

INF5120
"Modellbasert Systemutvikling"
"Modelbased System development"

Lecture 9: 13.03.2017

Arne-Jørgen Berre

arneb@ifi.uio.no or Arne.J.Berre@sintef.no

Course parts (16 lectures) - 2017

- January (1-3) (Introduction to Modeling, Business Architecture and the Smart Building project):
- 1-16/1: Introduction to INF5120
- 2-23/1: Modeling structure and behaviour (UML and UML 2.0 and metamodeling) - (establish Oblig groups)
- 3-30/1: WebRatio for Web Apps/Portals and Mobile Apps – and Entity/Class modeling – (Getting started with WebRatio)

- February (4-7) (Modeling of User Interfaces, Flows and Data model diagrams, Apps/Web Portals - IFML/Client-Side):
- 4-6/2: Business Model Canvas, Value Proposition, Lean Canvas and Essence
- 5-13/2: IFML – Interaction Flow Modeling Language, WebRatio advanced – for Web and Apps
- 6-20/2: BPMN process, UML Activ.Diagrams, Workflow and Orchestration modelling value networks
- 7-27/2: Modeling principles – Quality in Models
- 27/2: Oblig 1: Smart Building – Business Architecture and App/Portal with IFML WebRatio UI for Smart Building

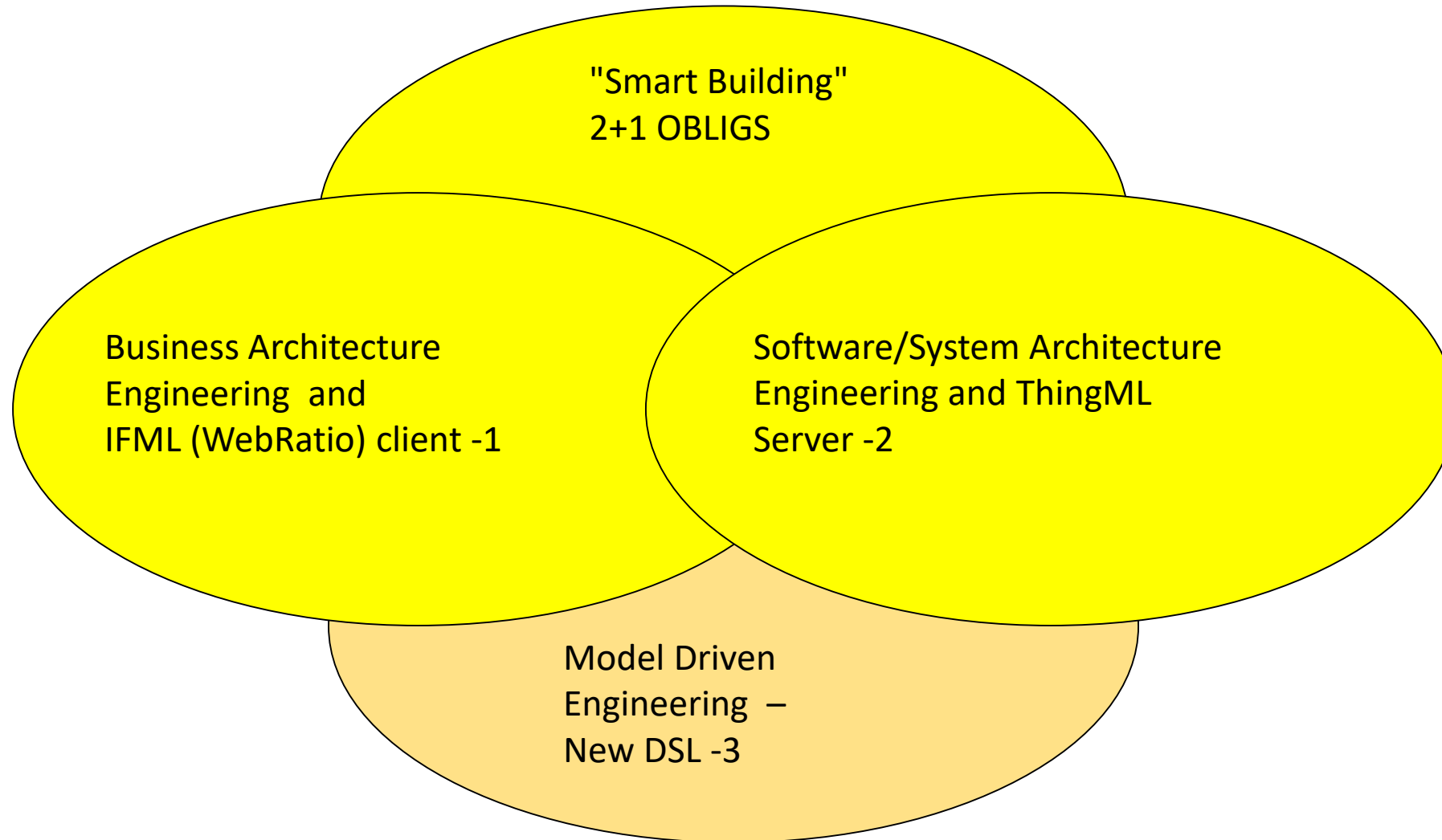
- March (8-11) (Modeling of IoT/CPS/Cloud, Services and Big Data – UML SM/SD/Collab, ThingML Server-Side):
- 8-6/3: Basis for DSL and ThingML -> UML State Machines and Sequence Diagrams
- 9-13/3: ThingML DSL - UML Composite structures, State Machines and Sequence Diagrams II
- 10-20/3: Guest lecture, "Experience with Modelling", Anton Landmark, SINTEF
- 11-27/3: ThingML part 2 and UML Service Modeling, Architectural models, SoaML. Role modeling and UML Collaboration diagrams

- April/May (12-14) (MDE – Creating Your own Domain Specific Language):
- 12-3/4: Model driven engineering – Metamodels, DSL, UML Profiles, EMF, Sirius Editors – intro to Oblig 3
- 3/4: Oblig 2: Smart Building – Internet of Things control with ThingML – Raspberry Pi, Wireless sensors (temperature, humidity), actuators (power control) – individual part

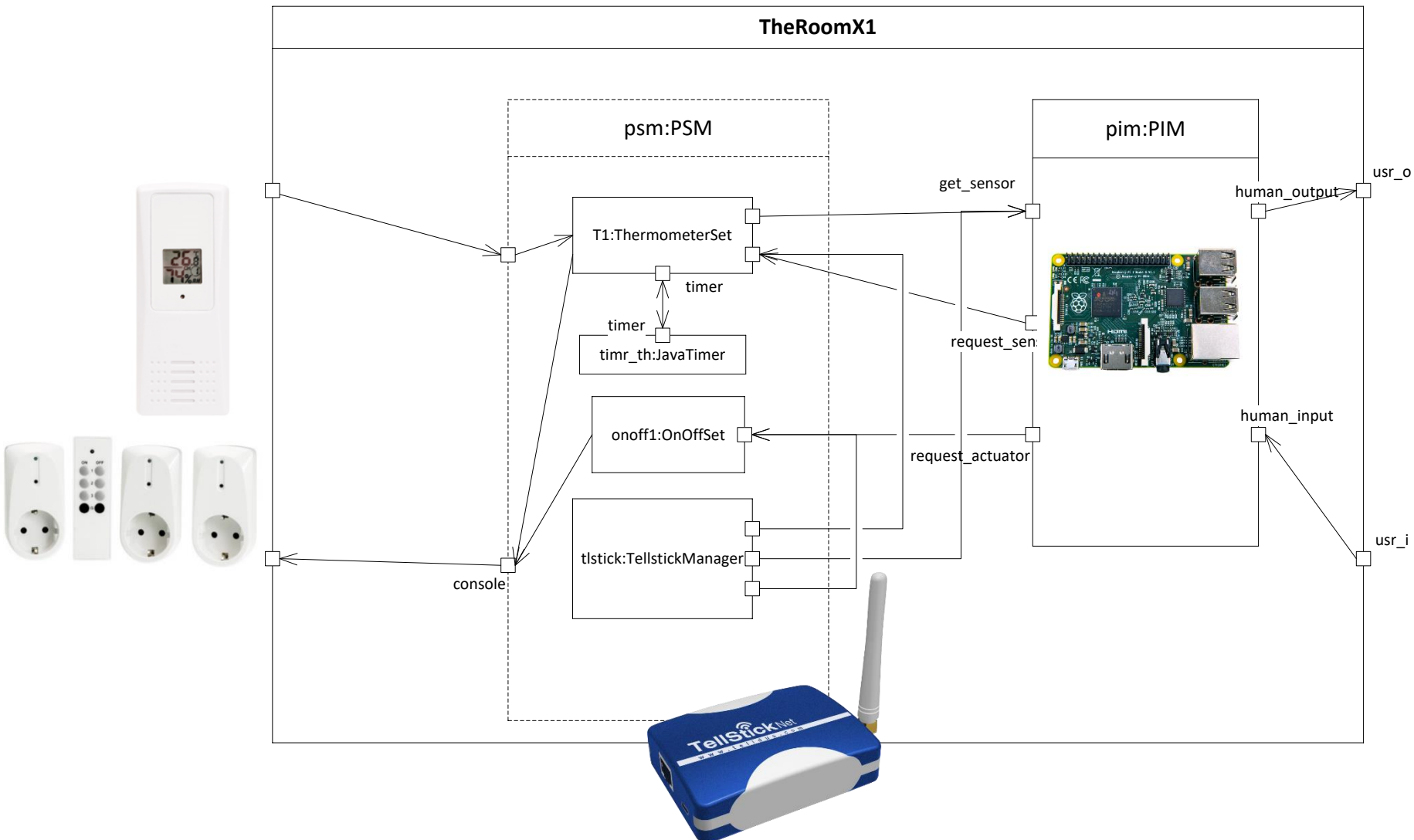
- EASTER – 10/4 og 17/4
- 3/4: Oblig 2: Smart Building – Group par delivery
- 13-24/4: MDE transformations, Non Functional requirements
- 1. Mai – Official holiday
- 14-8/5: SmartBuilding – Integrating App with Server side
- 8/5: Oblig 3 - Your own Domain Specific Language

- May (15-17): (Bringing it together)
- 15-15/5: Summary of the course – Final demonstrations
- 16-22/5: Previous exams – group collaborations (No lecture)
- 17-29/5: Conclusions, Preparations for the Exam by old exams
- June (Exam)
- 13/6: Exam (4 hours), June 13th, 0900-1300

Course components



Smart Building – server side



Using
ThingML Domain
Specific Modeling
Language

- Related to UML
Sequence Diagrams
and State Machines

Why ThingML? Installation/Execution

CPS-LO

CPS-L0: Why ThingML? Installation and Execution

- Why ThingML?
 - CPS with constrained resources
 - Abstraction
 - Separation of Concerns
- What is ThingML?
 - a domain-specific language
 - textual with visualization to UML
 - focused on asynchronous messaging between state machines

Why ThingML?

- The idea of ThingML is to develop
 - a practical model-driven software engineering tool-chain
 - which targets resource constrained embedded systems
 - such as low-power sensor and microcontroller based devices
- ThingML – thing modeling language
 - refers to Internet of Things – another way to say cyber-physical
- ThingML provides several compilers
 - for C, C++, Java, UML, etc
 - which makes its products usable and portable

ThingML and abstraction

- We want functionality, but we also want to get our (mental) arms around the problem
 - because we are going to improve our solution
 - because we are going to correct our solution
 - because we are going to evolve our solution
 - because we are going to enhance our solution
- We want constructs that help comprehend our system
- We want constructs that help distribute the development of the system

ThingML main constructs

- a **thing** is something that behaves as a state machine,
 - has local attributes
 - communicates asynchronously through ports
- **messages** are sent asynchronously between things
- **instances** are established and connected
- ThingML contains a native action language

ThingML and separation of concerns

- Separation of concerns
 - often overlap with abstractions
 - are ways that allow us to focus on a manageable part of reality
 - are ways that makes it possible to distribute work
 - are ways that divides the work and makes it possible to plan
- We will go through many ways to separate concerns
 - PSM and PIM
 - Composite states
 - Concurrent things

The most updated installation instructions

- Follow the Readme.rd at:
 - <https://github.com/SINTEF-9012/ThingML/blob/master/README.md>
- Go to the tutorial of ThingML and walk through it:
https://github.com/HEADS-project/training/tree/master/1.ThingML_Basics
 - Copy the “Hello World” model and execute it!



brice-morin Update README.md

8efc5f2 on Feb 9

5 contributors

279 lines (191 sloc) | 13.2 KB

Raw Blame History

ThingML

The ThingML approach is composed of *i*) a **modeling language**, *ii*) a set of **tools** and *iii*) a **methodology**. The modeling language combines well-proven software modeling constructs for the design and implementation of distributed reactive systems:

- statecharts and components (aligned with the UML) communicating through asynchronous message passing
- an imperative platform-independent action language
- specific constructs targeted at IoT applications.

The ThingML language is supported by a set of tools, which include editors, transformations (e.g. export to UML) and an advanced multi-platform code generation framework, which support multiple target programming languages (C, Java, Javascript). The [methodology](#) documents the development processes and tools used by both the IoT service developers and the platform experts.

HEADS ThingML modelling language basics



Learn the basics of ThingML. How to write you first program in a platform independent way and compile it to different platforms ranging from an Arduino microcontroller to a plain Java program. Also learn how to write platform specific components and link to exiting APIs or libraries.

This tutorial covers:

- Installing the ThingML tools (editor and compilers)
- Basic ThingML syntax and constructs (through a set of examples)
- Compiling to 4 different platforms (Java, NodeJS, Posix C, Arduino)
- Running the compiled code on the 4 different platforms
- Writing ThingML platform independent components
- Specializing ThingML platform-independent components to different platforms
- Writing an additional example program in ThingML

Note:

- Some documentation on the ThingML syntax can be found on the [ThingML web site](#) . It is currently incomplete but might be useful.
- A lot of ThingML example programs can be found [here](#).

0. Installing the HEADS tools for ThingML

To follow this tutorial, you need to have the ThingML editor and compilers. They are released as plugins for the Eclipse IDE. There are two different options for installing the ThingML tools:

- [Download the latest HEADS IDE Eclipse bundle](#). This bundle contains all the HEADS plugins already installed.

ThingML installation for Java starting from HEADS IDE

- ThingML is the modeling language for code generation
 - Look at <http://thingml.org> and <http://heads-project.eu/>
- Download HEADS-IDE from <http://headside.gforge.inria.fr/>
 - It includes Eclipse and most of the necessary plugins
- Install Plantuml from <http://plantuml.sourceforge.net/updatesitejuno>
- and Graphviz from www.graphviz.org
- Install a proper Oracle Java JDK (we do not use a JRE).
 - Configure the Eclipse to point to that Java JDK
 - Windows / Preferences / Java / Installed JREs
- Go to the tutorial of ThingML and walk through it:
https://github.com/HEADS-project/training/tree/master/1.ThingML_Basics

Java and Maven

- We are going (in the first place) to use Java as our target language. In the sequel we may rather use C
- When using Java with ThingML, we also use Maven
 - What you have downloaded with the HEADS IDE also includes a Maven plugin
 - See also <https://maven.apache.org/what-is-maven.html>
- Note: you have to set a real JDK as the JRE
 - Window/Preferences/ click Java/InstalledJREs and then Add... and put in the jdk top folder.

How to compile and run

- Right-click on the configuration file
 - HEADS/ThingML
 - java/swing
 - There will be warning and notices, but beware of errors
- Open folder thingml-gen\java\CPS
 - Right-click on pom.xml
 - Run As
 - Maven Build
 - First time for a project a dialog comes up
 - Give it an appropriate specific name
 - Fill in Goal: *clean install exec:java*

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

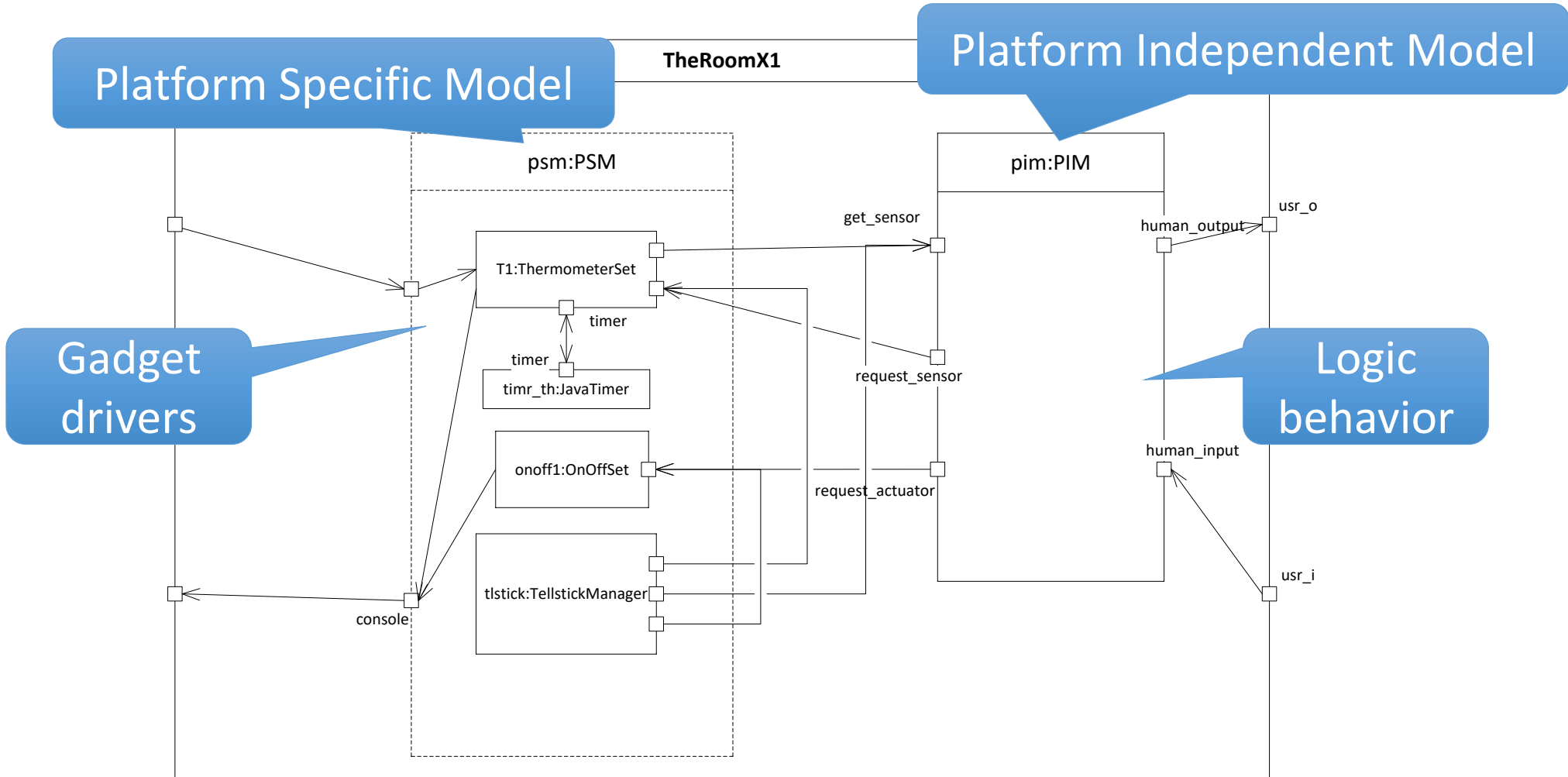
Tellstick



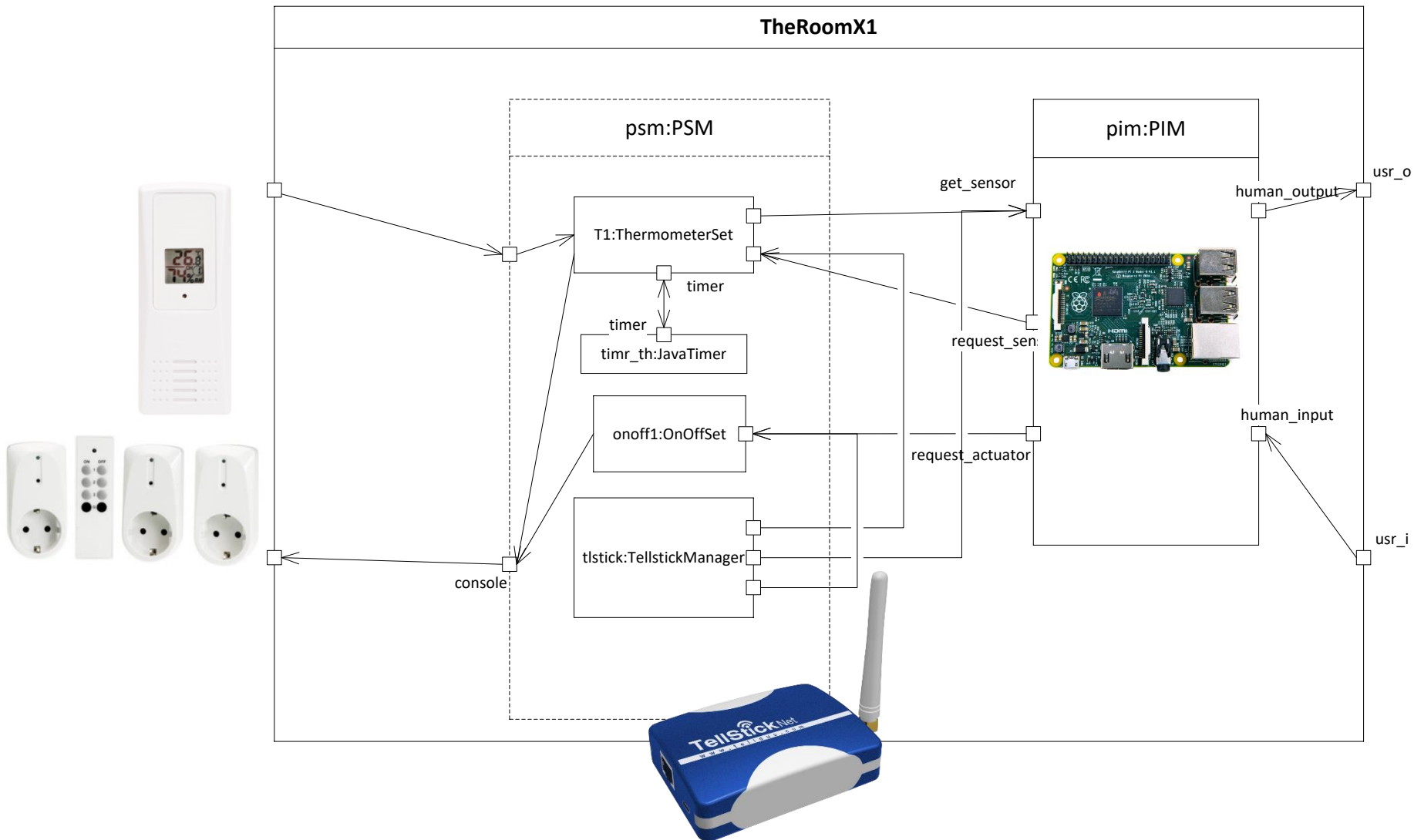
Processor



The software elements

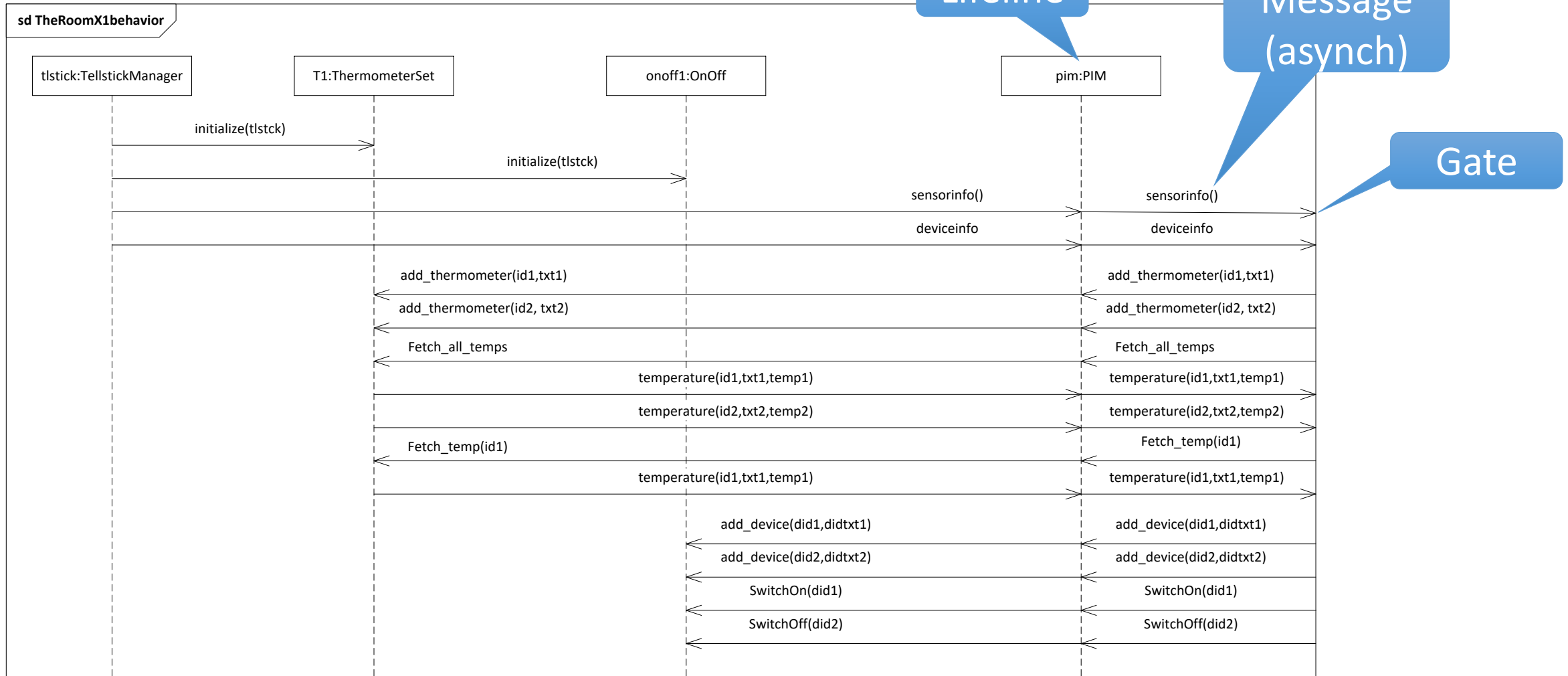


In one picture



Sequence Diagram

The Room X1 Behavior



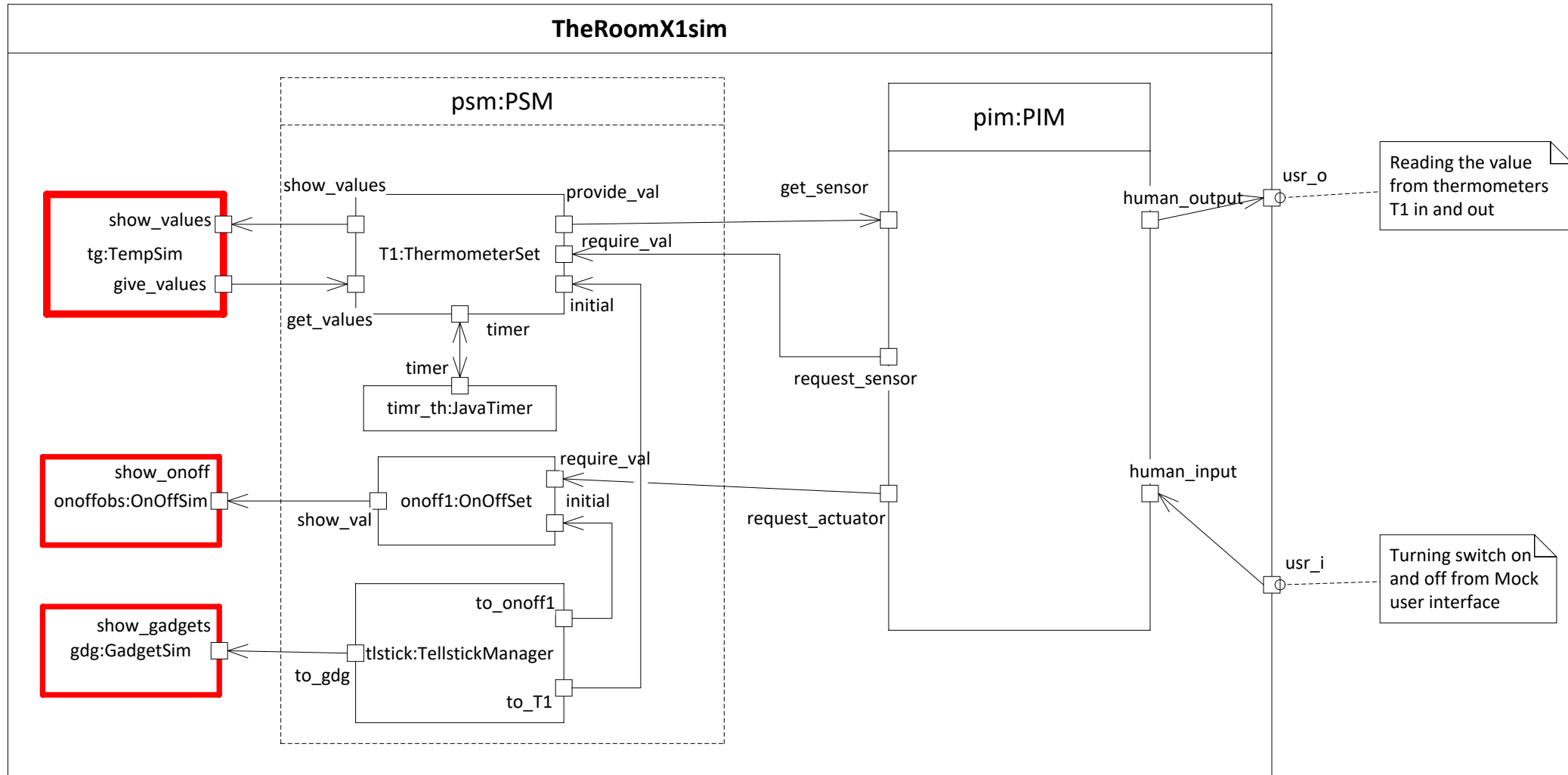
Simulation

- 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

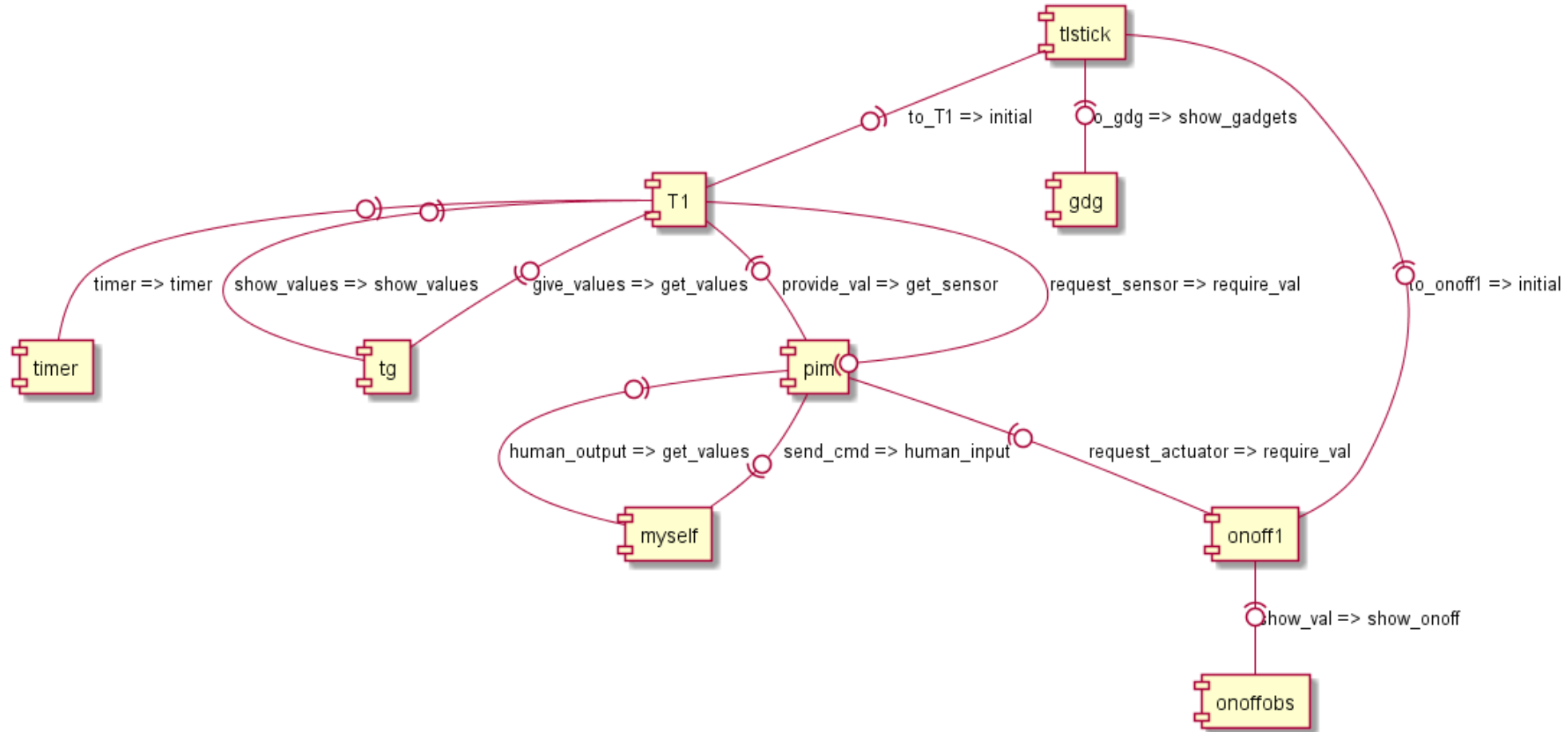
connector onoff1.show_val => onoffobs.show_onoff

// 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

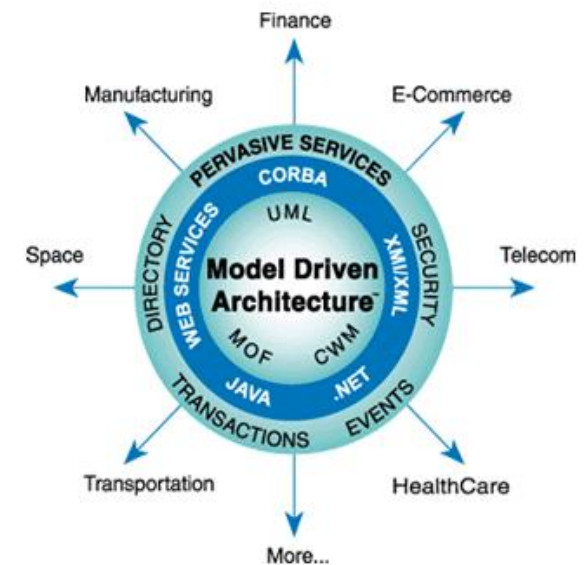
// SIMULATION
connector tg.give_values => T1.get_values
}
```

The Room X1 – ThingML config visualized via PlantUML



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

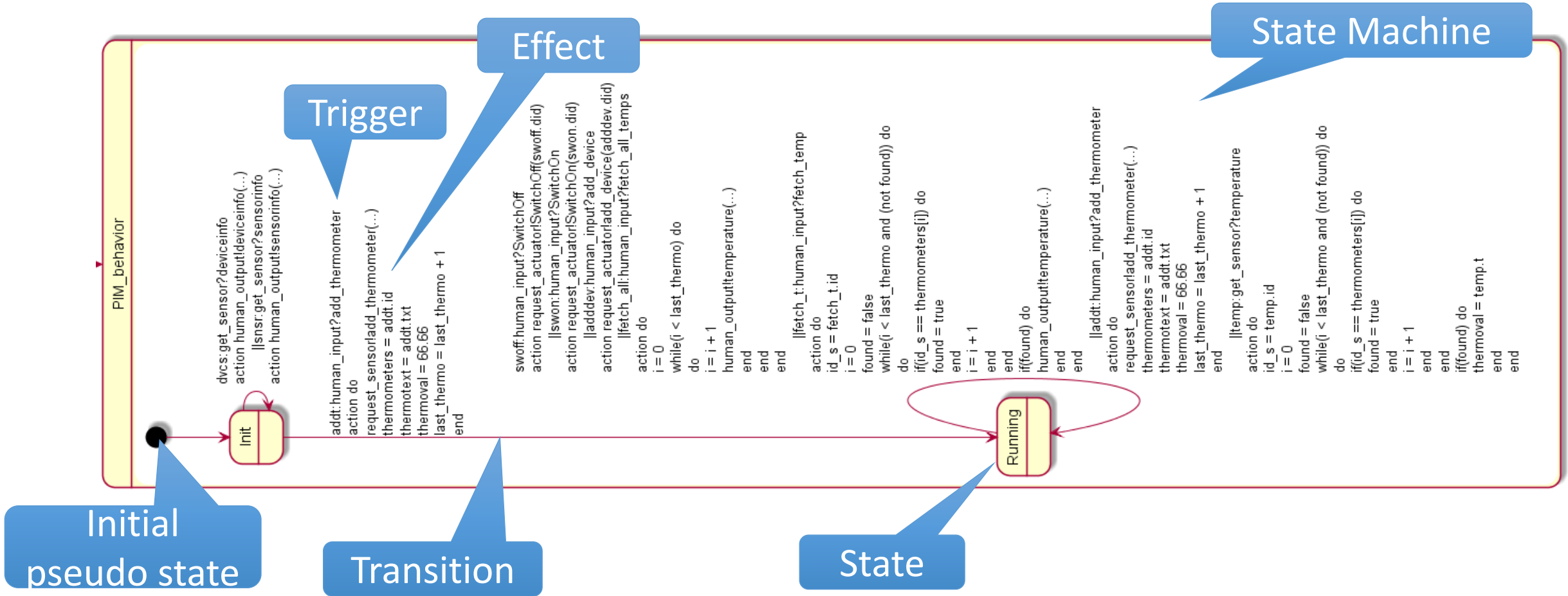
- PSM = Platform Specific Model
 - what will be replaced when the platform (low level) is changed
 - in our case it also means changing from simulation platform to real platform
 - The drivers
- PIM = Platform Independent Model
 - what will be stable regardless of platform changes
 - The application 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 1 – PIM in text (1)

Initial pseudo state

Transition

Trigger

Effect

Message

State Machine

State

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
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

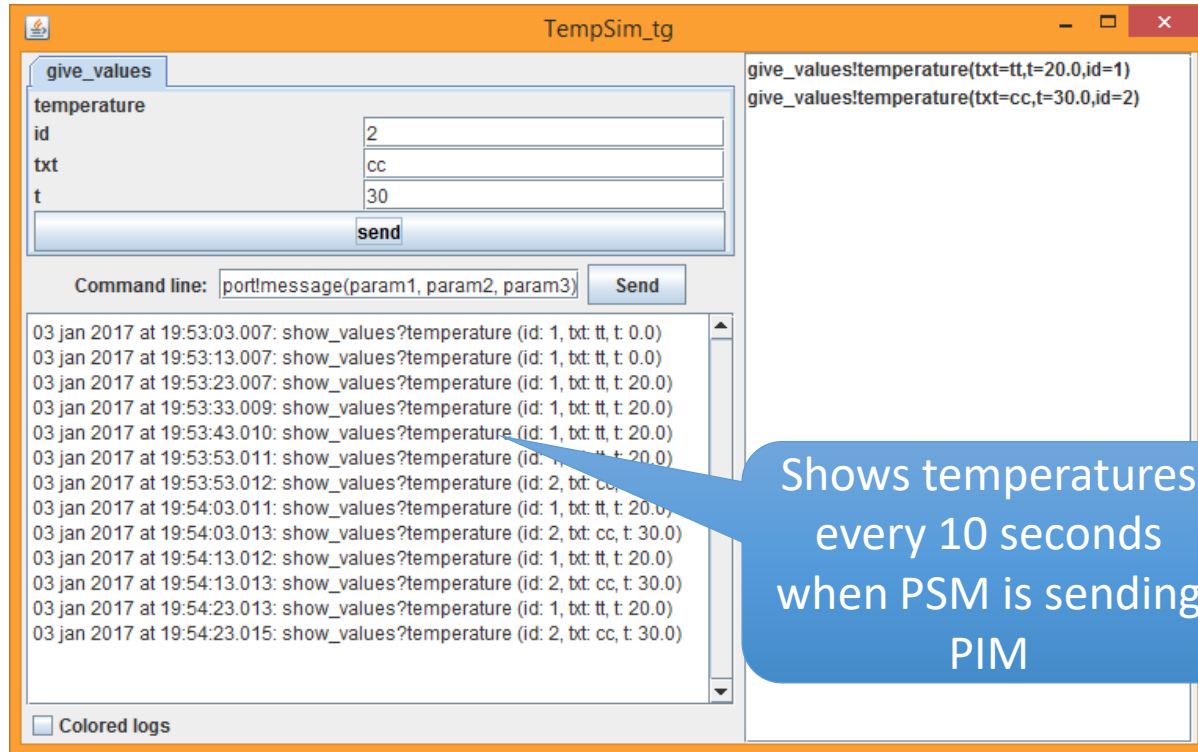
The @mock interface

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

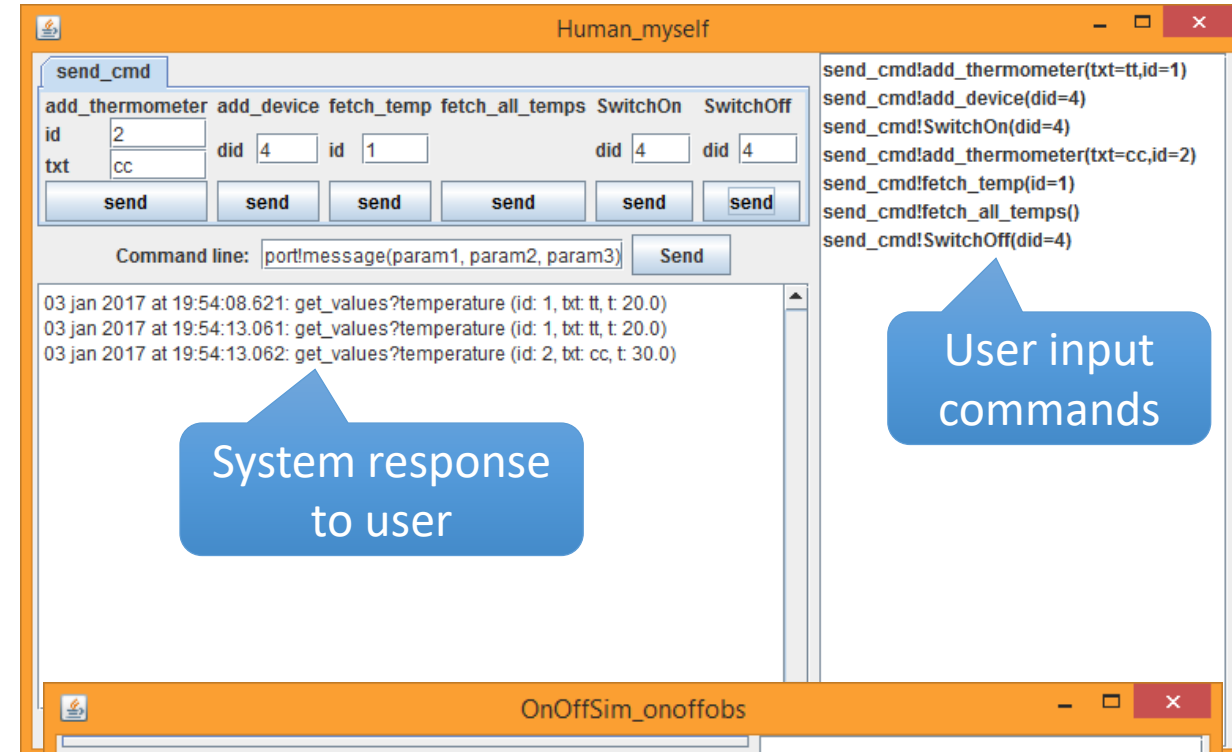
thing GadgetSim includes GeneralMsg
@mock "true"
{provided port show_gadgets {
receives sensorinfo, deviceinfo
}
}

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

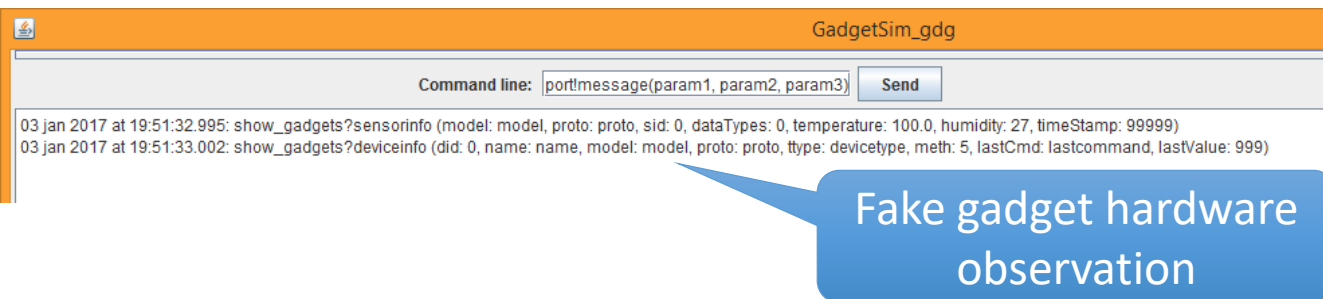
The Room X1 – A simulated execution



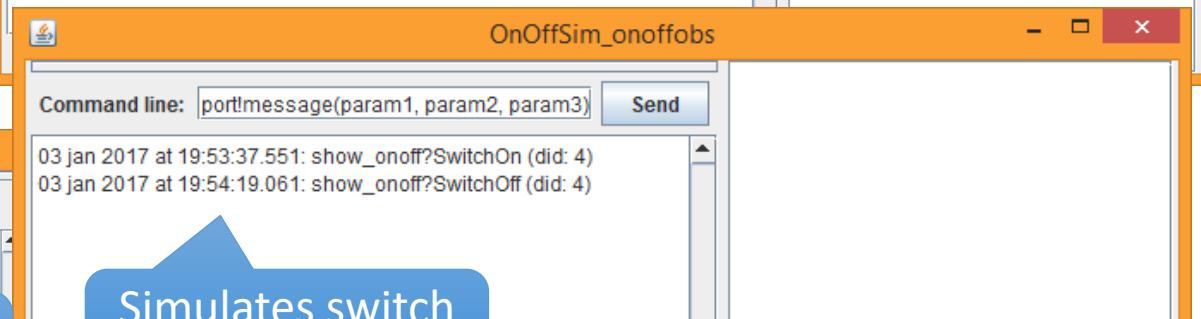
The TempSim_tg window features a 'give_values' section with input fields for 'id' (2), 'txt' (cc), and 't' (30), and a 'send' button. Below this is a 'Command line' field with the text 'port!message(param1, param2, param3)' and another 'Send' button. The main area is a log window displaying a series of 'show_values?temperature' messages with timestamps and parameters. A blue callout bubble points to the log with the text: 'Shows temperatures every 10 seconds when PSM is sending PIM'.



The Human_myself window has a 'send_cmd' section with buttons for 'add_thermometer', 'add_device', 'fetch_temp', 'fetch_all_temps', 'SwitchOn', and 'SwitchOff'. Each button has associated input fields for 'id' and 'did'. Below the buttons is a 'Command line' field and a 'Send' button. The log area shows 'get_values?temperature' messages. A blue callout bubble points to the log with the text: 'System response to user'. Another blue callout bubble points to the right side of the window with the text: 'User input commands'.



The GadgetSim_gdg window has a 'Command line' field with the text 'port!message(param1, param2, param3)' and a 'Send' button. The log area shows 'show_gadgets?sensorinfo' and 'show_gadgets?deviceinfo' messages. A blue callout bubble points to the log with the text: 'Fake gadget hardware observation'.



The OnOffSim_onoffobs window has a 'Command line' field with the text 'port!message(param1, param2, param3)' and a 'Send' button. The log area shows 'show_onoff?SwitchOn' and 'show_onoff?SwitchOff' messages. A blue callout bubble points to the log with the text: 'Simulates switch – on or off'.

The Room X1 – PIM – a summary

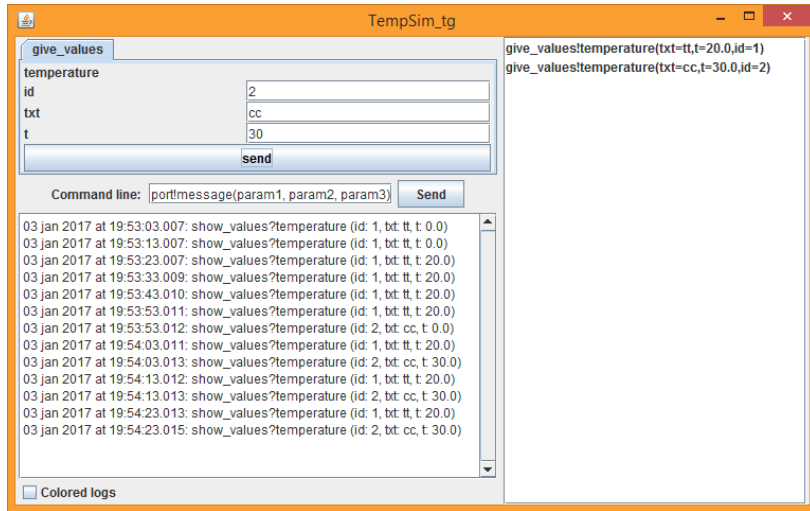
- X1'PIM is a thing where the state machine is rather simple
- X1'PIM functions according to the behavior specified
- X1'PIM does more than that – it functions for more traces than those specified by the sequence diagram
 - Many people would after trying it think it worked well
- X1'PIM is not perfect,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

From Simulation to the Real System

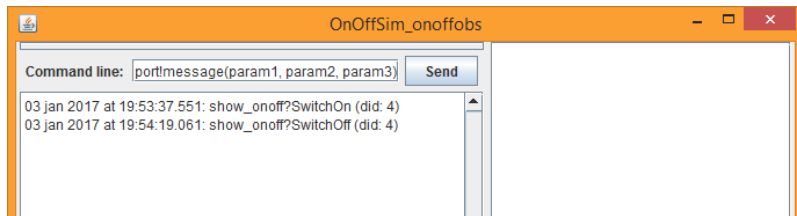
From Simulation to the Real System

- 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



Simulated

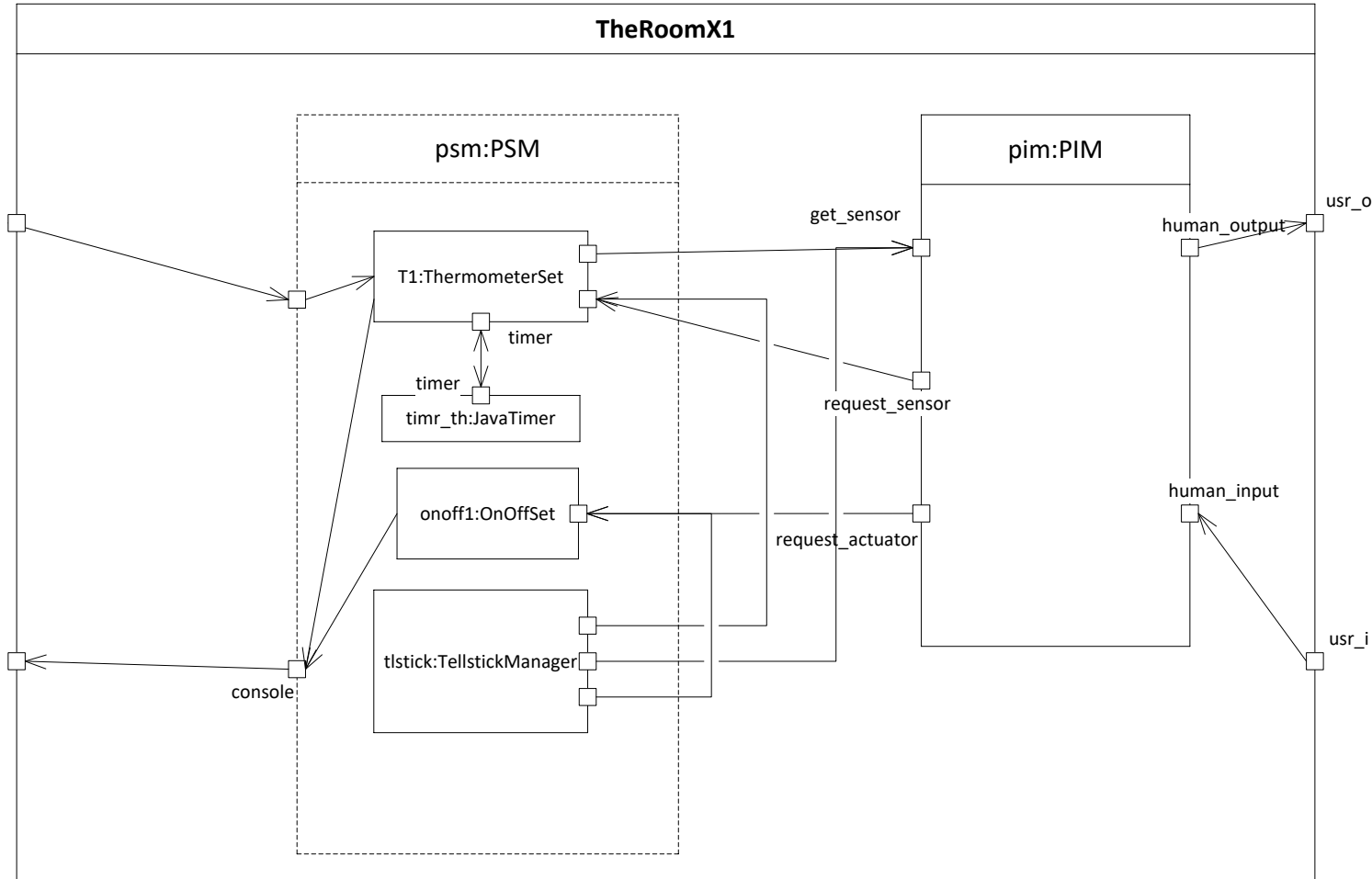
Real



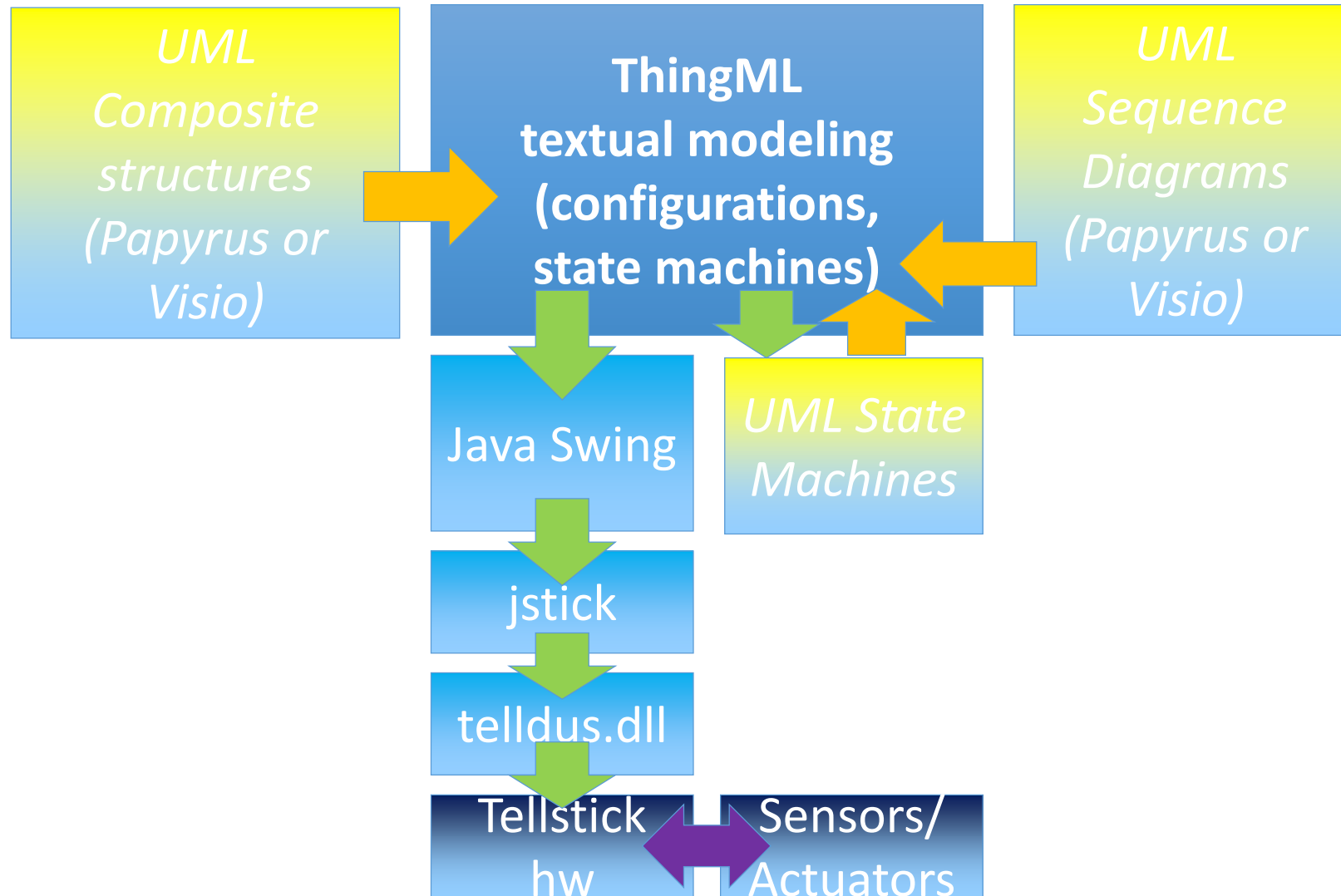
Real



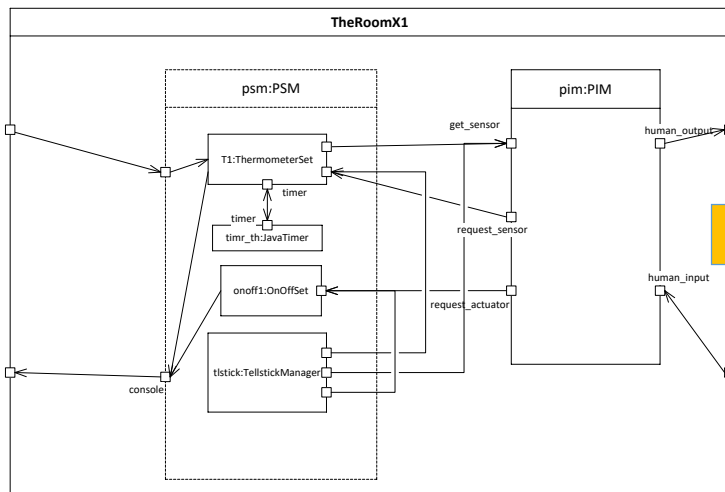
The Room X1: Architecture of the real system



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

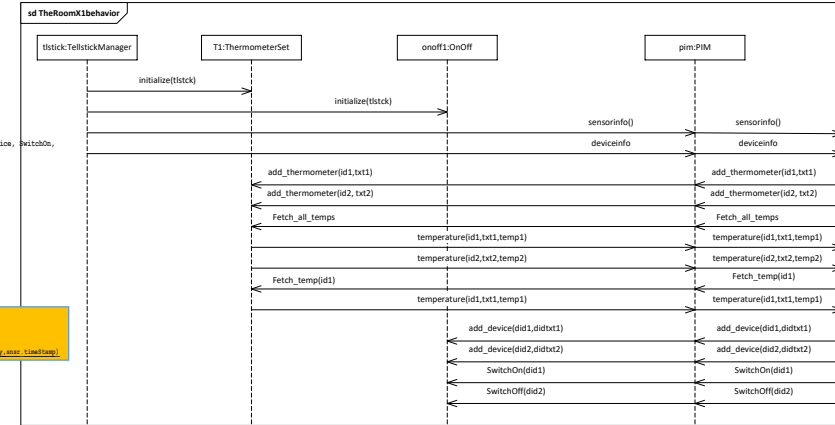
connector pim.request_sensor => H1.require_val
connector pim.request_actuator => onoff1.require_val
connector T1.provide_val => pim.get_sensor

connector myself.get_values => pim.human_output
connector pim.human_input => myself.get_values
}
    
```

```

thing PIM includes GeneralMsg, TemperatureMsg, HumidityMsg, OnOffMsg {
provided port get_sensor {
receives timing, temperature, humidity, sensorinfo, deviceinfo
}
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 con get_sensor/sensorinfo
action do
// nothing
end
}
state Running {
transition -> Running
event user get_sensor/sensorinfo
action do
human_output!sensorinfo(usr.mdl,usr.protocol,usr.sld,usr.dataType,usr.temperature,usr.humidity,usr.timing)
end
}
}
    
```



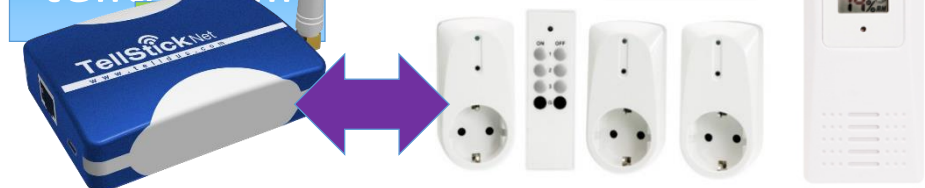
Java Swing

jstick

tellus dll

```

PIM_sensor
...
add_device(did,tx)
...
add_thermometer(d1,tx1)
...
add_hygrometer(d1,tx1)
...
switchon(did)
...
switchoff(did)
    
```

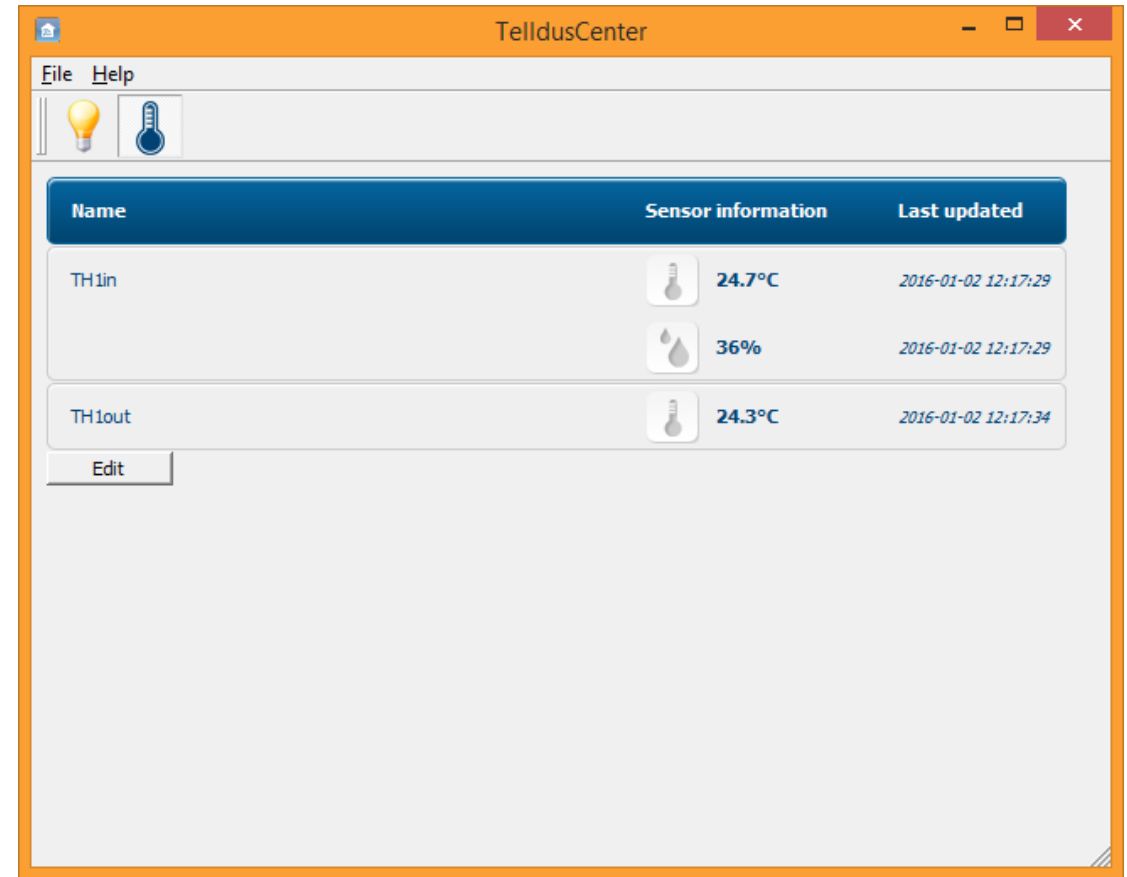
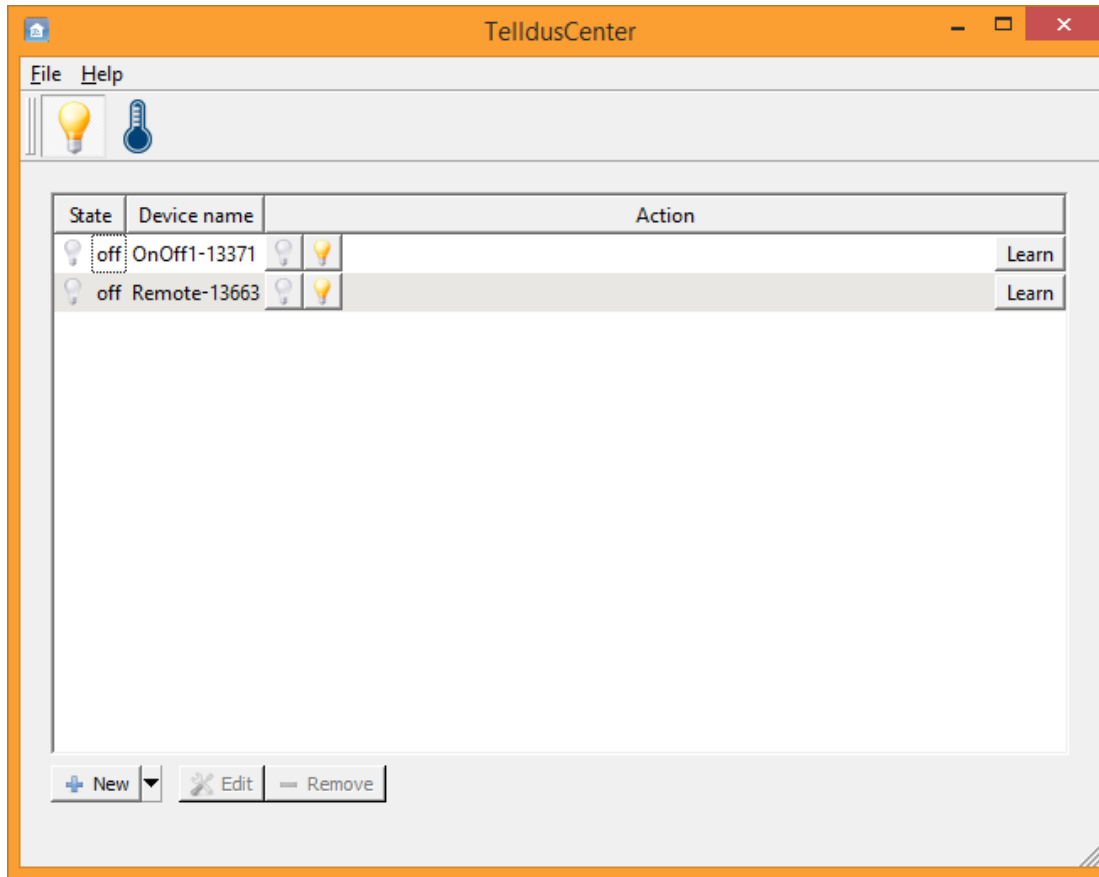


Tellstick from Telldus

- 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
end
```

&sensor_list&

'.getModel()'

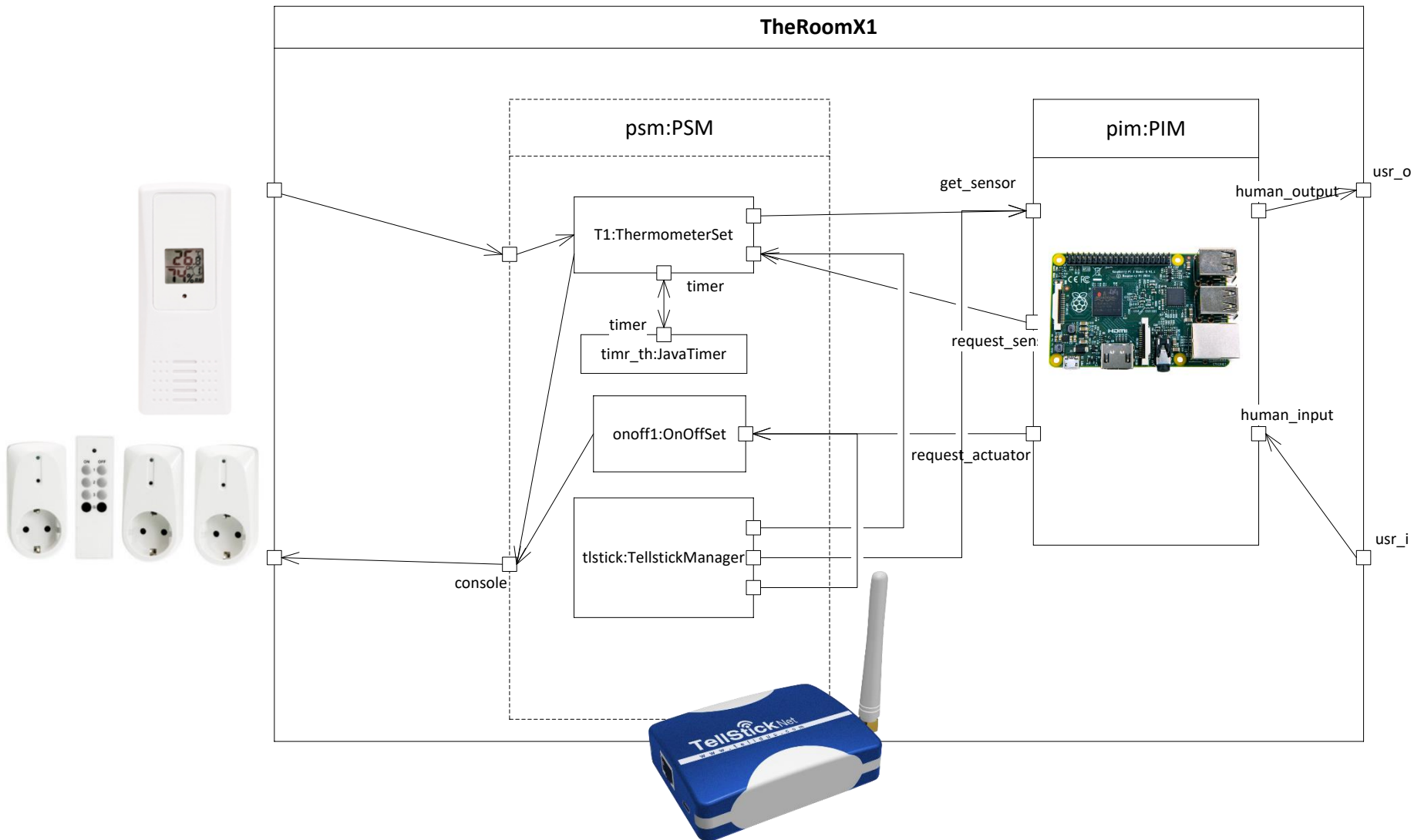
PSM code: The two principles of ThingML kick-down

- `'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

L2: The Room X2 with Thermostat

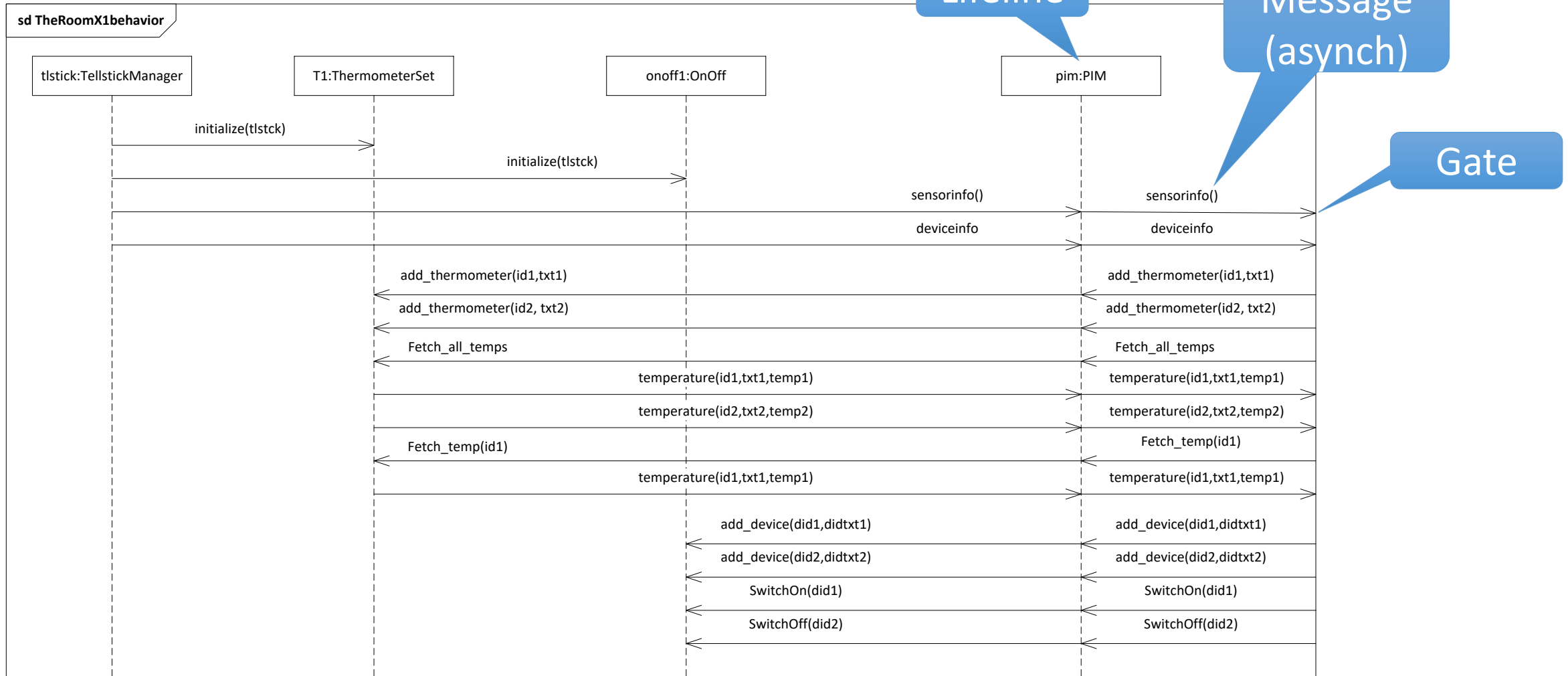
X1: Recap The Room

In one picture

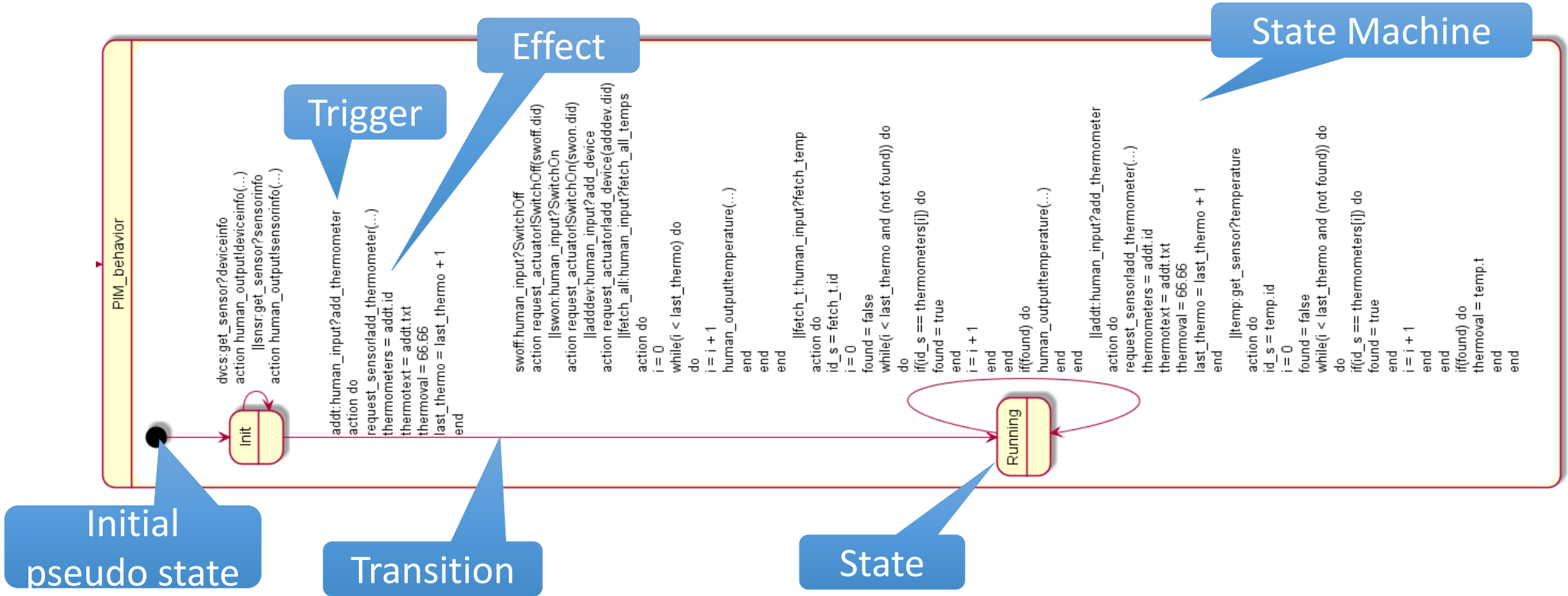


Sequence Diagram

The Room X1 Behavior



The Room X1 – PIM state machine visualized



The Room 1 – PIM in text (1)

Initial pseudo state

Transition

Trigger

Effect

Message

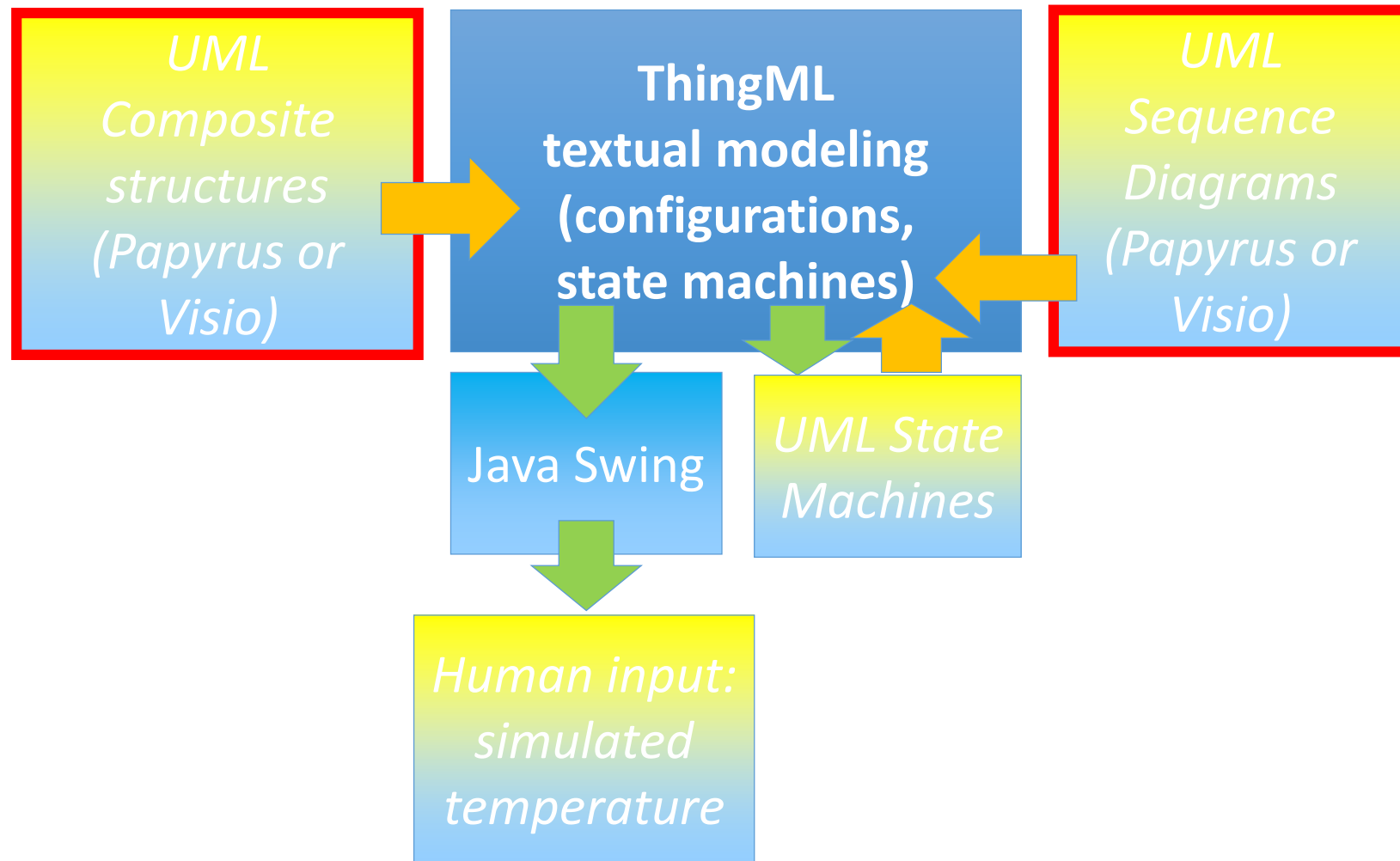
State Machine

State

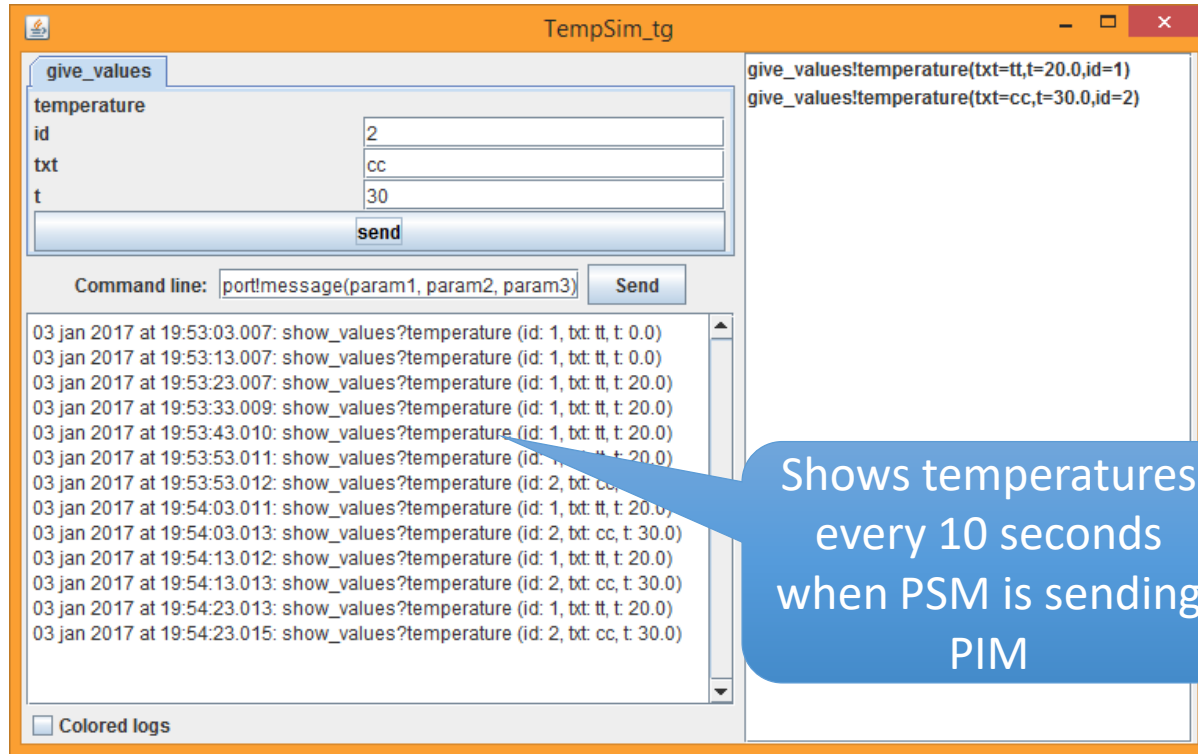
Port

```
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
end
}
```

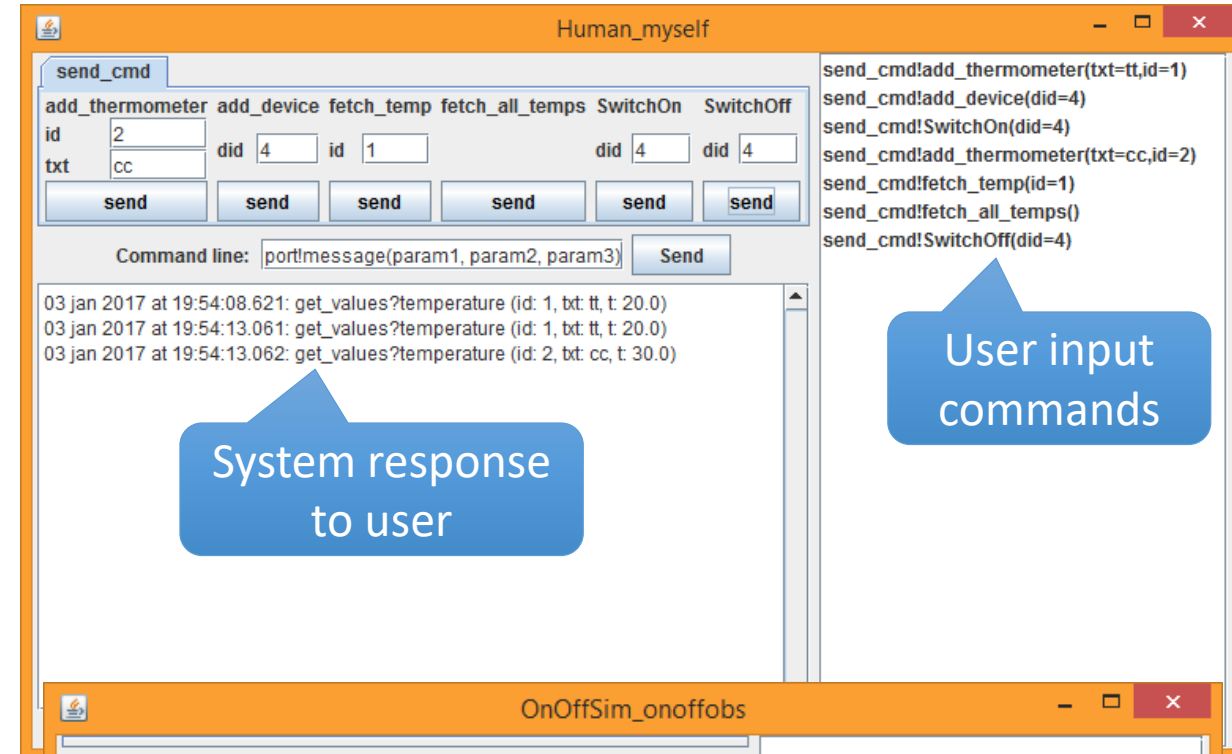
Simulating the laboratory



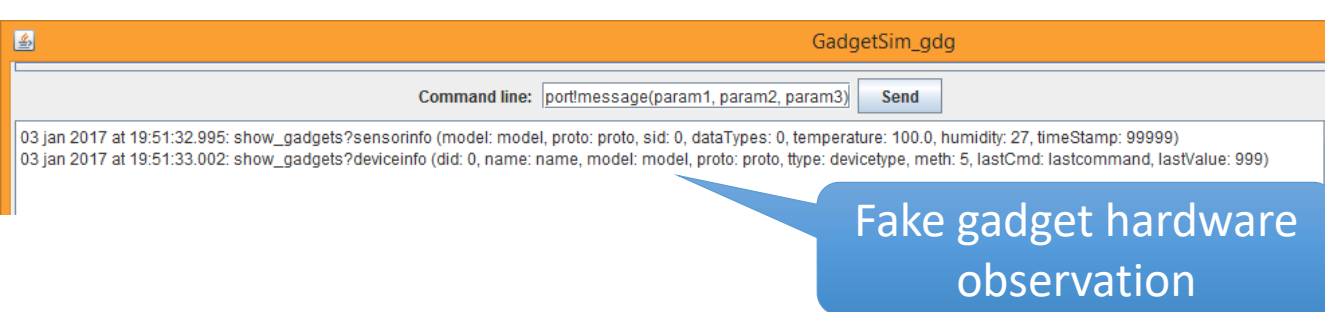
The Room X1 – A simulated execution



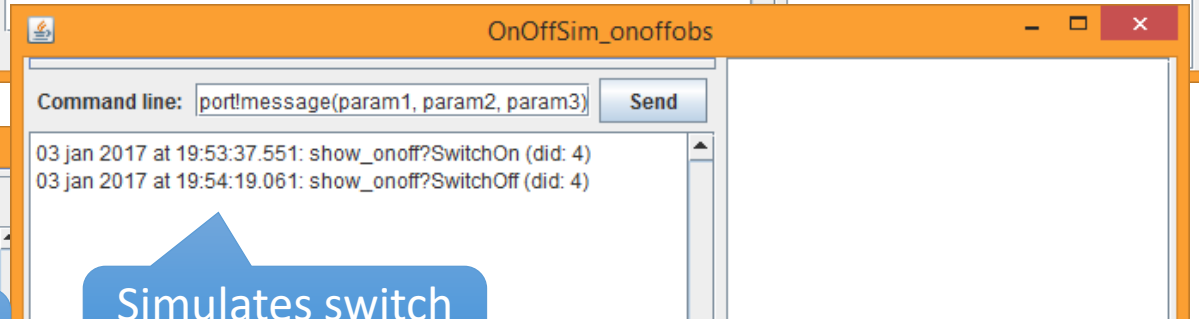
The TempSim_tg window features a 'give_values' section with input fields for 'id' (2), 'txt' (cc), and 't' (30), and a 'send' button. Below this is a 'Command line' field with the text 'port!message(param1, param2, param3)' and another 'Send' button. The main area is a log window displaying a series of messages: '03 jan 2017 at 19:53:03.007: show_values?temperature (id: 1, txt: tt, t: 20.0)' through '03 jan 2017 at 19:54:23.015: show_values?temperature (id: 2, txt: cc, t: 30.0)'. A blue callout bubble points to the log with the text: 'Shows temperatures every 10 seconds when PSM is sending PIM'.



The Human_myself window has a 'send_cmd' section with buttons for 'add_thermometer', 'add_device', 'fetch_temp', 'fetch_all_temps', 'SwitchOn', and 'SwitchOff'. Each button has associated input fields for 'id' and 'did'. Below these are six 'send' buttons. A 'Command line' field contains 'port!message(param1, param2, param3)' and a 'Send' button. The log shows three messages: '03 jan 2017 at 19:54:08.621: get_values?temperature (id: 1, txt: tt, t: 20.0)', '03 jan 2017 at 19:54:13.061: get_values?temperature (id: 1, txt: tt, t: 20.0)', and '03 jan 2017 at 19:54:13.062: get_values?temperature (id: 2, txt: cc, t: 30.0)'. A blue callout bubble points to the log with the text: 'System response to user'. On the right side, a list of commands is shown: 'send_cmd!add_thermometer(txt=tt,id=1)', 'send_cmd!add_device(did=4)', 'send_cmd!SwitchOn(did=4)', 'send_cmd!add_thermometer(txt=cc,id=2)', 'send_cmd!fetch_temp(id=1)', 'send_cmd!fetch_all_temps()', and 'send_cmd!SwitchOff(did=4)'. A blue callout bubble points to this list with the text: 'User input commands'.



The GadgetSim_gdg window has a 'Command line' field with 'port!message(param1, param2, param3)' and a 'Send' button. The log shows two messages: '03 jan 2017 at 19:51:32.995: show_gadgets?sensorinfo (model: model, proto: proto, sid: 0, dataTypes: 0, temperature: 100.0, humidity: 27, timeStamp: 99999)' and '03 jan 2017 at 19:51:33.002: show_gadgets?deviceinfo (did: 0, name: name, model: model, proto: proto, ttype: devicetype, meth: 5, lastCmd: lastcommand, lastValue: 999)'. A blue callout bubble points to the log with the text: 'Fake gadget hardware observation'.



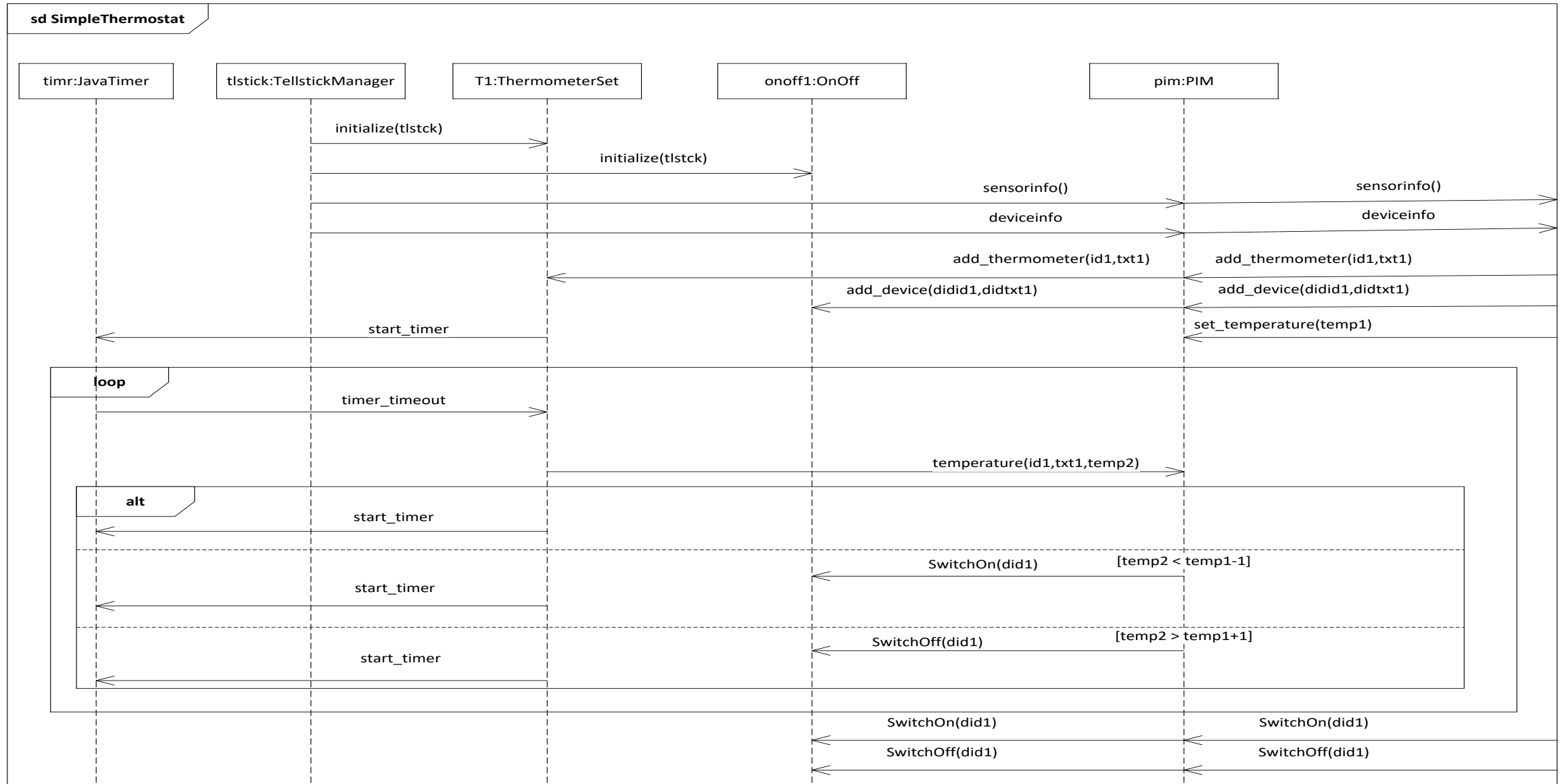
The OnOffSim_onoffobs window has a 'Command line' field with 'port!message(param1, param2, param3)' and a 'Send' button. The log shows two messages: '03 jan 2017 at 19:53:37.551: show_onoff?SwitchOn (did: 4)' and '03 jan 2017 at 19:54:19.061: show_onoff?SwitchOff (did: 4)'. A blue callout bubble points to the log with the text: 'Simulates switch – on or off'.

X2A: The first thermostat

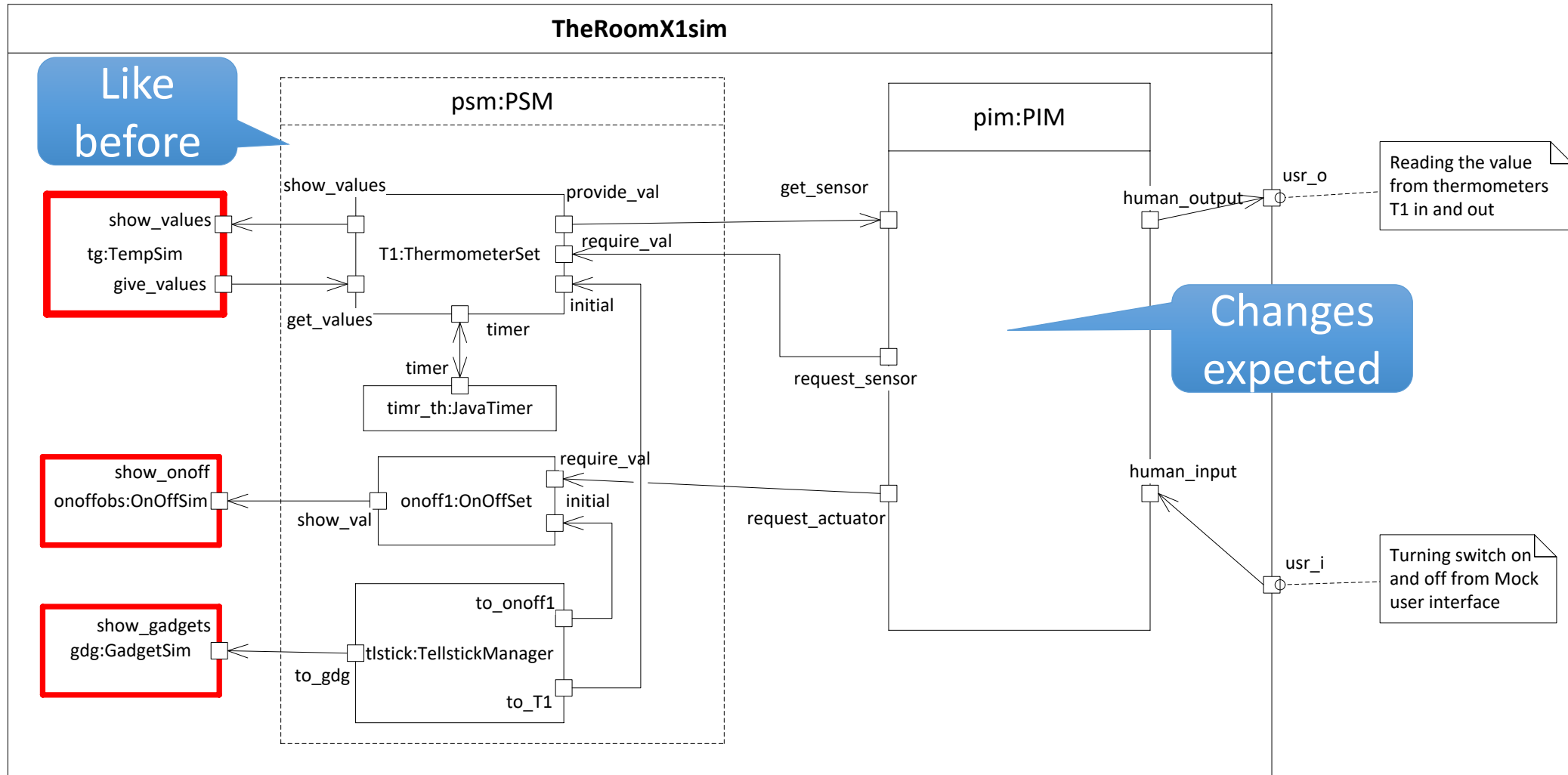
X2A: The Room with a simple Thermostat

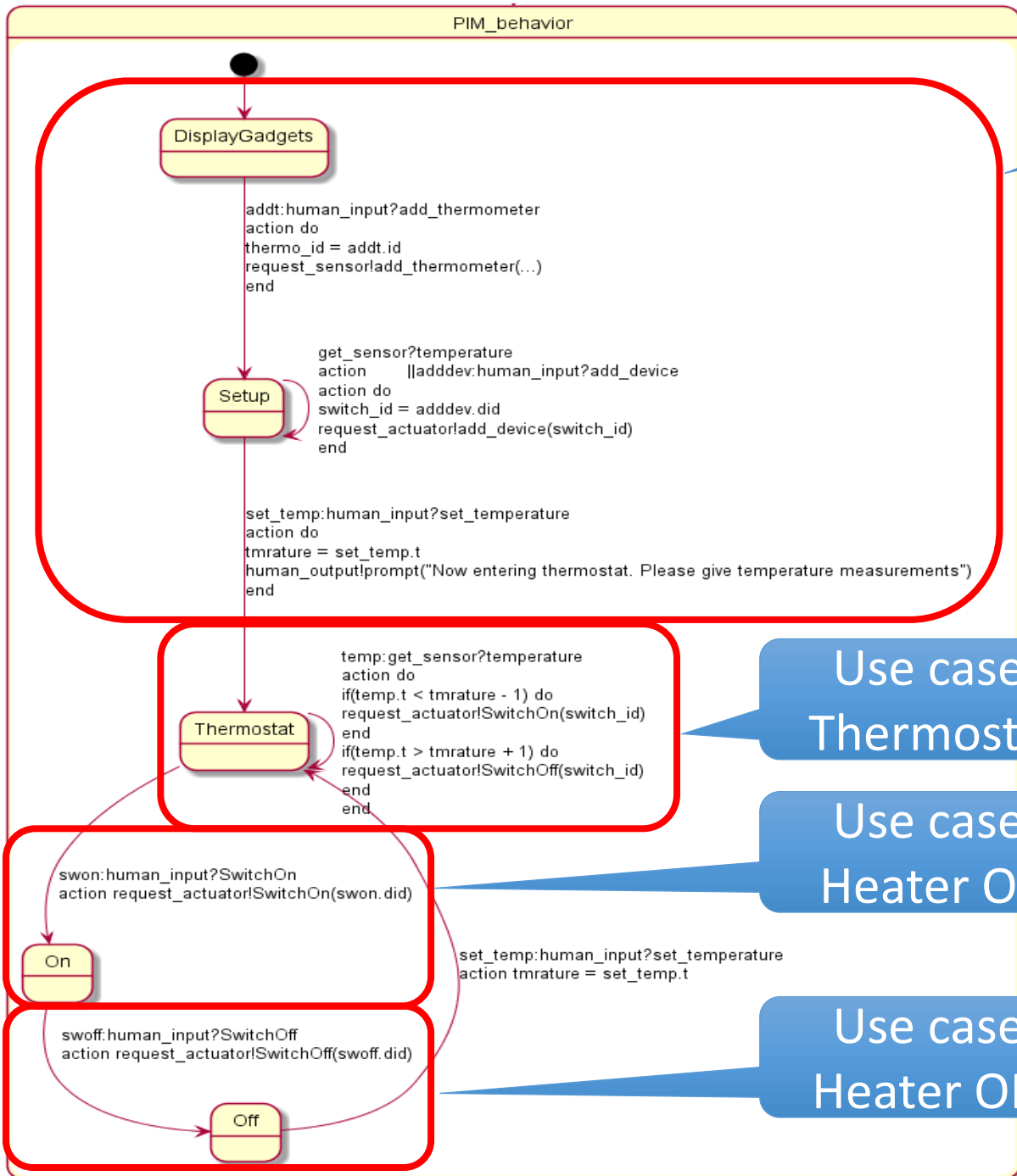
- Our room X2 has
 - One thermometer
 - One switch (on/off) that turns heat on or off
- The functionality requirements are
 - Keep the room temperature within a comfort range of temperatures
 - Directly turn switch ON or OFF
- We assume that in Norway the temperature will fall if there is no heating, and rise when there is heating
- Our first solution attempt for the thermostat:
 - When the temperature is below the bottom threshold, switch on
 - When the temperature is above the upper threshold, switch off

Behavior of the simple Thermostat



The Room X2 – Simulation architecture as X1





Similar to X1

The PIM behavior now changes

Use case: Thermostat

Use case: Heater ON

Use case: Heater OFF

We execute the simulated system

- We follow closely the behavioral description given by the sequence diagram
 - Provide the adequate input
 - Check that the generated output is according to the spec
- If we can walk through all the variants of the sequence diagram, and the generated output is as specified, then the state machine is consistent with the interaction

Execution (4 windows)

The image displays four overlapping windows from a simulation environment:

- TempSim_tg**: A window titled "Temperature simulation" with a "give_values" tab. It contains input fields for "id" (1), "txt" (T), and "t" (23), and a "send" button. The log shows a series of "show_values?temperature" messages with timestamps and parameters (id: 1, bt: T, t: 23.0).
- Human_myself**: A window titled "User Interface" with a "send_cmd" tab. It features buttons for "add_thermometer", "add_device", "SwitchOn", "SwitchOff", and "set_temperature", each with associated input fields and a "send" button. The log shows "send_cmd" messages for these actions and a "get_values?prompt" message.
- GadgetSim_gdg**: A window titled "Fake gadget info" with a "Command line" input field and a "Send" button. The log shows "show_gadgets?sensorinfo" and "show_gadgets?deviceinfo" messages.
- OnOffSim_onoffobs**: A window titled "Switch operations" with a "Command line" input field and a "Send" button. The log shows a sequence of "show_onoff?SwitchOn" and "show_onoff?SwitchOff" messages.

Are we happy now?

- The state machine PIM is consistent with the Interaction SimpleThermostat
- but the behavioral specification in a sequence diagram is not complete – it does not cover all situations

Observations when we simulate

- The state machine specifies a very strict order between the states
Thermostat, On and Off
 - but there is no logical reason for this order
 - The user should freely be able to move between these running states
- The default duration between temperature signals may not be perfect for all simulations
 - We should be able to set the temperature cycle

Observation of the state machine specification

- We have two states that relate to initial setup of the thermostat
- We have three states that relate to running the Room X2
- The specification does not in itself highlight this distinction between setup and running situations

X2B: Composite States

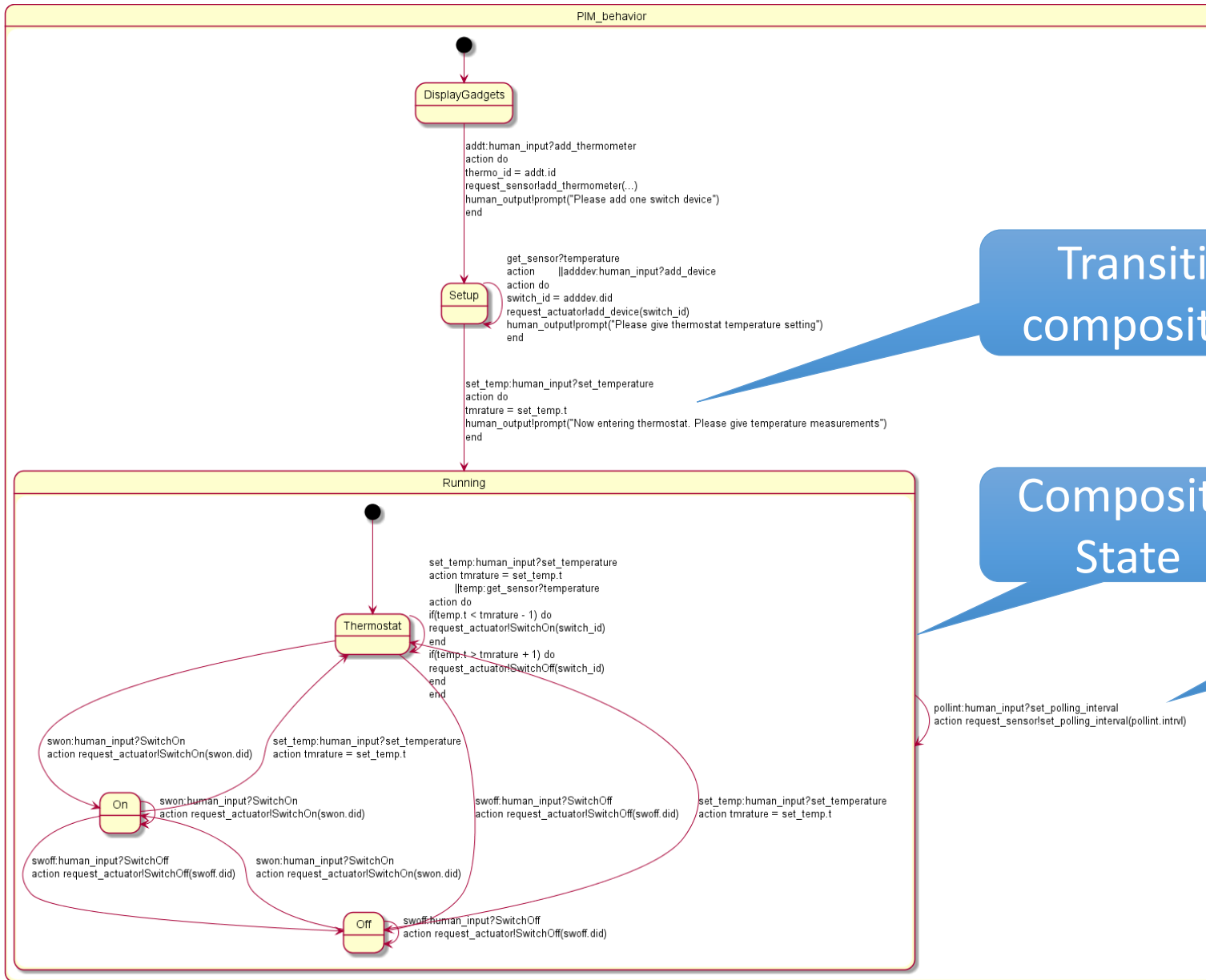
X2B: The Room with composite state

- We introduce composite state
 - as a way to group states for better overview
 - as a means to achieve less redundancy
- We also show how easy it is to introduce a new service
 - SetPollingInterval: how often the temperature is checked

The Room X2B

- Let the user move freely between Thermostat, On, Off
- Wrap a Running state around (Thermostat, On, Off)
- Introduce a new service *set_polling_interval* which will set the duration between temperature measures

The Room X2B PIM behavior



Transition to composite state

Composite State

Transition on composite state

Composite State in ThingML

```
statechart PIM_behavior init DisplayGadgets {
  state DisplayGadgets {...}
  state Setup { ...
    transition -> Running ...
  }
  composite state Running init Thermostat keeps history {
    state Thermostat {
      transition -> Thermostat ...
      transition -> On ...
      transition -> Off ...
      transition -> Thermostat ...
    }
    state On {
      transition -> Off ...
      transition -> On ...
      transition -> Thermostat ...
    }
    state Off {
      transition -> Off ...
      transition -> On ...
      transition -> Thermostat ...
    }
  }
  transition -> Running ...
}
```

Transition to
composite state

Composite
State

Note: keeps history
(not shown in UML diagram)

Transition on
composite state

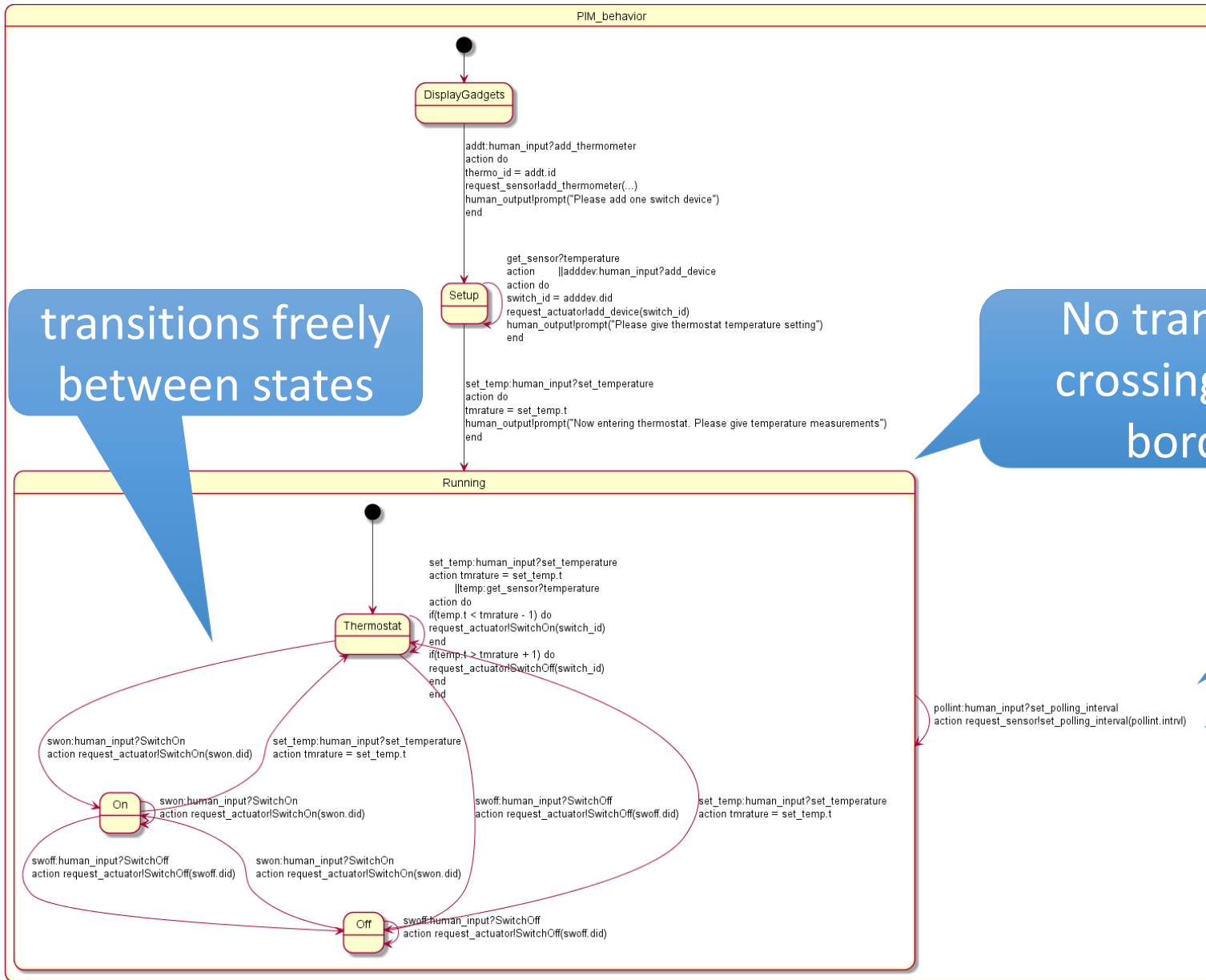
The Semantics of a Composite State

- In ThingML transitions can only go between states on the same level
- There may be simple and composite states on same level
- Any trigger will trigger on the innermost level where it matches
- If there is no match on one level, the next level out will be attempted

Semantics of Composite States (history)

- When a composite state is entered the first time, the inner state given by the **init**-clause will be entered
- When a composite state is re-entered, and it has no “keeps history” clause, it will also go to the state of the init-clause
- When a composite state with a **keeps history** clause is entered, it will return to the last inner state where it was before it left the composite state

The Room X2B PIM behavior



transitions freely between states

No transition crossing state border

Adding set_polling_interval has no effect on existing transitions

Practical to put transition here instead of at every inner state (keeps history)

Separation of Concerns – Why?

- Think and reason locally – keep your focus
- Apply structuring means to
 - Identify and name areas of concerns that are manageable
 - Encapsulate
 - Hide / Show
- You may separate behavior as well as structure
 - Separated between PSM and PIM (structure)
 - Composite states define chunks of behavior

Are we happy now with The Room X2B?

- The Room is according to its specification, but there may still be some problems we would like to mitigate
- Simulation is effective, but simulation is a way of abstraction that may disguise important details of reality
 - Here when running the real system, we realize that the switch is being set unnecessarily
 - Logically there is no problem that a switch is turned on when it is already on, but in practice this may probably wear the switch out long before it needed to

X2C: Smarter switching

X2C: Actuators may be worn out if applied too frequently

- We observe that switches are applied all the time
 - This may wear out the hardware too soon
 - Intelligent use of composite states will help
- We look at more than happy day scenarios

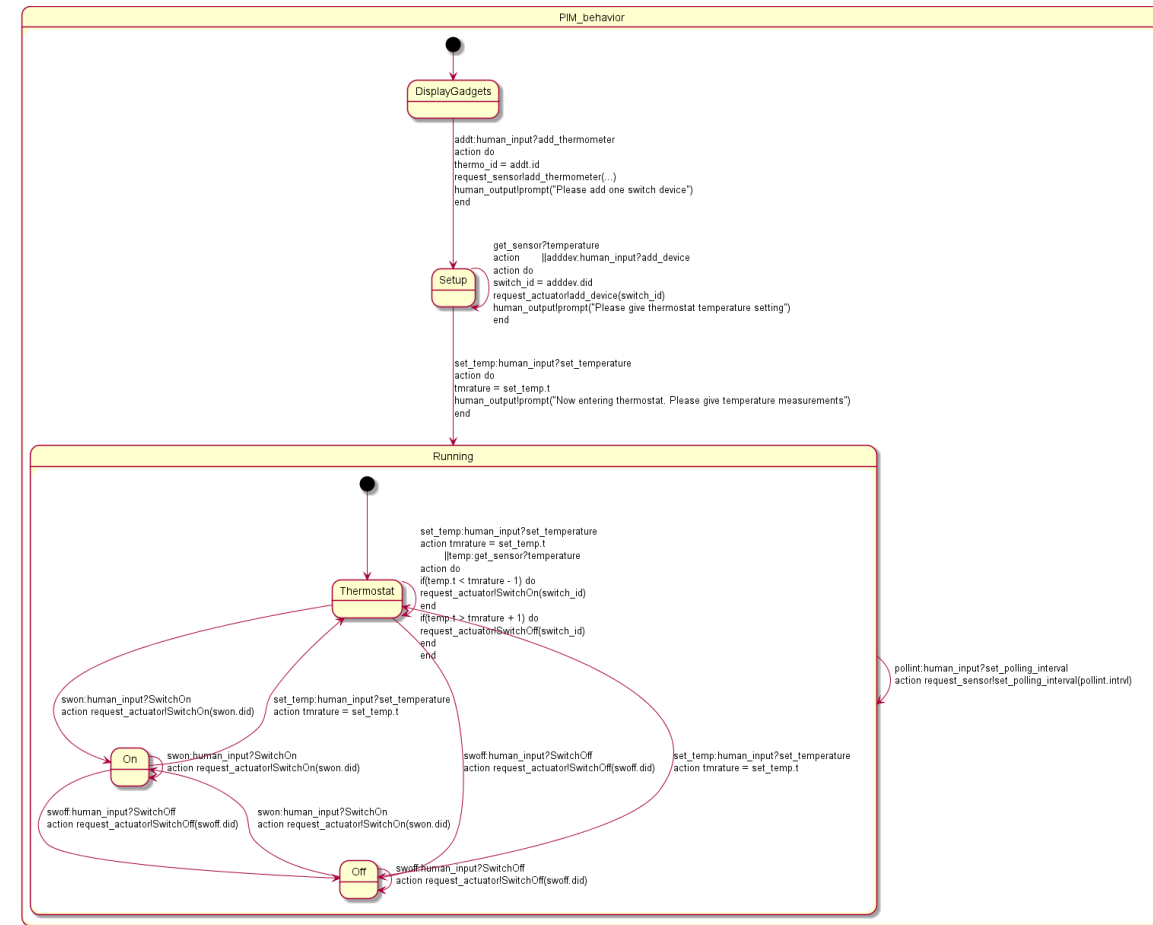
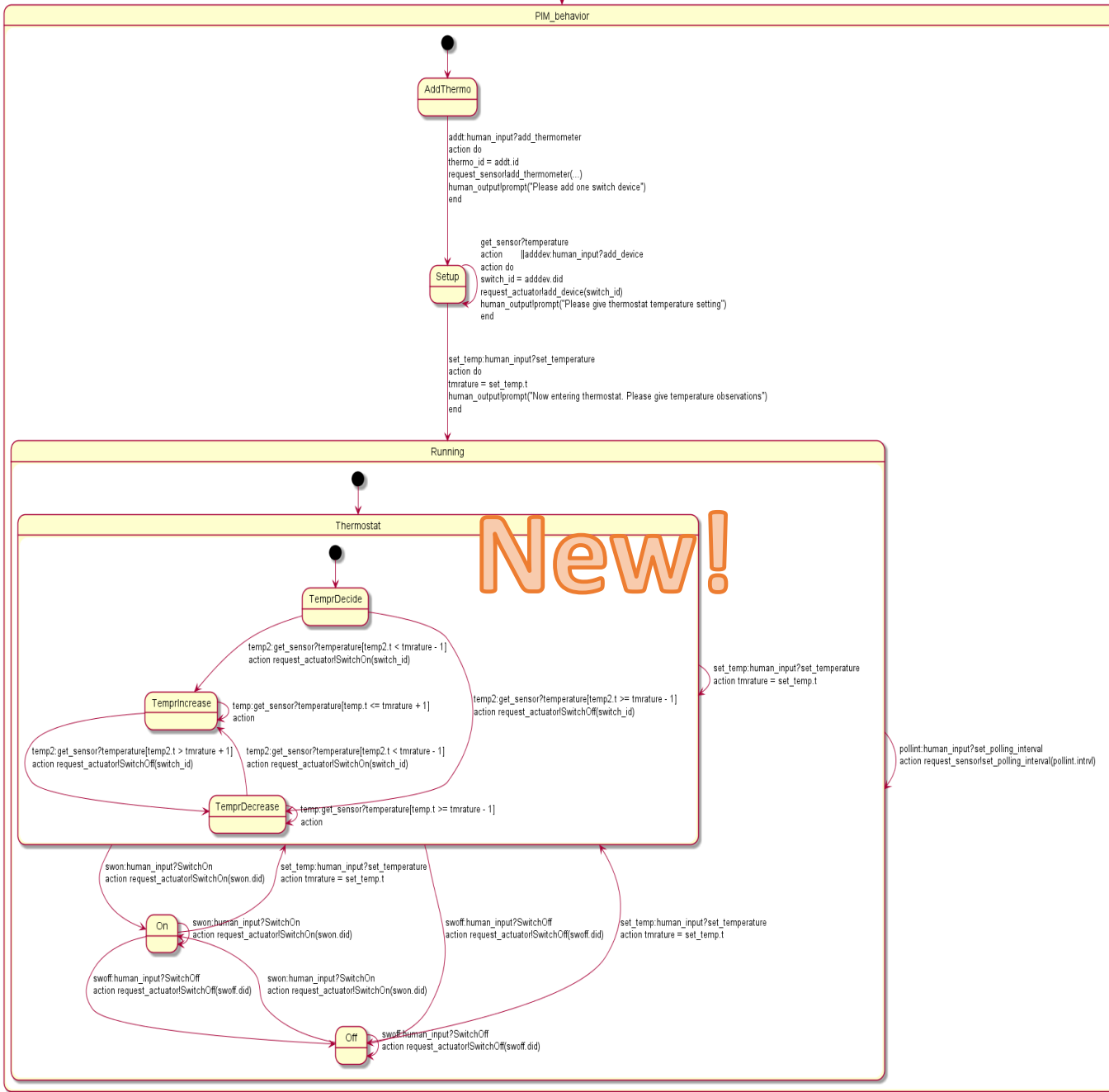
Goal: Reduce or remove the redundant switching

- We want to reduce or remove unnecessary application of the switches due to the risk of wearing the switches out prematurely
- Switching to ON is unnecessary if it is already ON
 - and the temperature should be increasing
- Switching to OFF is unnecessary if it is already OFF
 - and the temperature should be decreasing

Separation of concerns

- The problem we want to mitigate only concerns the thermostat functionality
 - This should mean that our solution should only affect the Thermostat state, and all other states and transitions should remain untouched

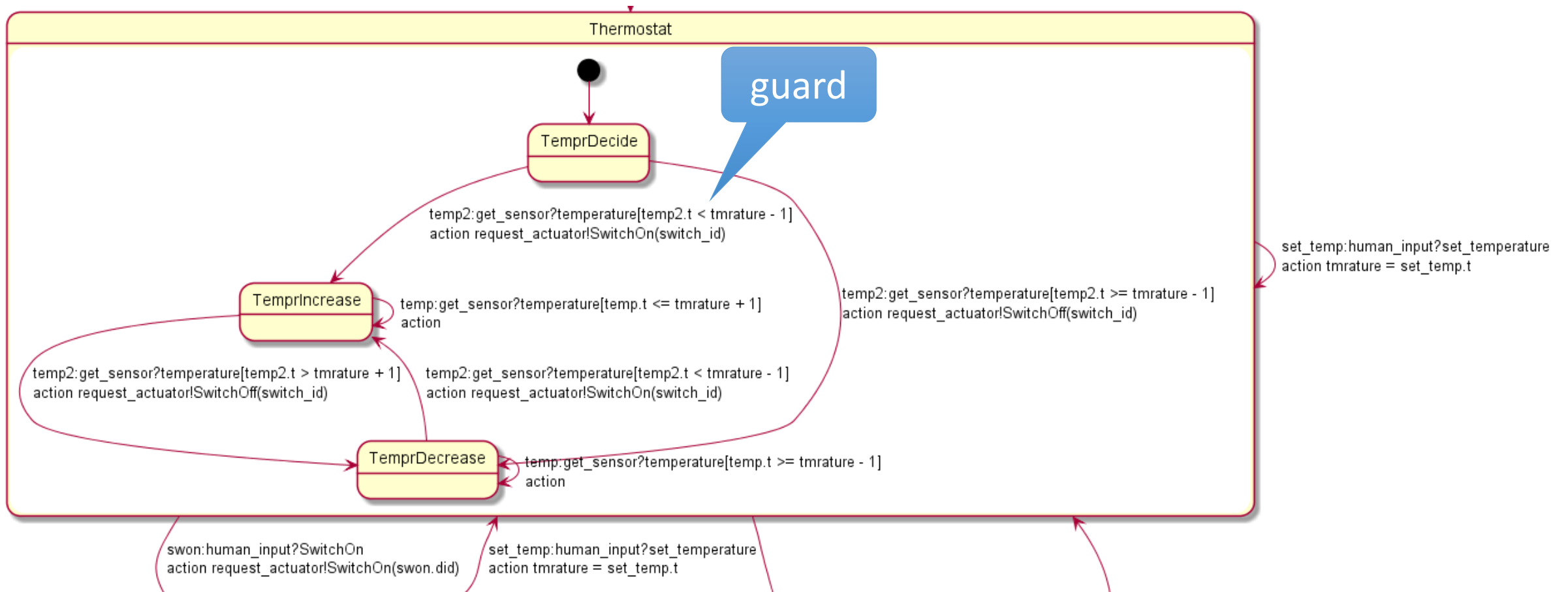
Thermostat revisited, everything else stable



Our solution

- We propose to make Thermostat a composite state
 - and include two inner states TemprIncrease and TemprDecrease with the obvious state invariants that the temperature should increase in TemprIncrease and decrease in TemprDecrease
- This is not entirely sufficient since when we enter the Thermostat we must always determine the adequate position of the switch
 - and for that purpose we introduce a third state TemprDecide

The Thermostat with inner states



The Thermostat in ThingML

```
composite state Thermostat init TemprDecide {
// notice that we are NOT keeping history
  state TemprDecide {
    transition -> TemprDecrease
    event temp2:get_sensor?temperature
    guard temp2.t>=tmrature-1 // OFF as much possible
    action do
      request_actuator!SwitchOff(switch_id)
    end

    transition -> TemprIncrease
    event temp2:get_sensor?temperature
    guard temp2.t<tmrature-1
    action do
      request_actuator!SwitchOn(switch_id)
    end
  }

  state TemprIncrease{
// Invariant: Switch is ON and temperature should increase
    transition -> TemprIncrease
    event temp:get_sensor?temperature
    guard temp.t<=tmrature+1
    // increasing until well above desired temperature
    action do // nothing
    end

    transition -> TemprDecrease
    event temp2:get_sensor?temperature
    guard temp2.t>tmrature+1
    action do
      request_actuator!SwitchOff(switch_id)
    end
  }
}
```

guard

```
state TemprDecrease{
// Invariant: Switch is OFF and temperature should decrease
  transition-> TemprDecrease
  event temp:get_sensor?temperature
  guard temp.t>=tmrature-1 // it should keep decreasing until
  well below the desired temperature
  action do // nothing
  end

  transition -> TemprIncrease
  event temp2:get_sensor?temperature
  guard temp2.t<tmrature-1
  action do
    request_actuator!SwitchOn(switch_id)
  end
}

// Transitions from Thermostat to states on same level
transition -> On
event swon:human_input?SwitchOn
action do
  request_actuator!SwitchOn(swon.did)
end
transition -> Off
event swoff:human_input?SwitchOff
action do
  request_actuator!SwitchOff(swoff.did)
end
transition -> Thermostat
event set_temp:human_input?set_temperature
action do
  tmrature = set_temp.t
end
} //end of Thermostat
```

The Room X2C: Summary

- We introduced more complexity in state Thermostat to mitigate a problem of reality, namely that setting switches frequently may wear the hardware
- We were able to confine our changes to the single state Thermostat
 - but it became a composite state with 3 inner states
- Is our system perfect now?
 - It is quite good for happy day scenarios, but how does it handle the awkward events?

X2D: Robustification 1

X2D: The Room must handle any signal at any time

- First robustification approach: cover all possible signals
- Show how composite states are useful for concise description of the robustification with minimal interference

Our Room must be more robust

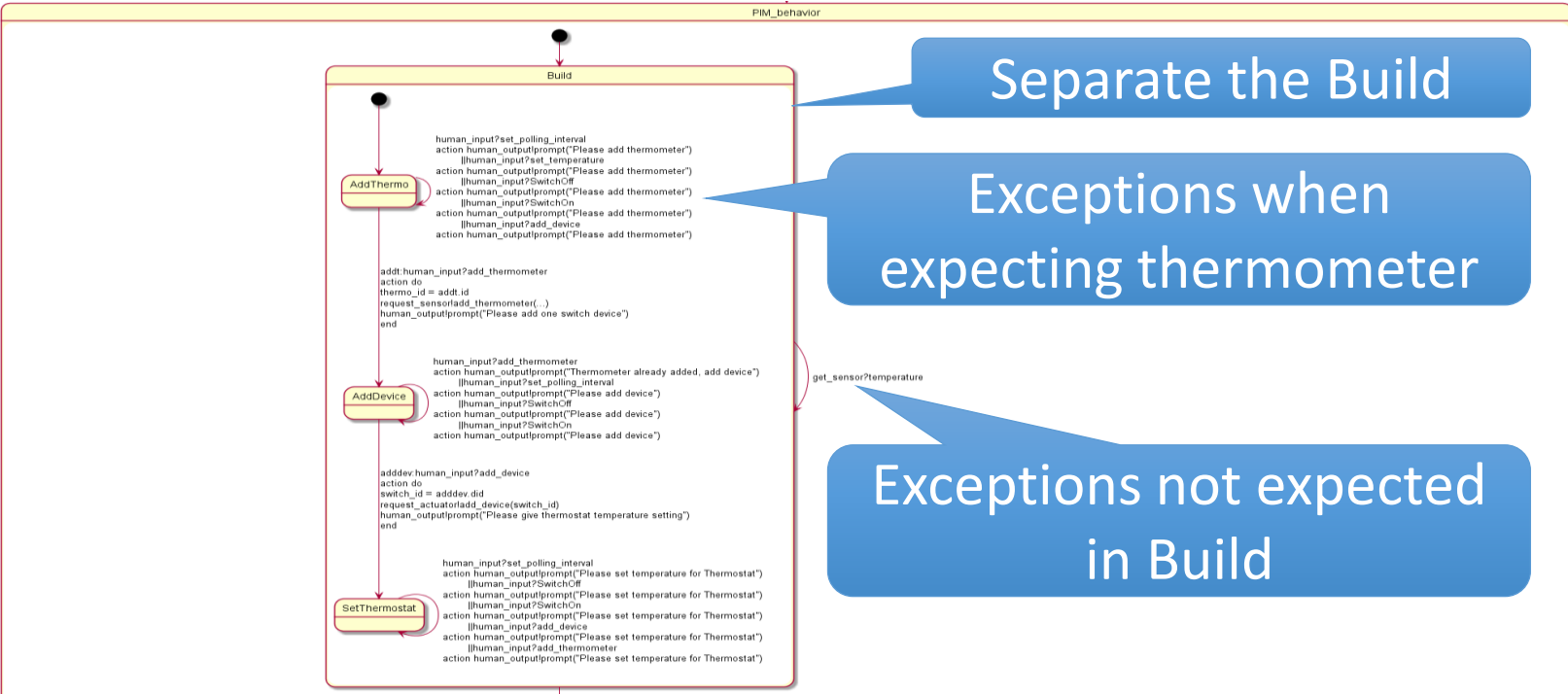
- The Room X2C works when nothing unexpected happens
 - Such a room could function for years
- What about the unexpected?
 - How can we know anything about the unexpected? Would that not be counterintuitive since we cannot expect the unexpected?

The Beauty of State Machines

- Finite State Machines are finite!
 - There is a finite number of states
 - and the number is in our cases a small number
 - There is a finite number of signals to handle
 - and the number is in our cases a reasonably small number
 - There is a finite possible number of unique transitions
 - and in principle we can define them all
- A State captures the whole history up till now
 - Think locally for every state

Making the initial building of The Room more concise

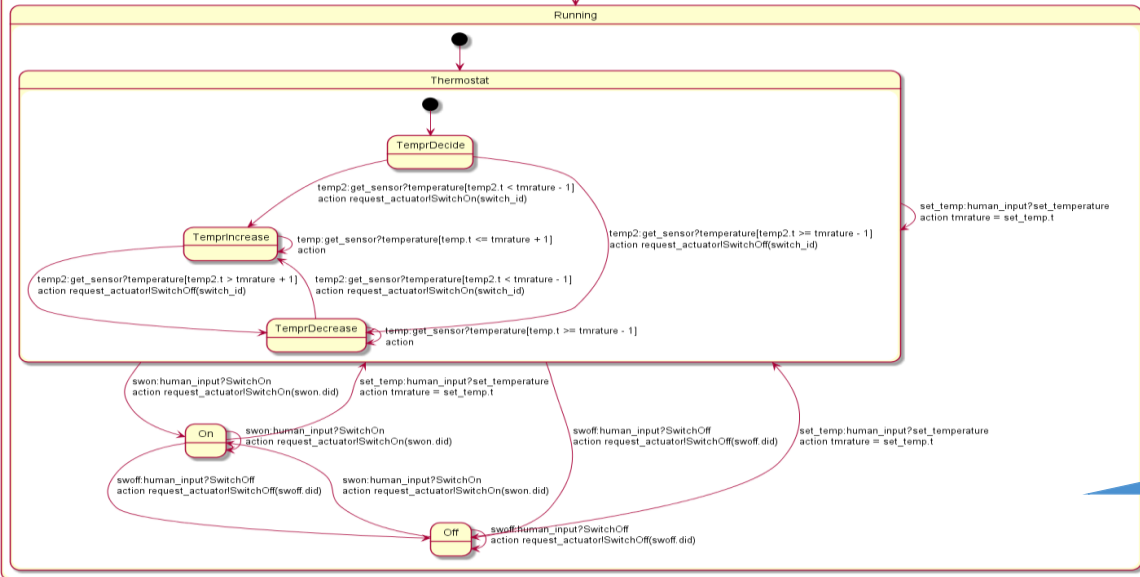
- Walking through the initial building of The Room, we realize that we should control the order more directly
- More control may not always be a bad thing
- We introduce the composite state Build to distinguish the setup from the Running
 - Clear separation of concerns



Separate the Build

Exceptions when expecting thermometer

Exceptions not expected in Build



Exceptions not expected in Running

Inside Running, robustification has no impact

The Room X2D – covering all signals

The unexpected in ThingML (1)

Separate the Build

```
composite state Build init AddThermo keeps history {  
  state AddThermo {  
    transition -> AddDevice  
    event addt:human_input?add_thermometer  
    action do  
      thermo_id=addt.id  
      request_sensor!add_thermometer(thermo_id,addt.txt)  
      human_output!prompt("Please add one switch device")  
      // SIMULATION: prompting on console for the user to react properly  
    end  
    transition -> AddThermo // Cover other messages  
    event human_input?add_device  
    event human_input?SwitchOn  
    event human_input?SwitchOff  
    event human_input?set_temperature  
    event human_input?set_polling_interval  
    action do  
      human_output!prompt("Please add thermometer")  
    end  
    // temperature is handled on Build level  
  }  
}
```

Exceptions when
expecting thermometer

The unexpected in ThingML (2)

```
// Normal transition to the Running state
transition -> Running
event set_temp:human_input?set_temperature
action do
    tmrature = set temp.t
    human_output!prompt("Now entering thermostat. Please give temperature observations")
    // SIMULATION: prompting on console for the user to react properly
end

//Escape situations
transition -> Build
event get_sensor?temperature
    // just discard, the thermostat is not running, yet
} // end Build
```

Exceptions not expected
in Build

The unexpected in ThingML (3)

```
// Transitions of the composite state Running
transition -> Running
event pollint:human_input?set_polling_interval
action do
    // just forward the polling interval instructions to the PSM
    request_sensor!set_polling_interval(pollint.intrvl)
end
transition -> Running
event temp:get_sensor?temperature
    // just discard - this should only happen when in On or Off states

// Messages that should not occur, but may occur
transition -> Running
event human_input?add_thermometer
event human_input?add_device
action do
    human_output!prompt("Adding gadgets has been done and then blocked")
end
// Messages the cannot occur - since they are always handled
transition -> Running
event human_input?SwitchOn
event human_input?SwitchOff
event human_input?set_temperature
action do
    human_output!prompt("INTERNAL ERROR: Impossible messages at PIM.Running")
end
} // end Running
} // end PIM_behavior
} // end PIM_thing
```

Normal situation, not related to the thermostat function as such

Exceptions not expected in Running

Human input which is misplaced, but very possible

Technical software firewall: our analysis shows this cannot happen, but we still catch it

The Room X2D: First Robustification, all signals covered

- Since finite state machines are finite, exploit this!
- Walk through all transitions and have a conscious attitude to what the effects should be
- Apply composite states for concise description
- Distinguish between
 - Normal situations within happy day scenarios
 - Possible situations from which we need some recovery
 - Impossible situations that we still catch to cover own errors

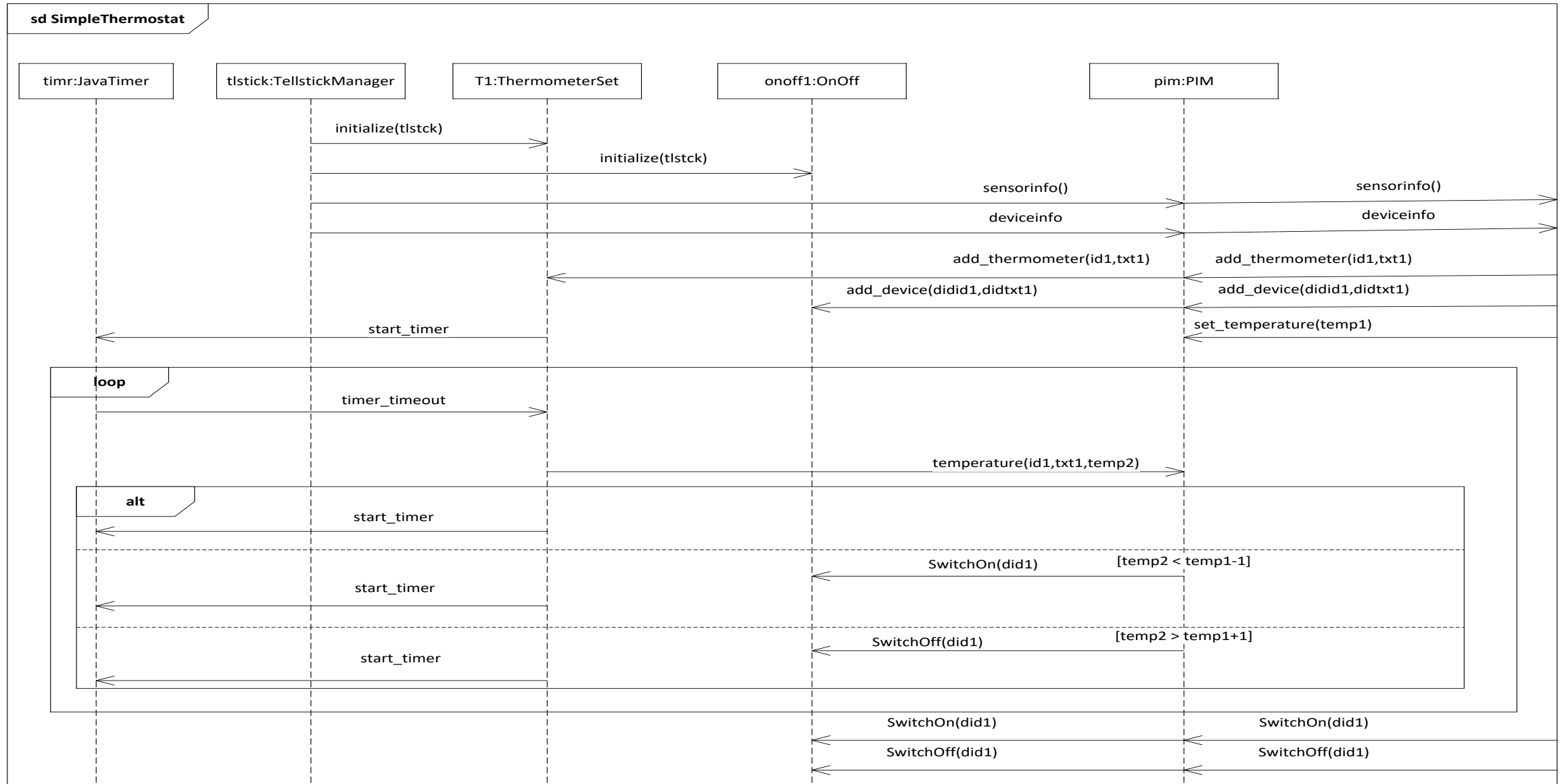
Are we now happy with our Room Thermostat at X2D?

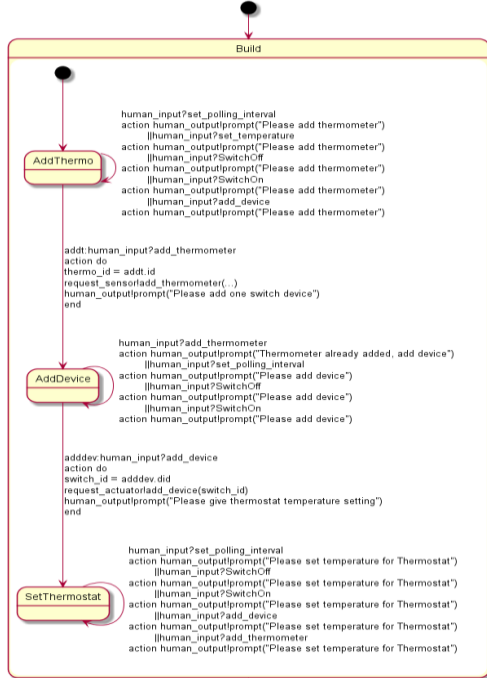
- We have now a fairly well built software logic (PIM)
- A good product is the best motivator for new requests!
 - Maintenance starts when the software is made available
 - New and better functionality can be imagined
- Reality is also a reference for what is needed
 - What about unreliable gadgets?
 - What about intentional attacks?

L3: The Room X3 –
unreliable external components

Recap of X2 – the Thermostat

Behavior of the simple Thermostat

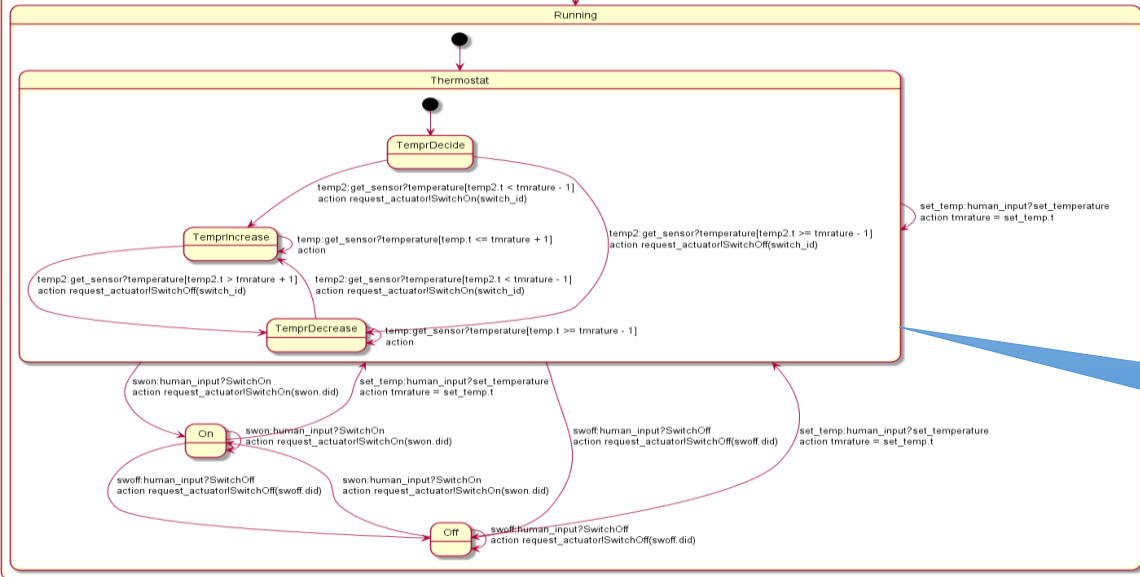




Composite states for separation of concerns

Concise description of transitions for the whole composite structure

The Room X2D – Software becoming internally robust



All possible signals covered

Optimizing actuator – apply switch only when needed

human_input?set_temperature
action human_outputprompt("INTERNAL ERROR: Impossible messages at PIM.Running")
|human_input?SwitchOn
action human_outputprompt("INTERNAL ERROR: Impossible messages at PIM.Running")
|human_input?add_device
action human_outputprompt("Adding gadgets has been done and then blocked")
|human_input?add_thermometer
action human_outputprompt("Adding gadgets has been done and then blocked")
|item.get_sensor?temperature |pollint:human_input?set_polling_interval
action request_sensorset_polling_interval[pollint.intw]

The Room X3 – guarding returns
from external sources

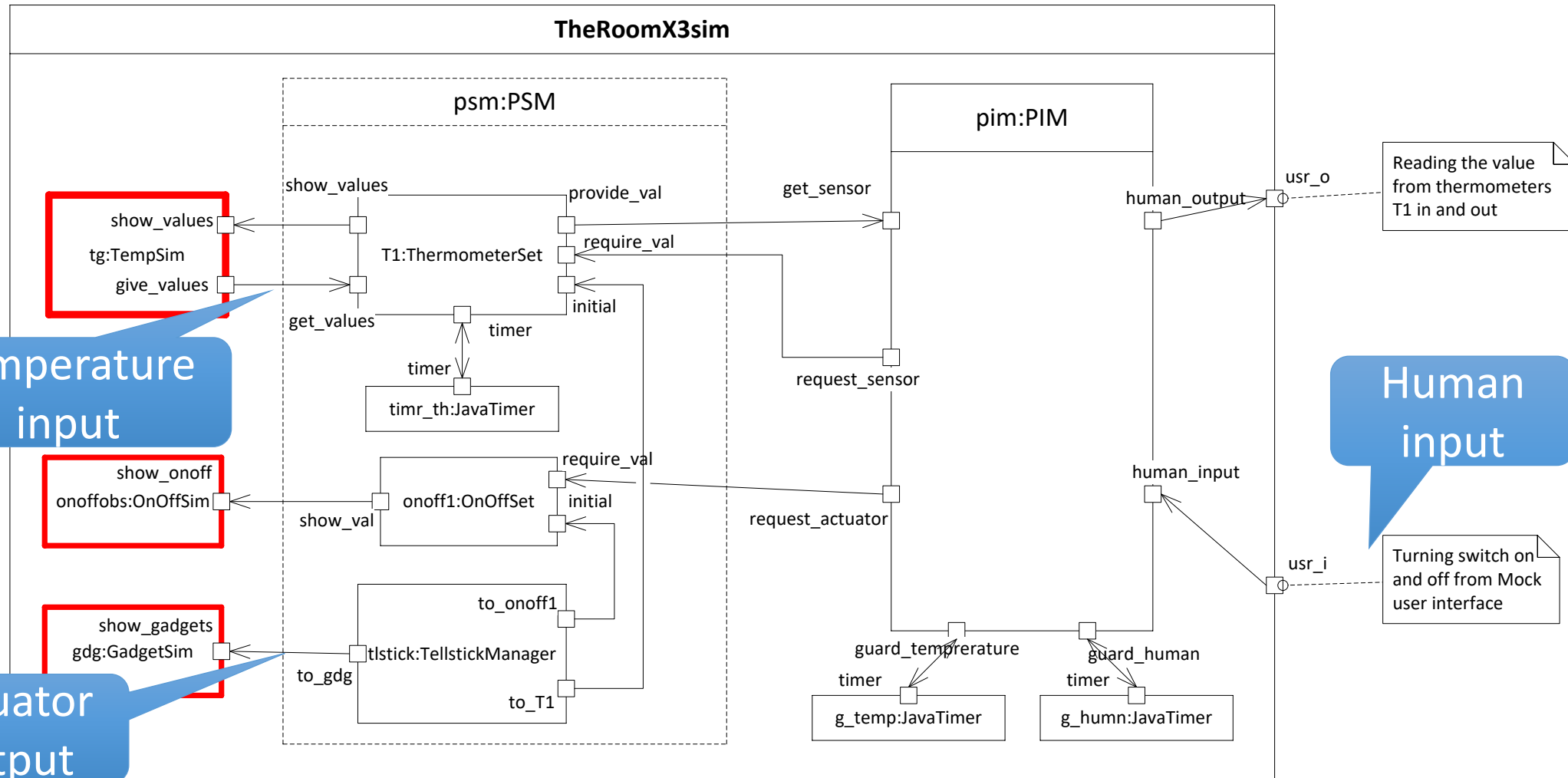
What we cover and what we do not cover in X2

- We cover
 - all possible signals in every state
 - some hardware constraint/problem: do not wear out the switches
- We do not cover
 - that externals e.g. the user by mistake fail to respond
 - that some technical gadgets may fail
 - that somebody may want to harm our system

The Room X3: The system environment

- Any real system relates to its environment
 - We cannot control the environment
 - What we can do, is to observe the environment and react to it
- One particular challenge is when the environment is expected to deliver input, and it fails to do so
- In The Room our environment consists of
 - Human user
 - Input from thermometer
 - Output to switch

The Room X3 – Simulated Environment



The Room X3: Guarding Response

- We cannot force the thermometer to send us temperatures and we cannot force the user to give the necessary input
- We observe that response is late by applying timers
 - We start a timer when we wait for a response
 - We stop the timer when we have received the expected response
- In The Room we expect
 - temperature from the thermometer (in Running)
 - building operations from the user (in Build)

Timers in ThingML (1)

```
configuration CPS {  
    ...  
    instance g_temp:TimerJava  
    instance g_humn:TimerJava  
    instance timer : TimerJava  
  
    // PSM  
    ...  
    connector T1.timer => timer.timer  
  
    // PIM  
    ...  
    connector pim.guard_temperature =>g_temp.timer  
    connector pim.guard_human => g_humn.timer  
}
```

- Soft timers in ThingML are instances of a Timer thing
- With Java object code there is a TimerJava specialization
- The timer object
 - sends timer_timeout
 - receives timer_start, timer_cancel
- The timer client (here PIM)
 - receives timer_timeout
 - sends timer_start, timer_cancel

Timers in ThingML (2)

```
thing fragment TimerMsgs {
    // Start the Timer
    message timer_start(delay : Integer);
    // Cancel the Timer
    message timer_cancel()@debug "false";
    // Notification that the timer has expired
    message timer_timeout();
}
thing fragment Timer includes TimerMsgs {
    provided port timer {
        sends timer_timeout
        receives timer_start, timer_cancel
    }
}
thing fragment TimerClient includes TimerMsgs {
    required port timer {
        receives timer_timeout
        sends timer_start, timer_cancel
    }
}
```

Timers guarding expected escapes from a state

```
required port guard_temperature {
  receives timer_timeout
  sends timer_start, timer_cancel
}

required port guard_human {
  receives timer_timeout
  sends timer_start, timer_cancel
}

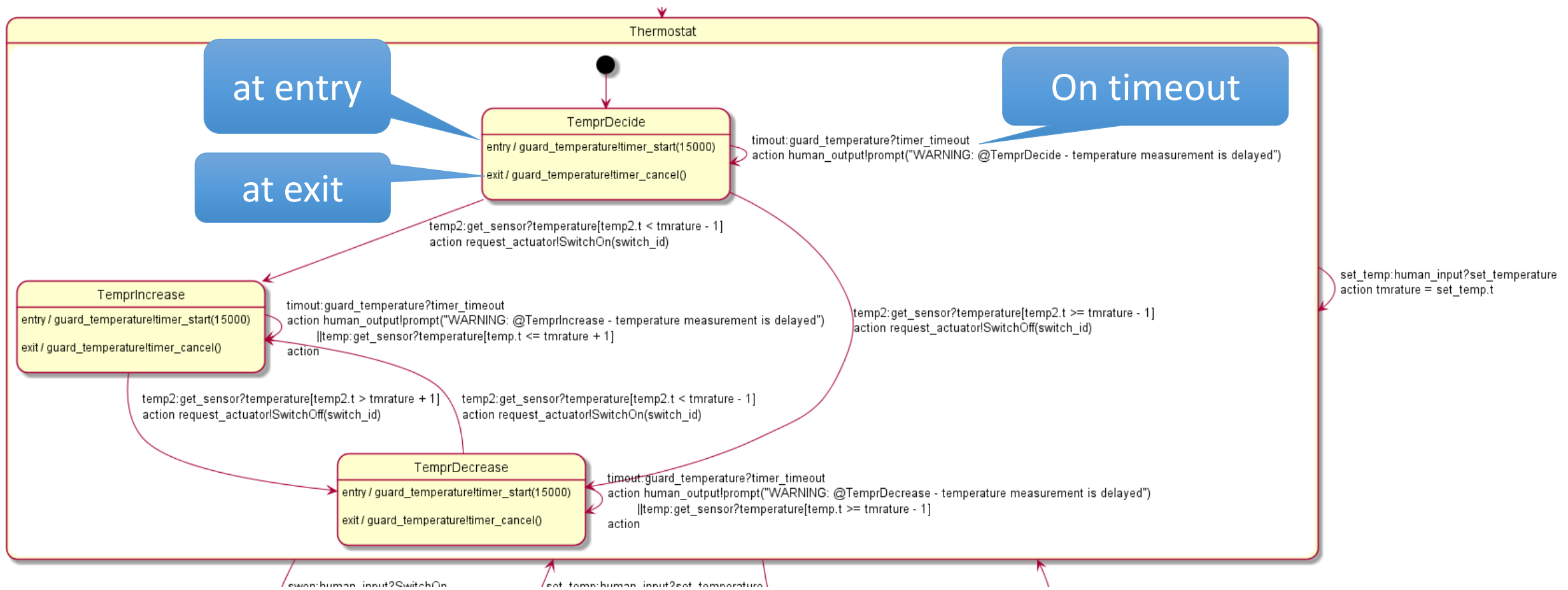
statechart PIM_behavior init Build {
  composite state Build init AddThermo keeps history {
    on entry guard_human!timer_start(30000) // 30s to do the whole build
    on exit guard_human!timer_cancel()
    ...
    transition -> Build
    event tmout:guard_human?timer_timeout
    action do
      human_output!prompt("Please continue doing the build")
    end
  } // end Build
}
```

When entering state Build,
start the timer

When exiting state Build,
cancel the timer

On timeout, perform a
recovery action

Guarding missing temperature measurements



Executing The Room X3

The screenshot shows a window titled "Human_myself" with a control panel and a log window. The control panel includes buttons for "add_thermometer", "add_device", "SwitchOn", "SwitchOff", "set_temperature", and "set_polling_interval". Each button has associated input fields for parameters like "id", "txt", "did", "short", "t", and "intrvl". A "Command line:" field contains the text "portmessage(param1, param2, param3)" and a "Send" button. The log window displays a series of messages with timestamps and prompts, such as "20 jan 2017 at 11:01:49.783: get_values?prompt (txt: Please continue doing the build of the temperature control)".

```
send_cmd!add_thermometer(txt=tt,id=1)
send_cmd!add_device(did=4)
send_cmd!set_temperature(t=21.0)
send_cmd!set_polling_interval(intrvl=25)
send_cmd!set_polling_interval(intrvl=5000)
send_cmd!set_polling_interval(intrvl=25000)
```

Command line: portmessage(param1, param2, param3) Send

20 jan 2017 at 11:01:49.783: get_values?prompt (txt: Please continue doing the build of the temperature control)
20 jan 2017 at 11:02:19.784: get_values?prompt (txt: Please continue doing the build of the temperature control)
20 jan 2017 at 11:02:37.716: get_values?prompt (txt: Please add one switch device)
20 jan 2017 at 11:02:44.061: get_values?prompt (txt: Please give thermostat temperature setting)
20 jan 2017 at 11:03:11.037: get_values?prompt (txt: Now entering thermostat. Please give temperature observations)
20 jan 2017 at 11:04:32.738: get_values?prompt (txt: WARNING: @TemprDecide - temperature measurement is delayed)
20 jan 2017 at 11:04:57.737: get_values?prompt (txt: WARNING: @TemprIncrease - temperature measurement is delayed)

Colored logs

Too slow giving human input

Temperature cycle slower than guarding timer

The Room X3B – actuator failing

Failing actuators

- The Room X3 took care of missing expected input
- The Room X3B shall look at problematic output
 - The output from The Room is on the switch
 - The communication with the switch is only one way
 - The Room controlling unit can know what the most recent signal to the switch has been, but ...
- How can we assert that the switch is on (or off)?

Is the switch on or off?

- The Room X3 only knows what is the most recent sent signal to the switch
 - The Room X3 does not know what the state of the switch is
- Solution 1: Enhance protocol with ack-signal
 - Problem: This is hardware dependent, and our switch does not have means to send signals back
- Solution 2: Observe some effects of the switch
 - Camera to observe an associated lamp
 - Observe whether expected changes in temperature actually occur

We decide to observe changes in temperature

- If we think that the switch is on, we believe that the temperature should be rising
 - We are in `TemprIncreasing` state
- If we think that the switch is off, we believe that the temperature should be falling
 - We are in `TemprDecreasing` state

Our simple discover and recover strategy

- Observe development of temperature, rising or falling
- If in state *TemprIncreasing* and temperature is falling, we try and switch ON again
- If in state *TemprDecreasing* and temperature is rising, we try and switch OFF again
- If in state *ON* and temperature is falling, we try and switch ON again
- If in state *OFF* and temperature is rising, we try and switch OFF again

TemprIncrease

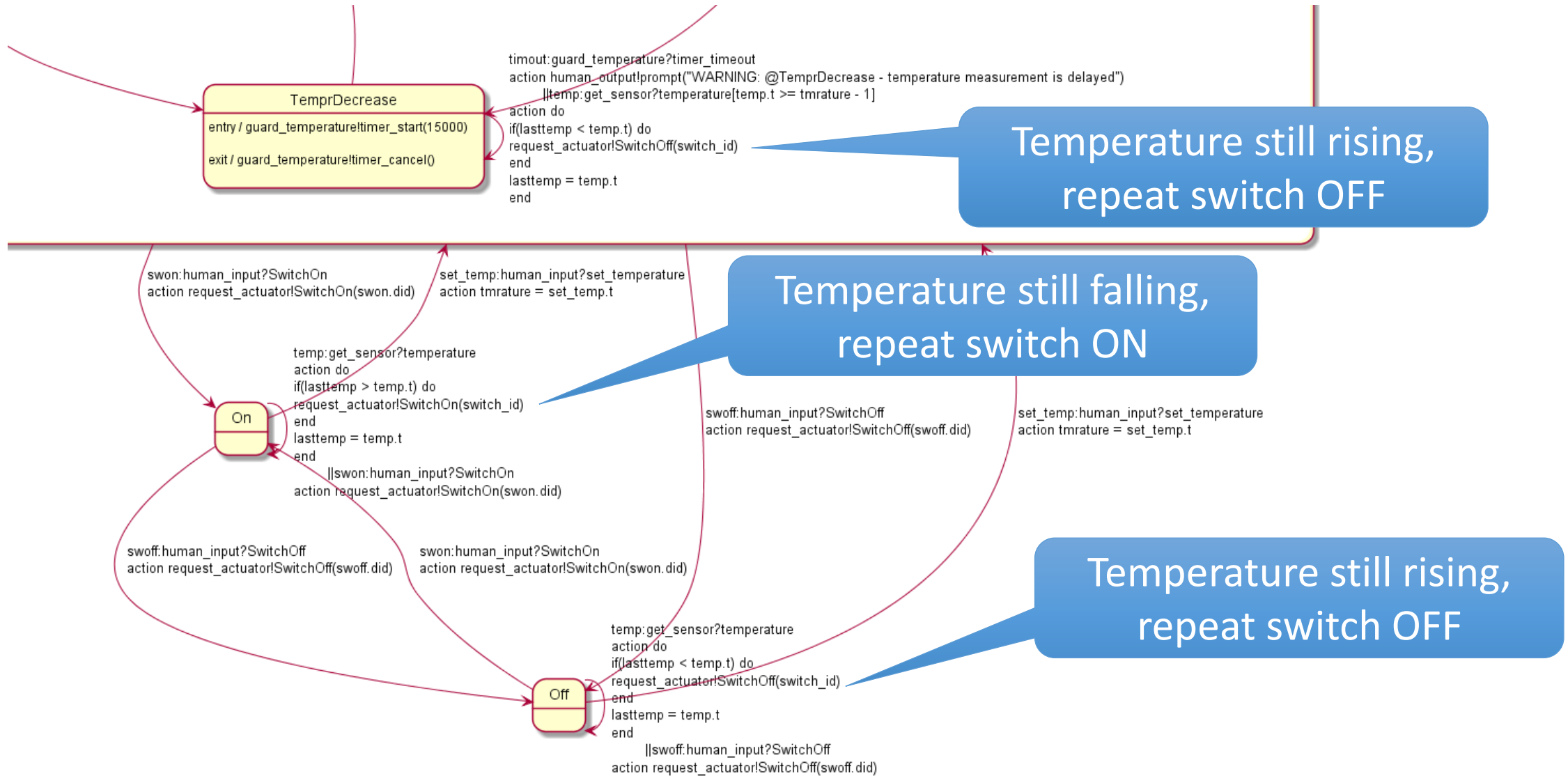
```
state TemprIncrease{ // Invariant: Switch is ON and temperature should increase
on entry guard_temperature!timer_start(15000)
on exit guard_temperature!timer_cancel()
  transition -> TemprIncrease
  event temp:get_sensor?temperature
  guard temp.t<=tmrature+1
  action do
    if (lasttemp>temp.t) request_actuator!SwitchOn(switch_id)
    // the temperature is still falling even though switch should be ON, reactivate
    lasttemp = temp.t
  end

  transition -> TemprDecrease
  event temp2:get_sensor?temperature
  guard temp2.t>tmrature+1
  action do
    request_actuator!SwitchOff(switch_id)
    lasttemp = temp2.t
  end

  transition -> TemprIncrease
  event timeout:guard_temperature?timer_timeout
  action do
    human_output!prompt("WARNING: @TemprIncrease - temperature measurement is delayed")
  end
end
}
```

Temperature still falling,
repeat switch ON

and graphically



The Room X3 – implications of a challenging environment

- X3 guards the expected inputs with timers
- X3 guards the expected results of output with clever observation and recovery
- X3 has modifications that are due to the challenging environment
 - but to test X3 it is a lot easier to execute the simulated version!
- X3 in reality would have to
 - manipulate thermometers e.g. by removing batteries
 - manipulate switches e.g. by physically altering them