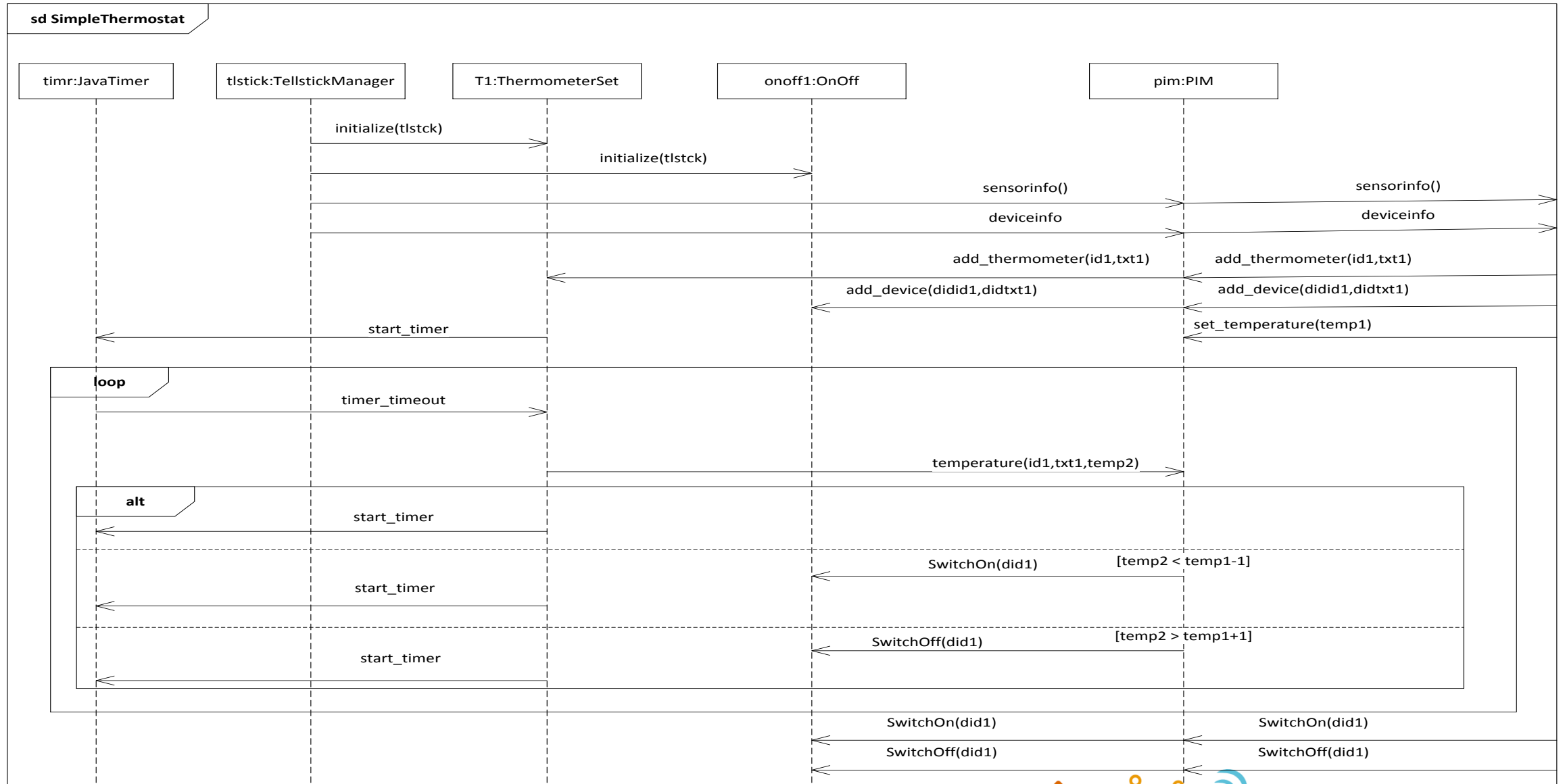


## L3: The Room X3 – unreliable external components

# Recap of X2 – the Thermostat

# Behavior of the simple Thermostat



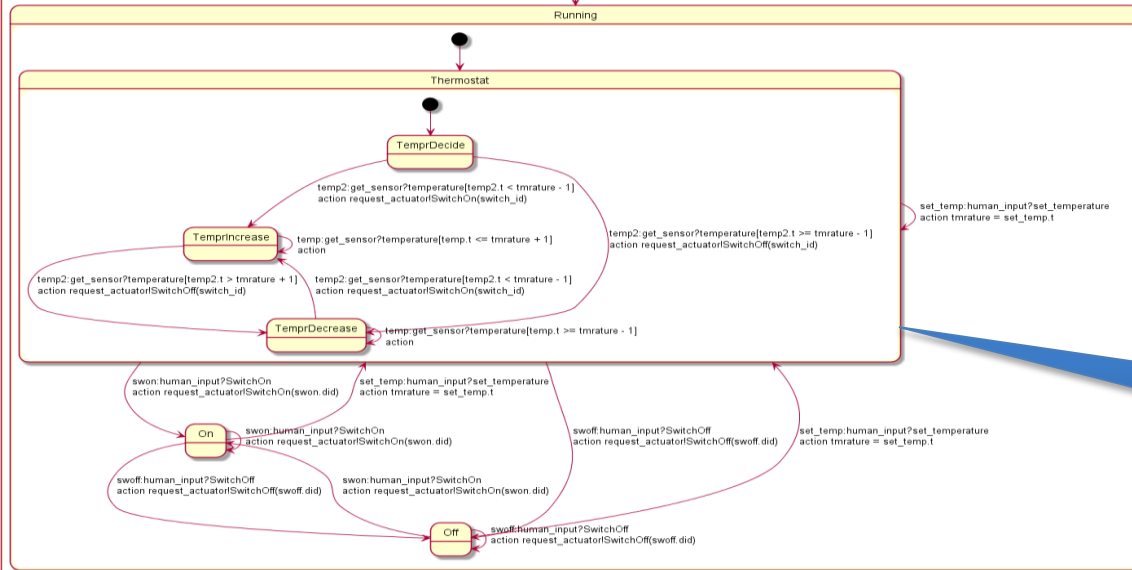
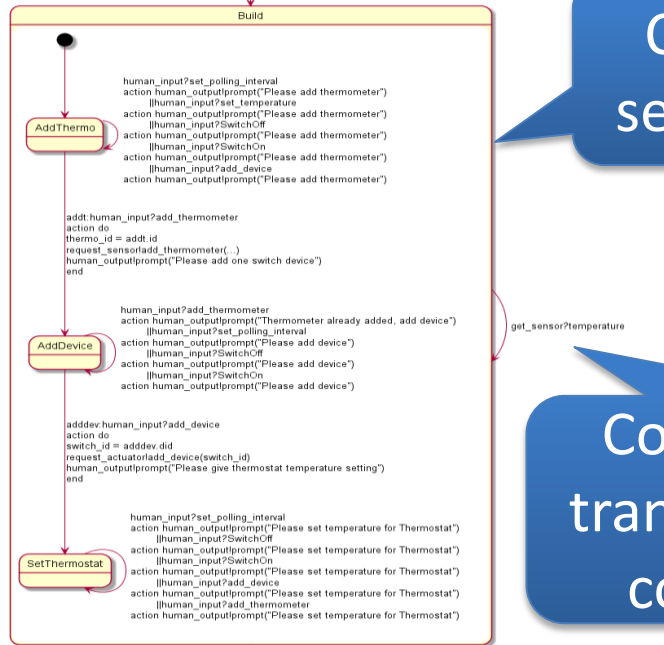
# The Room X2D – Software becoming internally robust

Composite states for  
separation of concerns

Concise description of  
transitions for the whole  
composite structure

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")  
[temp\_get\_sensor?temperature [pollint:human\_input?set\_polling\_interval  
action request\_sensor?set\_polling\_interval[pollint:intrv]

# The Room X3 – guarding returns from external sources

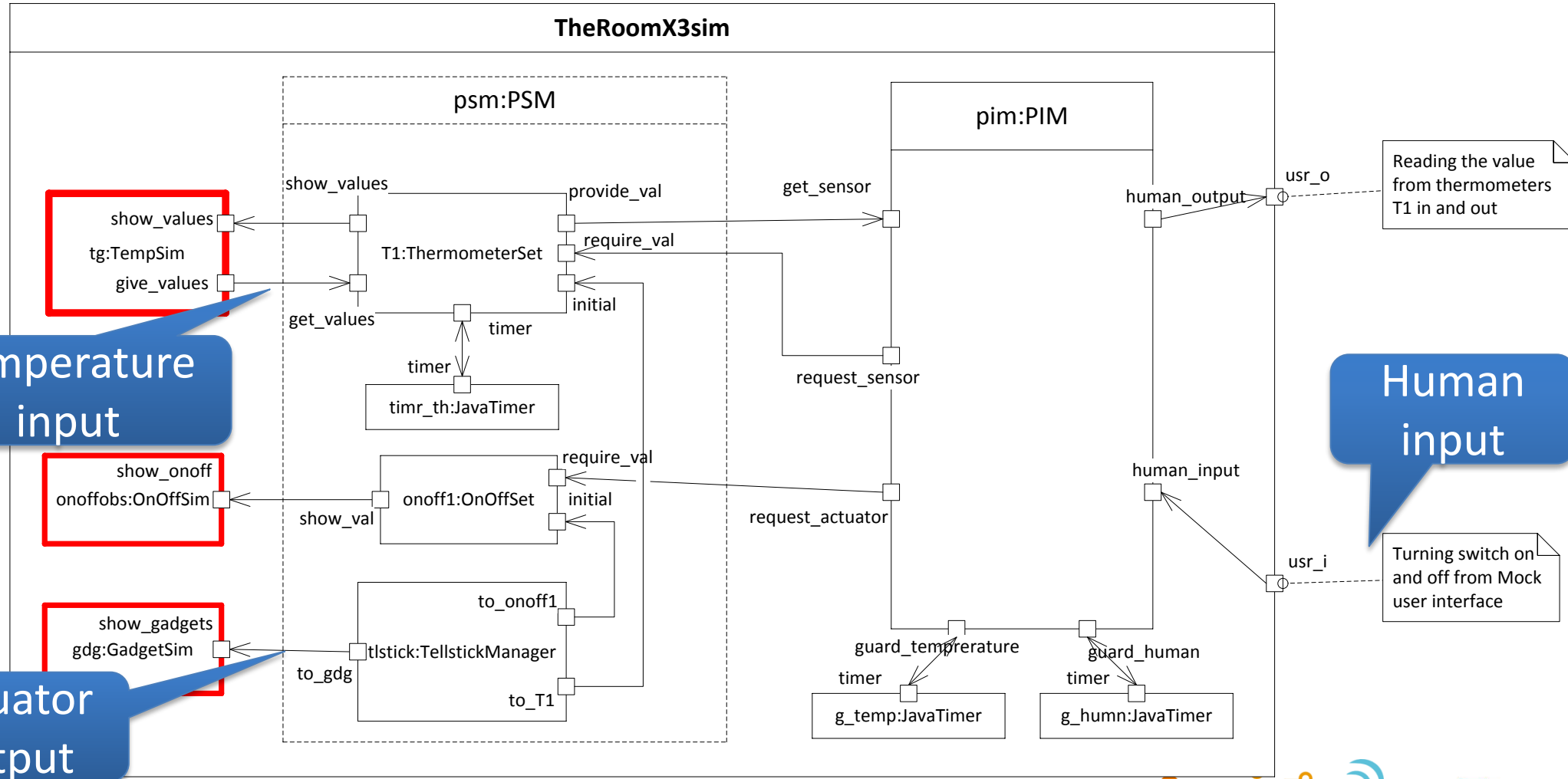
# What we cover and what we do not cover in X2

- 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
}

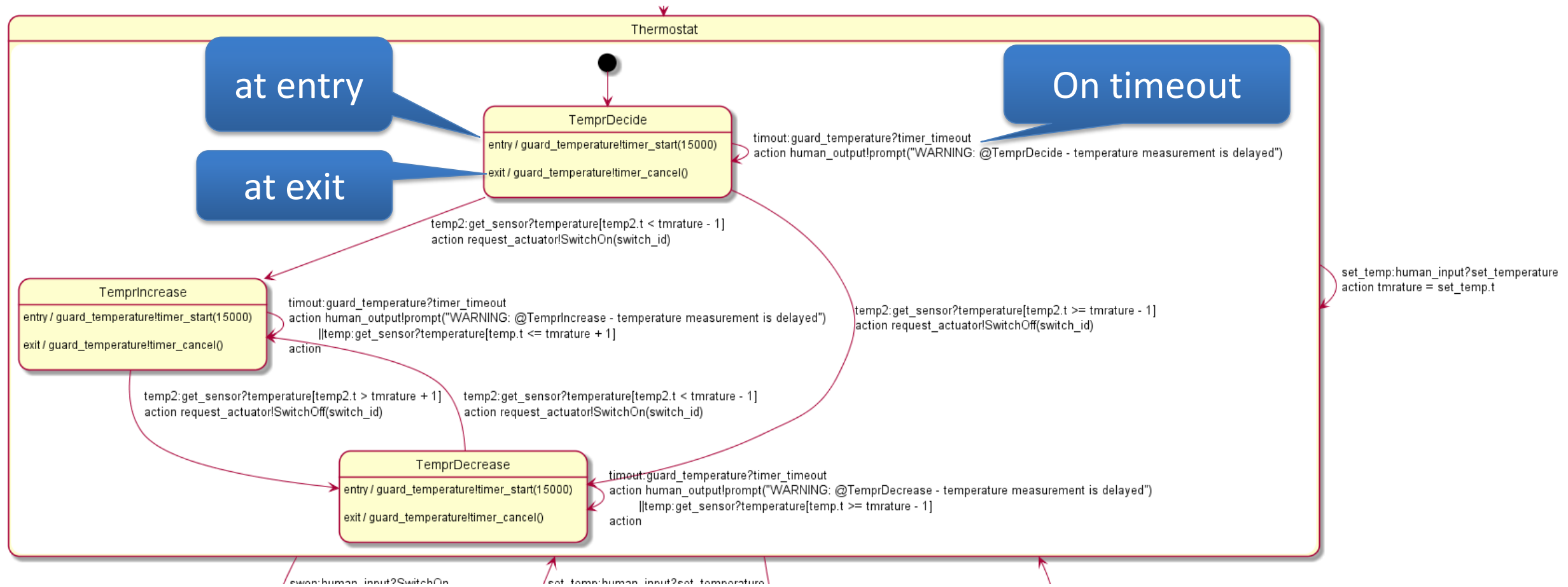
statechart PIM_behavior init Build {
  composite state Build init AddThermo keeps history {
    on entry guard_human!timer_start(30000) // 30s to do the whole build
    on exit guard_human!timer_cancel()
    ...
    transition -> Build
    event tmout:guard_human?timer_timeout
    action do
      human_output!prompt("Please 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 tab labeled "send\_cmd". The interface includes several input fields and buttons for sending commands:

- add\_thermometer:** id: 1, txt: tt, send button
- add\_device:** did: 4, send button
- SwitchOn:** did: short, send button
- SwitchOff:** did: short, send button
- set\_temperature:** t: 21, send button
- set\_polling\_interval:** intrvl: 25000, send button

Below these is a "Command line:" field with the text "portmessage(param1, param2, param3)" and a "Send" button.

The log area at the bottom displays the following messages:

```
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)
```

On the right side of the window, a list of commands is shown:

```
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)
```

Two blue callout boxes point to the log area:

- "Too slow giving human input" points to the first two log entries.
- "Temperature cycle slower than guarding timer" points to the last two log entries.

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

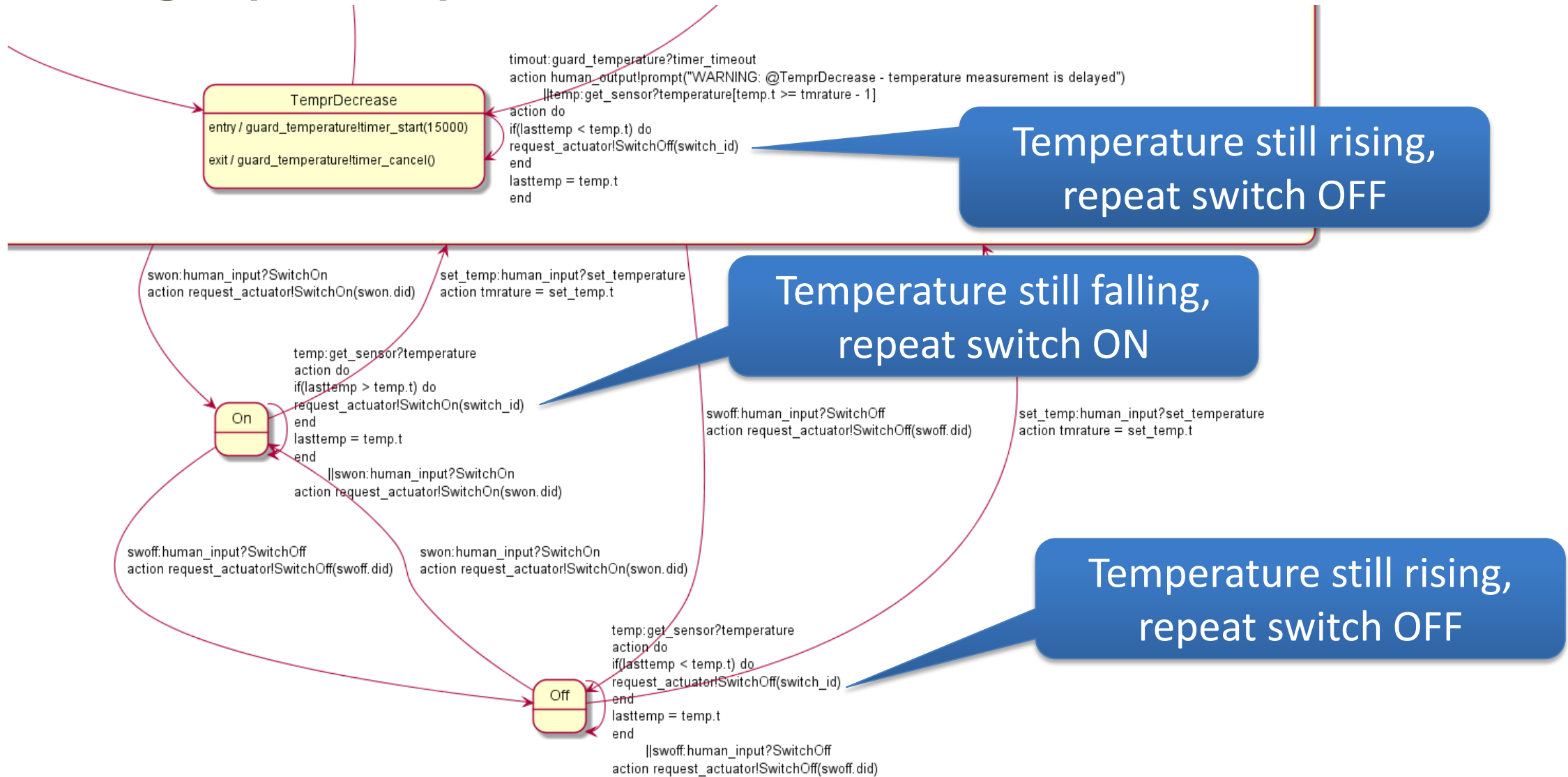
```
state TemprIncrease{ // Invariant: Switch is ON and temperature should increase
on entry guard_temperature!timer_start(15000)
on exit guard_temperature!timer_cancel()
  transition -> TemprIncrease
  event temp:get_sensor?temperature
  guard temp.t<=tmrature+1
  action do
    if (lasttemp>temp.t) 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
}
```

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

# Consortium

