Ponder2 - The Self Managed Cell

Anatomy of a Policy Service and Practical Exercises

Kevin Twidle, Emil Lupu
Department of Computing
Imperial College London
http://ponder2.net

Overview

- Tutorial Installation
- Policies, Events and Managed Objects
- The Self Managed Cell Architecture
- PonderTalk
- Body Sensor Node Example and Exercises
- Authorisation Policies and Exercises
- Writing a new Managed Object in Java

Tutorial Installation

- Use the CD, USB key or http://ponder2.net/tutorial.zip
- Unzip Ponder2Tutorial.zip to your Desktop
- Open Ponder2Tutorial/index.html for documentation
- Open a Terminal or CMD window for running Ponder2
  - cd Desktop/Ponder2Tutorial
  - cd c:\Documents and Settings\name\Desktop\Ponder2Tutorial
  - ant rmi or ./ant rmi to test the installation
- Windows users may have to set JAVA_HOME if *ant* complains about **tool.jar** not being found.
  set JAVA_HOME=c:\Program Files\Java\jdk1.5.0_12
- These slides are in Ponder2Tutorial/Ponder2Tutorial.pdf

Tutorial Installation

- Use the CD, USB key or http://ponder2.net/tutorial.zip
- Unzip Ponder2Tutorial.zip to your Desktop
- Open Ponder2Tutorial/index.html for instructions
- Open a Terminal or CMD window for running Ponder2
  - cd Desktop/Ponder2Tutorial
  - cd c:\Documents and Settings\name\Desktop\Ponder2Tutorial
  - ant rmi or ./ant rmi to test the installation
- Windows users may have to set JAVA_HOME if *ant* complains about **tool.jar** not being found.
  set JAVA_HOME=c:\Program Files\Java\jdk1.5.0_12
- These slides are in Ponder2Tutorial/Ponder2Tutorial.pdf

Demo Time

Try the built-in client shell

ant run

In another terminal window

telnet localhost 13570

$ ls

$ cd factory

$ ls -l

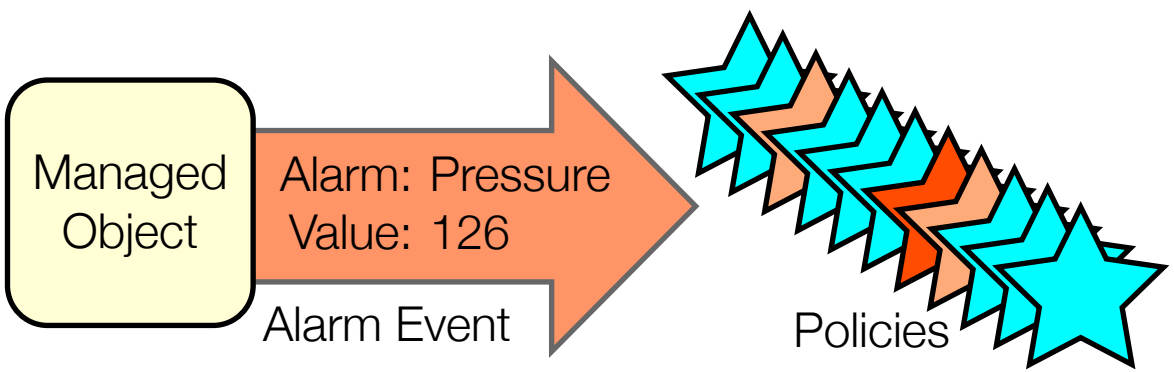
Policies

Rules governing choices in the behaviour of systems

- Obligation *or* Event Condition Action(ECA) Policies
- Authorisation Policies
- Can be dynamically changed: loaded, enabled, disabled without interrupting the system
- Are specified for groups of objects, often before objects are instantiated

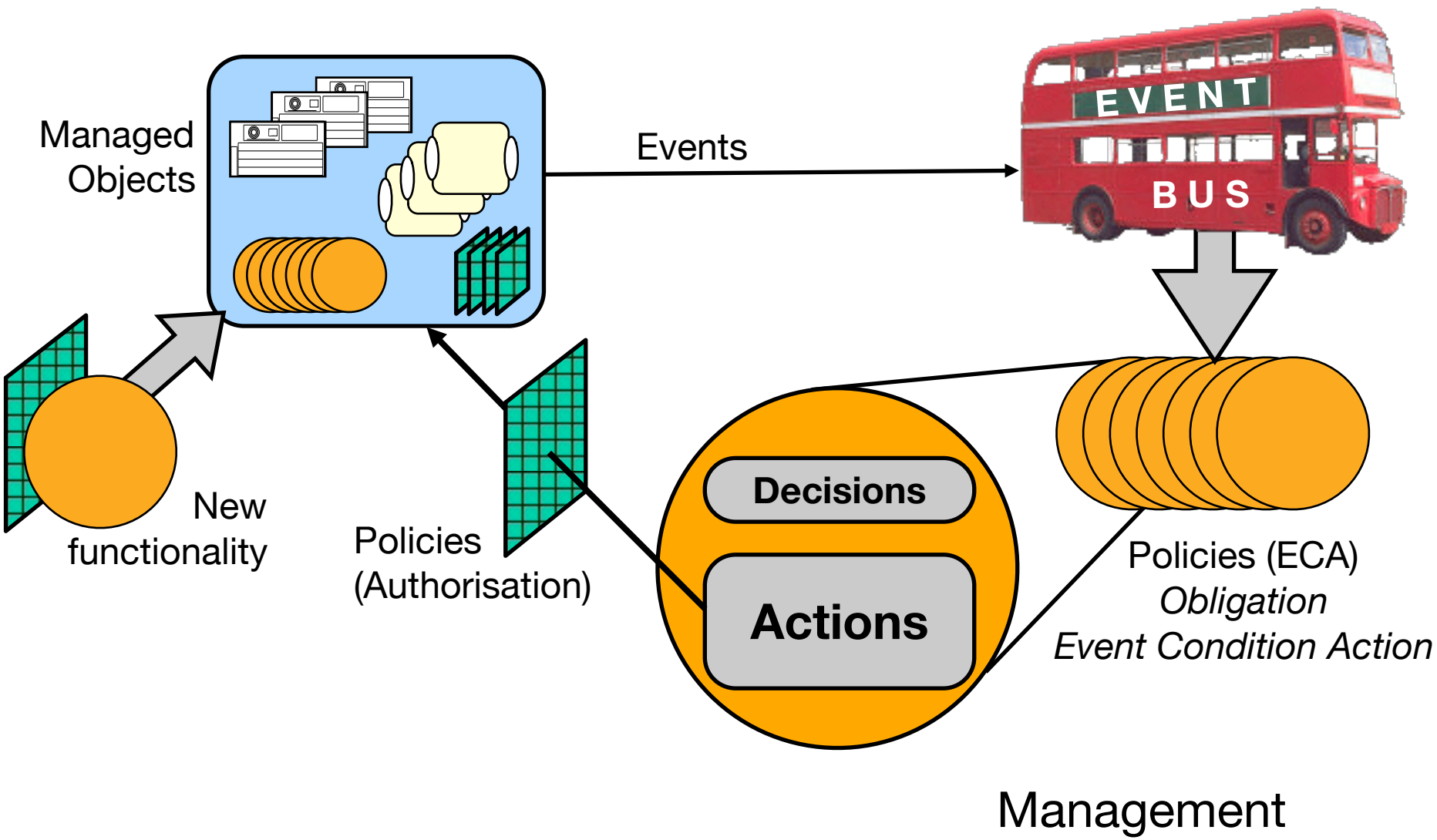
Events

- Event = notification with named attributes.
- Created by Managed Objects.
- Trigger ECA policies.



- Can integrate with one or several external event systems through adapter objects. e.g xmlBlaster

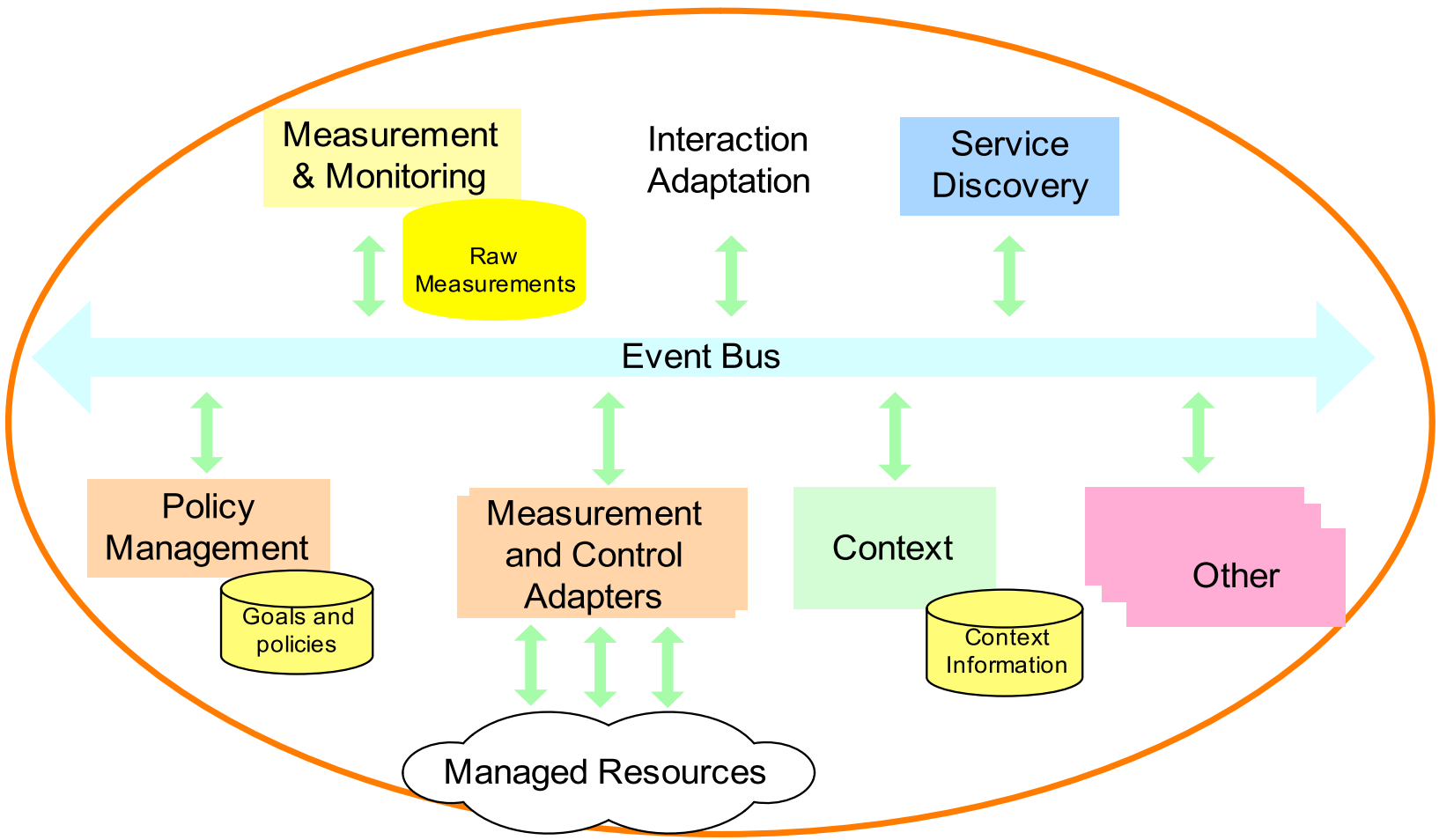
Policy-based closed adaptation loop



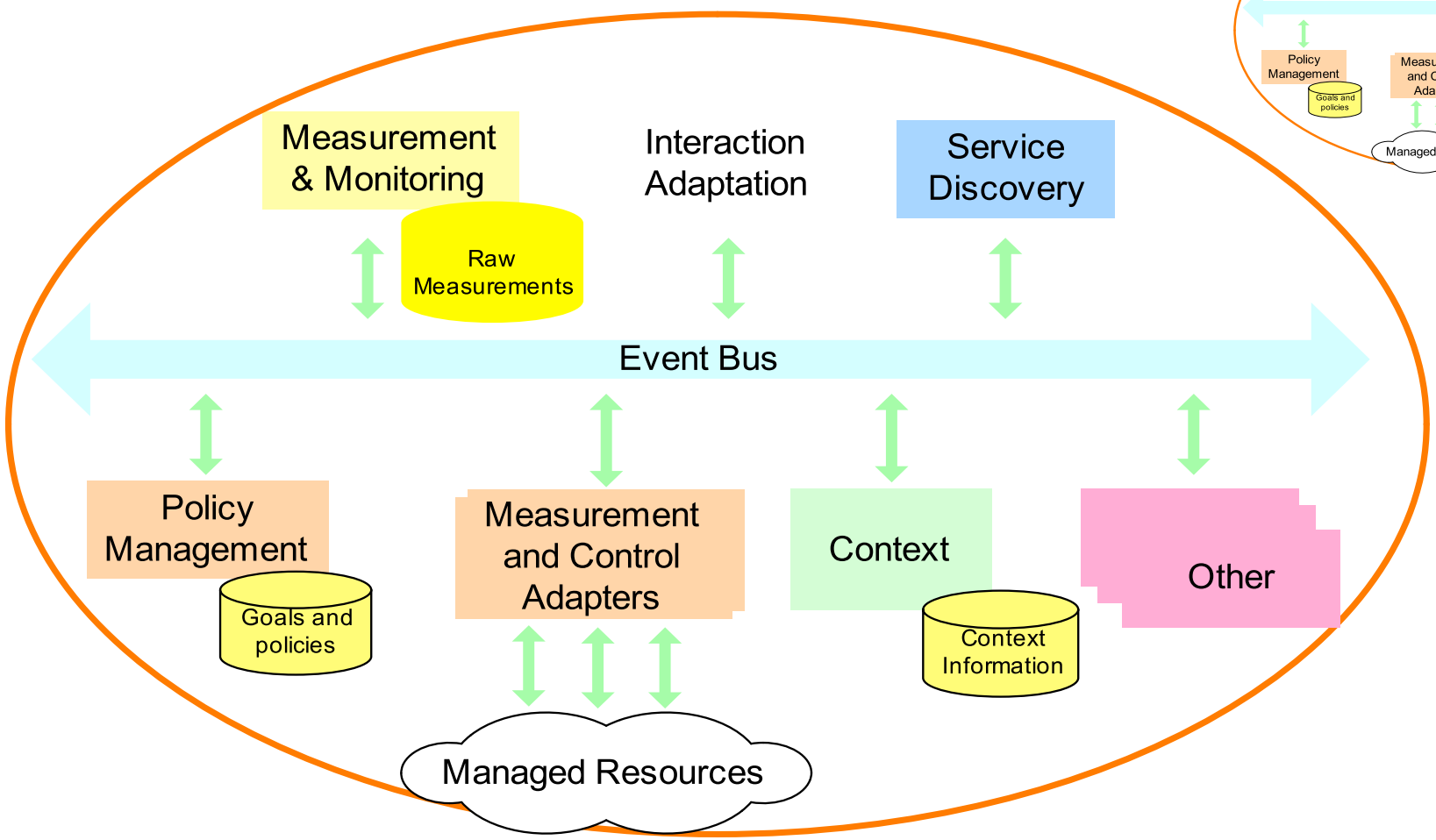
What is a Self Managed Cell?

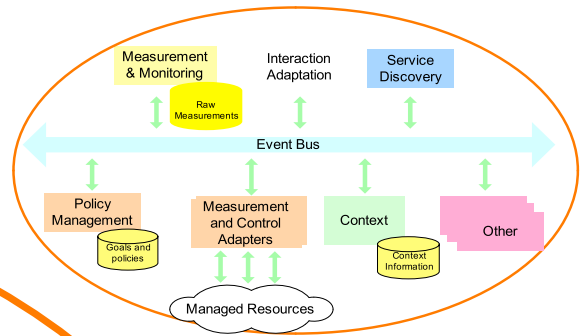
- A set of hardware and software components forming an administrative domain that is able to function autonomously and thus capable of self-management
- Management services interact with each other through asynchronous events propagated through a content-based event bus
- Policies define the management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

Self-Managed Cell (SMC)

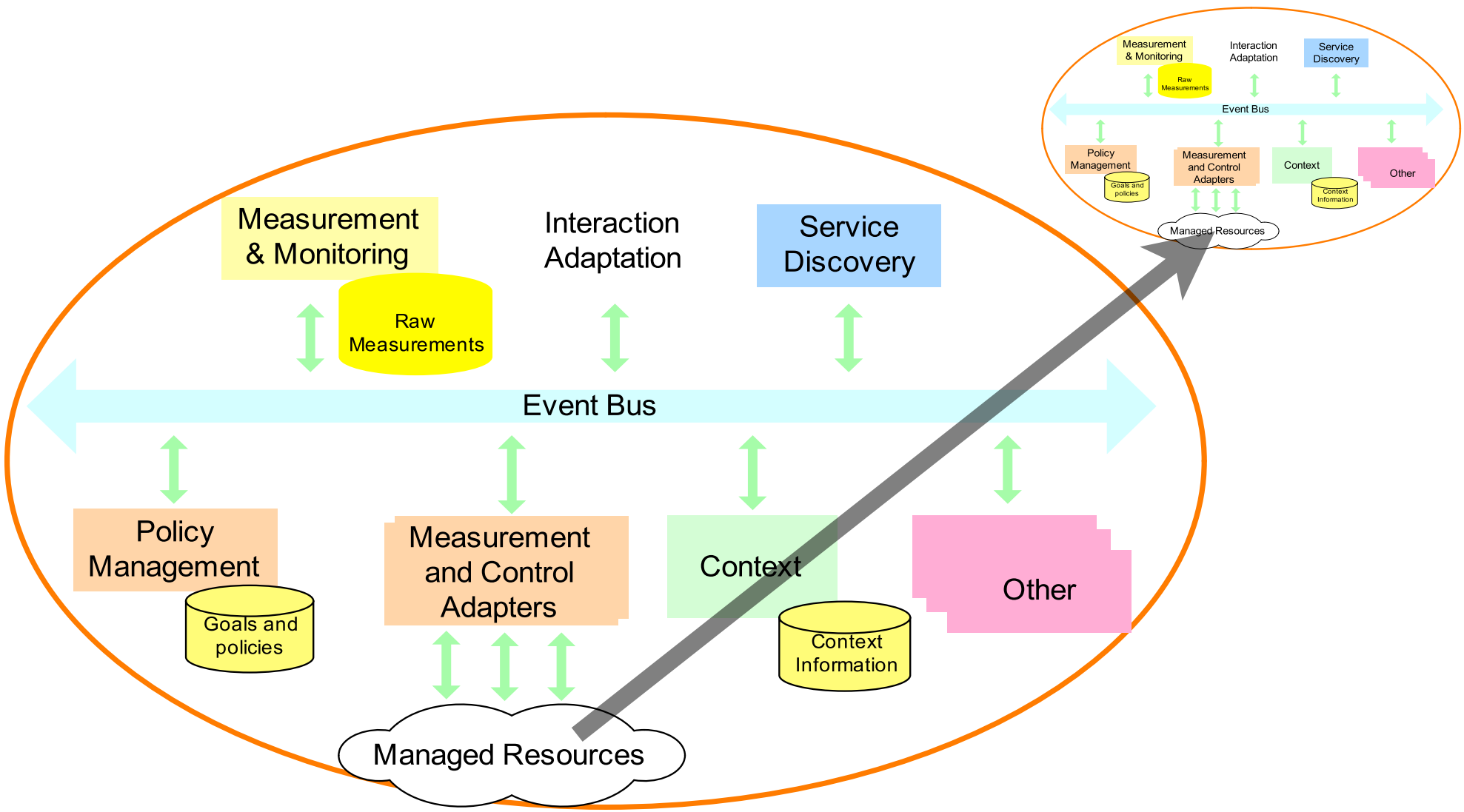


Self-Managed Cell (SMC)





Self-Managed Cell (SMC)



ECA Policy Example

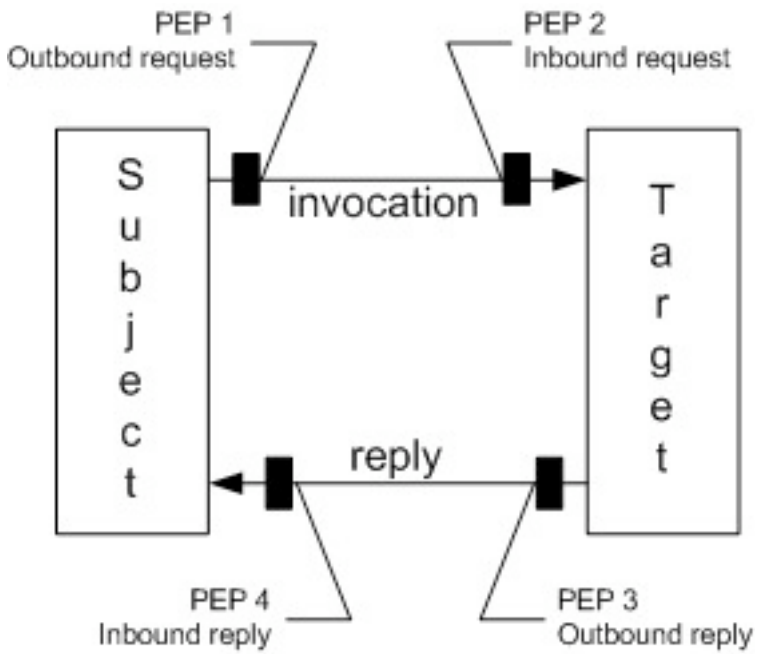
```
// Create the event type for new values from sensors, from tut2.p2
bsnvalue := root/factory/event
                    create: #( "name" "newValue" "oldValue" "rate").
root/event at: "bsnValue" put: event.
```

```
policy := root/factory/ecapolicy create.
policy event: root/event/bsnValue;                                  // From tut6.p2
       condition: [ :name :oldValue :newValue |
          // Add a debugging print statement
          root print: "Got  "+ name + " old " + oldValue + " new " + newValue.

          name == "HEART1" & (newValue > 130) & (oldValue <= 130)

         ];
       action: [
             domain/bsn/BP1 setSampleRate: 500 .
             root/alarm setAlarm: true; show.
         ].
domain/policy at: "hearthigh" put: policy.
policy active: true.
```

Authorisation Policy

Authorisation policies allow or deny requests from one managed object to another

- PEP1 and PEP4 are used to enforce authorisation policies for the subject side
- PEP2 and PEP3 are used to enforce authorisation policies for the target side



PEP1 protects the subject from revealing sensitive data

- e.g. sending a password to a phishing site, or accessing a blacklisted site

PEP2 protects the target from bad requests and bad data

- c.f. a traditional firewall or home router

PEP3 protects the target from giving out sensitive data

- protects privacy should result of an action contain info that should not be revealed

PEP4 protects the subject from receiving bad data

- e.g. buffer exploit

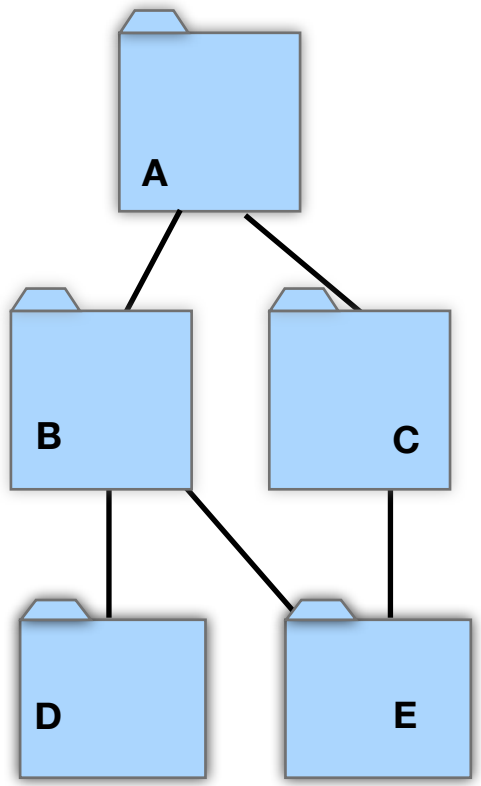
Authorisation Policy Example

```
// nurseauth.p2
// Don't allow general nurses to set the sensor rate
root/tauthdom at: "n1" put:
                (newauthpol subject: root/hospital/bedlam/nurse
                            action: "setSampleRate:"
                            target: root/hospital/bedlam/ward
                             focus: "t").
root/tauthdom/n1 reqneg.
root/tauthdom/n1 active: true.


// Allow ward nurses to set the sensor rate
root/tauthdom at: "n2" put:
                (newauthpol subject: root/hospital/bedlam/ward/Crippen/nurse
                            action: "setSampleRate:"
                            target: root/hospital/bedlam/ward/Crippen/bed
                             focus: "st").
root/tauthdom/n2 active: true.
```

See http://ponder2.net for full authorisation documentation

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

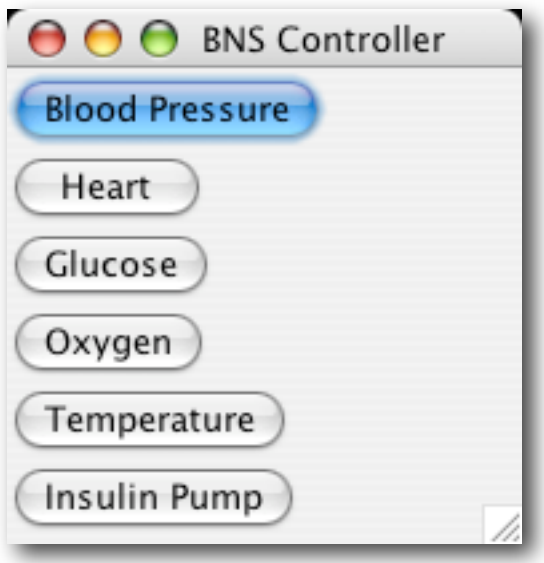


Managed Objects

- Self-contained entities capable of receiving messages
- Three pre-defined types of Managed Objects: domains, event templates, policies
- Addressed by name
  - e.g. root/policy/temppolicy
- Also basic value types including *Strings, Numbers, Arrays, Hashes, XML* etc.
- New Managed Objects e.g. device adaptors are written in Java with simple annotations to manage messages

The SMC in action

- Devices are started using the Body Sensor Node Controller
- Devices may be started and stopped by clicking on the buttons or by closing the individual device windows. Close the controller to terminate it.
- To run the BSN controller use
  ./ant bsn



The SMC in action

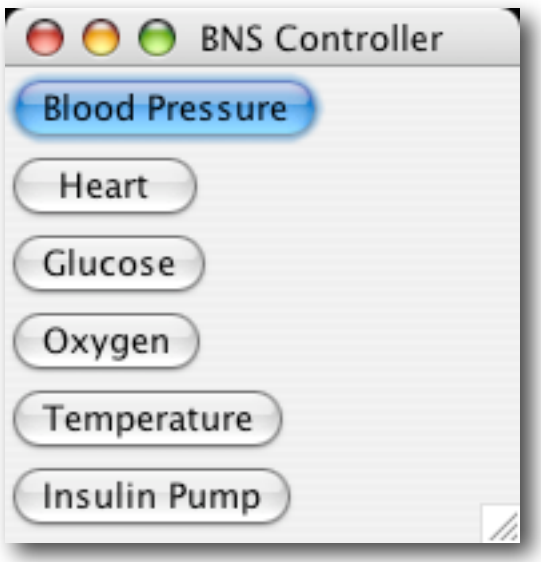
- Devices are started using the Body Sensor Node Controller
- Devices may be started and stopped by clicking on the buttons or by closing the individual device windows. Close the controller to terminate it.
- To run the BSN controller use
  ./ant bsn

Demo Time

./ant bsn

./ant tut6

- Click on different buttons to start and remove the various sensors
- Get BP and Heart windows
- Try raising the Heart Rate



ECA Policy Example

```
// Create the event type for new values from sensors, from tut2.p2
bsnvalue := root/factory/event
                    create: #( "name" "newValue" "oldValue" "rate").
root/event at: "bsnValue" put: event.
```

```
policy := root/factory/ecapolicy create.
policy event: root/event/bsnValue;                                  // From tut6.p2
       condition: [ :name :oldValue :newValue |
          // Add a debugging print statement
          root print: "Got  "+ name + " old " + oldValue + " new " + newValue.

          name == "HEART1" & (newValue > 130) & (oldValue <= 130)

         ];
       action: [
             root/bsn/BP1 setSampleRate: 500 .
             root/alarm setAlarm: true; show.
         ].
root/policy at: "hearthigh" put: policy.
policy active: true.
```

ECA Policy Example

```
// Create the event type for new values from sensors, from tut2.p2
bsnvalue := root/factory/event
            create: #("name" "newValue" "oldValue" "rate").
root/event at: "bsnValue" put: event.

polic
policy event: root/event/bsnValue;                    // From tut6.p2
        // Add a debugging print statement
        root print: "Got " + name + " old " + oldValue + " new " + newValue.
        name == "HEART1" & (newValue > 130) & (oldValue <= 130)

        ];
      action: [
            root/bsn/BP1 setSampleRate: 500 .
            root/alarm setAlarm: true; show.
        ].
root/policy at: "hearthigh" put: policy.
policy active: true.
```

Demo Time

- Now turn off the policy to raise the alarm

telnet localhost 13570

$ ls policy

$ root/policy/hearthigh active: false.

$

- Try raising and lowering the Heart Rate

Ponder2 Language - PonderTalk

- Ponder2 Managed Objects are written in Java
- Need a language that can be used to specify Managed Objects and the messages to be sent to them
- Ponder2 version 1 used XML as a configuration and control language - very messy!
- PonderTalk is based on Smalltalk
    - **Smalltalk language was designed to send messages to objects**

Basic Syntax

- PonderTalk is a sequence of statements
- Statements are like sentences they are separated with a full-stop (period)
- A statement specifies a receiver (object) and a message (command) to be sent to the receiver
- The receiver returns another object (or itself) in response to a message

```
object message. object message. (object message) message
```

Message Types

- Unary
  - No argument values

```
train stop.
```

- Binary
  - One argument value

```
train + carriage.
```

- Keyword
  - One or more arguments

```
train goto: "London" at: 1400.
```

Unary Message

- Messages with no arguments
- c.f. function calls with no arguments

PonderTalk	Java
`train start`	`train.start()`
`train stop`	`train.stop()`

Binary Message

- Symbol and argument sent to receiver

```
train + carriage.
```

- Train is sent the message “+ carriage”. Train adds the carriage to the back of its carriages and returns itself (the train). This allows the following:

```
train + carriage1 + carriage2 + carriage3.
```

- c.f.

```
((train + carriage1) + carriage2) + carriage3.
```

PonderTalk	Java
`train + carriage`	`train.plus(carriage)`
`“string1” == “string2”`	`“string1”.equals(“string2”)`

Keyword Message

- Functions with one or more arguments
- All arguments are named

PonderTalk	Java
`train setSpeed: 60`	`train.setSpeed(60)`
`train goto: “London” at: 1200`	`train.setOffFor(“London”, 1200)`

- Keyword arguments are referred to as a concatenation
  e.g. goto:at:
  (this will be used later when writing Java objects)

Precedence

- Unary > Binary > Keyword
- Same Precedence - Left to Right

As Written	Precedence
`train stop + carriage`	`(train stop) + carriage`
`train + carriage start`	`train + (carriage start)`
`(train + carriage) start`	`(train + carriage) start`
`train stop start`	`(train stop) start`
`train speed: train max`	`train speed: (train max)`
`3 + 4 * 5`	`( 3 + 4 ) * 5`
~~`3 plus: 4 times: 5`~~	`(3 plus: 4) times: 5`

Context variables

- Context variables are named objects that are referred to in the context of the statement. They are not referenced through the domain structure
- May be used as temporary variables
- Use *assignment* to create a context variable then use as any other object

```
// Hold a factory object
alarmfact := root load: "AlarmDisplay".
// Create an instance
alarmfact create.
// Hold a number (unrelated to alarmfact)
size := 43.
size := size * 2.
```

Return Values

- The return value from a command sent to an object is determined by the object
  - A train getting an extra carriage may return itself
  - A factory (c.f. class) receiving a create message will return a new object of the correct type
  - In a Java Managed Object a void method returns the Managed Object itself
- The following returns a new Domain Managed Object

```
root/factory/domain create.
```

Blocks

```
myBlock := [:msg | root print: msg].
```

- Statements can be grouped into blocks
- Blocks are objects
- Blocks can take arguments
- Block execution is delayed
- Blocks are closures
- Blocks return the result of their last statement
- c.f. dynamic functions

Blocks - Execution

- A simple block

```
myBlock := [root print: "Hello, World!"].
```

- To execute it

```
myBlock value.
```

- or

```
[root print: "Hello, World!"] value.
```

Blocks - Execution

- A simple block

```
myBlock := [root print: "Hello, World!"].
```

- To execute it

```
myBlock value.
```

- or

```
[root print: "Hello, World!"] value.
```

==

```
root print: "Hello, World!".
```

Blocks - Closure

- A block is a closure

```
message := "Hello, World!".
myBlock := [root print: message].

message := "Goodbye, Cruel World!".
myBlock value.
```

- Prints “Hello, World!”

Blocks - Arguments

- A block can declare arguments
- value: keywords are used to give the arguments values

```
myBlock := [:arg1 :arg2 | arg1 > arg2].
```

- To execute the block

```
myBlock value: 30 value: 20.
```

- What does this statement return?

Cascaded Messages

- Shorthand for sending a series of messages to the same object

```
// Separate statements
event addArg: "a1".
event addArg: "a2".
event addArg: "a3".
```

- Cascaded statements use Semicolon - ";"

```
// A cascaded statement
event addArg: "a1";
      addArg: "a2";
      addArg: "a3".
```

- See Policy examples

Basic Types

- Basic Types include Domains, Arrays, Hashes, Strings, Numbers and XML
- root is type SelfManagedCell - print, trace, load, import
- SelfManagedCell is a Domain (c.f. System in Java)
- Domains are Hashes
- Most objects can be treated as collections
- Full documentation in

```
Ponder2Tutorial/index.html
```

Factory Managed Object

- Used to create a new Managed Object
- Loading new Managed Object code (currently only from a Jar or Java class file) produces a Factory Object

```
factory := root load: "MyManagedObject".
root load: "net.ponder2.myproject.MyManagedObject".
```

- Accepts Managed Object specific create commands and returns a new instance of a Managed Object

```
newobj1 := factory create.
newobj2 := factory create: "argument".
newobj2 := factory name: "Sheep" initialValue: 12.
```

Basic Operations

- See PonderTutorial/index.html-Ponder2 Docs-P2Hash
- Hashes - at:, at:put:

```
root at: "newName" put: managedObject.
(root at: "newName") msg.
root/newName msg.
```

- Hashes - creation using an array

```
newHash := #( name value name value ) asHash.
```

- Collections, do:, collect:

```
root/mydomain do: [ :name :obj |
                        root/copydomain at: name put: obj.
                    ]
```

Bootstrapping Ponder2

- SMC is just an empty domain
- Import domain factory
- Create domains
- Import basic factories
- Read more PonderTalk

```
// Bootstrap code for Ponder2

// Import the Domain code
// and create the default domains
domainFactory := root load: "Domain".
root
   at: "factory" put: domainFactory create;
   at: "policy" put: domainFactory create;
   at: "event" put: domainFactory create.

// Put the domain factory into the factory directory
root/factory at: "domain" put: domainFactory.

// Import event and policy factories
root/factory
   at: "event" put: ( root load: "EventTemplate" );
   at: "ecapolicy" put: ( root load: "ObligationPolicy" ).

// Delete the variable
// System remove: "domainFactory".
```

Bootstrapping Ponder2

- SMC is just an empty domain
- Import domain factory
- Create domains
- Import basic factories
- Read more PonderTalk

```
// Bootstrap code for Ponder2

// Import the Domain class
// and create the default domains
domainFactory := root load: "Domain".
root
   at: "factory" put: domainFactory create;
   at: "policy" put: domainFactory create;
   at: "event" put: domainFactory create.

// Put the domain factory into the factory directory
root/factory at: "domain" put: domainFactory.

// Import event and policy factories
root/factory
   at: "event" put: ( root load: "EventTemplate" );
   at: "ecapolicy" put: ( root load: "ObligationPolicy" ).

// Delete the variable
// System remove: "domainFactory".
```

Demo Time

ant empty

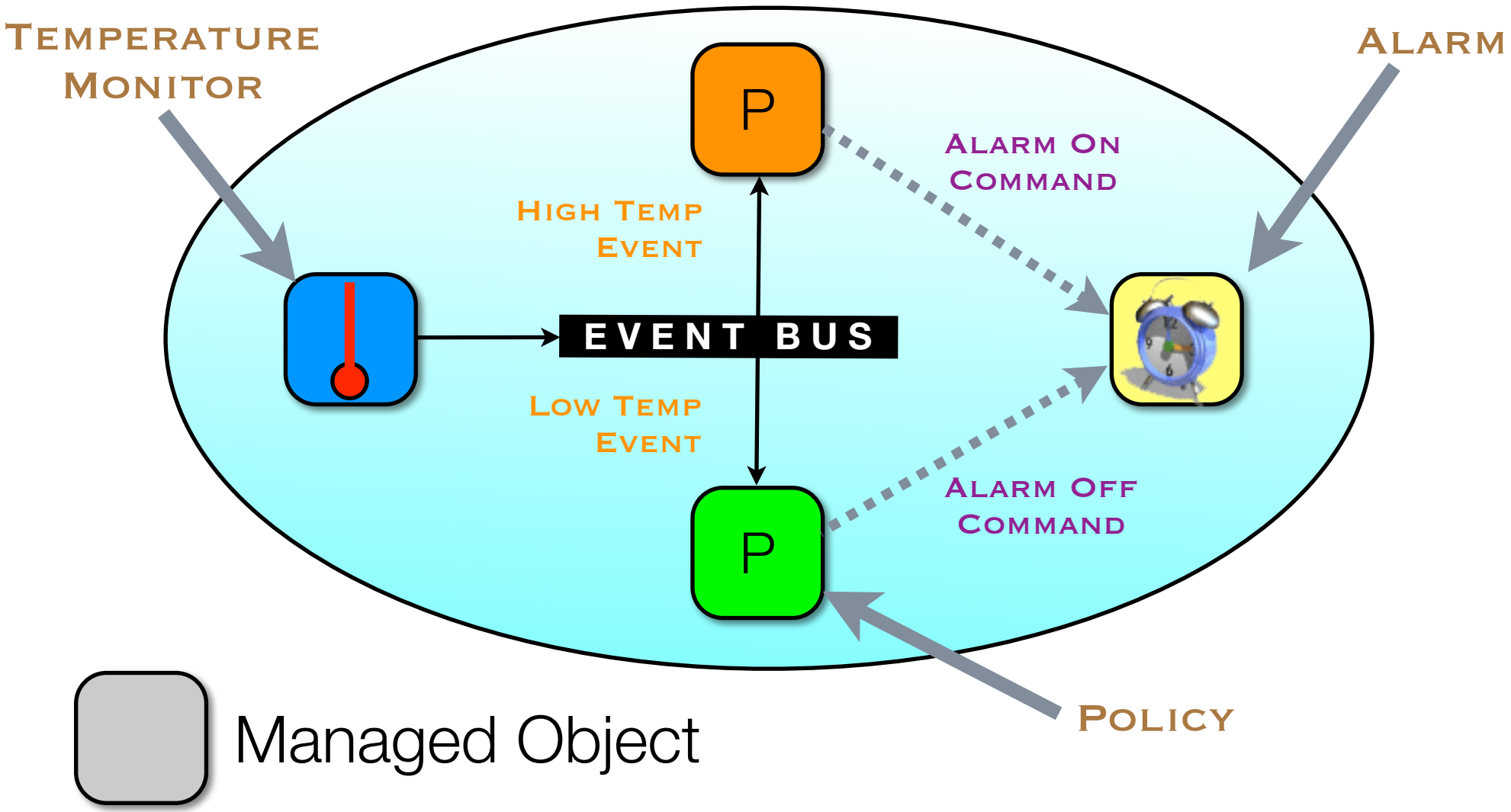
telnet localhost 13570

$ ls

$ read boot.p2

$ ls

Example Events and Policies



PonderTalk in Action

- refer to an object send it commands
- Domain - at:put:, at:, remove:, list
- Root domain - load:
- Factory - create
- Alarm - show, hide

/factory/alarm
/alarm

```
// Import the Alarm display
root/factory at: "alarm"
                put: (root load: "AlarmDisplay").

// Create an alarm instance -->
root at: "alarm"
     put: (root/factory/alarm create).
```

```
// Import the Alarm display
alarmfact := root load: "AlarmDisplay".
root/factory at: "alarm" put: alarmfact.

// Create an alarm instance -->
alarminst := root/factory/alarm create.
root at: "alarm" put: alarminst.
```

Documentation in doc directory

PonderTalk in Action

- refer to an object send it commands
- Domain - at:put:, at:, remove:, list
- Root domain - load:
- Factory - create
- Alarm - show, hide

/factory/alarm
/alarm

```
// Import the Alarm display
root/factory/alarm
        put: (root load: "AlarmDisplay").

// Create an alarm instance
root at: "alarm"
     put: (root/factory/alarm create).
```

```
// Import the Alarm display
alarmfact := root load: "AlarmDisplay".
root/factory at: "alarm" put: alarmfact.

// Create an alarm instance -->
alarminst := root/factory/alarm create.
root at: "alarm" put: alarminst.
```

Demo Time

ant tut1

telnet localhost 13570
$ ls
$ root/alarm show.

Documentation in doc directory

Events

- Notification with named values
- Event Templates created from the Event Factory
- Events created using Event Templates
- Events contain named arguments and their values
- Events trigger policies

```
// Create template /event/toohigh
newevent := root/factory/event.

root/event at: "toohigh"
            put: (newevent create:
                    #( "message" "value" )
                  ).
```

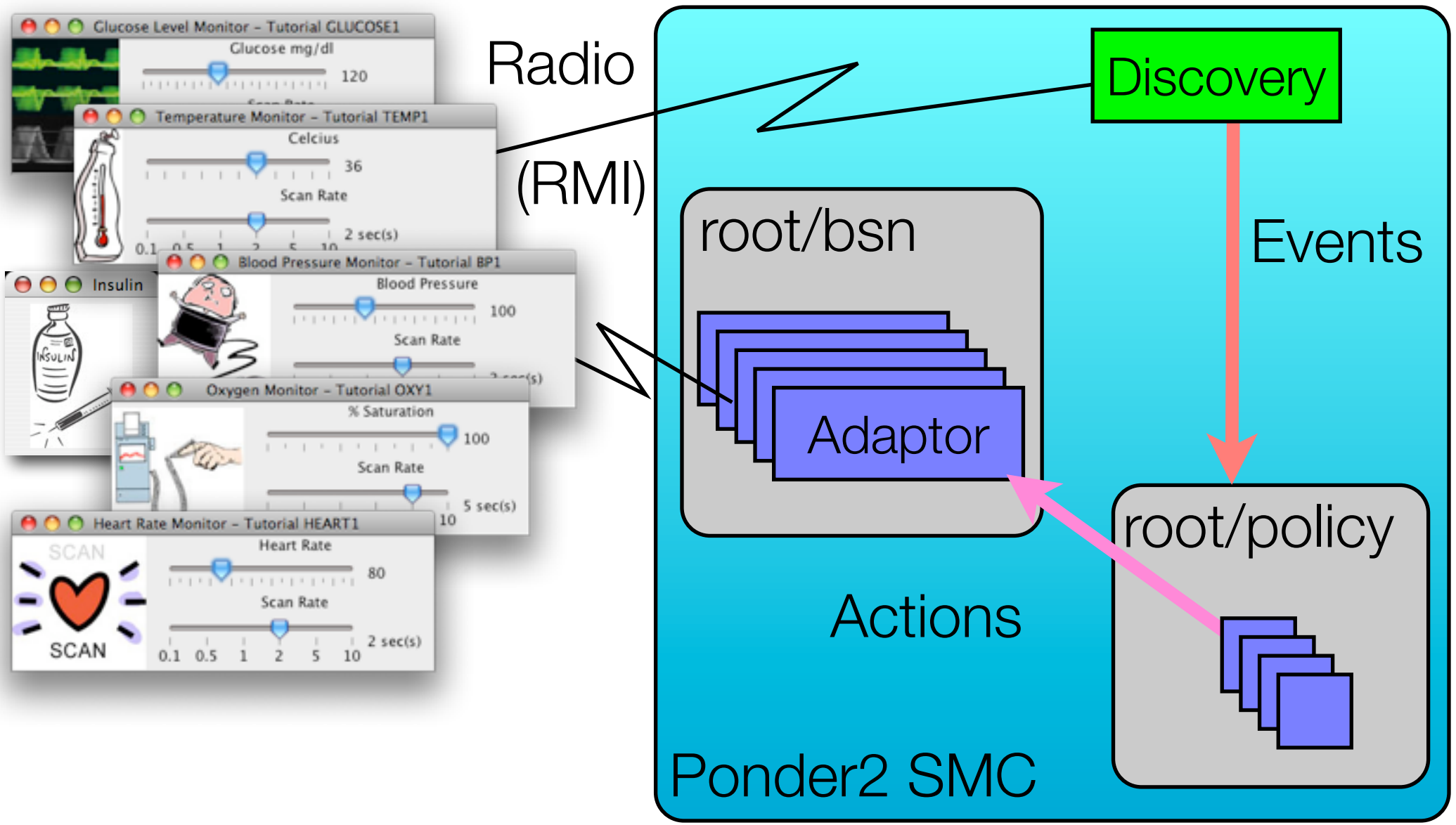
```
// Create template /event/toohigh
newevent := root/factory/event.

sensorevent := newevent create:
                    #( "message" "value" )
root/event at: "toohigh"
            put: sensorevent.
```

Policies

- Created with policy factory
- Dynamically associate an event with a policy
- Can be activated and deactivated
- Are managed objects. Can be moved, deleted, created, activated, deactivated by other policies

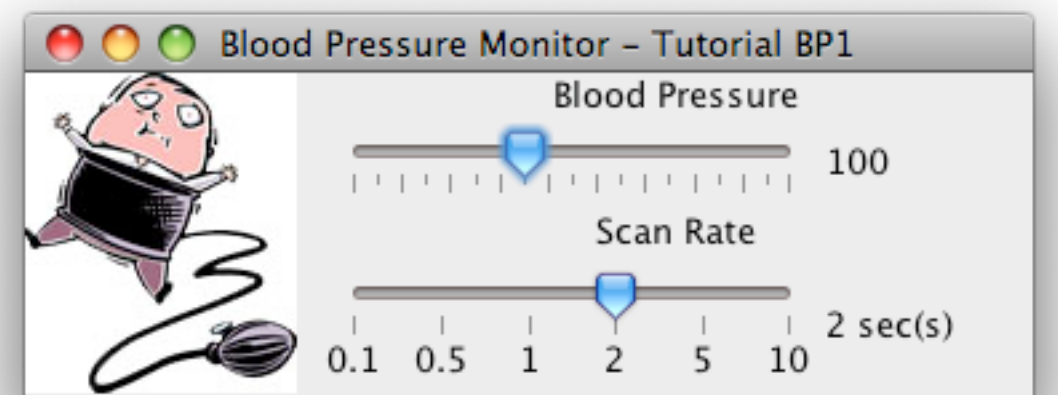
```
// Create policy /policy/toohigh
newpolicy := root/factory/ecapolicy.

root/policy at: "toohigh"
             put: (newpolicy create).

root/policy/toohigh
   event: root/event/toohigh;
   condition: [ :value | value > 10 ];
   action: [ :message |
     root print: "Got event " + message ];
   setActive: true.
```

Body Sensor Network (BSN) Example



BSN Simulation

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



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

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

```
newevent := root/factory/event.

// define root/event/newBSN
root/event
    at: "newBSN"
    put: ( newevent create: #("name" "type") ).

// example of creating an event
root/event/newBSN create: #("Temp1" "TempMon").
```

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

```
// Create discovery policy
newpolicy := root/factory/ecapolicy.

newBSN := newpolicy create.
newBSN
    event: root/event/newBSN;
    action: [ :name :type |
                root/template/bsnAdaptor
                        create: name
                        setActive: type
            ];
    setActive: true.
```

Heart Rate Policy

- on heartrate(value)

  if (value>130)
  && oldValue<=130

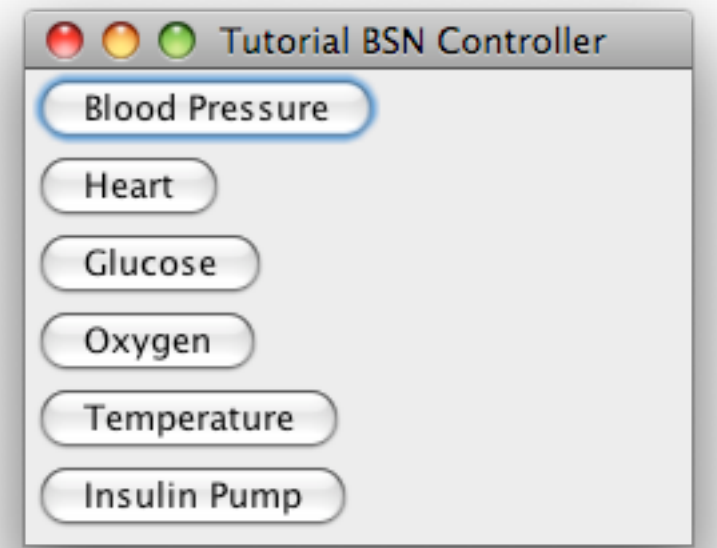
- do
  /bsn/BP1
  .set(sensorRate=1)

  /alarm.alarm(on)
  /alarm.show

```
// Create blood pressure policy
newpolicy := root/factory/ecapolicy.

hearthigh := newpolicy create.
hearthigh
   event: root/event/bsnvalue;
   condition: [ :name :newValue :oldValue |
                     name == "HEART1"
                     && ( newValue > 130 )
                     && ( oldValue < 130 ) ];
   action: [
         root/bsn/BP1  setRate: 1.
         root/alarm setAlarm: true; show ];
   setActive: true.

root/policy at: "hearthigh" put: hearthigh
```

BSN Simulation

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



BSN Simulation

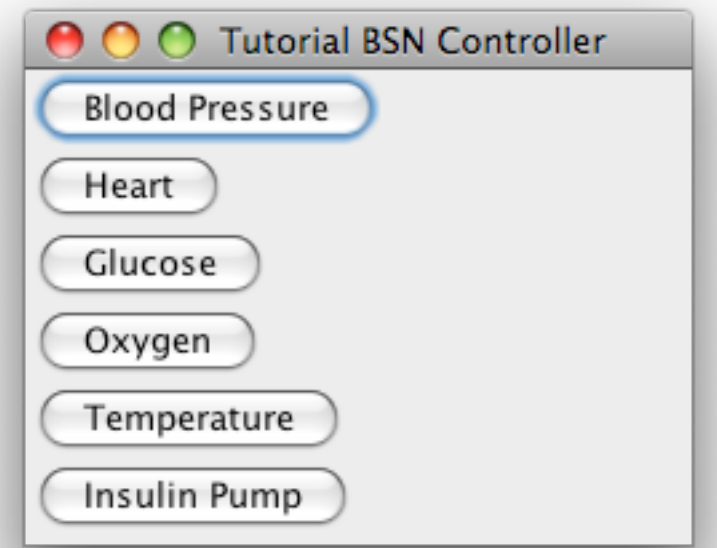
- Devices are started using the BSN Controller
- Devices may be started and stopped by clicking on the buttons or by closing the individual device windows. Close the controller to terminate it.
- To run the BSN controller use ./ant bsn

Demo Time

ant bsn

ant tut6

- Click on different buttons to start and remove the various sensors
- Get BP and Heart windows
- Try raising the Heart Rate



Exercise 1 - Policy writing

- Detect high glucose level, activate Insulin pump
- ex1.p2 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 - Policy writing

- Detect high glucose level, activate Insulin pump
- ex1.p2 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 Time

- Use the exercise documentation found in **index.html**
- At the bottom click on **Exercise 1**
- Open **ex1.p2** and fill in the blanks
- Run it with **./ant ex1**

Exercise 1 - New/Lost Pump policy

- on event newPump(name)
  create new pumpadaptor in /bsn/name
- Pumpadaptor create: takes argument 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 - Alarm commands

- Factory Commands
- `create`
- `create: “title”`
- Operation Commands
- `alarm title: “title”`
- `alarm setAlarm: true`
- `alarm show`
- `alarm hide`

Notes - BSNAdaptor commands

- Factory Commands
- `create: “name”`
- Operation Commands
- `rate: “value”`

Notes - PumpAdaptor commands

- Factory Commands
- `create: "name"`
- Operation Commands
- `injectDose: number`

Notes - Running your PonderTalk

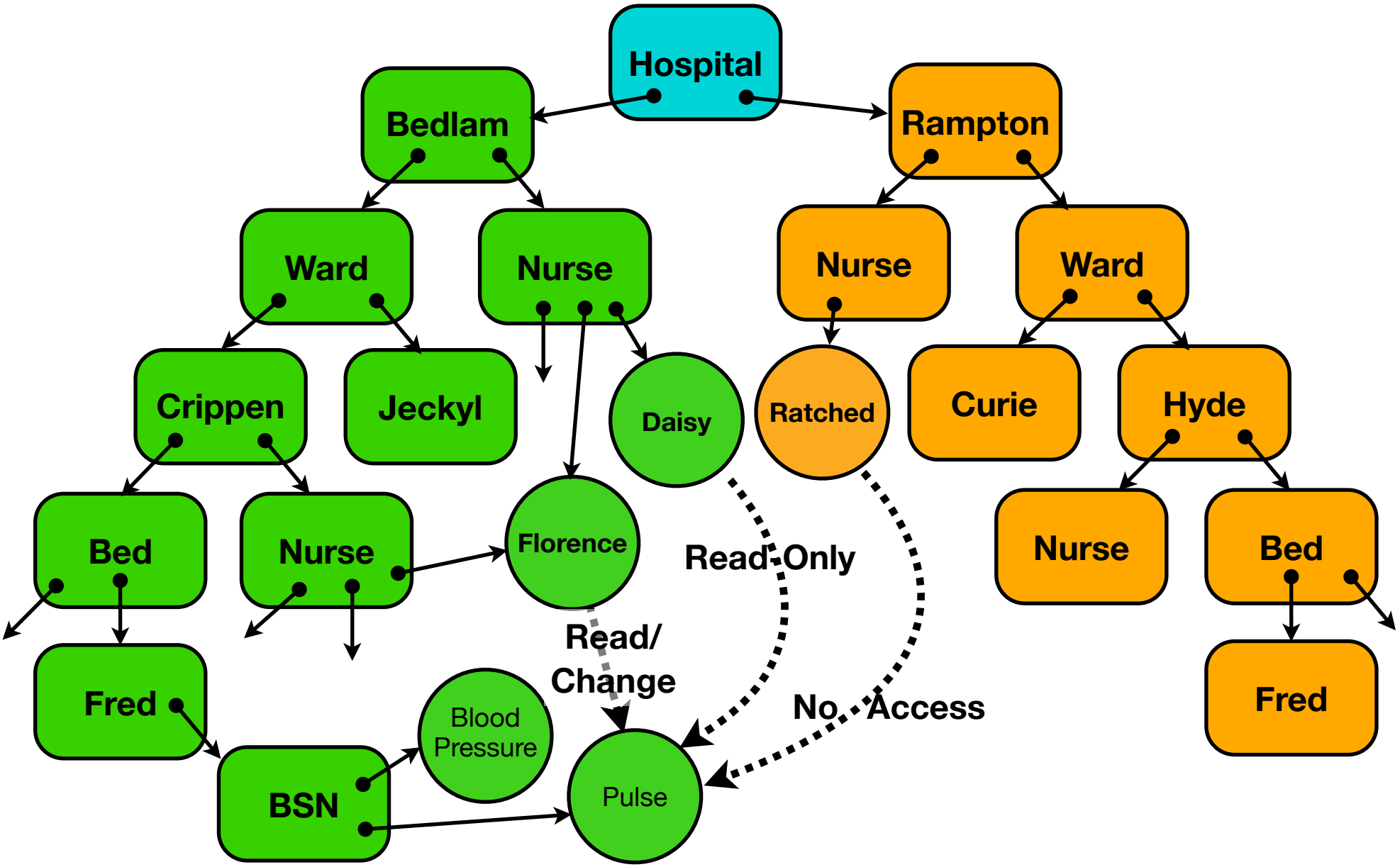
- Create your PonderTalk in a new file called ex1.p2
- The ant file will run the complete tutorial example and will read ex1.p2 when using the following
- ant ex1
- Add a little PonderTalk at a time to ex1.p2, then run it. When it's working, add a little more, then run it. etc. etc. Use the shell to inspect your objects.
- Use print statements for debugging

```
root print: "Value now " + value.
```

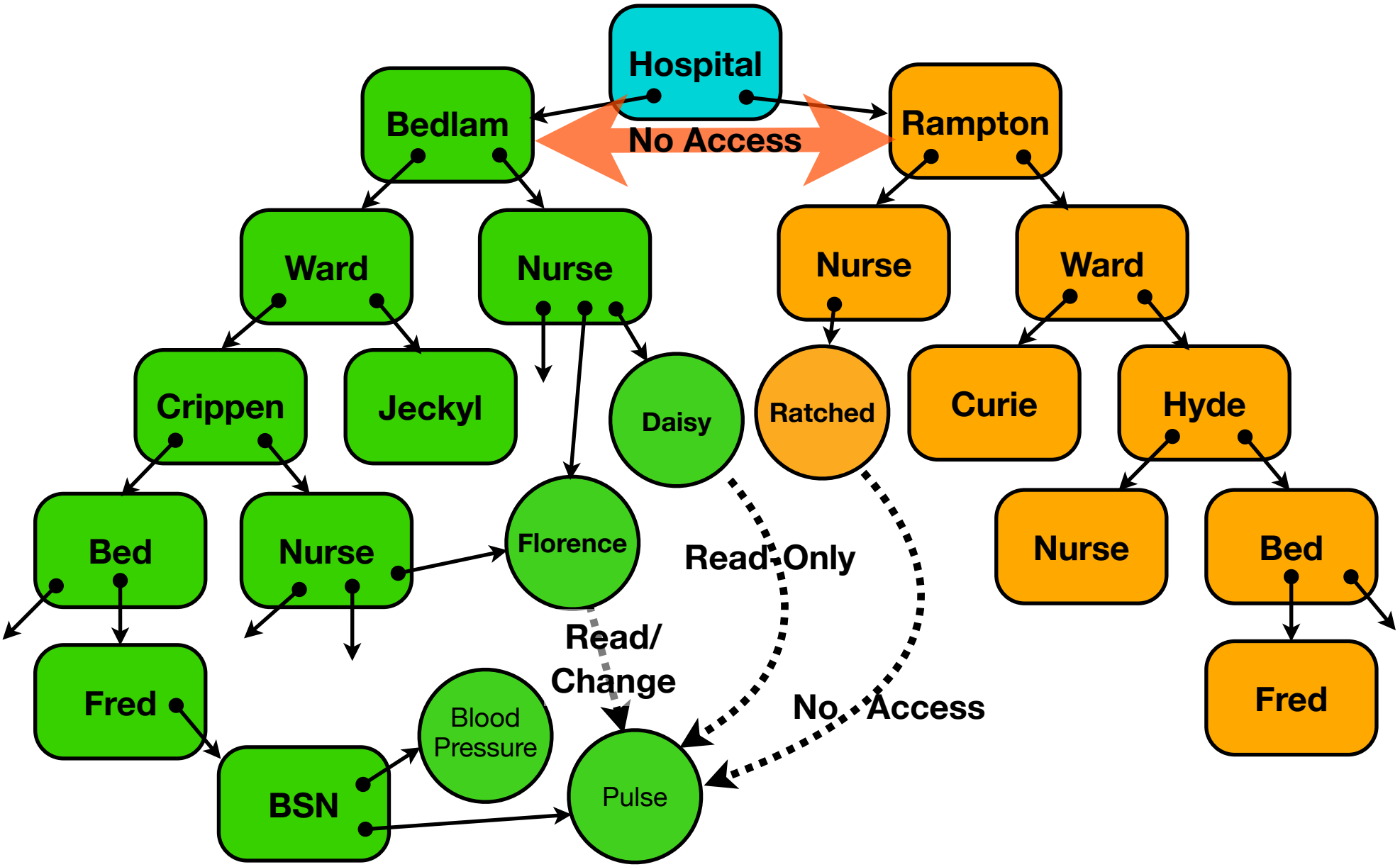
Hospital Example

- Hospital
  - Nurses - who work at the hospital
  - Wards
    - Nurses - who work in the ward
    - Beds
      - Patient
      - Body Sensor Nodes

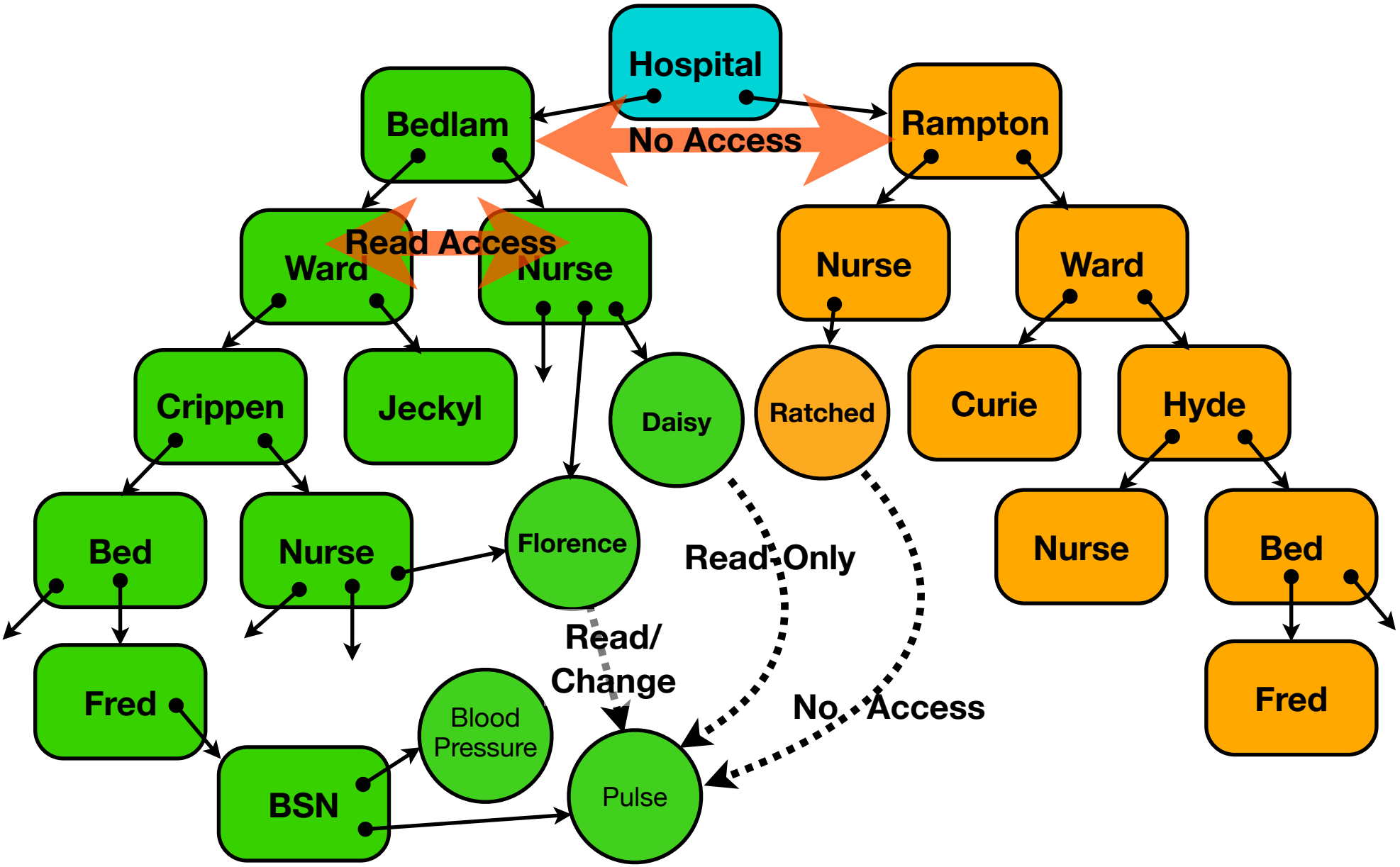
Hospital Example



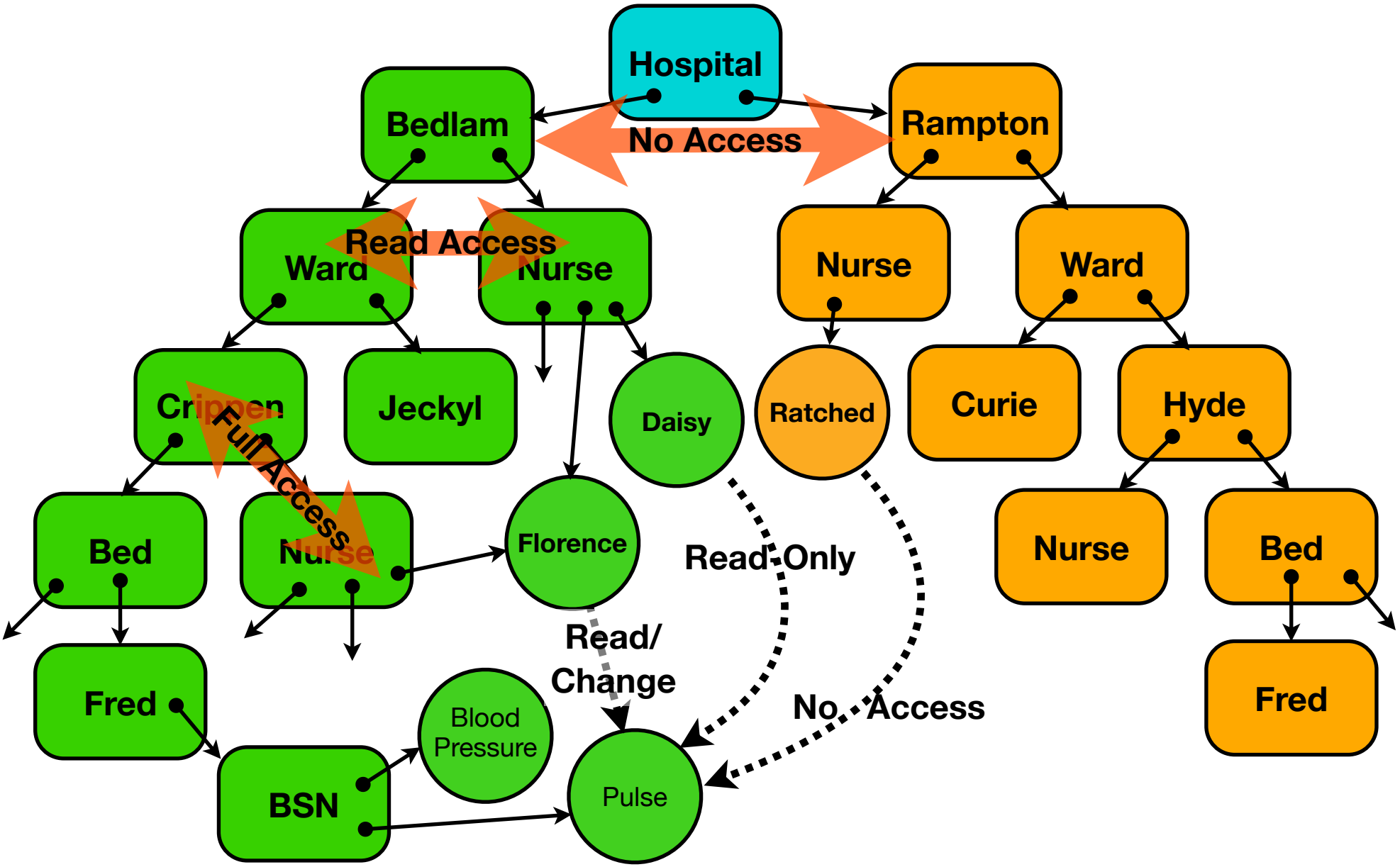
Hospital Example



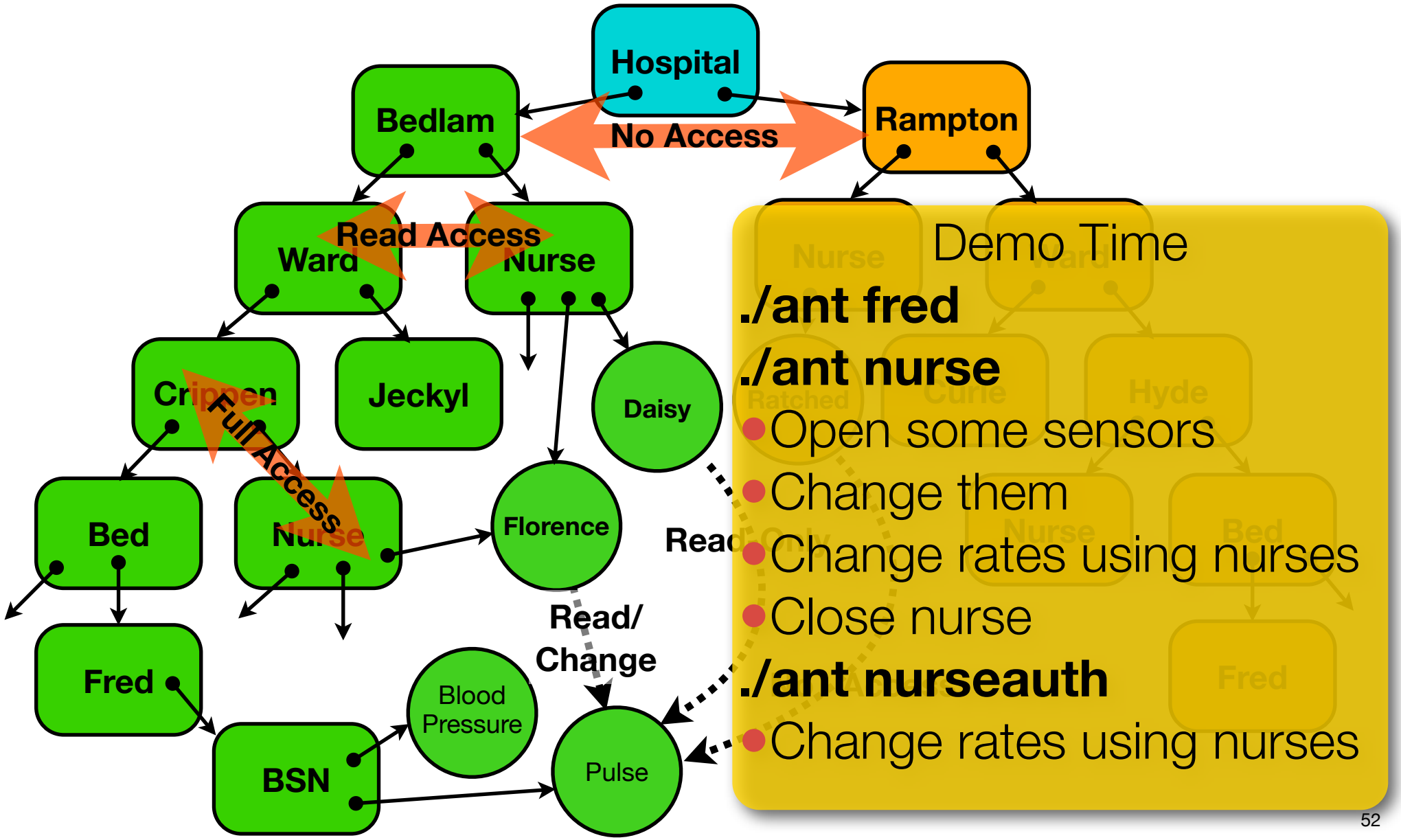
Hospital Example



Hospital Example



Hospital Example



Exercise 3

- Comes before exercise 2!
- Write an authorisation policy to allow nurse Ratched from Rampton hospital to access patient Fred's readings but not to change them
- You will then use the shell to disable and enable this policy to see its the immediate effects upon nurse Ratched

Exercise 3

- Comes before exercise 2!
- Write an authorisation policy to allow nurse Ratched from Rampton hospital to access patient Fred's readings but not to change them
- You will then use the shell to disable and enable this policy to see its the immediate effects upon nurse Ratched

Exercise Time

- Use the exercise documentation found in **index.html**
- At the bottom click on **Exercise 3**
- Open **ex3.p2** and fill in the blanks
- Run it with **./ant ex3**

Writing a new Managed Object

- A Managed Object is a Java class
- PonderTalk messages converted to method calls
- Constructors called by factory messages
- Instance methods called by operational messages
- Mapping done by @Ponder2op Java annotation
  - Annotations introduced in Java 1.4
  - Provides compile time and run time information
  - e.g. @Override to indicate an overriding method
  - uses apt Annotation Processing Tool instead of javac

Writing a new Managed Object

```
package net.ponder2.managedobject;

import java.util.HashMap;
import java.util.Map;

import net.ponder2.apt.Ponder2op;
import net.ponder2.ManagedObject;
import net.ponder2.objects.P2Object;

/**
 * Implements a hash or dictionary.
 * This object holds name/managed object pairs.
 * Objects may be added and removed.
 */
public class MyManagedObject implements ManagedObject {

  private Map<String, P2Object > data;

  /**
   * Creates an empty hash
   */
  @Ponder2op("create")
  MyManagedObject() {
    data = new HashMap<String, P2Object >();
  }

  /**
   * Creates a hash with a particular minimum size
   */
  @Ponder2op("size:")
  MyManagedObject(int size) {
    data = new HashMap<String, P2Object >(size);
  }

  /**
   * adds an object to the hash
   */
  @Ponder2op("at:put:")
  P2Object store(String name, P2Object obj) {
    data.put(name, obj);
    return obj;
  }

  /**
   * retrieves an object by name
   */
  @Ponder2op("at:")
  P2Object p2_operation_at(String name) {
    return data.get(name);
  }

  /**
   * removes the named object from the hash
   */
  @Ponder2op("remove:")
  P2Object p2_operation_remove(String name) {
    return data.remove(name);
  }

  /**
   * get the number of objects in the hash
   */
  @Ponder2op("size")
  int p2_operation_size() {
    return data.size();
  }

}
```

Writing a new Managed Object

```
package net.ponder2.managedobject;

import java.util.HashMap;
import java.util.Map;

import net.ponder2.apt.Ponder2op;
import net.ponder2.ManagedObject;
import net.ponder2.objects.P2Object;

/**
 * Implements a hash or dictionary.
 * This object holds name/managed object pairs.
 * Objects may be added and removed.
 */
public class MyManagedObject implements ManagedObject {

  private Map<String, P2Object > data;

  /**
   * Creates an empty hash
   */
  @Ponder2op("create")
  MyManagedObject() {
    data = new HashMap<String, P2Object >();
  }

  /**
   * Creates a hash with a particular minimum size
   */
  @Ponder2op("size:")
  MyManagedObject(int size) {
    data = new HashMap<String, P2Object >(size);
  }

  /**
   * adds an object to the hash
   */
  @Ponder2op("at:put:")
  P2Object store(String name, P2Object obj) {
    data.put(name, obj);
    return obj;
  }

  /**
   * returns the object by name
   */
  @Ponder2op("at:")
  P2Object p2_operation_at(String name) {
    return data.get(name);
  }

  /**
   * removes the named object from the hash
   */
  @Ponder2op("remove:")
  P2Object p2_operation_remove(String name) {
    return data.remove(name);
  }

  /**
   * get the number of objects in the hash
   */
  @Ponder2op("size")
  int p2_operation_size() {
    return data.size();
  }

}
```

Indicates that this is to be a Managed Object

Writing a new Managed Object

```
package net.ponder2.managedobject;

import java.util.HashMap;
import java.util.Map;

import net.ponder2.apt.Ponder2op;
import net.ponder2.ManagedObject;
import net.ponder2.objects.P2Object;

/**
 * Implements a hash or dictionary.
 * This object holds name/managed object pairs.
 * Objects may be added and removed.
 */
public class MyManagedObject implements ManagedObject {

  private Map<String, P2Object > data;

  /**
   * Creates an empty hash
   */
  @Ponder2op("create")
  MyManagedObject() {
    data = new HashMap<String, P2Object >();
  }

  /**
   * Creates a hash with a particular minimum size
   */
  @Ponder2op("size:")
  MyManagedObject(int size) {
    data = new HashMap<String, P2Object >(size);
  }

  /**
   * adds an object to the hash
   */
  @Ponder2op("at:put:")
  P2Object store(String name, P2Object obj) {
    data.put(name, obj);
    return obj;
  }

  /**
   * retrieves an object by name
   */
  @Ponder2op("at:")
  P2Object p2_operation_at(String name) {
    return data.get(name);
  }

  /**
   * removes the named object from the hash
   */
  @Ponder2op("remove:")
  P2Object p2_operation_remove(String name) {
    return data.remove(name);
  }

  /**
   * get the number of objects in the hash
   */
  @Ponder2op("size")
  int p2_operation_size() {
    return data.size();
  }

}
```

The factory will map create to this constructor

Writing a new Managed Object

```
// Import and create MyManagedObject
factory := root load: "MyManagedObject".
objDB := factory create.
```

The factory will map create to this constructor

```
package net.ponder2.managedobject;

import java.util.HashMap;
import java.util.Map;

import net.ponder2.apt.Ponder2op;
import net.ponder2.ManagedObject;
import net.ponder2.objects.P2Object;

/**
 * Implements a hash or dictionary.
 * This object holds name/managed object pairs.
 * Objects may be added and removed.
 */
public class MyManagedObject implements ManagedObject {

  private Map<String, P2Object > data;

  /**
   * Creates an empty hash
   */
  @Ponder2op("create")
  MyManagedObject() {
    data = new HashMap<String, P2Object >();
  }

  /**
   * Creates a hash with a particular minimum size
   */
  @Ponder2op("size:")
  MyManagedObject(int size) {
    data = new HashMap<String, P2Object >(size);
  }

  /**
   * adds an object to the hash
   */
  @Ponder2op("at:put:")
  P2Object p2_operation_atPut(String name, P2Object obj) {
    data.put(name, obj);
    return obj;
  }

  /**
   * retrieves an object by name
   */
  @Ponder2op("at:")
  P2Object p2_operation_at(String name) {
    return data.get(name);
  }

  /**
   * removes the named object from the hash
   */
  @Ponder2op("remove:")
  P2Object p2_operation_remove(String name) {
    return data.remove(name);
  }

  /**
   * get the number of objects in the hash
   */
  @Ponder2op("size")
  int p2_operation_size() {
    return data.size();
  }

}
```

Writing a new Managed Object

```
package net.ponder2.managedobject;

import java.util.HashMap;
import java.util.Map;

import net.ponder2.apt.Ponder2op;
import net.ponder2.ManagedObject;
import net.ponder2.objects.P2Object;

/**
 * Implements a hash or dictionary.
 * This object holds name/managed object pairs.
 * Objects may be added and removed.
 */
public class MyManagedObject implements ManagedObject {

  private Map<String, P2Object > data;

  /**
   * Creates an empty hash
   */
  @Ponder2op("create")
  MyManagedObject() {
    data = new HashMap<String, P2Object>();
  }

  /**
   * Creates a hash with a particular minimum size
   */
  @Ponder2op("size:")
  MyManagedObject(int size) {
    data = new HashMap<String, P2Object >(size);
  }

  /**
   * adds an object to the hash
   */
  @Ponder2op("put:")
  P2Object p2_operation_put(String name, P2Object obj) {
    data.put(name, obj);
    return obj;
  }

  /**
   * retrieves an object by name
   */
  @Ponder2op("at:")
  P2Object p2_operation_at(String name) {
    return data.get(name);
  }

  /**
   * removes the named object from the hash
   */
  @Ponder2op("remove:")
  P2Object p2_operation_remove(String name) {
    return data.remove(name);
  }

  /**
   * get the number of objects in the hash
   */
  @Ponder2op("size")
  int p2_operation_size() {
    return data.size();
  }

}
```

```
// Import and create MyManagedObject
factory := root load: "MyManagedObject".
objDB := factory create.
objDB2 := factory size: 25.
```

The factory will map size: to this constructor

Writing a new Managed Object

Keyword message at:put: with two args will invoke this method

```
package net.ponder2.managedobject;

import java.util.HashMap;
import java.util.Map;

import net.ponder2.apt.Ponder2op;
import net.ponder2.ManagedObject;
import net.ponder2.objects.P2Object;

/**
 * Implements a hash or dictionary.
 * This object holds name/managed object pairs.
 * Objects may be added and removed.
 */
public class MyManagedObject implements ManagedObject {

  private Map<String, P2Object > data;

  /**
   * Creates an empty hash
   */
  @Ponder2op("create")
  MyManagedObject() {
    data = new HashMap<String, P2Object >();
  }

  /**
   * Creates a hash with a particular minimum size
   */
  @Ponder2op("size:")
  MyManagedObject(int size) {
    data = new HashMap<String, P2Object >(size);
  }

  /**
   * adds an object to the hash
   */
  @Ponder2op("at:put:")
  P2Object store(String name, P2Object obj) {
    data.put(name, obj);
    return obj;
  }

  /**
   * retrieves an object by name
   */
  @Ponder2op("at:")
  P2Object p2_operation_at(String name) {
    return data.get(name);
  }

  /**
   * removes the named object from the hash
   */
  @Ponder2op("remove:")
  P2Object p2_operation_remove(String name) {
    return data.remove(name);
  }

  /**
   * get the number of objects in the hash
   */
  @Ponder2op("size")
  int p2_operation_size() {
    return data.size();
  }

}
```

Writing a new Managed Object

Keyword message at:put: with two args will invoke this method

```
package net.ponder2.managedobject;

import java.util.HashMap;
import java.util.Map;

import net.ponder2.apt.Ponder2op;
import net.ponder2.ManagedObject;
import net.ponder2.objects.P2Object;

/**
 * Implements a hash or dictionary.
 * This object holds name/ma
 * Objects may be added and
 */
public class MyManagedObject implements ManagedObject {

  private Map<String, P2Object > data;

  /**
   * Creates an empty hash
   */
  @Ponder2op("create")
  MyManagedObject() {
    data = new HashMap<String, P2Object >();
  }

  /**
   * Creates a hash with a particular minimum size
   */
  @Ponder2op("size:")
  MyManagedObject(int size) {
    data = new HashMap<String, P2Object >(size);
  }
```

```
  /**
   * adds an object to the hash
   */
  @Ponder2op("at:put:")
  P2Object store(String name, P2Object obj) {
    data.put(name, obj);
    return obj;
  }

  /**
   * retrieves an object by name
   */
  @Ponder2op("at:")
  P2Object p2_operation_at(String name) {
    return data.get(name);
  }

  /**
   * removes the named object from the hash
   */
  @Ponder2op("remove:")
  P2Object p2_operation_remove(String name) {
    return data.remove(name);
  }

  /**
   * get the number of objects in the hash
   */
  @Ponder2op("size")
  int p2_operation_size() {
    return data.size();
  }

}
```

```
// Add an object to the object database
objDB at: "Fred" put: root/objs/fred.
```

Writing a new Managed Object

```
package net.ponder2.managedobject;

import java.util.HashMap;
import java.util.Map;

import net.ponder2.apt.Ponder2op;
import net.ponder2.ManagedObject;
import net.ponder2.objects.P2Object;

/**
 * Implements a hash or dictionary.
 * This object holds name/managed object pairs.
 * Objects may be added and removed.
 */
public class MyManagedObject implements ManagedObject {

  private Map<String, P2Object > data;

  /**
   * Creates an empty hash
   */
  @Ponder2op("create")
  MyManagedObject() {
    data = new HashMap<String, P2Object >();
  }

  /**
   * Creates a hash with a particular minimum size
   */
  @Ponder2op("size:")
  MyManagedObject(int size) {
    data = new HashMap<String, P2Object >(size);
  }

  /**
   * adds an object to the hash
   */
  @Ponder2op("at:put:")
  P2Object store(String name, P2Object obj) {
    data.put(name, obj);
    return obj;
  }

  /**
   * retrieves an object by name
   */
  @Ponder2op("at:")
  P2Object p2_operation_at(String name) {
    return data.get(name);
  }

  /**
   * removes the named object from the hash
   */
  @Ponder2op("remove:")
  P2Object p2_operation_remove(String name) {
    return data.remove(name);
  }

  /**
   * get the number of objects in the hash
   */
  @Ponder2op("size")
  int p2_operation_size() {
    return data.size();
  }

}
```

Exercise 2 - A new managed Object

- create a new timer Managed Object with commands
  sleepFor: num_of_secs event: root/event/tick
  cancel
- After secs seconds it generates the named event with no arguments. Cancel cancels the timer.
- Write PonderTalk with a new Event and a new Policy to set the Alarm
- Create a class called Timer.

Exercise 2 - A new managed Object

- create a new timer Managed Object with commands sleepFor num_of_secs event: root/event/tick cancel
- After secs seconds it generates the named event with no arguments. Cancel cancels the timer.
- Write PonderTalk with a new Event and a new Policy to set the Alarm
- Create a class called Timer.

Exercise Time

- Use the exercise documentation found in **index.html**
- At the bottom click on **Exercise 2**
- Open **ex2.p2** and fill in the blanks
- Run it with **./ant ex2**

Exercise 2 - Thread Notes

- Use a Thread for timing.

```
@Ponder2op(....)
mymethod(... OID event ...)

new Thread() {
   run() {
          Thread.sleep(secs*1000);
           event.operation(null, “create”);
   }
   catch (InterruptedException e) {
   }
};
```

Exercise 2 - Running notes (Unix)

- You do not need the tutorial BSN policies
- You can use tut1.p2 to set up the alarm
- Create your PonderTalk in ex2.p2
- Compile and run as

```
ant ex2
```

Acknowledgements

Imperial College London

Kevin Twidle

Morris Sloman

Sye-Loong Keoh

Alberto Schaeffer

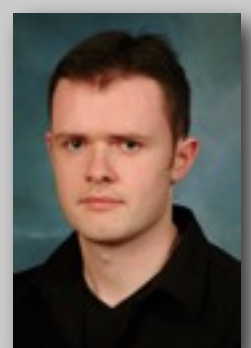
Naranker Dulay

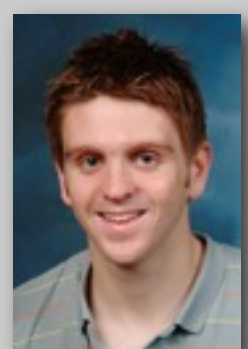
Emil Lupu

UNIVERSITY of GLASGOW

Joe Sventek

Stephen Strowes





Steven Heeps

Generated Adaptor Class

```
import java.util.HashMap;
import java.util.Map;

import net.ponder2.objects.P2Object;
import net.ponder2.exception.Ponder2OperationException;
import net.ponder2.exception.Ponder2Exception;
import net.ponder2.ManagedObject;

/**
 * Adaptor object for managed object
 *
 * @author Auto generated by annotation processor tool
 */
public class SampleObjectP2Adaptor extends
net.ponder2.P2ObjectAdaptor {

  /**
   * The map of create operations to constructors
   */
  private final static Map<String, CreateOperation> create;
  /**
   * The map of instance operations to methods
   */
  private final static Map<String, InstanceOperation> operation;

  // Create the call tables when the class is loaded
  static {
    create = new HashMap<String, CreateOperation>();
    operation = new HashMap<String, InstanceOperation>();

    // Create operation 'create' calls constructor for SampleObject
    create.put("create", new CreateOperation() {
      @Override
      public net.ponder2.ManagedObject call(P2Object obj, P2Object
source, String operation, P2Object... args)
          throws net.ponder2.exception.Ponder2Exception {
        return new SampleObject();
      }
    });

    // Create operation 'size:' calls constructor for SampleObject
    create.put("size:", new CreateOperation() {
      @Override
      public net.ponder2.ManagedObject call(P2Object obj, P2Object
source, String operation, P2Object... args)
          throws net.ponder2.exception.Ponder2Exception {
        return new SampleObject(args[0].asNumber());
      }
    });

    // Operation 'at:put:' calls p2_operation_at_put
    operation.put("at:put:", new InstanceOperation() {
      @Override
      public P2Object call(P2Object thisObj,
net.ponder2.ManagedObject obj, P2Object source, String operation,
P2Object... args)
          throws net.ponder2.exception.Ponder2Exception {
        net.ponder2.objects.P2Object value =
            ((SampleObject)obj).p2_operation_at_put(args[0].asString(),
args[1]);
        return value;
      }
    });

    // Operation 'at:' calls p2_operation_at
    operation.put("at:", new InstanceOperation() {
      @Override
      public P2Object call(P2Object thisObj,
net.ponder2.ManagedObject obj, P2Object source, String operation,
P2Object... args)
          throws net.ponder2.exception.Ponder2Exception {
        net.ponder2.objects.P2Object value =
            ((SampleObject)obj).p2ß_operation_at(args[0].asString());
        return value;
```

Cell Policy Service