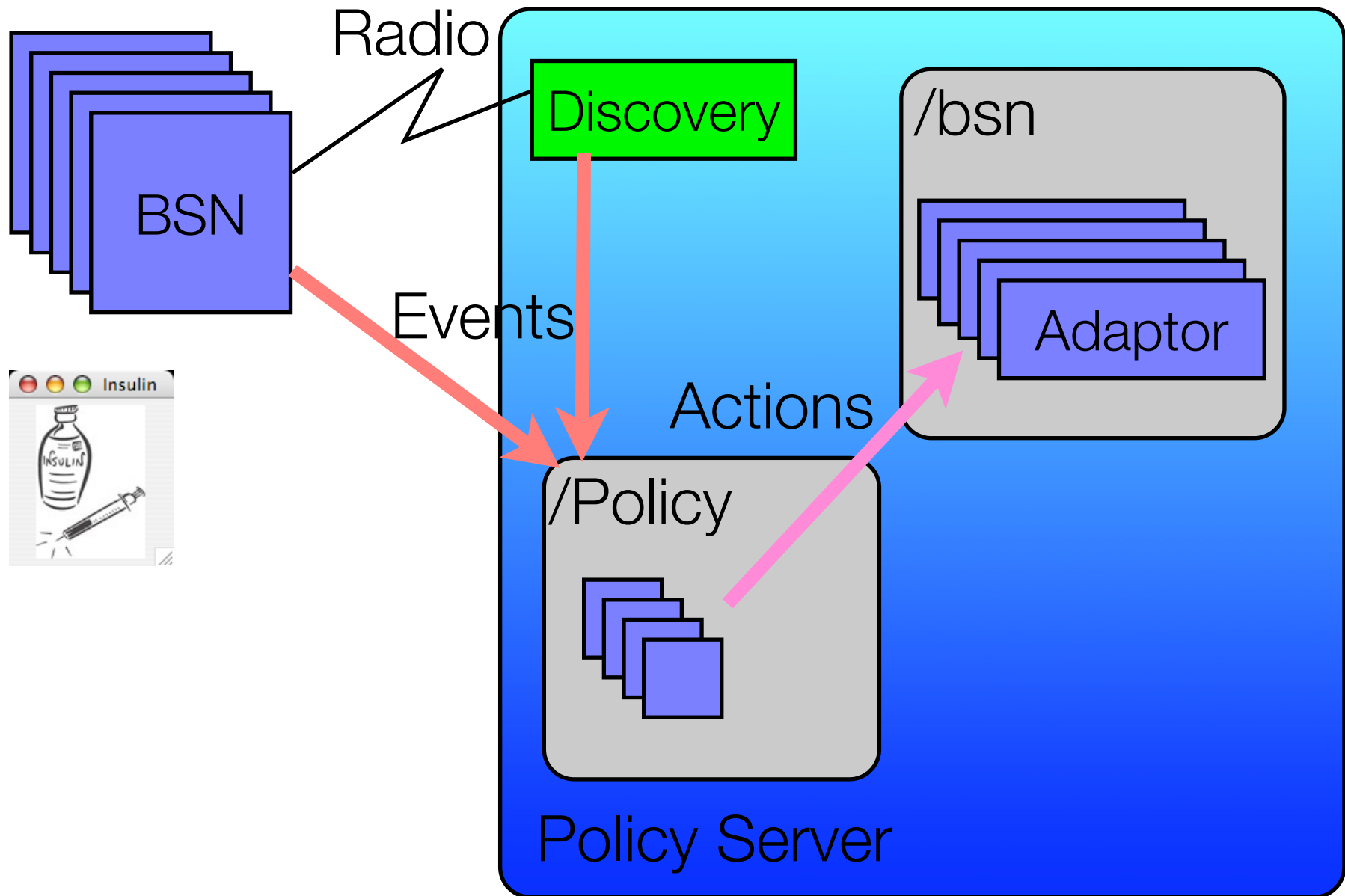
```
<use name="/Policy">
  <add name="toohigh">

  <use name="/Template/policy">

    <!-- Create a policy -->
    <create type="obligation"
      event="/Event/toohigh"
      active="true">
      <!-- We need this arg -->
      <arg name="msg"/>
      <!-- Do this action -->
      <action>
        <use name="/alarm"
          alarm="on"
          title="!msg;"/>
      </action>
    </create>
  </use> </add> </use>

<use name="mypolicy"
  active="false"/>
```

# Body Sensor Node Example

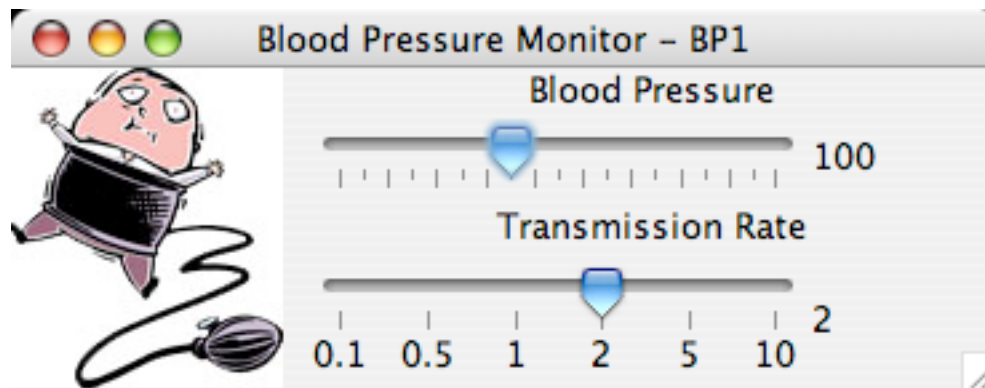




# BSN Simulation

---

- Five different discoverable BSNs and an Insulin pump can be run
- Each BSN can have its value changed and the rate at which it sends that value

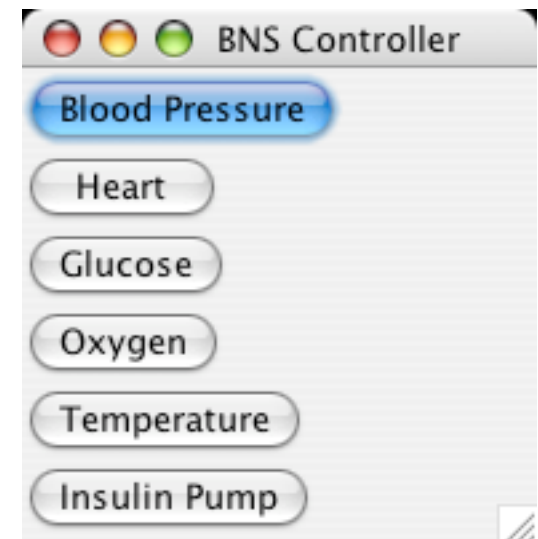


- BSN windows can be closed to simulate them going out of range

# BSN Simulation

---

- BSNs are started using the BSN Controller
- BSNs may be started and stopped by clicking on the buttons or by closing the individual BSN windows. Close the controller to terminate it.
- To run the BSN controller use  
ant bsn                      Unix  
bsn.bat                      Windows



# Discovery Event

---

- Discovery managed object issues events when BSN is detected or lost
- A policy creates or removes the appropriate adaptor managed object
- Adaptor object acts as proxy for the BSN and can receive commands for them e.g. **setrate**

```
<use name="/Event">  
  <add name="newBSN">  
    <use name="/Template/event">  
      <create>  
        <arg name="type"/>  
        <arg name="name"/>  
      </create>  
    </use>  
  </add>  
</use>
```

# Discovery Policy

---

- Discovery managed object issues events when BSN is detected or lost
- A policy creates or removes the appropriate adaptor managed object
- Adaptor object acts as proxy for the BSN and can receive commands for them e.g. **setrate**

```
<use name="/Template/policy">  
  <create type="obligation"  
    event="/Event/newBSN"  
    active="true" debug="true">  
    <arg name="type"/>  
    <arg name="name"/>  
    <action>  
      <use name="/bsn">  
        <add name="!name;">  
          <use name=  
            "/Template/bsnadaptor">  
            <create name="!name;"  
              type="!type;"/>  
          </use>  
        </add>  
      </use>  
    </action>  
  </create>  
</use>
```

# Blood Pressure Policy

---

- on bp(value)

if (value>150)  
  && oldValue<=150

do

  /bsn/HEART1  
  .set(sensorRate=1)

  /alarm(alarm=on).show

```
<use name="/Policy">
  <add name="bphigh">
    <use name="/Template/policy">
      <create type="obligation"
        event="/Event/bsnvalue" active="true">
        <arg name="name"/>
        <arg name="oldValue"/>
        <arg name="newValue"/>
        <condition>
          <AND>
            <EQ>!name; <!-- -->BP1</EQ>
            <GT>!newValue; <!-- -->150</GT>
            <LE>!oldValue; <!-- -->150</LE>
          </AND>
        </condition>
        <action>
          <use name="/bsn/HEART1">
            <set rate="0.1"/>
          </use>
          <use name="/alarm" alarm="on">
            <show/>
          </use>
        </action>
      </create>
    </use>
  </add>
</use>
```

# Event Filter Policy

---

```
<use name="/">
  <add name="eventfilter">
    <use name="/Template/eventfilter">
      <create event="/Event/bsnvalue"/>
    </use>
  </add>
</use>
<!-- Create a policy for filtering the events from the BSNs -->
<use name="/Policy">
  <add name="filter">
    <use name="/Template/policy">
      <create type="obligation" event="/Event/bsnevent" active="true">
        <arg name="name"/>
        <arg name="newValue"/>
        <action>
          <use name="/eventfilter">
            <filter name="!name;" value="!newValue;"/>
          </use>
        </action>
      </create>
    </use>
  </add>
```

# Bootstrap Demo

---

- SMC is just an empty Domain
- Import Domain Template

- Create Domain

- Be Happy

```
<!-- Import the template for creating domains -->  
<use name="/">  
  <!-- /.add("domaintemplate",import(Domain) -->  
  <add name="domaintemplate">  
    <use name="/">  
      <import name="Domain"/>  
    </use>  
  </add>  
  <!-- /.add("Template",/domaintemplate.create()) -->  
  <add name="Template">  
    <use name="/domaintemplate">  
      <create/>  
    </use>  
  </add>  
</use>
```

# To Do

---

- Deletion semantics
- External references with Dump and Restore
- More external protocols
- Freeze and Restart systems
- General JAVA Swing Managed Object
- Access Control Policy



# Exercise 1 - Policy writing

---

- Detect high glucose level, activate Insulin pump
- `ex1.xml` contains basic event definitions for the pump
- You need policies to create and remove a `pumpadaptor` instance.
- You need policy to detect glucose over 180 and inject a dose of insulin every 10 seconds (change glucose rate)
- You need a policy to detect glucose under 180 and raise the glucose monitoring rate.
- Extra points for adding the alarm (`/alarm`) into the mix

# Exercise 1 - New/Lost Pump policy

---

- on event newPump(name)  
create new pumpadaptor in /bsn/name
- Pumpadaptor create takes attribute name="!name;"
- on event lostPump(name)  
remove /bsn/name

# Exercise 1 - Glucose Policies

---

- glucosehigh policy

```
on event bsnvalue(name, newValue)
  if name == GLUCOSE1 && newValue > 180
    /bsn/GLUCOSE1.set(rate=10)
    /bsn/IPUMP1.inject(dose=3)
```

- glucosenormal policy

```
on event bsnvalue(name, newValue)
  if name == GLUCOSE1 && newValue <= 180
    /bsn/GLUCOSE1.set(rate=2)
```

# Notes - Policy Conditions

---

- Optional condition  
    <condition> logical condition </condition>
- Where logical condition is  
    <and>logical conditions</and>  
    <or>logical conditions</or>  
    <gt>value1 <!-- --> value2</gt> - value1>value2
- Also <ge>,<lt>,<le>,<eq>,<ne>,<not>
- Event arguments can be used by name as  
    !argname; e.g. <ge> !value; <!-- --> 25 </ge>

# Notes - Alarm commands

---

- `<create [title="string"]/>`
- `<use [title="string"] [alarm="on|off"]>`
  - `<show/>`
  - `<hide/>`

# Notes - BSNAdaptor commands

---

- `<create name="string"/>`  
  `<use>`  
    `<set rate="value"/>`

# Notes - PumpAdaptor commands

---

- <create name="string"/>  
 <use>  
 <inject dose="value"/>

# Notes - Running your XML

---

- The file tutorial.xml reads all the tutorial specifications in the appendix. You need this to run your XML.
- Create your XML in a new file e.g. ex1.xml
- Add `<eval name="ex1.xml">` to tutorial.xml
- Run using
  - ant tutorial                  Unix
  - tutorial.bat                Windows
- Add a little at a time to ex1.xml, then run it. When working, add a little more, then run it. etc. etc. Use the shell to inspect your objects.



# Exercise 2 - A new managed Object

---

- create a new timer Managed Object with commands  
    <wake duration="secs" event="name"/>  
    <cancel>
- After secs seconds it generates the named event with no arguments. Cancel cancels the timer.
- Write XML with new Event and new Policy to set Alarm
- Copy and rename NullManagedObject.java, call it Timer.java
- Code contains examples of attribute and command reading

# Exercise 2 - Thread Notes

---

- Use a Thread for timing.

```
new Thread() {  
    run() {  
        Thread.sleep(secs*1000);  
        Event.sendEvent("/Event/ename", string1, str2 ...);  
    }  
    catch (InterruptedException e) {  
    }  
};
```

# Exercise 2 - Running notes (Unix)

---

- You do not need the tutorial BSN policies
- You can use alarm.xml to set up the alarm
- Create your xml in e.g. ex2.xml
- Compile and run as

```
javac -d classes -cp ponder2.jar *.java
java -cp ponder2.jar:classes -rmi - \
    -boot alarm.xml
    -boot ex2.xml
```

# Exercise 2 - Running notes (Windows)

---

- You do not need the tutorial BSN policies
- You can use alarm.xml to set up the alarm
- Create your xml in e.g. ex2.xml
- Compile and run using

`compile.bat`

`smc.bat -boot alarm.xml -boot ex2.xml`

# Exercise 2 - XML Notes

---

- All XML held in Class TaggedElement e.g. xml

String tag = xml.getName(); - get the tag name

String att = xml.getAttribute("attname"); - get attribute

int size = xml.elements(); - get number of child elements

Element e = xml.getChild(1); - get second child element

where e is either type TaggedElement or TextElement

String text = ((TextElement)e).toString()