HEADS
Heterogenous and Distributed
Services for the Future Computing Continuum

The Room X1

CPS-L1

CPS-L1: The Room X1 – simple home automation

- Separation of concerns
  - PIM and PSM
  - Kick-down
- Simulation
  - Why?
  - How?
  - How to map to real CPS?

The Room X1: Smart Home Basics

- Our CPS will be a very basic Smart Home
  - The brain is a processor that runs java
    - This can be a PC or even a Raspberry Pi
  - A **Telldus Tellstick Duo** is connected via USB to the processor
  - The Tellstick controls a few **gadgets**
    - One (or more) on-off **switch(es)** wrapped up in an electric plug
    - One (or more) **thermometer(s)**

Specification of The Room X1 functionality

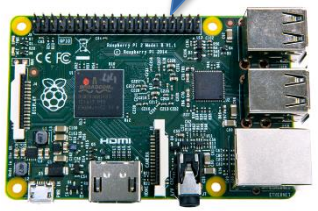
- Observe individually the temperature sensors
- Turn the on-off switches ON or OFF
- That’s it!

The gadgets

Sensor(s)

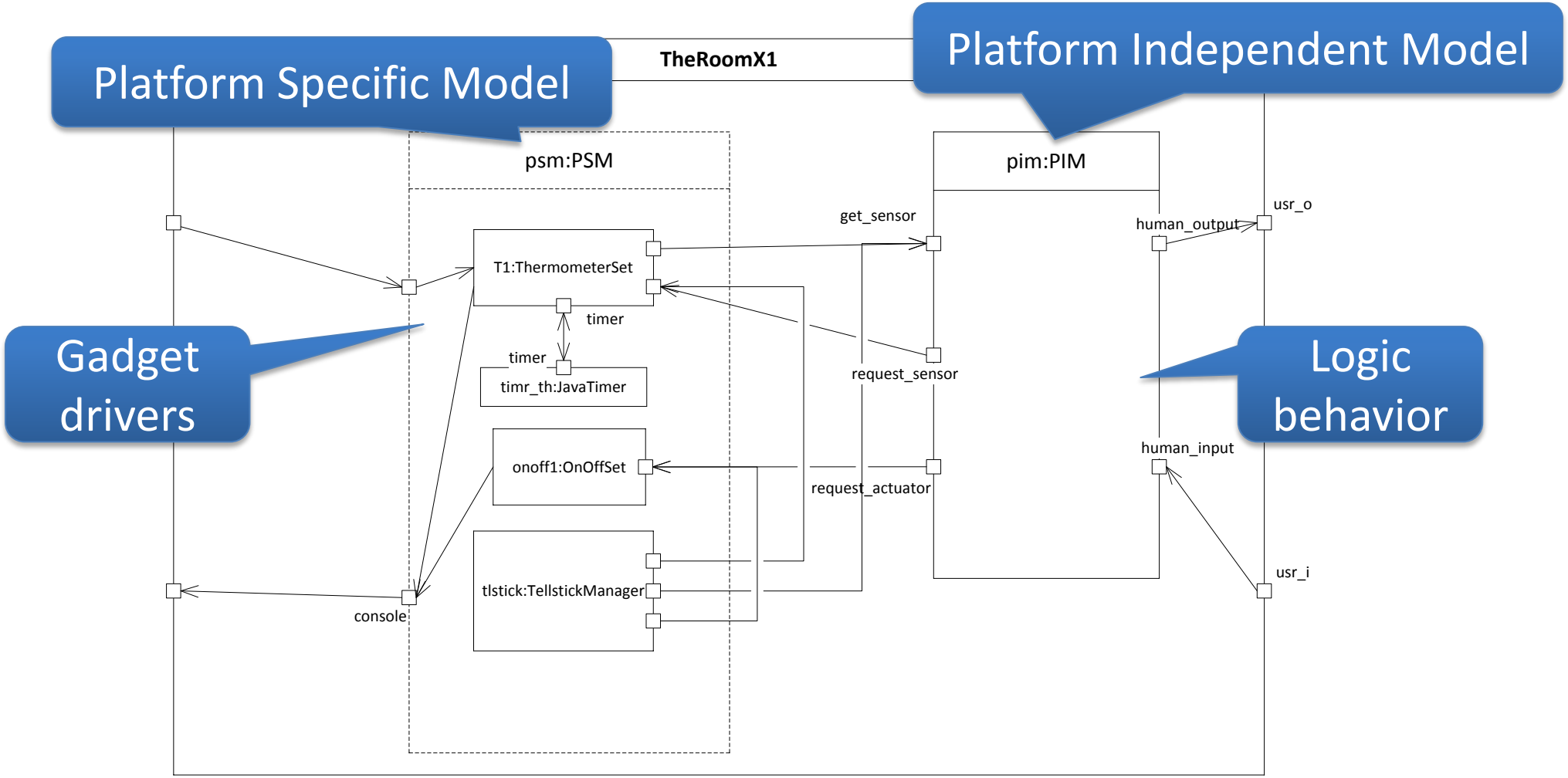
Actuator

Processor

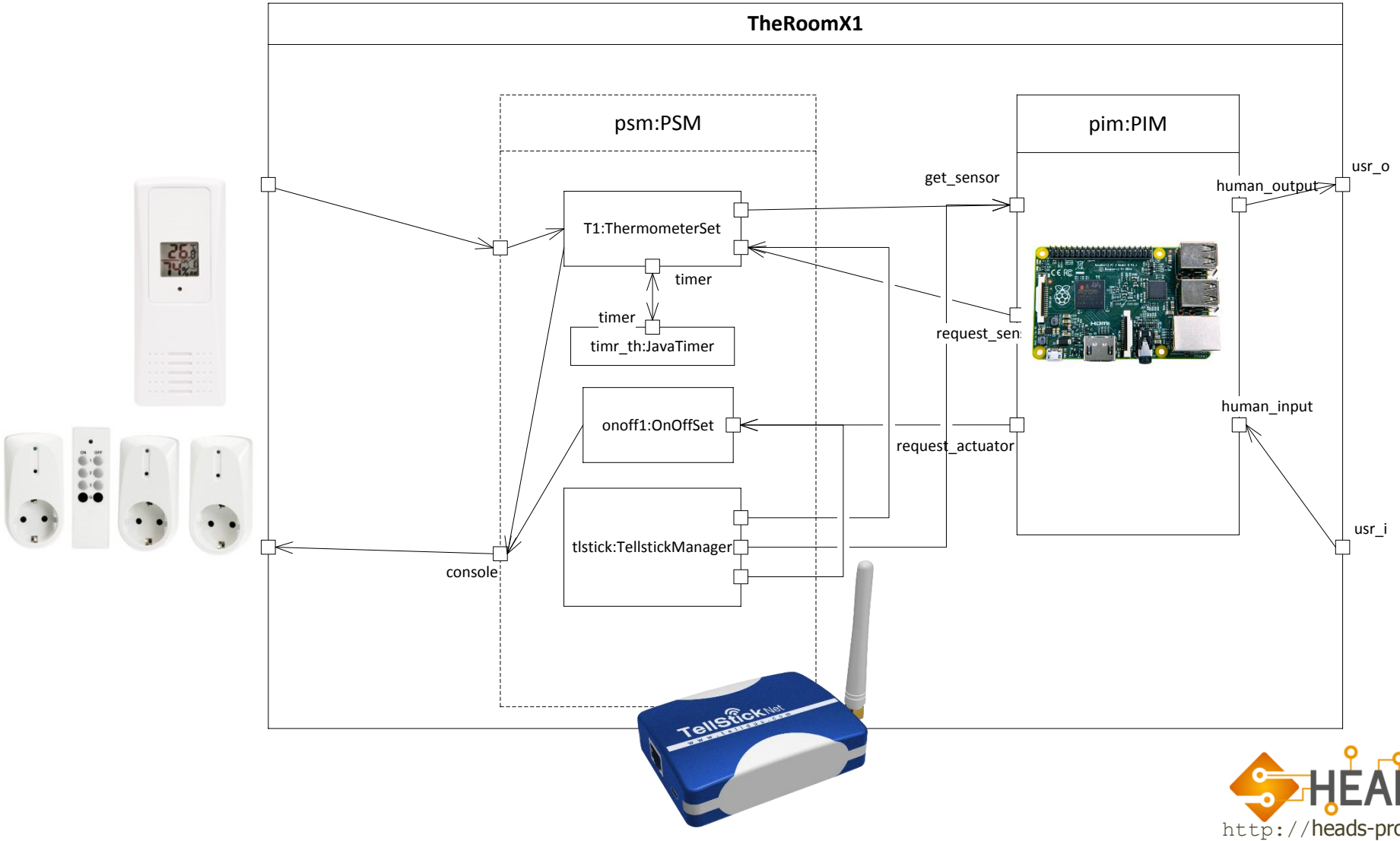


Tellstick

The software elements

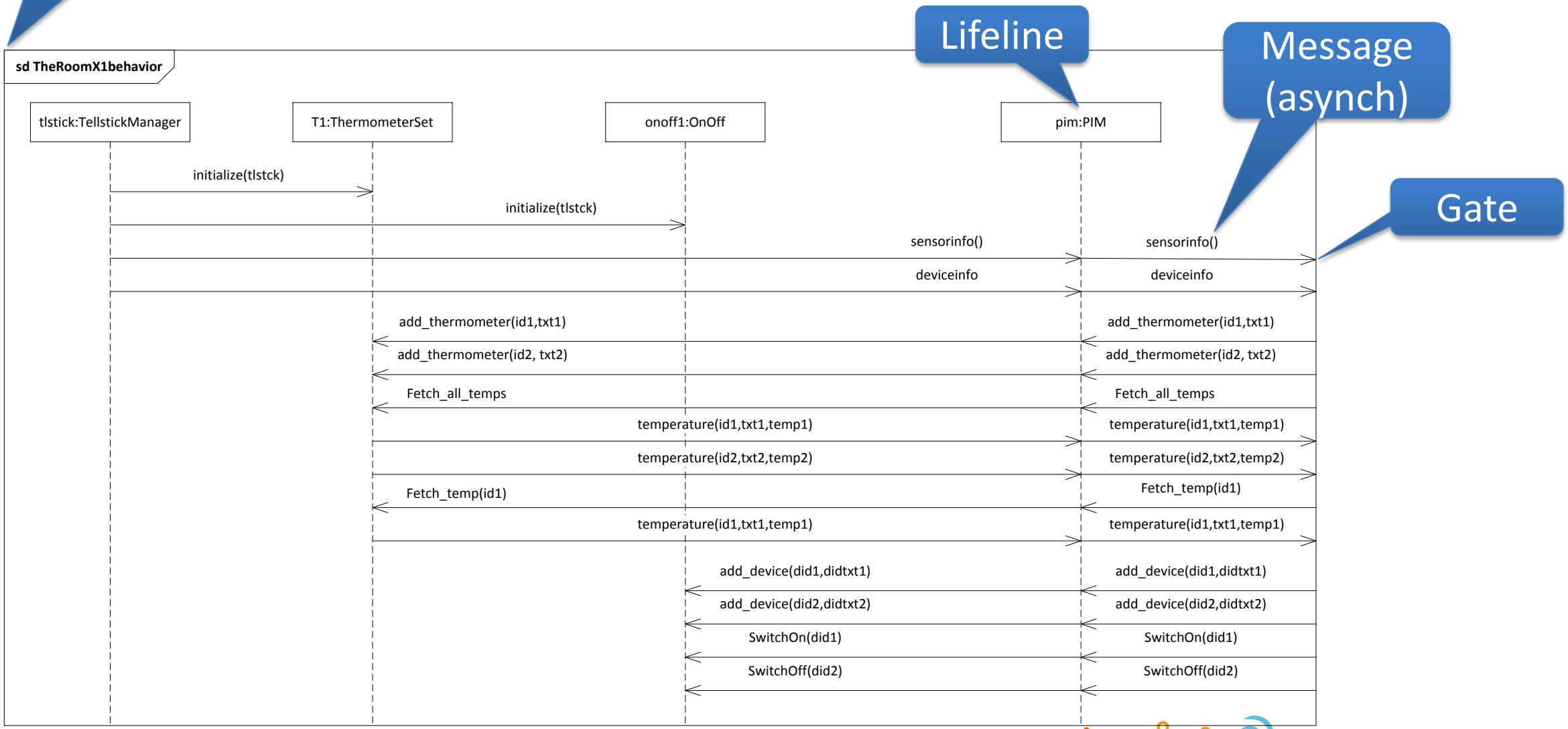


In one picture



Sequence Diagram

he Room X1 Behavior



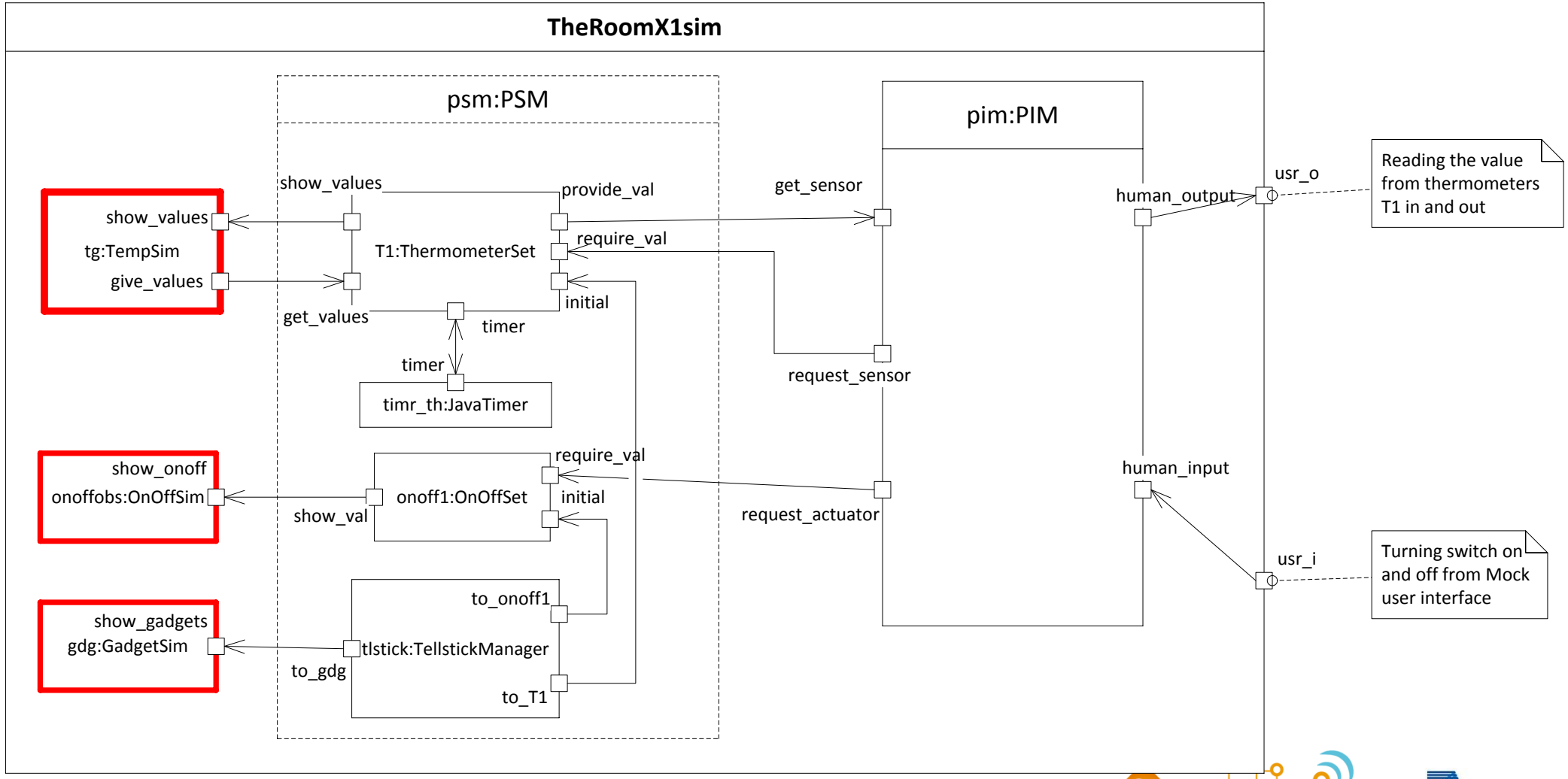
Simulation

- to execute a system in a fictitious setting
- Why?
  - You do not have the proper hardware available
    - because it does not exist, yet
    - because you have not bought it
  - During development you need more resources / power
    - to secure functionality before optimization
    - to provide better testing facilities
    - to provide better measurement opportunities

The Room X1 Simulation

- We use simulation for The Room X1
  - such that everybody can run it without buying or getting the real gadgets
  - to use the ThingML Eclipse on PCs for quicker turnaround and better debugging facilities than the Raspberry Pi or other low performance (but much cheaper) microcontrollers
  - And we can manipulate the temperature much faster
- Simulation environment using mock dialogues
  - Gadget startup; Thermometer set; Switch set;
  - Human interface;

The Room X1 – Simulation architecture

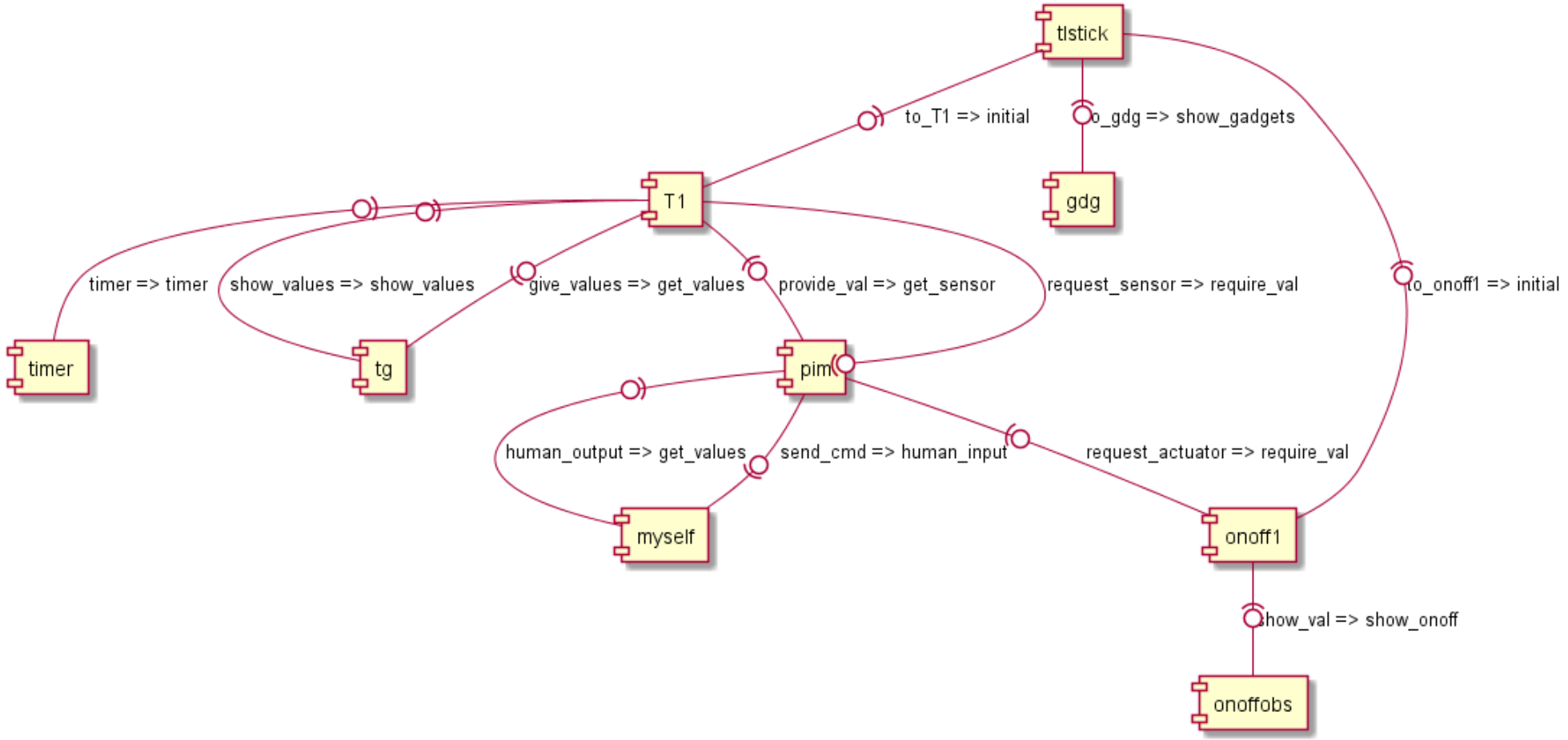


The Room X1 in ThingML – the configuration

```
import "psm_sim.thingml"
import "pim.thingml"
import "io.thingml"
import "javatimer.thingml"

configuration CPS {
instance tlstick:TellstickManager
instance T1:ThermometerSet
instance onoff1:OnOffSet
instance pim:PIM
instance myself:Human
instance timer : TimerJava

// SIMULATION
instance tg:TempSim
instance onoffobs:OnOffSim
instance gdg:GadgetSim
```

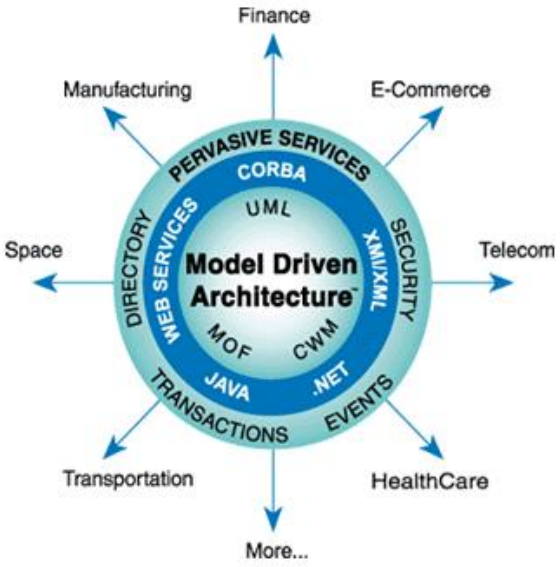
```
// PSM
connector tlstick.to_T1 => T1.initial
connector tlstick.to_gdg => gdg.show_gadgets
connector tlstick.to_onoff1 => onoff1.initial

connector T1.provide_val => pim.get_sensor
connector T1.timer => timer.timer
connector T1.show_values => tg.show_values

connector onoff1.show_val => onoffobs.show_onoff

// HMI
connector myself.send_cmd => pim.human_input

// PIM outwards
connector pim.request_sensor => T1.require_val
connector pim.request_actuator => onoff1.require_val
connector pim.human_output => myself.get_values

// SIMULATION
connector tg.give_values => T1.get_values
}
```

The Room X1 – ThingML config visualized via PlantUML



PSM and PIM (terms taken from OMG's MDA)

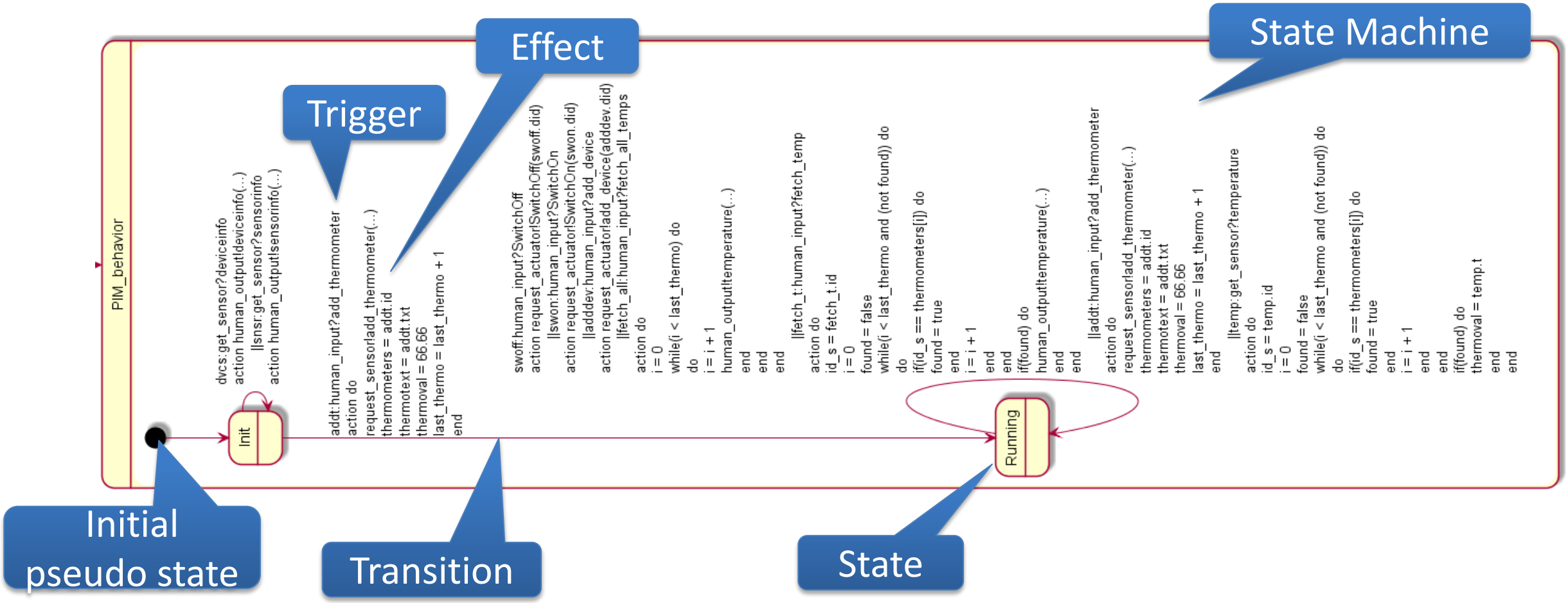
- PSM = Platform Specific Model
  - what will be replaced when the platform (low level) is changed
  - in our case it also means changing from simulation platform to real platform
  - The drivers
- PIM = Platform Independent Model
  - what will be stable regardless of platform changes
  - The application logic



The Room X1 – Application Logic

- The Room X1 PIM is only one state machine
- It is in X1 not much in addition to the PSM
- But it is the PIM that will grow and become more complex and more robust in versions to come
  - while the PSM will stay mostly unchanged

The Room X1 – PIM state machine visualized



The Room 2 – PIM in text (1)

Initial pseudo state

Transition

Trigger

State Machine

Effect

State

Message

Port

```
statechart PIM_behavior init Init {
    state Init {
        transition -> Init
        event snsr:get_sensor?sensorinfo
        action do
                human_output!sensorinfo(snsr.model,snsr.proto,snsr.sid,snsr.dataTypes,snsr.temperature,snsr.humidity,snsr.timeStamp
        )
        end
        transition -> Init
        event dvcs:get_sensor?deviceinfo
        action do
                human_output!deviceinfo(dvcs.did,dvcs.name,dvcs.model,dvcs.proto, dvcs.ttype,dvcs.meth,dvcs.lastCmd,dvcs.lastValue)
        end
        transition -> Running // adding the first thermometer will start the normal operation
        event addt:human_input?add_thermometer
        action do
            request_sensor!add_thermometer(addt.id,addt.txt)
            // we do some bookkeeping on thermometers both at the PSM and at the PIM
            thermometers[last_thermo]=addt.id
            thermotext[last_thermo]=addt.txt
            thermoval[last_thermo]=66.66 //to indicate no temperature has been received
            last_thermo=last_thermo+1 //increasing the number of thermometers in our set
        end
    }
```

The Room X1 – PIM in text (2)

```
state Running {

      transition -> Running
      event temp:get_sensor?temperature
      action do
            id_s=temp.id

            i=0
            found = false
            while (i<last_thermo and (not found)) do
                  if (id_s==thermometers[i]) do
                        found=true // trick to terminate while loop
                  end
                  i=i+1
            end
            if (found) do
                        thermoval[i-1]=temp.t
            end
      end
      transition -> Running
      event addt:human_input?add_thermometer
      action .......

      transition -> Running
      event fetch_t:human_input?fetch_temp
      action do

.......... // many transitions that go from Running to Running
}
```

The Room X1 – PIM in text (3) imports

```
// Base datatypes
import "datatypes.thingml"

/* PSM must be included */
import "psm_sim.thingml"
import "psm_datatypes_sim.thingml"
import "pim_messages.thingml"
```

The Room X1 – PIM in text (4) The thing port interface

```
thing PIM includes GeneralMsg, TemperatureMsg, OnOffMsg {
    provided port get_sensor {
        receives temperature, sensorinfo, deviceinfo
    }
    required port request_sensor {
        sends add_thermometer
    }
    required port request_actuator{
        sends add_device, SwitchOn, SwitchOff
    }
    provided port human_input {
        receives add_thermometer, add_device, fetch_temp, fetch_all_temps, SwitchOn, SwitchOff
    }
    required port human_output {
        sends temperature,  sensorinfo, deviceinfo
    }
```

The Room X1 – PIM in text (5) the thing properties

```
property thermometers:Integer[25] // Identifiers of the thermometers in the set
property thermotext:String[25] // corresponding explanatory text
property thermoval:Double[25] // storing the values received from the thermometers (through the PSM)
property last_thermo:Integer = 0 // number of thermometers in the set

// temporary variables
property id_s:Long // temporary id value (to be used with kick-down)
property temp_s:Double // temporary temperature value
property found:Boolean // temporary - true when item found in loop
property i:Integer // runner index in list
```

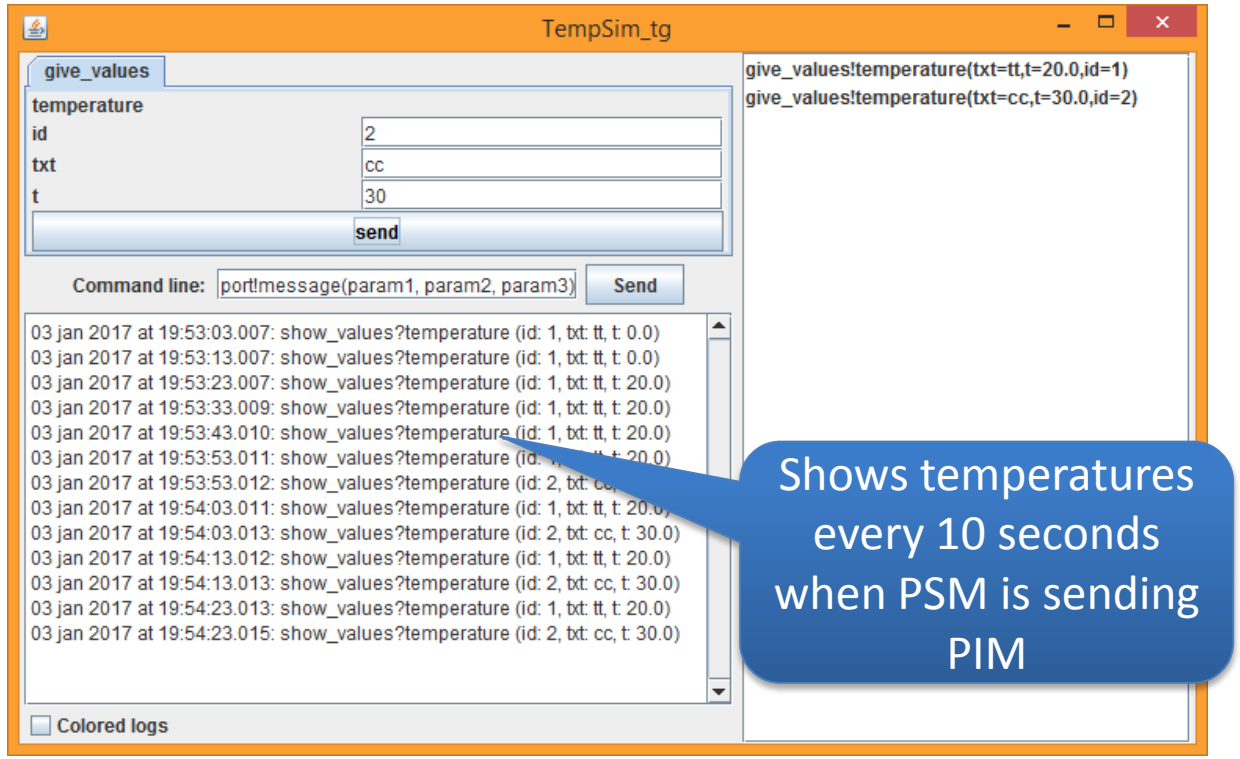
ThingML simple user interface

- By using Java and the compiler directive
- @mock “true”
- and compile with Swing, the compiler will provide a simple Swing window user interface

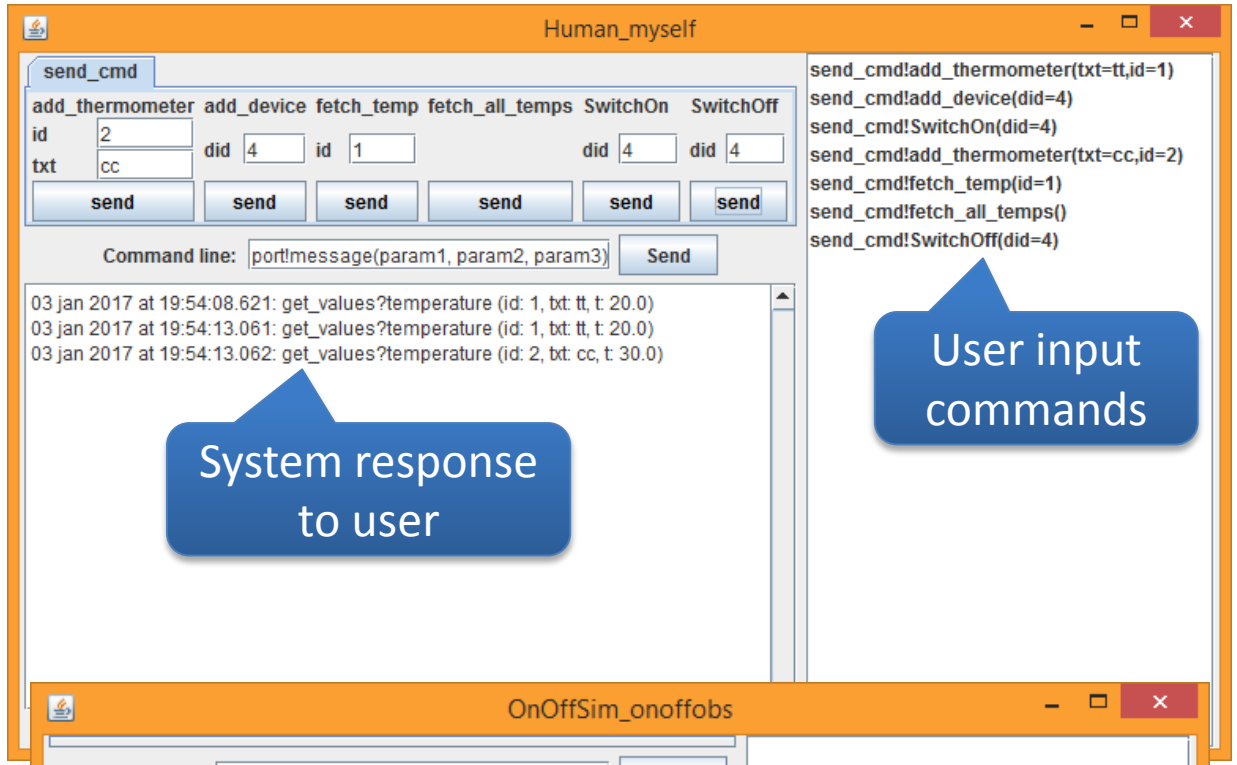
The @mock interface

```
//SIMULATION
thing TempSim includes TemperatureMsg
@mock "true"
{ required port give_values {
sends temperature
}
provided port show_values {
receives temperature
}
}

thing GadgetSim includes GeneralMsg
@mock "true"
{provided port show_gadgets {
receives sensorinfo, deviceinfo
}
}

thing OnOffSim includes OnOffMsg
@mock "true"
{provided port show_onoff {
receives SwitchOn, SwitchOff
}
}
```

The Room X1 – A simulated execution

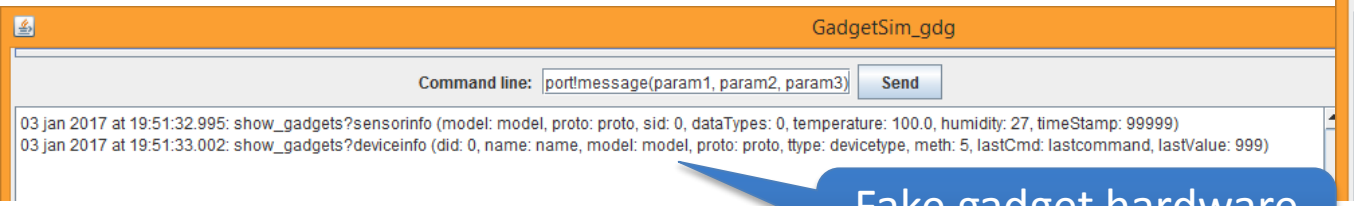


Shows temperatures every 10 seconds when PSM is sending PIM

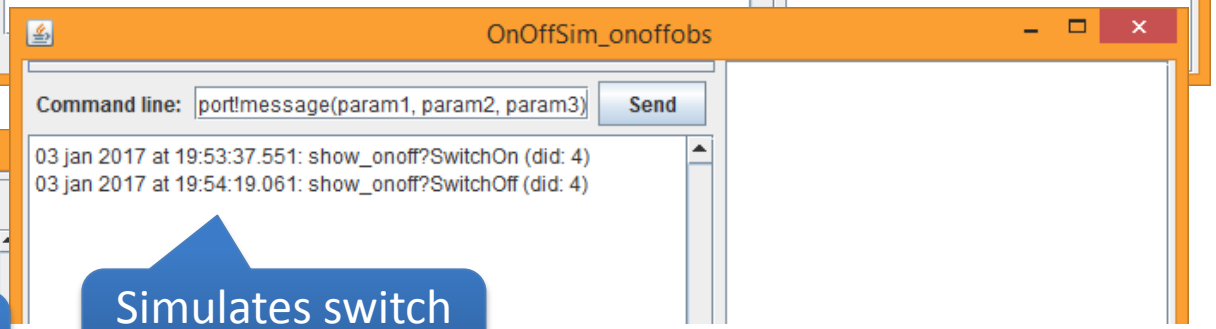


System response to user

User input commands



Fake gadget hardware observation



Simulates switch – on or off

The Room X1 – PIM – a summary

- X1'PIM is a thing where the state machine is rather simple
- X1'PIM functions according to the behavior specified
- X1'PIM does more than that – it functions for more traces than those specified by the sequence diagram
  - Many people would after trying it think it worked well
- X1'PIM is not perfect, it is not particularly robust
  - Try adding the same thermometer several times

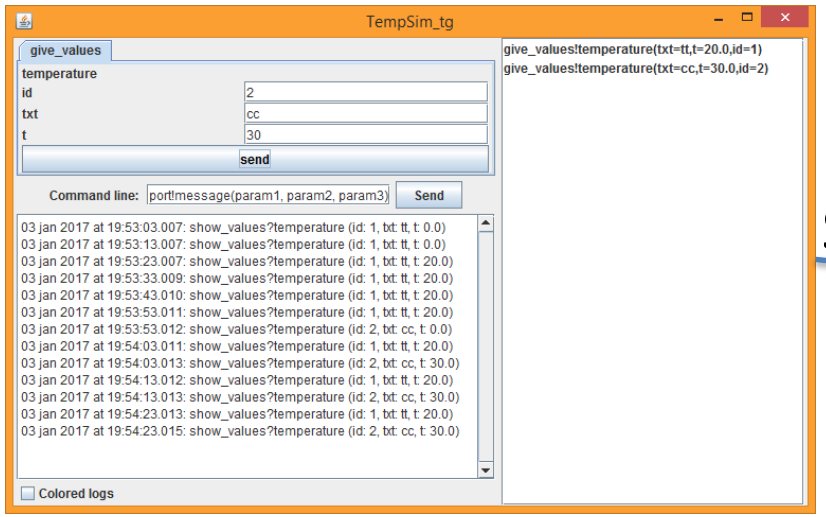
HEADS
Heterogenous and Distributed
Services for the Future Computing Continuum

From Simulation to the Real System

From Simulation to the Real System

- In Simulation
  - we have had an artificial environment
  - we may have applied abundant resources
- In the Real System
  - we need to hook up to the underlying physical devices
    - this requires driver software that may apply low level constructs in other languages than ThingML
  - we may have a scarcity of resources and may consider what functionality or operations to omit

The Room X1

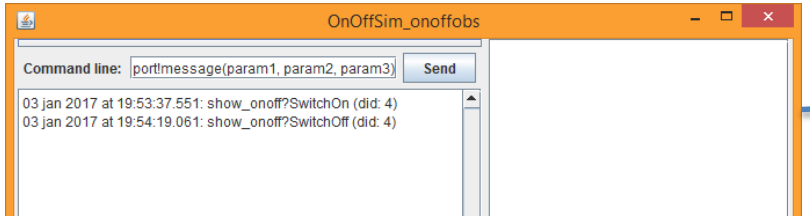


Simulated

Real

Thermometer(s)

Tellstick Duo

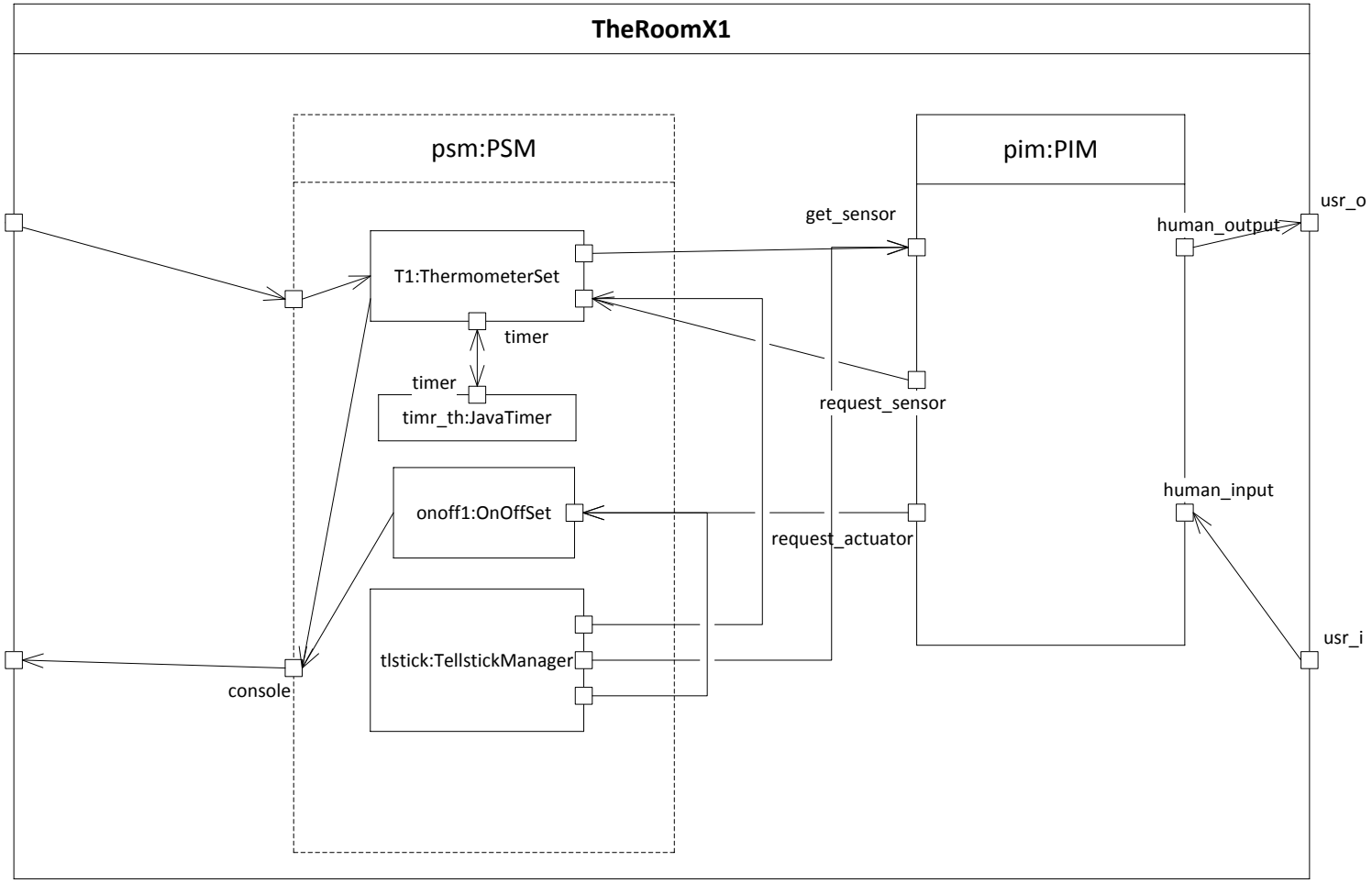


Simulated

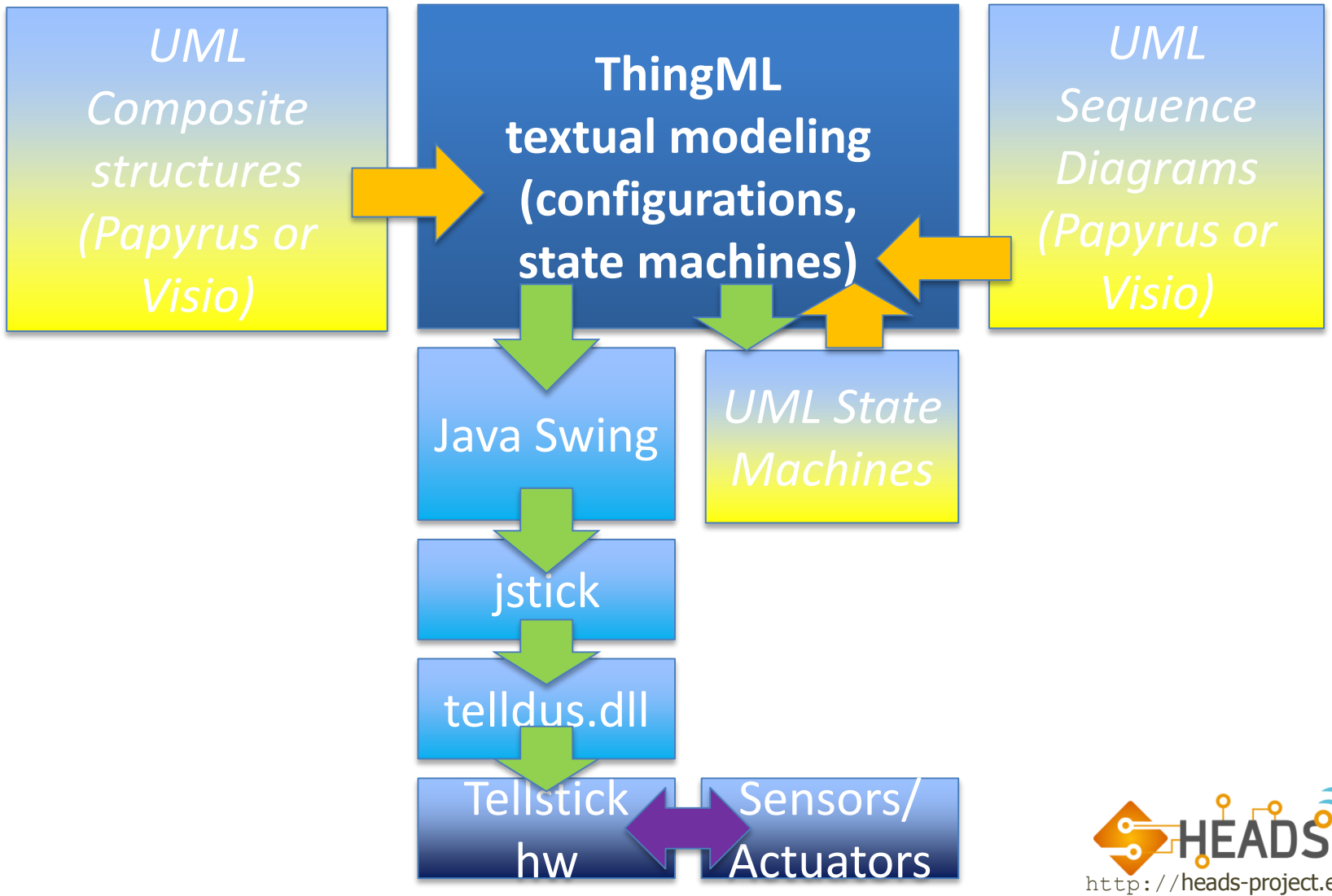
Real

Switch(es)

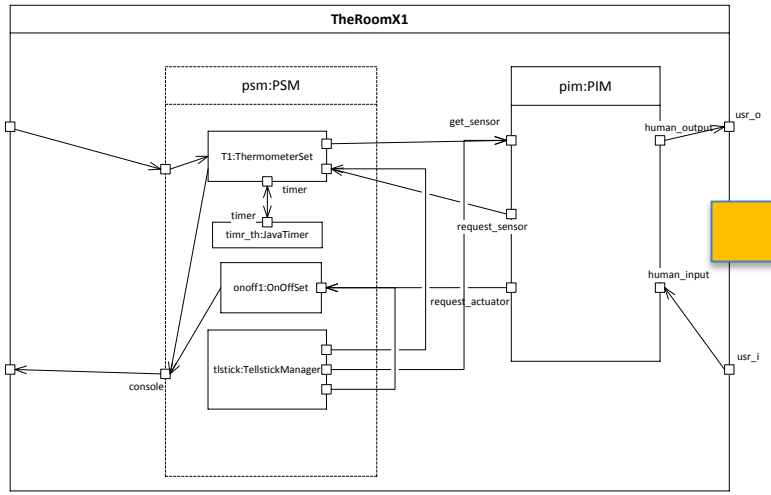
The Room X1: Architecture of the real system



The Elements of CPS Modeling in our course

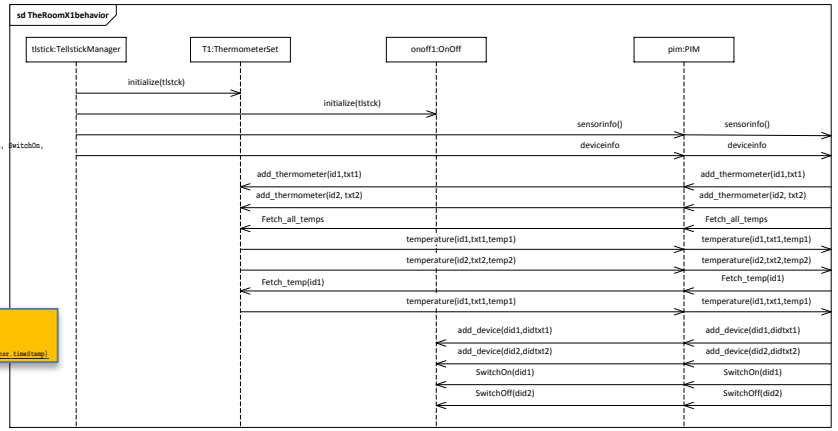


Same picture with our first CPS model




```
import "psm.thingml"
import "pim.thingml"
import "io.thingml"

configuration CPS {
instance tlstick:TellstickManager
instance T1:ThermometerSet
instance H1:HygrometerSet
instance onoff1:OnOffSet
instance pim:PIM
instance myself:Human

connector tlstick.to_T1 => T1.require_val
connector tlstick.to_H1 => H1.require_val
connector tlstick.to_pim => pim.get_sensor
connector tlstick.to_onoff1 => onoff1.require_val

connector pim.request_sensor => H1.require_val
connector pim.request_sensor => T1.require_val
connector H1.provide_val => pim.get_sensor
connector T1.provide_val => pim.get_sensor

connector pim.request_actuator => onoff1.require_val

connector myself => pim.human_input
connector pim.hu t => myself.get_values
}
```

```
thing PIM includes GeneralMsg, TemperatureMsg, HumidityMsg, OnOffMsg {
    provided port get_sensor {
        receives running,temperature, humidity,sensorinfo, deviceinfo
    }
    required port request_sensor {
        sends fetch_all_temps, fetch_temp, add_thermometer, fetch_all_hum, fetch_humidity, add_hygrometer
    }
    required port request_actuator{
        sends add_device, SwitchOn, SwitchOff
    }
    provided port human_input {
        receives fetch_temp, fetch_all_temps, add_thermometer, fetch_humidity, fetch_all_hum, add_hygrometer, add_device, SwitchOn,
    SwitchOff
    }
    required port human_output {
        sends temperature, humidity, sensorinfo, deviceinfo
    }

    statechart PIM_behavior init Init {
        state Init {
            transition -> Running
            event run: get_sensor?running
            action do
                // nothing
            end
        }
        state Running {
            transition -> Running
            event snsr:get_sensor?sensorinfo
            action do
                human_output!sensorinfo(snsr.model,snsr.protocol,snsr.sid,snsr.dataTypes,snsr.temperature,snsr.humidity,snsr.timeStamp)
            end
```

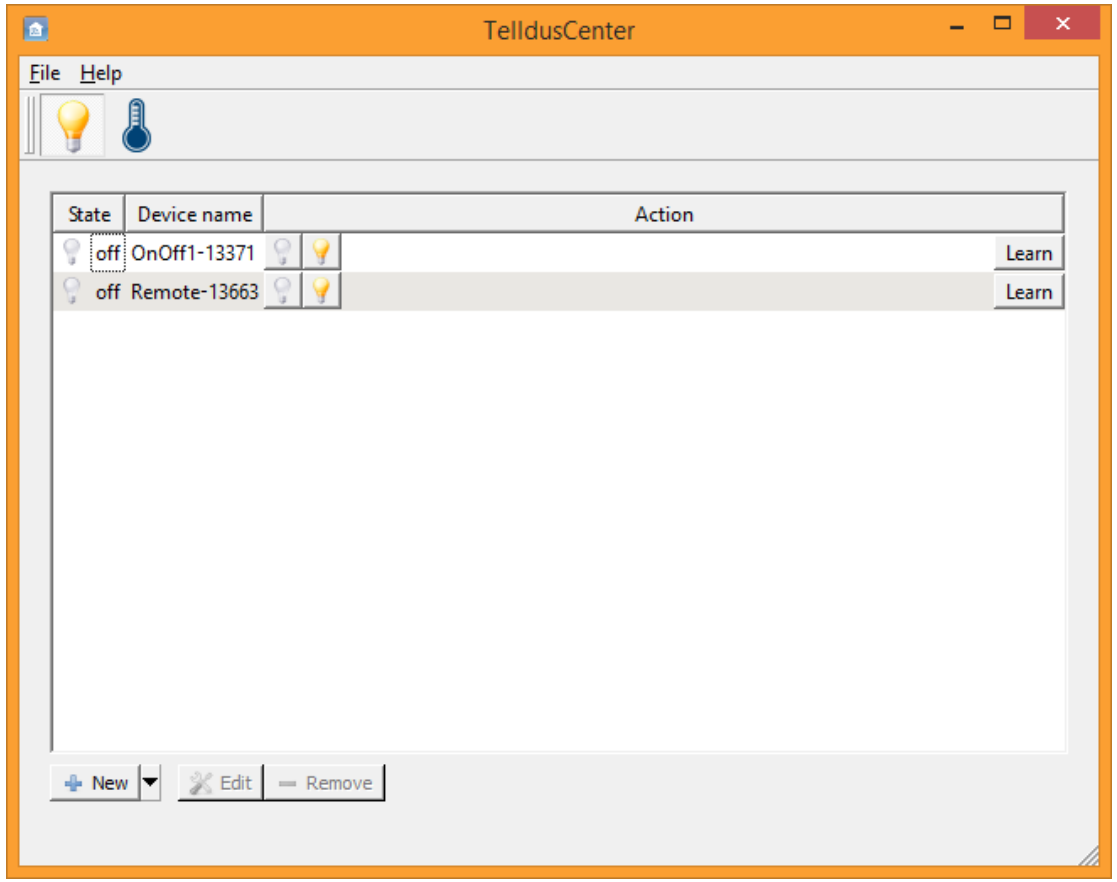


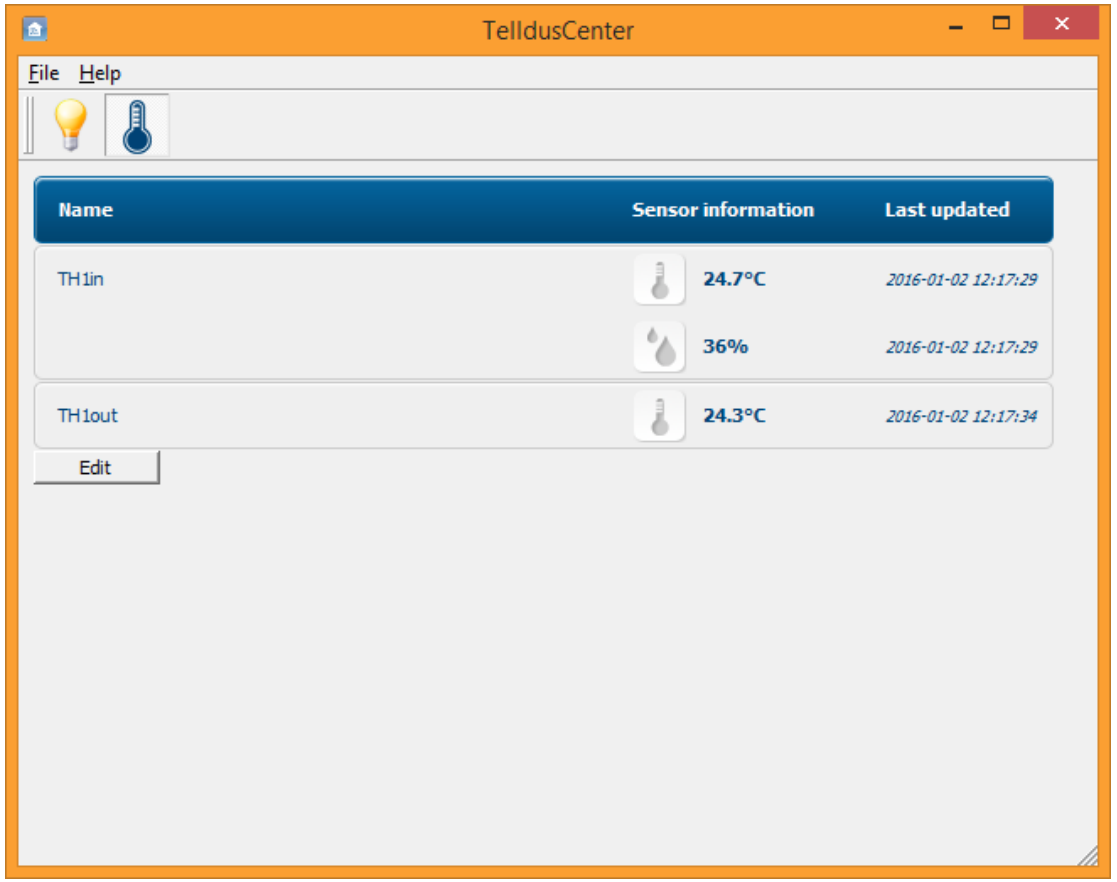
http://heads-project.eu

Tellstick from Telldus

- Tellstick Duo will be our thing controller
  - it is using RF on 433 MHz
- http://www.telldus.se/
- Go to the TellStick Duo page
  - http://telldus.se/produkt/tellstick-duo/
  - Install the software and try it
  - Get hold of:
    - One on-off switch
    - One thermometer (possibly two in one and a hygrometer)

The TellStick Duo software





Check that Telldus Service is running in Task Manager (on Windows)

Another piece of reality ...

- In our field development happens continuously and rapidly
- Last year Tellstick Duo from Telldus seemed a very reasonable choice
- This year it is obvious that Tellstick Duo is phased out
- Next year or even this year, we should find a replacement

Jstick – one Java driver for TellStick Duo

- http://jstick.net/
- And Github: https://github.com/juppinet/jstick
  - I needed to correct Tellstick.java by fixing bugs related to value of humidity sensors (in version 1.6)
  - The github gives a Maven project which you should install
- This may or may not be the best library for this, but it will probably do for now

PSM code: Relating to Maven

```
thing TellstickManager includes PSM_Msg, GeneralMsg
@maven_dep "<dependency>
<groupId>net.juppi</groupId>
<artifactId>jstick-api</artifactId>
<version>1.6</version>
</dependency>"
{ /* Ports may be defined here */
        required port to_T1 {
                sends initialize
        }
.....
```

@maven_dep defines a dependency in the maven file which is generated by ThingML compiler

PSM code: specific properties

```
/* properties defined here */
property ts : Tellstick // this is set in initialize() function
property sensor_list:Sensor[25] // removed at SIMULATION
property device_list:Device[25] // removed at SIMULATION
property i:Integer // runner index in list of sensors or devices
property s:Sensor // temporary Sensor removed at SIMULATION
property d:Device // temporary Device removed at SIMULATION
property model:String
property proto:String
```

Some properties may be specific to the real PSM, and some to the simulated PSM

PSM code: ThingML kick-down (to Java)

```
function observe_sensors() do
    // Now we send to PIM all the Sensor gadgets which are managed by that Tellstick
    ''&sensor_list& '=' ''&ts&'.getSensors().toArray('''&sensor_list& ');' // kick-down to tellstick
    i=0
    while (i<25)do
        s=sensor_list[i]
        if (not (s=='null')) // TODO find a way in ThingML to check existence?
        do
            model=''&s&'.getModel()'
            proto=''&s&'.getProtocol()'
            sid=''&s&'.getId()'
            dataTypes=''&s&'.getDataTypes()'
            temperature=''&s&'.getTemperature()'
            humidity=''&s&'.getHumidity()'
            timeStamp=''&s&'.getTimeStamp()'
            to_gdg!sensorinfo(model,proto,sid,dataTypes,temperature,humidity,timeStamp)
        end
        i=i+1
    end
end
```

&sensor_list&

‘.getModel()’

PSM code: The two principles of ThingML kick-down

- ‘.getModel()’
  - The single quote bracket indicates that the bracketed construct should be compiled directly as written in the target language
- &s&
  - The ampersand bracket asks the ThingML compiler to use the target language correspondent to the bracketed ThingML property
- Kick-down in ThingML are either statements or expressions

Consortium


SINTEF


informatics mathematics
Inria


M2M2on


tell.u


software AG


ATC
ATHENS TECHNOLOGY CENTER


HEADS
http://heads-project.eu


SEVENTH FRAMEWORK PROGRAMME