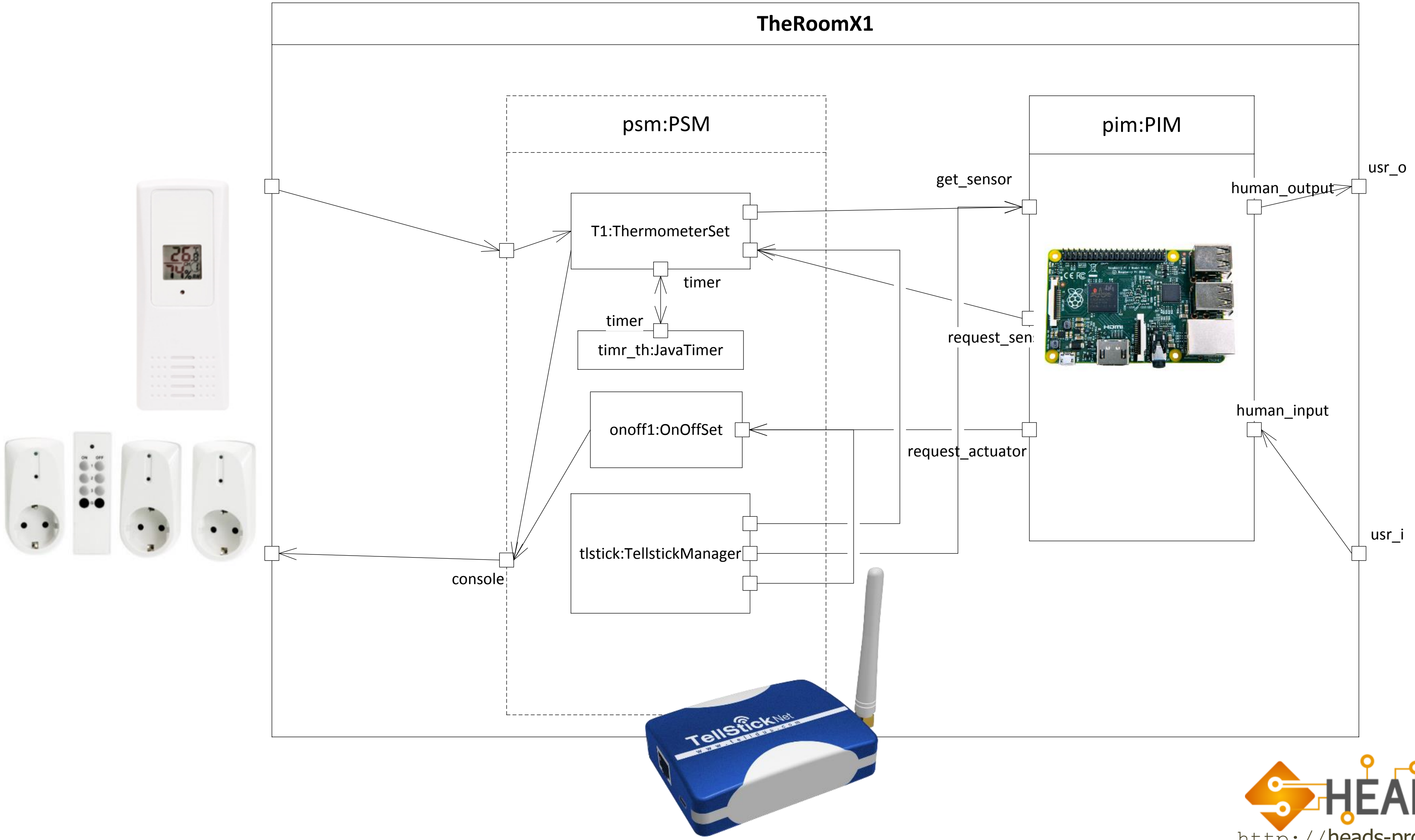
Heterogenous and Distributed Services for the Future Computing Continuum

L2: The Room X2 with Thermostat

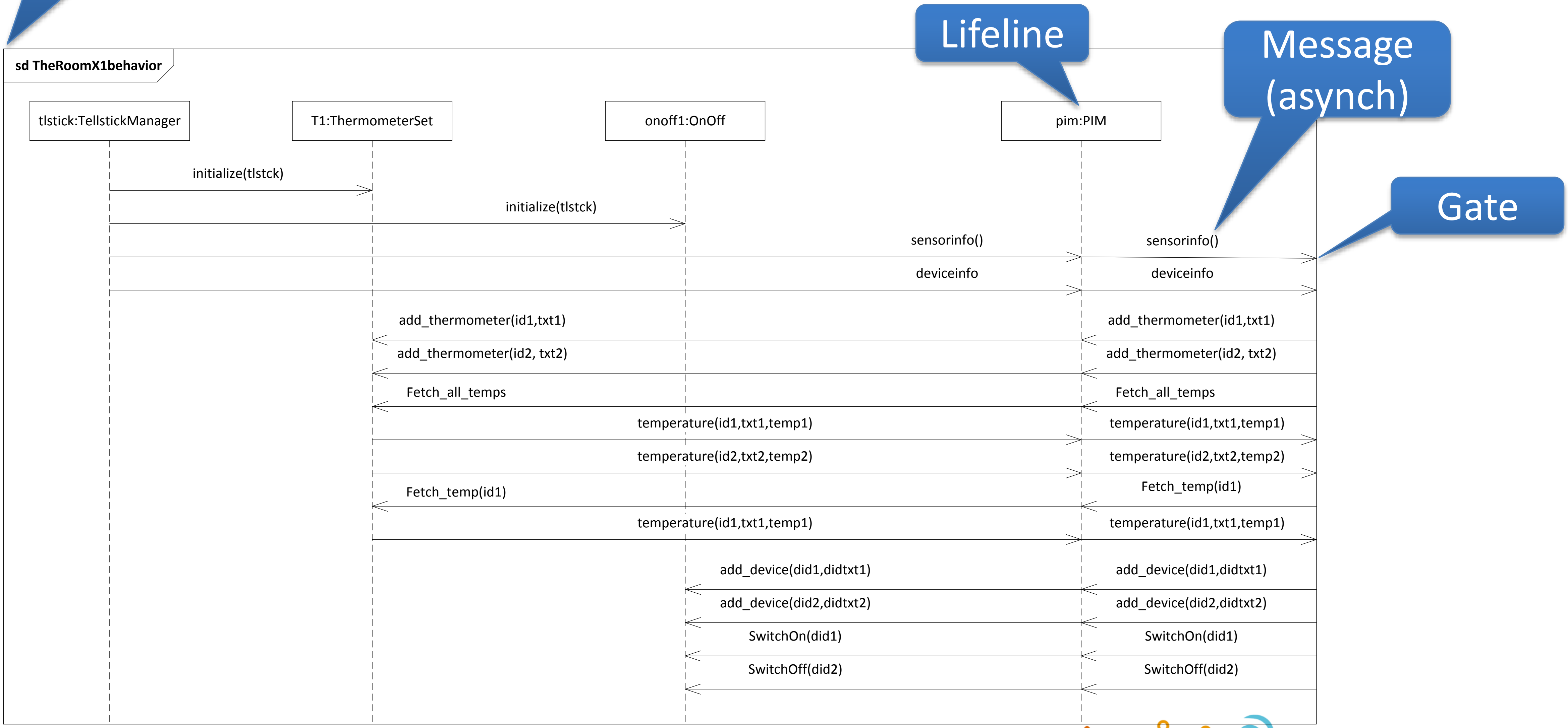
X1: Recap The Room

In one picture

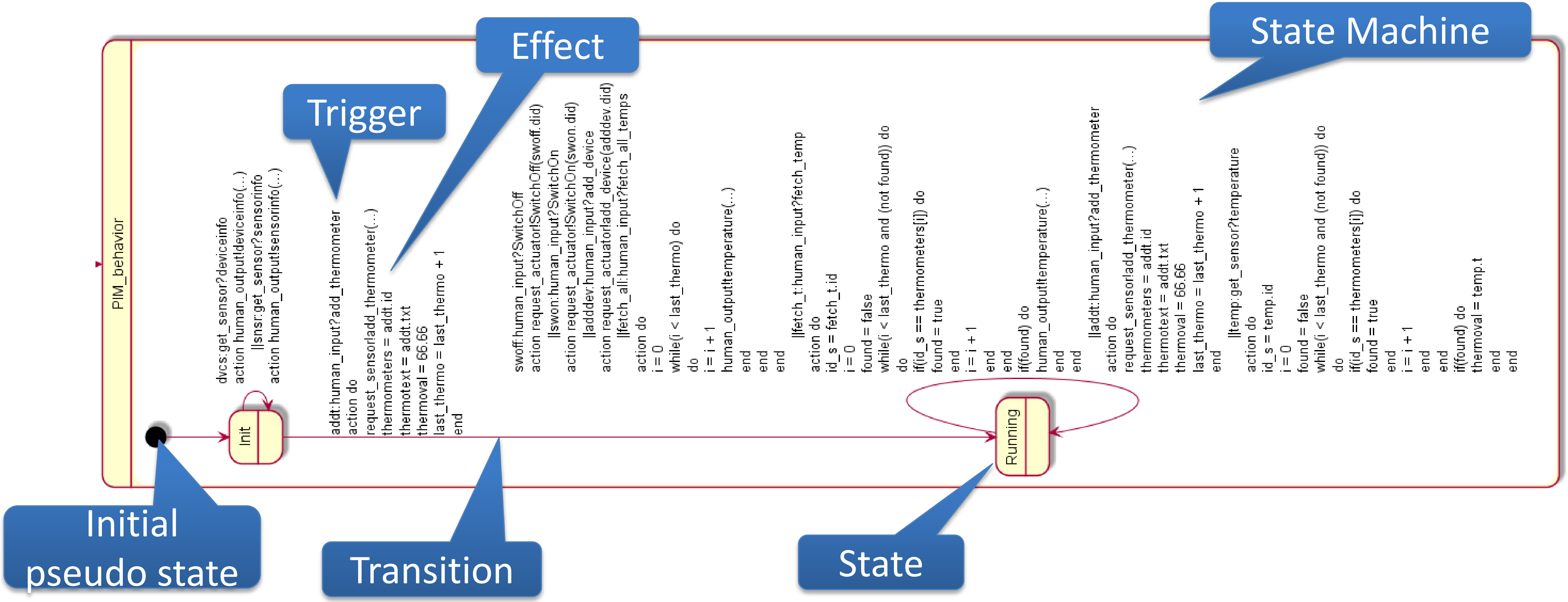


he Room X1 Behavior

Sequence Diagram



The Room X1 – PIM state machine visualized



http://heads-project.eu

The Room X2 – PIM in text (1)

Initial pseudo state

Transition

Trigger

State Machine

Effect

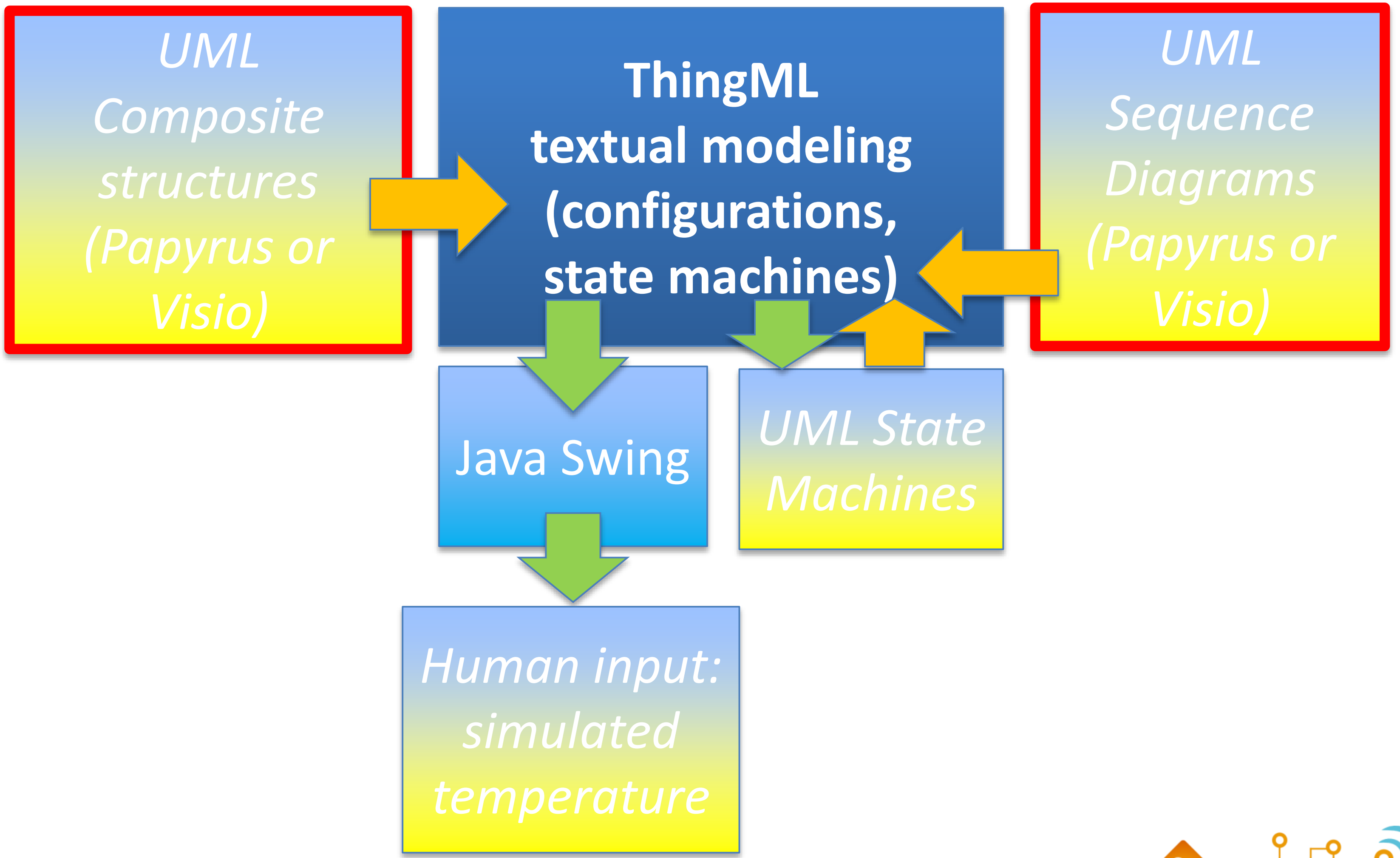
State

Message

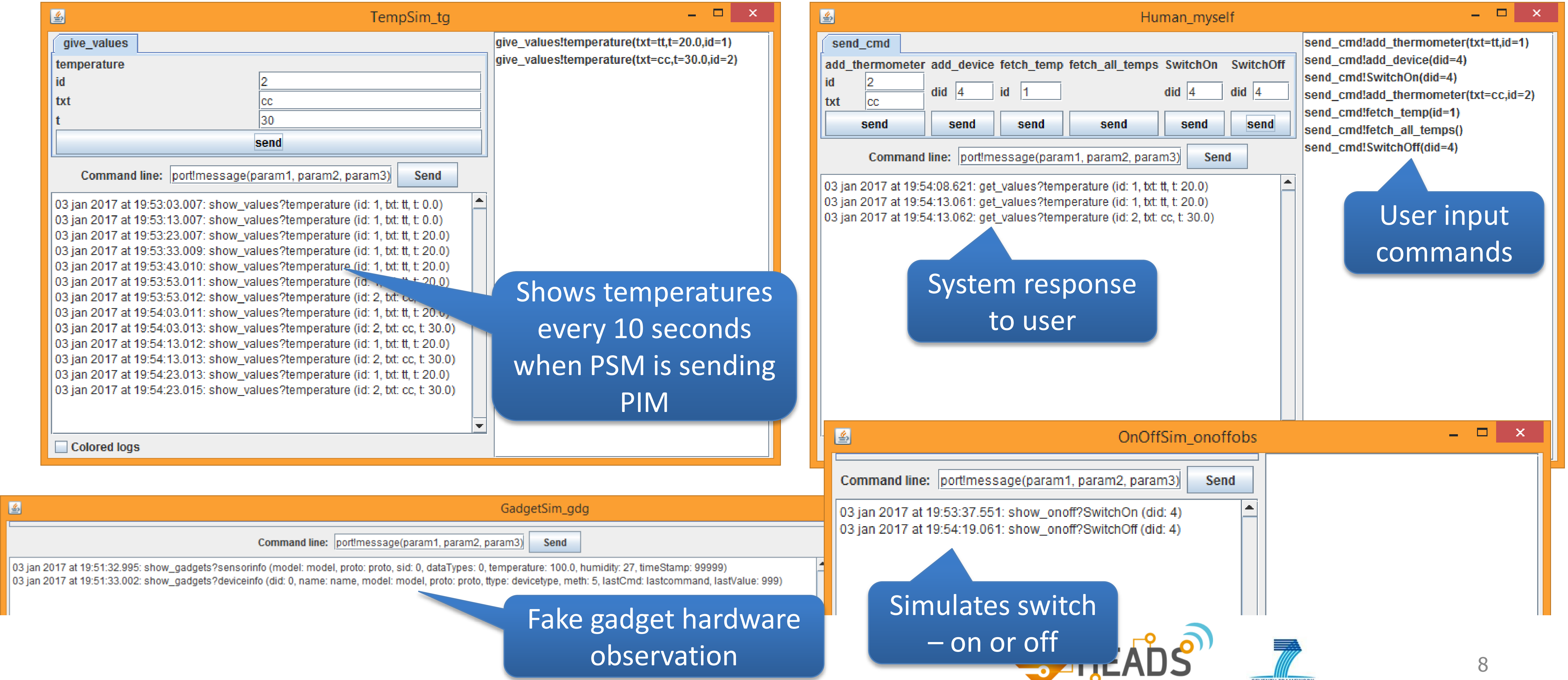
Port

```
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
	}
```

Simulating the laboratory



The Room X1 – A simulated execution



Heterogenous and Distributed Services for the Future Computing Continuum

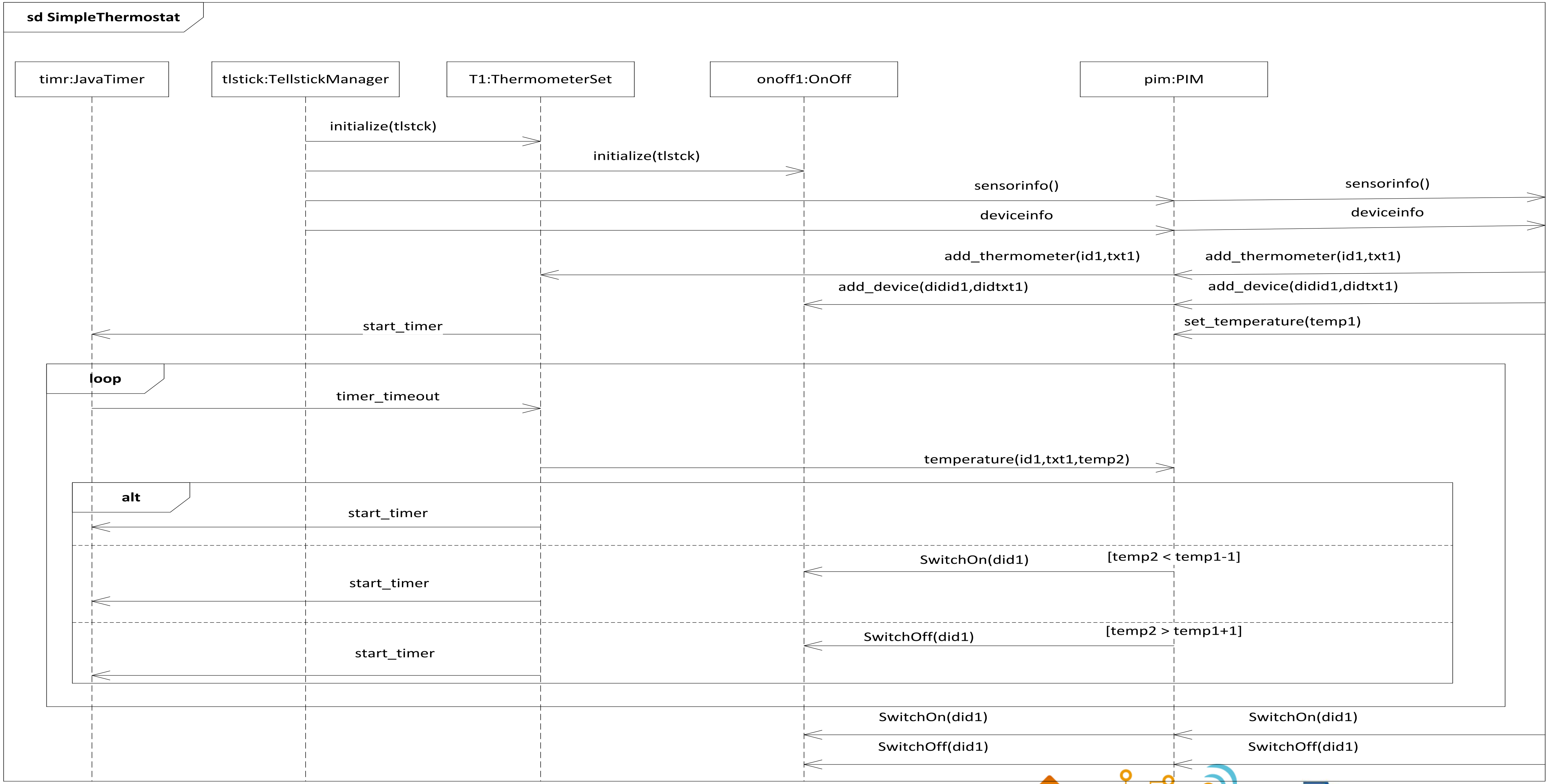
X2A: The first thermostat

SEVENTH FRAMEWORK PROGRAMME

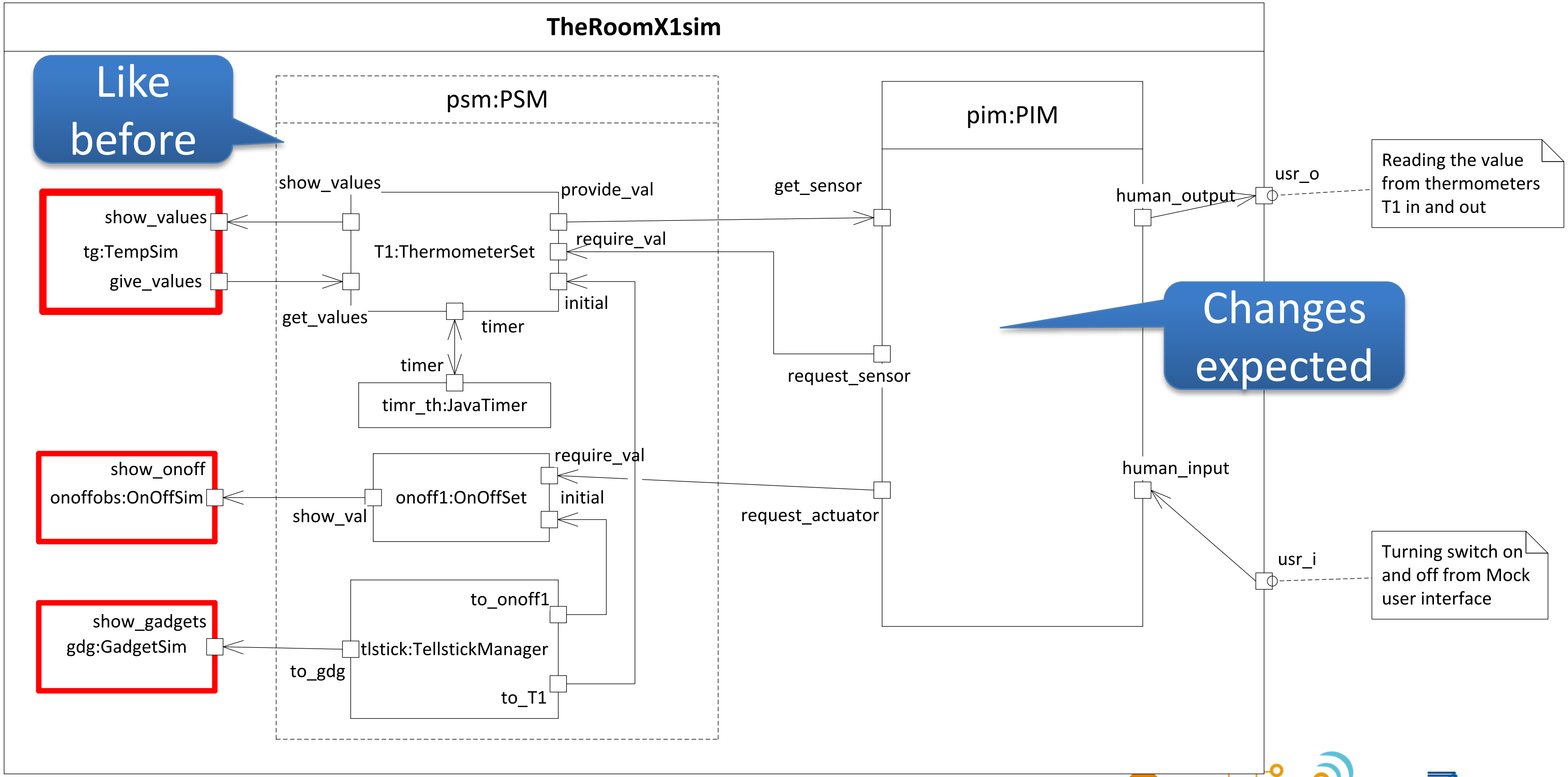
X2A: The Room with a simple Thermostat

- 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

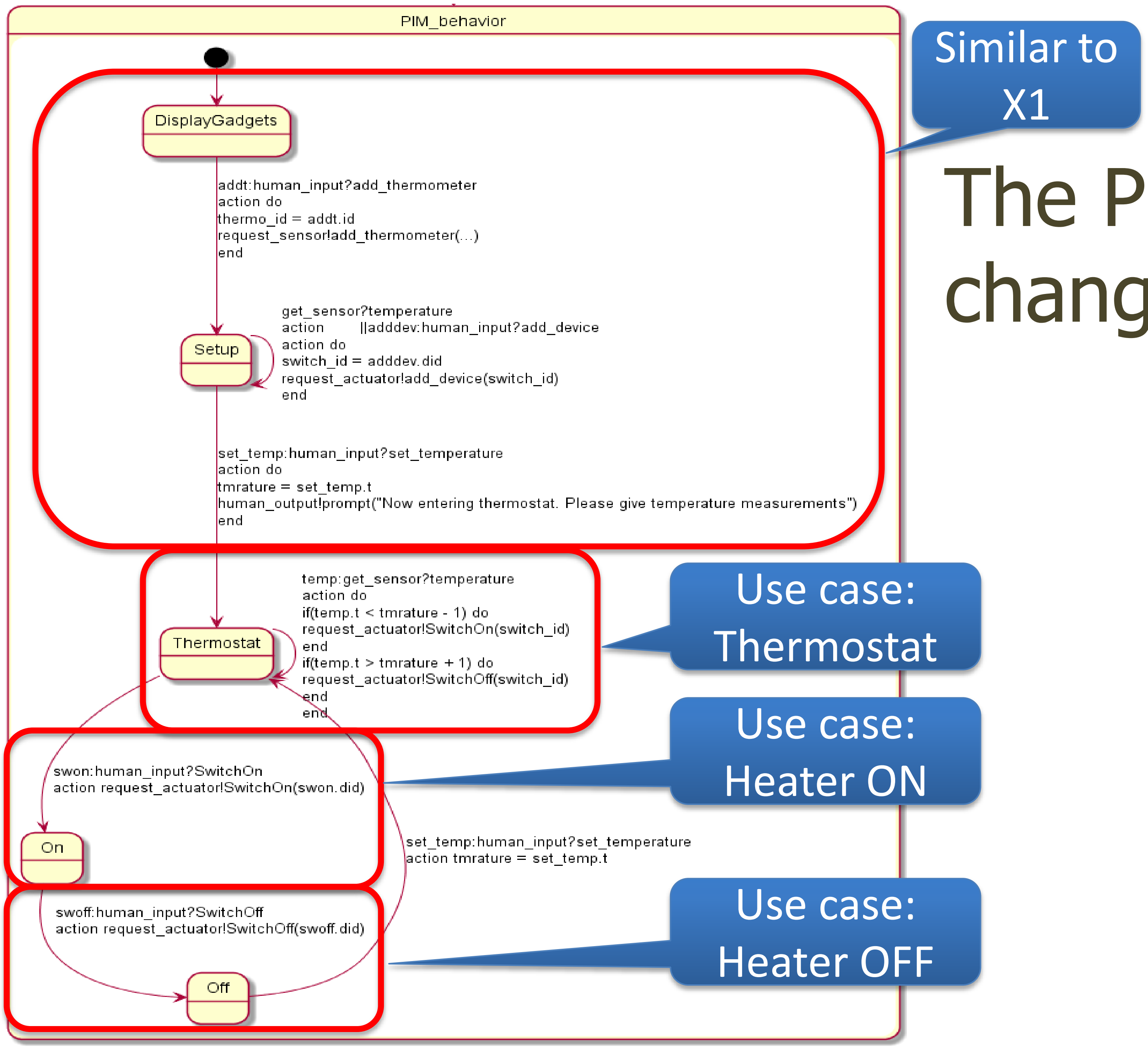


The Room X2 – Simulation architecture as X1



http://heads-project.eu

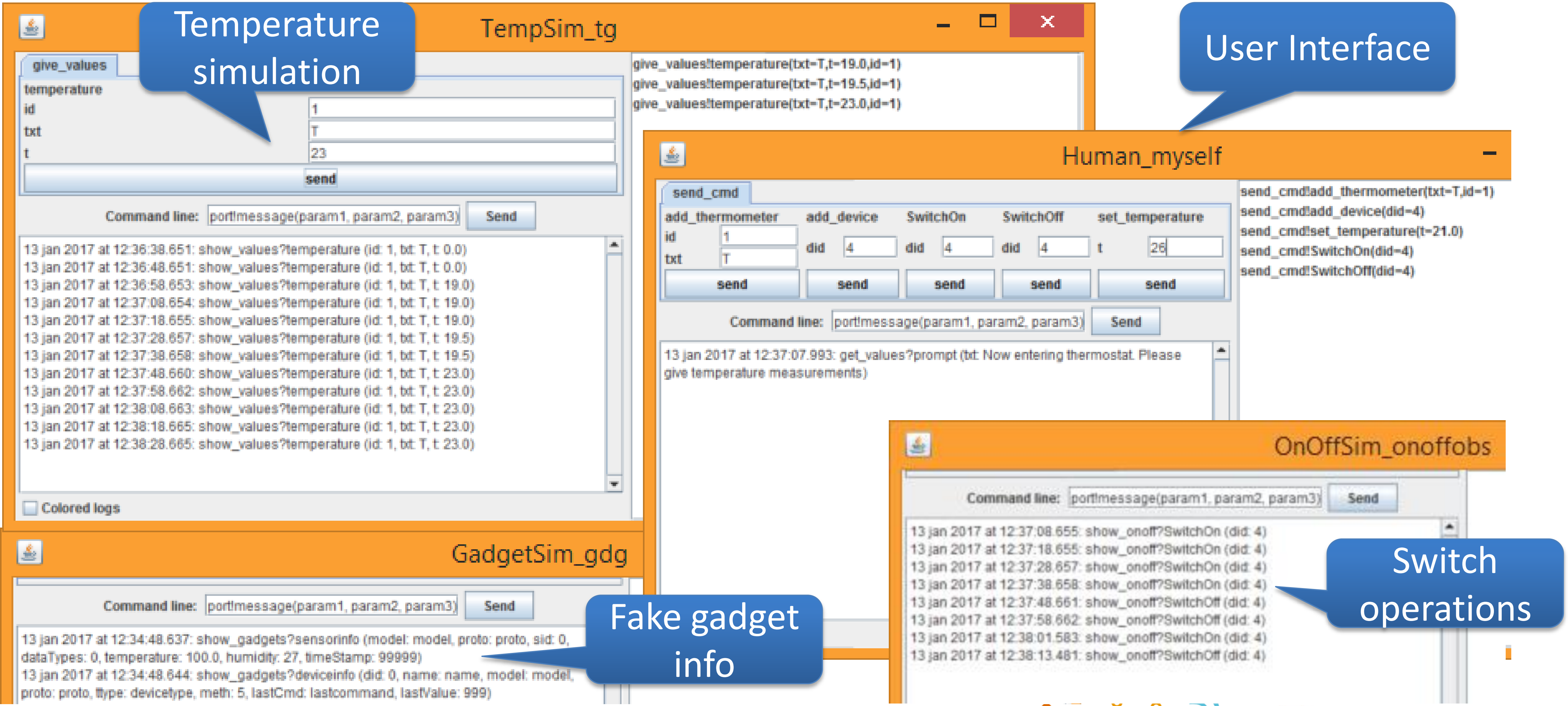
The PIM behavior now changes



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)



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

Heterogenous and Distributed Services for the Future Computing Continuum

X2B: Composite States

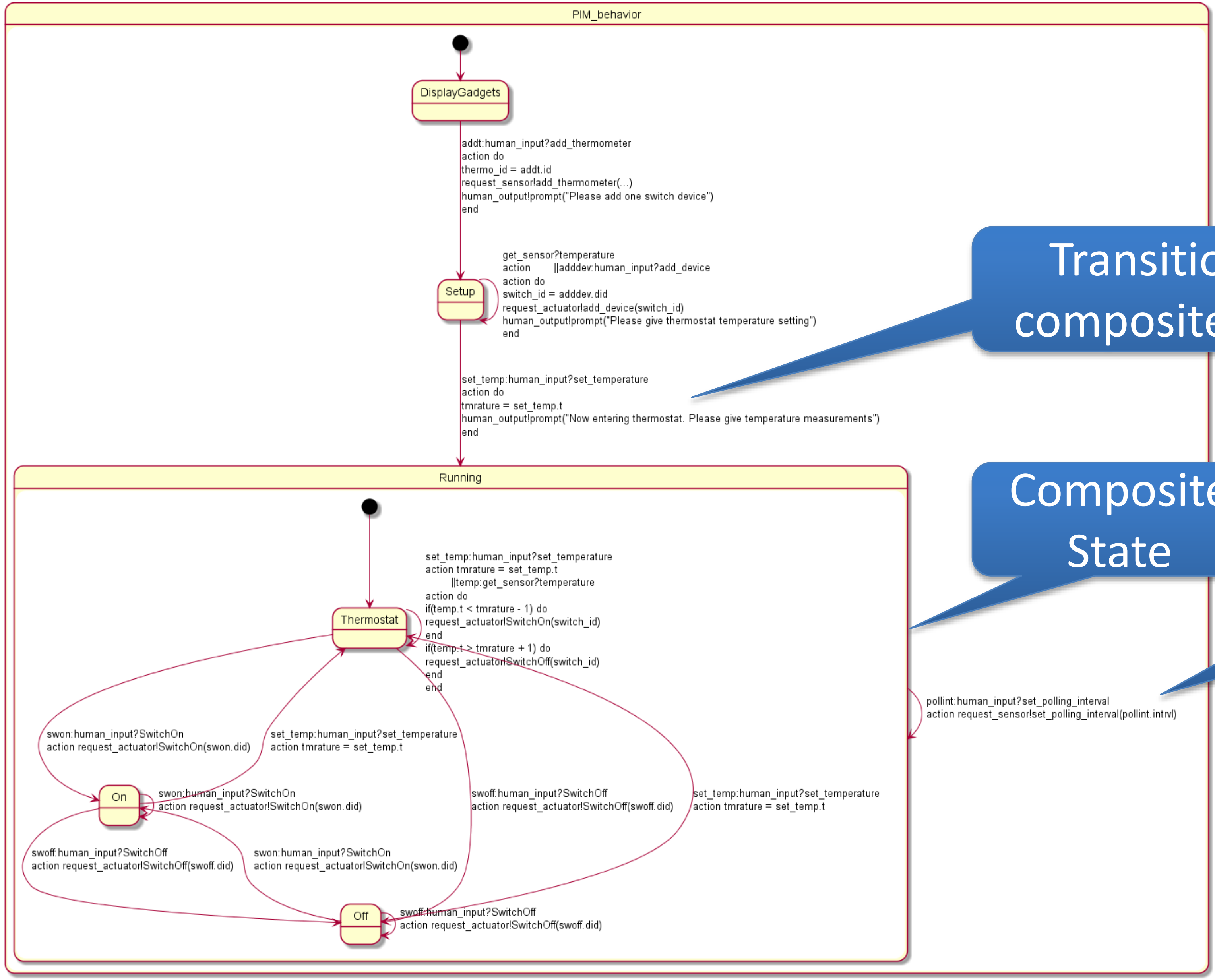
X2B: The Room with composite state

- 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

Note: **keeps history** (not shown in UML diagram)

Composite State

Transition on composite state

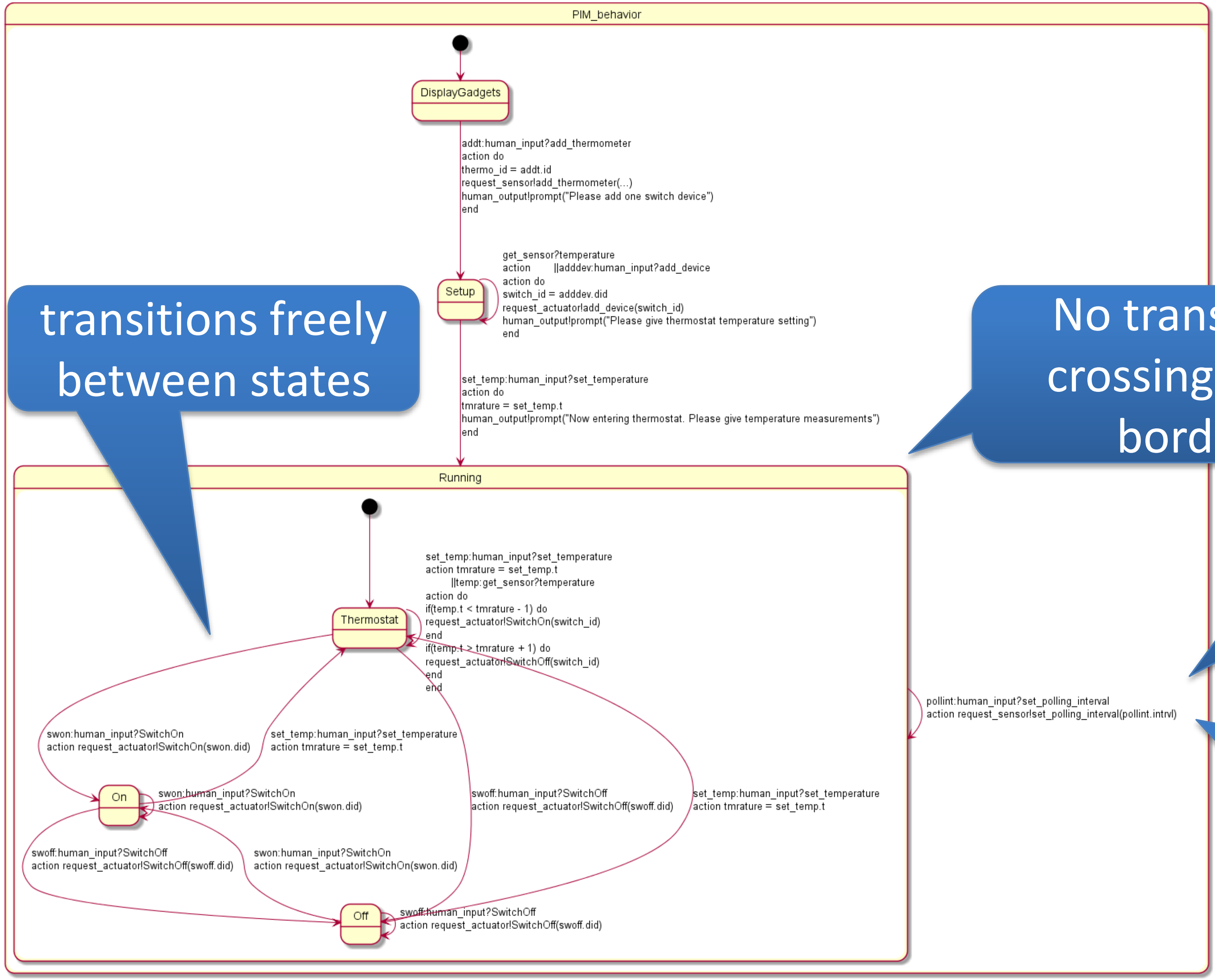
The Semantics of a Composite State

- 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

Heterogenous and Distributed Services for the Future Computing Continuum

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

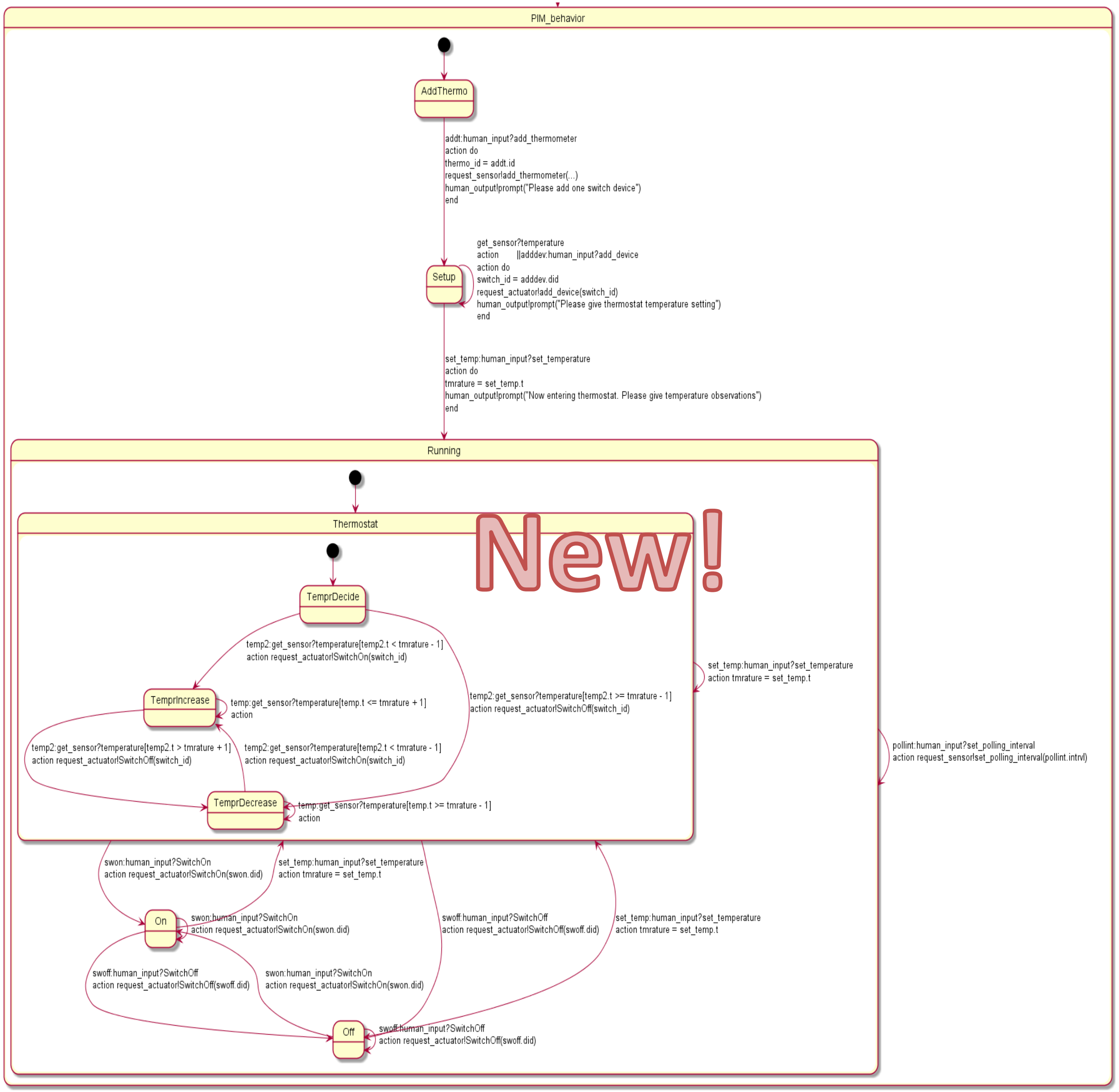
Goal: Reduce or remove the redundant switching

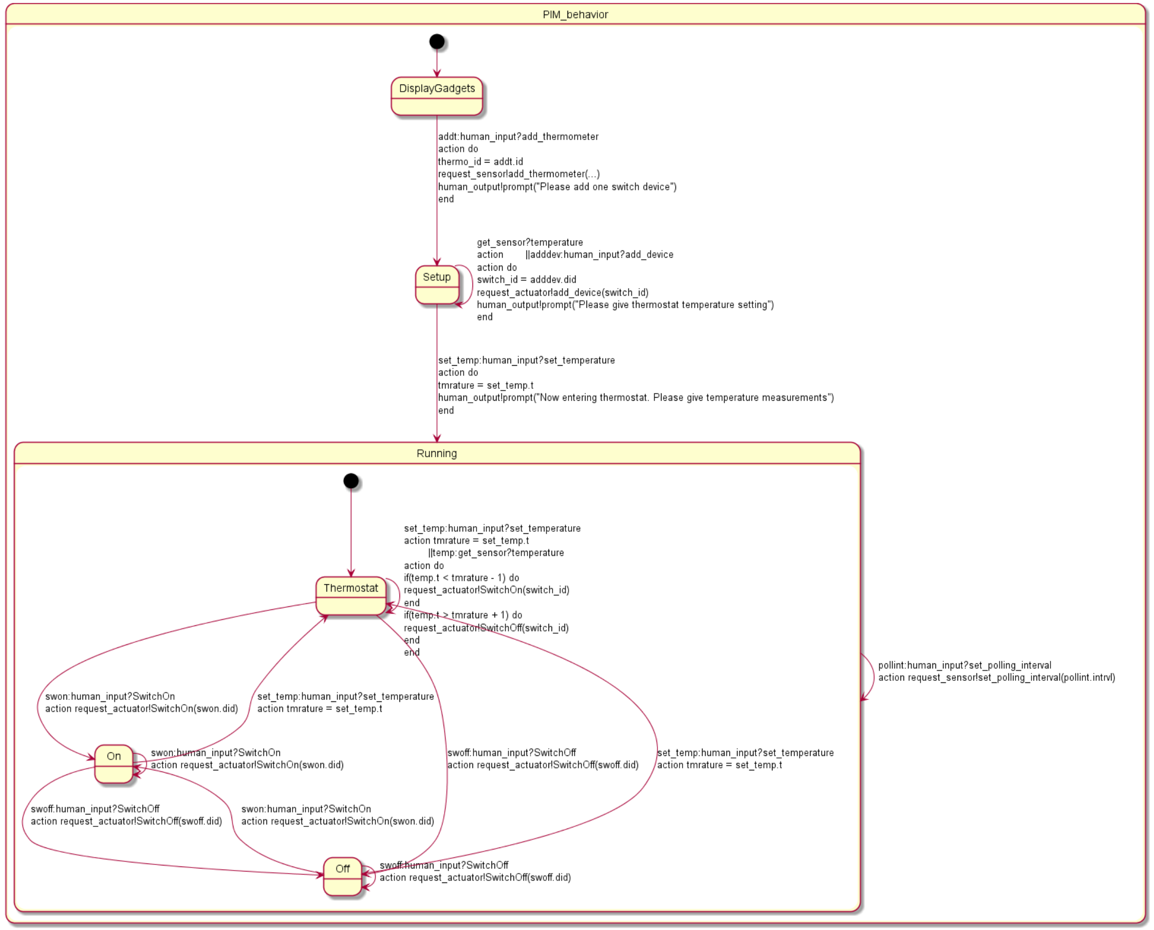
- We want to reduce or remove unnecessary application of the switches due to the risk of wearing the switches out prematurely
- Switching to ON is unnecessary if it is already ON
  - and the temperature should be increasing
- Switching to OFF is unnecessary if it is already OFF
  - and the temperature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

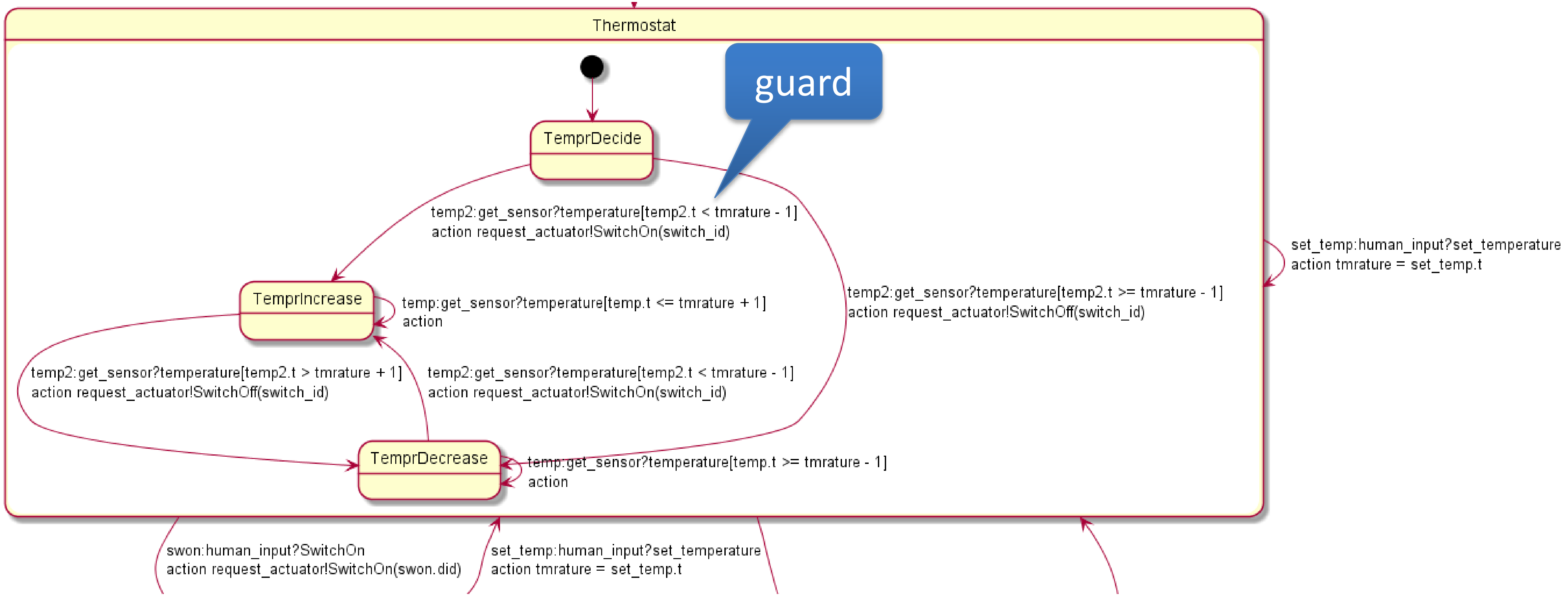




Our solution

- 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

guard

```
composite state Thermostat init TemprDecide {
// notice that we are NOT keeping history
      state TemprDecide {
            transition -> TemprDecrease
             vent temp2:get_sensor?temperature
              rd temp2.t>=tmrature-1   // OFF as much possible
              tion 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
      }
```

```
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?

Heterogenous and Distributed
Services for the Future Computing Continuum

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?

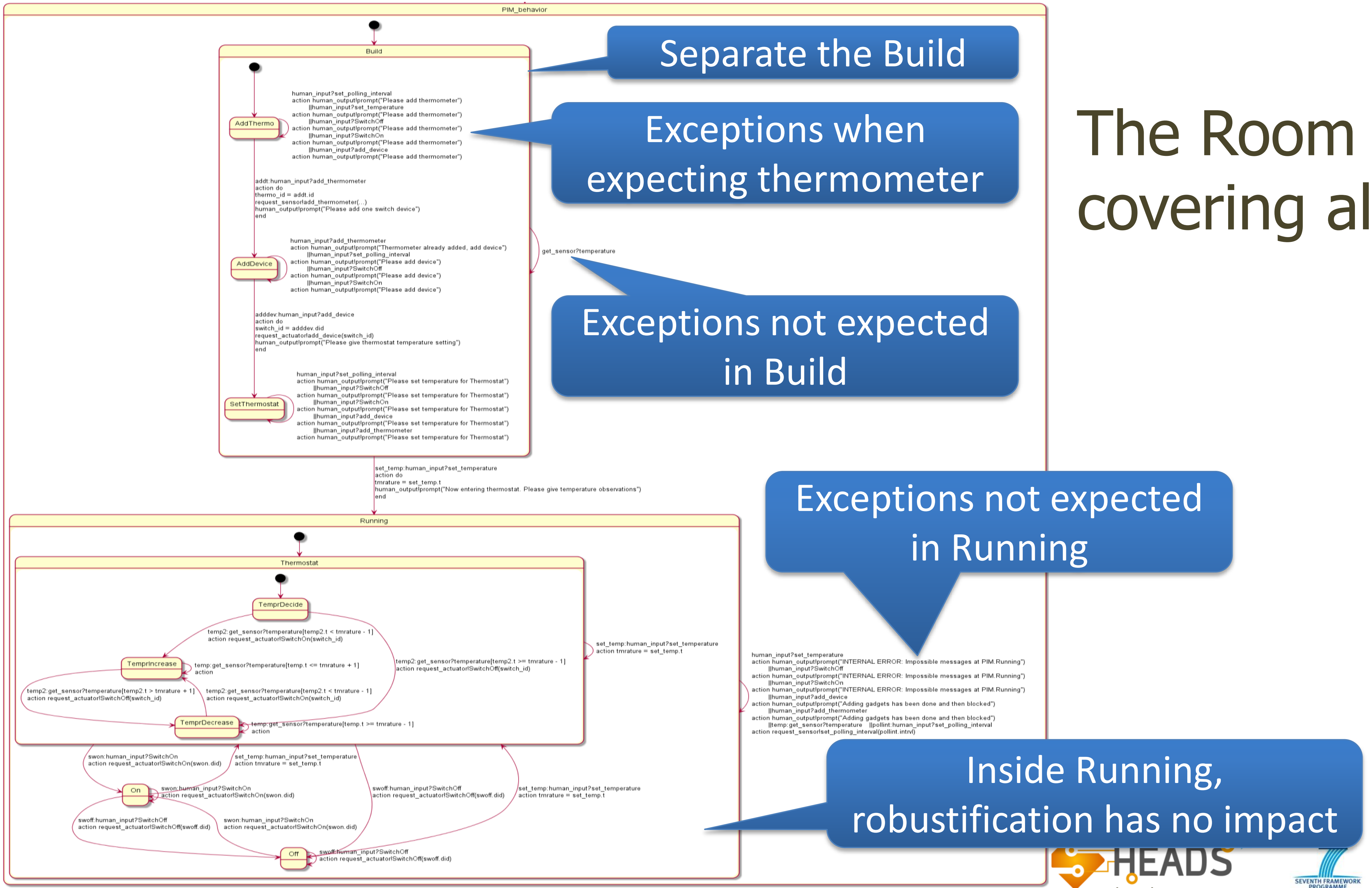
The Beauty of State Machines

- 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

The Room X2D – covering all signals



The unexpected in ThingML (1)

```
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
    }
```

Separate the Build

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

http://heads-project.eu

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?

Consortium

http://heads-project.eu