Petter Sandvik

Formal Stepwise Development of an In-House Lighting Control System

Turku Centre *for* Computer Science

TUCS Technical Report
No 1079, May 2013

Formal Stepwise Development of an In-House Lighting Control System

Petter Sandvik
Åbo Akademi University, Department of Information Technologies
Joukahaisenkatu 3-5 A, 20520 Turku, Finland
TUCS – Turku Centre for Computer Science
Joukahaisenkatu 3-5 A, 20520 Turku, Finland
petter.sandvik@abo.fi

TUCS Technical Report
No 1079, May 2013

Abstract

Automated control systems have become increasingly popular. These types of systems can be very complex, and in order for us to be able to understand and trust them, we need formally derived and verified models. In this paper, we utilise the Event-B formalism to stepwise model a system of automatic control of in-house lighting, with the aim of possible integration with other control systems.

TUCS Laboratory
Distributed Systems Laboratory

1 Introduction

Automatically controlled in-house lighting systems have become popular in many places. These systems, which can turn on or off lights based on factors such as the presence or absence of motion and the ambient light level, can be used to save energy and money by turning off lights that are deemed unnecessary for the moment as well as only turning on those lights that are deemed necessary. In this paper, we describe the modelling of a lighting control system in an adaptive house. For this purpose, we use Event-B [2] and the Rodin Platform [3, 8]. We model the system from an abstract specification, describing what the system should do, stepwise towards a more concrete model, describing how the system should do what it is supposed to. We aim to model our system in such a way that we later will be able to integrate it with other adaptive house control systems [12].

In order for our model of a lighting control system to be feasible, we must make some assumptions of the system we try to model. In this case, we assume that we have a house partitioned into areas called rooms, which do not necessarily coincide with the physical rooms but rather define discrete spaces in which all lights are switched on or off together. We also assume that our adaptive house contains two different types of sensors that can be used for controlling the lighting. Sensors of the first type, which we will refer to as *ambient light sensors*, typically base their sensor states on the ambient light level being low enough or high enough for lights to be enabled or disabled, respectively. Sensors of the second type, which we refer to as *motion sensors*, typically base their sensor states on the detection of some situation (such as motion) or not having detected the situation for some time. We assume that if any sensor detects motion and the ambient light level is low enough the lighting control system must turn the lights on, and if no motion has been detected for some time the lighting control system must turn the lights off. Furthermore, each room must have at least one ambient light sensor and at least one motion sensor associated with it, and each sensor must in turn be associated with at least one room. The reason for this is not only to make modelling easier, but has practical purposes as well; if there was a sensor not associated with any room its readings would not have any effect, and if there was a room with no sensors the system would have no input on which to base its decision to turn the lights on or off in that room. We therefore do not model rooms in which lighting is controlled manually. Finally, we do not model the internal workings of the sensors or give their outputs as numerical readings, but rather assume that their readings can be abstractly represented by some sensor states, which we will describe in Section 3.

We proceed as follows. In Section 2 we shortly describe the Event-B formalism to the extent used in this paper. Section 3 describes the process of modelling our lighting control system. In Section 4 we conclude and discuss future work.

2 Event-B

Event-B [2, 8] is a formal modelling language aimed at stepwise development of correct systems. The Event-B formalism is based on the Action Systems formalism [5, 13] and the B-Method [1]. Using Event-B, the development of a model in carried out step by step from an abstract specification towards more concrete ones, with the goal of reaching a model that is concrete enough to be implemented.

A model in Event-B consists of *machines* and *contexts*, as seen in Fig. 1. A machine consists of variables, invariants, and events. A machine may see a context, which contains constants, sets, and axioms about these. The invariants in the machine are Boolean predicates that must evaluate to true for every reachable state of the system, where the state is described by the variables and constants in the model [2].

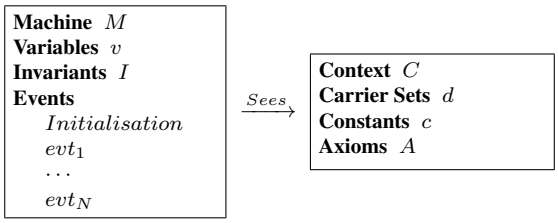

Figure 1: A machine M and a context C in Event-B

The events of a machine evaluate the variables (via event *guards*) and modify them (via event *actions*). An event for which all the guards evaluate to true is said to be enabled and can thereby execute and perform its actions, and if more than one event is enabled the choice of which one should execute is non-deterministic. If no event is enabled execution of the system is terminated, which can be desired or undesired (as in the form of a deadlock) depending on the model. Event-B action semantics are described using *before-after (BA) predicates* [2, 3], which describe relationships between system states before and after the execution of an event.

When using Event-B we employ a refinement-based approach to formal system development. Refinement provides a means for stepwise development of a system preserving correctness by gradually introducing new variables and events. Horizontal or superposition refinement [7, 11] refers to adding new variables and new events on top of the already existing variables and events. The added variables and events should be introduced in a consistent way with respect to the abstract machine. Vertical or data refinement [6] corresponds to replacing some abstract variables with more concrete variables and accordingly changing the events. In the development of the model we describe in this paper we use both refinement strategies.

In order to prove the correctness of each step of the development, we need to discharge a set of *proof obligations*. These are model semantics that can be math-

ematically expressed in the form of logical sequents. The Rodin Platform tool [8] generates proof obligations automatically and helps with discharging them, automatically or interactively [3, 4]. All of the proof obligations generated for each refinement step of our model can be discharged, thus ensuring the mathematical correctness of the complete model.

3 Modelling Our Lighting Control System

We model our lighting control system to control the lighting system in a house independently from other control systems. Lights will turn on or off based on input from sensors, which we model as being in different *sensor states* based on their readings. This separates the internal sensor readings from the situations that the lighting control system needs to respond to. We model two different types of sensors here. Sensors of the first type, which we will refer to as *ambient light sensors*, typically base their sensor states on the ambient light level being low enough or high enough for lights to be enabled or disabled, respectively. Sensors of the second type, which we refer to as *motion sensors*, typically base their sensor states on the detection of some situation (such as motion) or not having detected the situation for some time. As previously mentioned, we assume that if any sensor detects motion and the ambient light level is low enough the lighting control system must turn the lights on, and if no motion has been detected for some time the lighting control system must turn the lights off.

In Fig. 2 we show the sensor states for the motion sensors and the possible transitions between them. The ambient light sensors transition only from *sensor_lights_on* to *sensor_lights_off* and vice versa, and these transitions are therefore not shown.

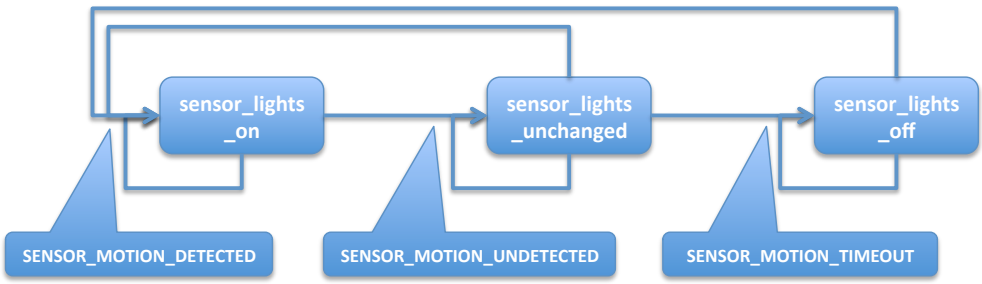

Figure 2: Motion sensor states and transitions in our model

Fig. 3 shows an overview of our lighting control system model and its refinements. In the following sections, we will shortly describe the development process, starting from the most abstract model and continuing with refinements. For the purpose of better readability axioms, invariants, guards, and actions are

labelled *axm*, *inv*, *grd* and *act*, respectively, followed by the number of the refinement in which that particular line was added, an underscore (_), and another number to uniquely identify each line.

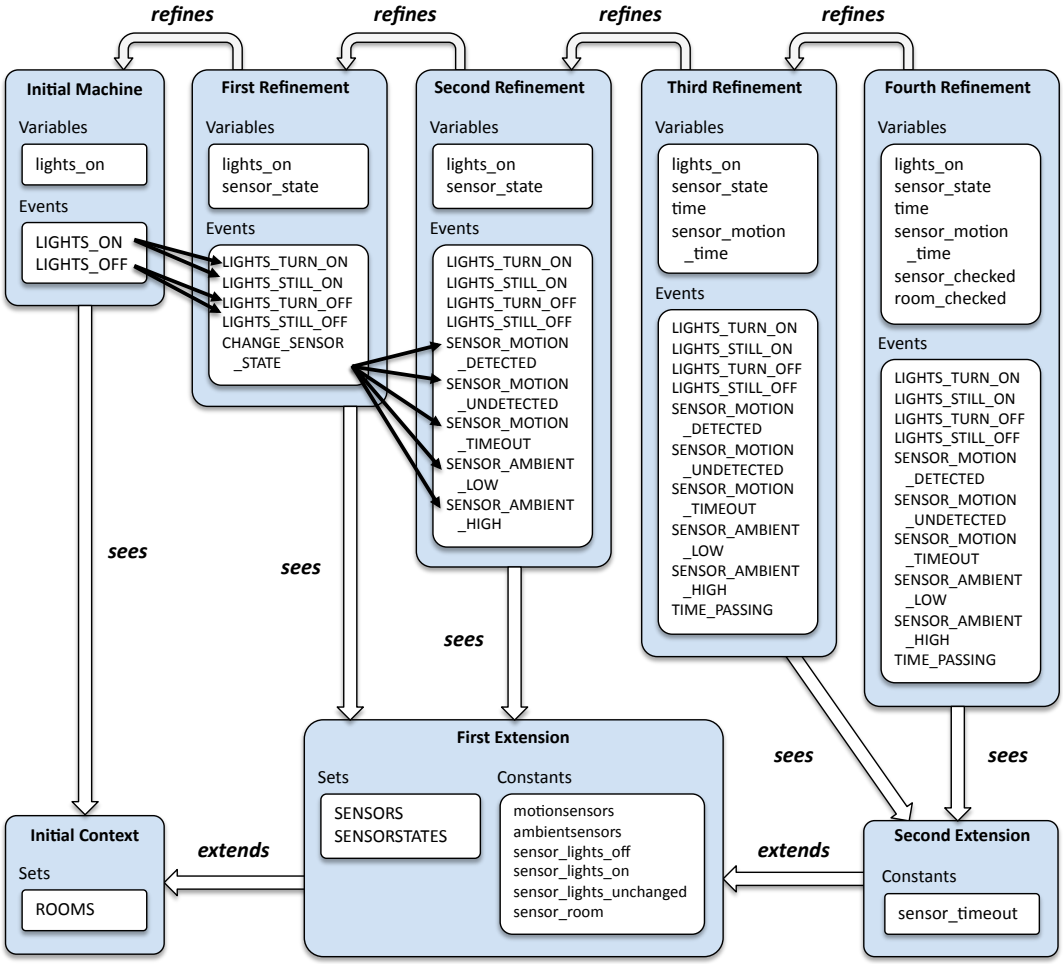

Figure 3: The refinement process of our lighting control system model

3.1 Initial Model

Our initial model is very abstract. In our initial context we have a finite set *ROOMS*. As previously mentioned, in our model a room does not necessarily represent a physical room in the house, but rather represents a discrete space in our adaptive house in which all the lights are switched on or off together. In our initial machine we therefore have a variable *lights_on* as a total function specifying a Boolean value for all rooms in our house. Initially the lights are turned off in all rooms.

CONTEXT SHL_C0

SETS

ROOMS

AXIOMS

axm0_1 : finite(ROOMS)

the number of rooms is finite

MACHINE SHL_M0
SEES SHL_C0
VARIABLES

lights_on

INVARIANTS

inv0_1 : lights_on $\in$ ROOMS $\rightarrow$ BOOL

lights on in a room, true or false

EVENTS
Initialisation

begin

act0_1 : lights_on $:=$ ROOMS $\times$ {FALSE}

initially all lights are off

end

In our initial machine we also have two abstract events: **LIGHTS_ON** for turning the lights on and **LIGHTS_OFF** for turning the lights off in any room. We will later refine these events into more concrete ones.

EVENT *LIGHTS_ON* $\widehat{=}$

any

r

where

grd0_1 : r $\in$ ROOMS

any room

then

act0_1 : lights_on(r) $:=$ TRUE

lights will be on

end

EVENT *LIGHTS_OFF* $\widehat{=}$

any

r

where

grd0_1 : r $\in$ ROOMS

any room

then

act0_1 : lights_on(r) $:=$ FALSE

lights will be off

end

3.2 First Refinement

In the first refinement, we introduce the finite sets *SENSORS* and *SENSORSTATES* into the context. The set *SENSORS* is partitioned into motion sensors (*motionsensors*) and ambient light sensors (*ambientsensors*), while the possible sensor states are *sensor_lights_off*, *sensor_lights_on* and *sensor_lights_unchanged*. We also introduce the constant *sensor_room* specifying that each sensor must belong to at least one room (axiom *axm1_4*) and there must be a sensor in each room. We further specify that each room must have at least one motion sensor and one ambient light sensor (axioms *axm1_6* and *axm1_7*).

CONTEXT SHL_C1
EXTENDS SHL_C0
SETS

SENSORS

SENSORSTATES

CONSTANTS

motionsensors

ambientsensors

sensor_lights_off

sensor_lights_on

sensor_lights_unchanged

sensor_room

AXIOMS

axm1_2 : finite(SENSORS)
the sensors are finite

axm1_3 : partition(SENSORS, motionsensors, ambientsensors)
the sensors are partitioned into motion sensors and ambient light sensors

axm1_4 : sensor_room ∈ SENSORS ⇿ ROOMS
each sensor belongs to at least one room

axm1_5 : partition(SENSORSTATES, {sensor_lights_off}, {sensor_lights_on}, {sensor_lights_unchanged})
sensors can indicate that lights should be on, lights should remain unchanged, or lights should be off

axm1_6 : ∀ra·ra ∈ ROOMS ⇒ (∃sm·sm ∈ motionsensors ∧ (sm ↦ ra) ∈ sensor_room)
in each room there is at least one sensor of the motion sensor type

axm1_7 : ∀rb·rb ∈ ROOMS ⇒ (∃sn·sn ∈ ambientsensors ∧ (sn ↦ rb) ∈ sensor_room)
in each room there is at least one sensor of the ambient light sensor type

In this refinement we also introduce a variable *sensor_state*, which is modelled as a total function specifying a sensor state for each sensor. However, the transitions between sensor states shown in Fig. 2 are not yet modelled, but instead an abstract event **CHANGE_SENSOR_STATE** is introduced that later will be

refined into more concrete events for transitioning between sensor states. In this refinement the previous abstract events for changing *lights_on* are refined into the separate events **LIGHTS_TURN_ON** for lights going from off to on, **LIGHTS_STILL_ON** for lights remaining on, **LIGHTS_TURN_OFF** for lights going from on to off and **LIGHTS_STILL_OFF** for lights remaining off in a room. The guards *grd1_2*, *grd1_3* and *grd1_4* in these events specify the conditions that must hold for each of these events to be enabled, and at any time only one of these four events can be enabled for each room. We have here assumed that for the lights to turn from off to on there must be at least one motion sensor and one ambient sensor in the room indicating that the lights should be on, otherwise the lights will remain off. However, for lights that are on to turn off we require that all motion sensors in that room indicate that the lights should be off, otherwise the lights will remain on.

MACHINE SHL_M1
REFINES SHL_M0
SEES SHL_C1
VARIABLES

lights_on

sensor_state

INVARIANTS

inv1_3 : $sensor\_state \in SENSORS \rightarrow SENSORSTATES$

EVENTS
Initialisation

begin

act0_1 : $lights\_on := ROOMS \times \{FALSE\}$
initially all lights are off

act1_2 : $sensor\_state := SENSORS \times \{sensor\_lights\_off\}$
initally all sensor think that the light should be off

end

EVENT *LIGHTS_TURN_ON* $\widehat{=}$
refines *LIGHTS_ON*

any

r

where

grd0_1 : $r \in ROOMS$
any room

grd1_2 : $lights\_on(r) = FALSE$
lights are off

grd1_3 : $\exists sa \cdot sa \in motionsensors \land (sa \mapsto r) \in sensor\_room \land sensor\_state(sa) = sensor\_lights\_on$
at least one motion sensor thinks lights should be turned on

grd1_4 : $\exists sb \cdot sb \in ambientsensors \land (sb \mapsto r) \in sensor\_room \land sensor\_state(sb) = sensor\_lights\_on$
at least one ambient sensors think lights could be tuned on

then

act0_1 : lights_on(r) := TRUE
lights will be on

end

EVENT *LIGHTS_STILL_ON* $\widehat{=}$
refines *LIGHTS_ON*

any
r
where
grd0_1 : r ∈ ROOMS
any room
grd1_2 : lights_on(r) = TRUE
lights are on
grd1_3 : ¬(∀sa·sa ∈ motionsensors ∧ (sa ↦ r) ∈ sensor_room
⇒ sensor_state(sa) = sensor_lights_off)
not all motions sensors have timed out
then
skip
end

EVENT *LIGHTS_TURN_OFF* $\widehat{=}$
refines *LIGHTS_OFF*

any
r
where
grd0_1 : r ∈ ROOMS
any room
grd1_2 : lights_on(r) = TRUE
lights are on
grd1_3 : ∀sa·sa ∈ motionsensors ∧ (sa ↦ r) ∈ sensor_room
⇒ sensor_state(sa) = sensor_lights_off
all motions sensors in this room think lights should be turned off
then
act0_1 : lights_on(r) := FALSE
lights will be off
end

EVENT *LIGHTS_STILL_OFF* $\widehat{=}$
refines *LIGHTS_OFF*

any
r
where
grd0_1 : r ∈ ROOMS
any room
grd1_2 : lights_on(r) = FALSE
lights are off
grd1_3 : ¬(∃sa·sa ∈ motionsensors ∧ (sa ↦ r) ∈ sensor_room
∧ sensor_state(sa) = sensor_lights_on)
∨ ¬(∃sb·sb ∈ ambientsensors ∧ (sb ↦ r) ∈ sensor_room
∧ sensor_state(sb) = sensor_lights_on)
no motion sensor or no ambient sensor thinks lights should be on
then
skip

end

EVENT *CHANGE_SENSOR_STATE* $\widehat{=}$

any

s

t

where

grd1_1 : s ∈ SENSORS

any sensor

grd1_2 : t ∈ SENSORSTATES

any sensor state

then

act1_1 : sensor_state(s) := t set the state of that sensor

end

3.3 Second Refinement

In the second refinement step we do not add anything to the context, but in the machine we refine the previous abstract event **CHANGE_SENSOR_STATE** into several different events. We have three different events for motion sensors, corresponding to the transitions between states in Fig. 2, and two different events for ambient light sensors, corresponding to the ambient light level being low enough that lights could be turned on and high enough that lights should not be turned on, respectively. As we have replaced an abstract notion of sensor state with a concrete one, we use a witness (keyword *with*) in each event for the relation between the previous abstract parameter *t* and its refinement in each event.

EVENT *SENSOR_MOTION_DETECTED* $\widehat{=}$

refines *CHANGE_SENSOR_STATE*

any

s

where

grd2_1 : s ∈ motionsensors

any motion sensor

with

t : t = sensor_lights_on

then

act2_1 : sensor_state(s) := sensor_lights_on

motion detcted, lights should be on

end

EVENT *SENSOR_MOTION_UNDETECTED* $\widehat{=}$

refines *CHANGE_SENSOR_STATE*

any

s

where

grd2_1 : s ∈ motionsensors

any motion sensor

grd2_3 : sensor_state(s) ≠ sensor_lights_off

lights are not off

with

$t : t = sensor\_lights\_unchanged$

then

$act2\_1 : sensor\_state(s) := sensor\_lights\_unchanged$

motion not detected, lights should remain unchanged

end

EVENT *SENSOR_MOTION_TIMEOUT* $\widehat{=}$

refines *CHANGE_SENSOR_STATE*

any

s

where

$grd2\_1 : s \in motionsensors$

any motion sensor

$grd2\_3 : sensor\_state(s) = sensor\_lights\_unchanged$

lights should have been unchanged

with

$t : t = sensor\_lights\_off$

then

$act2\_1 : sensor\_state(s) := sensor\_lights\_off$

sensor timeout, lights should turn off

end

EVENT *SENSOR_AMBIENT_LOW* $\widehat{=}$

refines *CHANGE_SENSOR_STATE*

any

s

where

$grd2\_1 : s \in ambientsensors$

any ambient sensor

with

$t : t = sensor\_lights\_on$

then

$act2\_1 : sensor\_state(s) := sensor\_lights\_on$

ambient sensor indicates lights should be on

end

EVENT *SENSOR_AMBIENT_HIGH* $\widehat{=}$

refines *CHANGE_SENSOR_STATE*

any

s

where

$grd2\_1 : s \in ambientsensors$

any ambient sensor

with

$t : t = sensor\_lights\_off$

then

$act2\_1 : sensor\_state(s) := sensor\_lights\_off$

ambient sensor indicates lights should be off

end

3.4 Third Refinement

In the third refinement we introduce the notion of time. Each motion sensor has an associated timeout value via the *sensor_timeout* constant, and the variable *time* is used to model the global time. We also introduce a variable *sensor_motion_time* for modelling when sensors last detected motion, which will be used with the *sensor_timeout* constant to enable each sensor to transition from the *sensor_lights_unchanged* state to the *sensor_lights_off* state after a certain amount of time has passed.

The final three invariants added in this refinement are interesting provable properties regarding the sensor states. The first one, *inv3_7*, states that for all motions sensors that are in the *sensor_lights_on* state, there must be a notion of at what time the sensor last detected motion, i.e., the sensor must have detected motion at some point. The second one, *inv3_8*, makes a similar claim for sensors in the *sensor_lights_unchanged* state. The third of the last three new invariants, *inv3_9*, states that if a sensor is in the *sensor_lights_off* state, and there is a notion of at what time the sensor last detected motion, the difference between the current global time and the time the sensor detected motion must be larger than the timeout value for that sensor. Thus, a sensor will be in the *sensor_lights_off* state if either it has never detected motion at all or sufficient time has passed since the sensor last detected motion.

CONTEXT SHL_C3
EXTENDS SHL_C1
CONSTANTS

`sensor_timeout`

AXIOMS

`axm3_6 :` $\texttt{sensor\_timeout} \in \texttt{motionsensors} \rightarrow \mathbb{N}$
each motion sensor times out after this time

MACHINE SHL_M3
REFINES SHL_M2
SEES SHL_C3
VARIABLES

`lights_on`

`sensor_state`

`time`

`sensor_motion_time`

INVARIANTS

`inv3_5 :` $\texttt{time} \in \mathbb{N}$
global time

`inv3_6 :` $\texttt{sensor\_motion\_time} \in \texttt{motionsensors} \pfun \mathbb{N}$
the time when a sensor last detected motion

inv3_7 : $\forall sa \cdot sa \in motionsensors \land sensor\_state(sa) = sensor\_lights\_on \Rightarrow (sa \in dom(sensor\_motion\_time))$
if a sensor has detected motion, it has done so at a certain time

inv3_8 : $\forall sb \cdot sb \in motionsensors \land sensor\_state(sb) = sensor\_lights\_unchanged \Rightarrow (sb \in dom(sensor\_motion\_time))$
if a sensor has previously detected motion, it has done so at a certain time

inv3_9 : $\forall sc \cdot sc \in motionsensors \land sensor\_state(sc) = sensor\_lights\_off \Rightarrow (sc \in dom(sensor\_motion\_time) \Rightarrow ((time - sensor\_motion\_time(sc)) > sensor\_timeout(sc)))$
if a sensor has timed out, it has either never detected motion or detected it a longer time ago than its timeout value

Most of the events in our model remain unchanged in this refinement. We only update the **SENSOR_MOTION_DETECTED** event to make note of the time a motion sensor last detected motion, and the **SENSOR_MOTION_TIMEOUT** event to use the previously noted time to require that a longer time than the timeout value of the sensor has passed before the sensor enters the *sensor_lights_off* state. We also add an abstract event **TIME_PASSING** to update the global time in our model.

EVENT *SENSOR_MOTION_DETECTED* $\widehat{=}$
refines *SENSOR_MOTION_DETECTED*

any
s
where
grd2_1 : $s \in motionsensors$
any motion sensor
then
act2_1 : $sensor\_state(s) := sensor\_lights\_on$
lights should be on
act3_2 : $sensor\_motion\_time(s) := time$
update the time when motion was last detected
end

EVENT *SENSOR_MOTION_TIMEOUT* $\widehat{=}$
refines *SENSOR_MOTION_TIMEOUT*

any
s
where
grd2_1 : $s \in motionsensors$
any motion sensor
grd2_3 : $sensor\_state(s) = sensor\_lights\_unchanged$
lights should have been unchanged
grd3_4 : $(time - sensor\_motion\_time(s)) > sensor\_timeout(s)$
sensor has not detected motion for a time longer than its timeout value
then
act2_1 : $sensor\_state(s) := sensor\_lights\_off$
lights should turn off
end

EVENT *TIME_PASSING* $\widehat{=}$

begin

act3_1 : time := time + 1

update global time

end

3.5 Fourth Refinement

In the fourth refinement of our lighting control model, we do not add anything to the context. However, until now the ordering in which the events in the machine are able to execute has been non-deterministic. In this refinement step we change this by introducing two new variables into the machine; *sensor_checked* and *room_checked*. These variables keep track of which sensors have had their states checked and which rooms have been checked for whether their lights should be on or off, respectively, during the current time cycle. The model requires that the states of all sensors in a room have been checked before the checking of lighting status in a room can be performed, which is ensured by the invariants *inv4_13* and *inv4_14*, and that all the rooms have been checked before the time is updated, which is ensured by the guard *grd4_1* of the refined **TIME_PASSING** event. In this way we ensure that the lights will actually turn on or off when the situation warrants, instead of merely making it possible for them to do so.

MACHINE SHL_M4
REFINES SHL_M3
SEES SHL_C3
VARIABLES

lights_on

sensor_state

time

sensor_motion_time

sensor_checked

room_checked

INVARIANTS

inv4_10 : sensor_checked ∈ SENSORS ⇸ BOOL

any sensors may have been checked this cycle

inv4_11 : room_checked ∈ ROOMS ⇸ BOOL

any rooms may have been checked this cycle

inv4_12 : ∀sn·sn ∈ dom(sensor_checked) ⇒ sensor_checked(sn) = TRUE

if a sensor has been checked, its value is true

inv4_13 : ∀ro, so·ro ∈ ROOMS ∧ so ∈ SENSORS ∧ (so ↦ ro) ∈ sensor_room ∧ so ∉ dom(sensor_checked) ⇒ ro ∉ dom(room_checked)

if a sensor in a room has not been checked, that room has not been checked

inv4_14 : $\forall rf, sf \cdot rf \in ROOMS \land sf \in SENSORS \land (sf \mapsto rf) \in sensor\_room$
$\land\ rf \in dom(room\_checked) \Rightarrow sf \in dom(sensor\_checked)$
if a room has been checked, all sensors in that room have been checked

To ensure that the invariants hold, we have added guards *grd4_5* and *grd4_6* to all light events ensuring that we only check each room once during each time cycle, and only after all sensors associated with that room have been checked. Likewise, we add a guard *grd4_2* to each sensor event ensuring that each sensor is checked only once during each time cycle. Finally, the **TIME_PASSING** event is refined to require that all rooms have been checked before it will be enabled, and as the global time is updated it resets the *room_checked* and *sensor_checked* to empty sets.

EVENTS
Initialisation
begin
act0_1 : $lights\_on := ROOMS \times \{FALSE\}$
initially all lights are off
act1_2 : $sensor\_state := SENSORS \times \{sensor\_lights\_off\}$
initally all sensor think that the light should be off
act2_3 : $time := 0$
start at time 0
act2_4 : $sensor\_motion\_time := \varnothing$
no sensor has detected motion ever
act4_5 : $sensor\_checked := \varnothing$
and we haven't checked any sensors this cycle
act4_6 : $room\_checked := \varnothing$
and we haven't checked any rooms this cycle
end
EVENT *LIGHTS_TURN_ON* $\widehat{=}$
refines *LIGHTS_TURN_ON*
any
r
where
grd0_1 : $r \in ROOMS$
any room
grd1_2 : $lights\_on(r) = FALSE$
lights are off
grd1_3 : $\exists sa \cdot sa \in motionsensors \land (sa \mapsto r) \in sensor\_room$
$\land\ sensor\_state(sa) = sensor\_lights\_on$
at least one motion sensor thinks lights should be turned on
grd1_4 : $\exists sb \cdot sb \in ambientsensors \land (sb \mapsto r) \in sensor\_room$
$\land\ sensor\_state(sb) = sensor\_lights\_on$
at least one ambient sensors think lights could be tuned on
grd4_5 : $r \notin dom(room\_checked)$
we haven't checked the room yet
grd4_6 : $\forall sc \cdot sc \in SENSORS \land (sc \mapsto r) \in sensor\_room$
$\Rightarrow (sc \in dom(sensor\_checked))$
but we have checked all the sensors in the room

then

act0_1 : lights_on(r) := TRUE
lights will be on

act4_2 : room_checked(r) := TRUE
and note that we have checked this room

end

EVENT *LIGHTS_STILL_ON* $\widehat{=}$
refines *LIGHTS_STILL_ON*

any

r

where

grd0_1 : r ∈ ROOMS
any room

grd1_2 : lights_on(r) = TRUE
lights are on

grd1_3 : ¬(∀sa·sa ∈ motionsensors ∧ (sa ↦ r) ∈ sensor_room
⇒ sensor_state(sa) = sensor_lights_off)
not all motions sensors have timed out

grd4_5 : r ∉ dom(room_checked)
we haven't checked the room yet

grd4_6 : ∀sc·sc ∈ SENSORS ∧ (sc ↦ r) ∈ sensor_room
⇒ (sc ∈ dom(sensor_checked))
but we have checked all the sensors in the room

then

act4_2 : room_checked(r) := TRUE
note that we have checked this room

end

EVENT *LIGHTS_TURN_OFF* $\widehat{=}$
refines *LIGHTS_TURN_OFF*

any

r

where

grd0_1 : r ∈ ROOMS
any room

grd1_2 : lights_on(r) = TRUE
lights are on

grd1_3 : ∀sa·sa ∈ motionsensors ∧ (sa ↦ r) ∈ sensor_room
⇒ sensor_state(sa) = sensor_lights_off
all motions sensors in this room think lights should be turned off

grd4_5 : r ∉ dom(room_checked)
we haven't checked the room yet

grd4_6 : ∀sc·sc ∈ SENSORS ∧ (sc ↦ r) ∈ sensor_room
⇒ (sc ∈ dom(sensor_checked))
but we have checked all the sensors in the room

then

act0_1 : lights_on(r) := FALSE
lights will be off

act4_2 : room_checked(r) := TRUE
note that we have checked this room

end

EVENT *LIGHTS_STILL_OFF* $\widehat{=}$
refines *LIGHTS_STILL_OFF*

any

r

where

grd0_1 : $\mathtt{r} \in \mathtt{ROOMS}$

any room

grd1_2 : $\mathtt{lights\_on}(\mathtt{r}) = \mathtt{FALSE}$

lights are off

grd1_3 : $\neg(\exists \mathtt{sa} \cdot \mathtt{sa} \in \mathtt{motionsensors} \land (\mathtt{sa} \mapsto \mathtt{r}) \in \mathtt{sensor\_room} \land \mathtt{sensor\_state}(\mathtt{sa}) = \mathtt{sensor\_lights\_on}) \lor \neg(\exists \mathtt{sb} \cdot \mathtt{sb} \in \mathtt{ambientsensors} \land (\mathtt{sb} \mapsto \mathtt{r}) \in \mathtt{sensor\_room} \land \mathtt{sensor\_state}(\mathtt{sb}) = \mathtt{sensor\_lights\_on})$

no motion sensor or no ambient sensor thinks lights should be on

grd4_5 : $\mathtt{r} \notin \mathtt{dom}(\mathtt{room\_checked})$

we haven't checked the room yet

grd4_6 : $\forall \mathtt{sc} \cdot \mathtt{sc} \in \mathtt{SENSORS} \land (\mathtt{sc} \mapsto \mathtt{r}) \in \mathtt{sensor\_room} \Rightarrow (\mathtt{sc} \in \mathtt{dom}(\mathtt{sensor\_checked}))$

but we have checked all the sensors in the room

then

act4_2 : $\mathtt{room\_checked}(\mathtt{r}) := \mathtt{TRUE}$

note that we have checked this room

end

EVENT *SENSOR_MOTION_DETECTED* $\widehat{=}$
refines *SENSOR_MOTION_DETECTED*

any

s

where

grd2_1 : $\mathtt{s} \in \mathtt{motionsensors}$

any motion sensor

grd4_2 : $\mathtt{s} \notin \mathtt{dom}(\mathtt{sensor\_checked})$

as long as we haven't checked it yet this cycle

then

act2_1 : $\mathtt{sensor\_state}(\mathtt{s}) := \mathtt{sensor\_lights\_on}$

lights should be on

act3_2 : $\mathtt{sensor\_motion\_time}(\mathtt{s}) := \mathtt{time}$

update the time when motion was last detected

act4_3 : $\mathtt{sensor\_checked}(\mathtt{s}) := \mathtt{TRUE}$

note that we have checked this sensor

end

EVENT *SENSOR_MOTION_UNDETECTED* $\widehat{=}$
refines *SENSOR_MOTION_UNDETECTED*

any

s

where

grd2_1 : $\mathtt{s} \in \mathtt{motionsensors}$

any motion sensor

grd2_3 : $\mathtt{sensor\_state}(\mathtt{s}) \neq \mathtt{sensor\_lights\_off}$

lights are not off

grd4_2 : $s \notin dom(sensor\_checked)$
we haven't checked the sensor yet this cycle

then

act2_1 : $sensor\_state(s) := sensor\_lights\_unchanged$
motion not detected, lights should remain unchanged
act4_3 : $sensor\_checked(s) := TRUE$
note that we have checked this sensor

end

EVENT *SENSOR_MOTION_TIMEOUT* $\widehat{=}$
refines *SENSOR_MOTION_TIMEOUT*

any

s

where

grd2_1 : $s \in motionsensors$
any motion sensor
grd2_3 : $sensor\_state(s) = sensor\_lights\_unchanged$
lights should have been unchanged
grd3_4 : $(time - sensor\_motion\_time(s)) > sensor\_timeout(s)$
sensor has not detected motion for a time longer than its timeout value
grd4_2 : $s \notin dom(sensor\_checked)$
we haven't checked the sensor yet this cycle

then

act2_1 : $sensor\_state(s) := sensor\_lights\_off$
lights should turn off
act4_3 : $sensor\_checked(s) := TRUE$
note that we have checked this sensor

end

EVENT *SENSOR_AMBIENT_LOW* $\widehat{=}$
refines *SENSOR_AMBIENT_LOW*

any

s

where

grd2_1 : $s \in ambientsensors$
any ambient sensor
grd4_2 : $s \notin dom(sensor\_checked)$
we haven't checked the sensor yet this cycle

then

act2_1 : $sensor\_state(s) := sensor\_lights\_on$
ambient sensor indicates lights should be on
act4_3 : $sensor\_checked(s) := TRUE$
note that we have checked this sensor

end

EVENT *SENSOR_AMBIENT_HIGH* $\widehat{=}$
refines *SENSOR_AMBIENT_HIGH*

any

s

where

grd2_1 : $s \in ambientsensors$
any ambient sensor

grd4_2 : s ∉ dom(sensor_checked)
we haven't checked the sensor yet this cycle

then

act2_1 : sensor_state(s) := sensor_lights_off
ambient sensor indicates lights should be off
act4_3 : sensor_checked(s) := TRUE
and note that we have checked this sensor

end

EVENT *TIME_PASSING* ≙
refines *TIME_PASSING*

when

grd4_1 : dom(room_checked) = ROOMS
time increases only when we have checked all the rooms

then

act3_1 : time := time + 1
update global time
act4_2 : room_checked := ∅
we start checking rooms from scratch
act4_3 : sensor_checked := ∅
dito for sensors

end

With the fourth refinement of our model we have created a sufficiently concrete model of a lighting control system. We started from an abstract model of what the system should do, i.e., set the lights to be on or off in a room, and now we have a model of how that should be done, i.e., by checking the states of all the sensors in all the rooms and based on those decide for each room whether to switch the lights or let them remain unchanged.

4 Conclusion

Automated control systems can be very complex. Using the refinement approach to development, a system can be described at different levels of abstraction, and the consistency in and between levels can be proved mathematically. Refinement allows us to efficiently cope with complexity of distributed systems verification and gradually derive an implementation with the desired properties and behaviour [2].

We have created a model of an in-house lighting control system, starting from an abstract specification and stepwise introducing functionality until approaching a concrete implementation. In this model, all sensor readings, represented by abstract sensor states, will be checked each time cycle, and based on these the decision will be made to switch the lights in each room or let them remain unchanged in the present time cycle. In the future, we hope to integrate this control system with other formally developed control systems [12] to create an integrated model of such a system of systems [10].

Acknowledgements

The work described in this paper has been done as part of the FResCo project [9].

References

[1] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[2] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[3] J.-R. Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, and L. Voisin. Rodin: An Open Toolset for Modelling and Reasoning in Event-B. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(6):447–466, 2010.

[4] J.-R. Abrial, M. Butler, S. Hallerstede, and L. Voisin. An Open Extensible Tool Environment for Event-B. *Lecture Notes in Computer Science*, 4260/2006:588–605, 2006.

[5] R.J.R. Back and R. Kurki-Suonio. Decentralization of Process Nets with Centralized Control. In *Proceedings of the 2nd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 131–142, 1983.

[6] R.J.R. Back and K. Sere. Stepwise Refinement of Action Systems. *Structured Programming*, 12(1):17–30, 1991.

[7] R.J.R. Back and K. Sere. Superposition Refinement of Reactive Systems. *Formal Asp. Comput.*, 8(3):324–346, 1996.

[8] Event-B and the Rodin Platform. `http://www.event-b.org/` (Accessed May 2013).

[9] FResCo: High-quality Measurement Infrastructure for Future Resilient Control Systems. `https://research.it.abo.fi/research/distributed-systems-laboratory/projects/fresco` (Accessed May 2013).

[10] M. Jamshidi. *System of Systems Engineering: Innovations for the 21st Century*. New York: John Wiley & Sons, 2009.

[11] S.M. Katz. A Superimposition Control Construct for Distributed Systems. *ACM Transactions on Programming Languages and Systems*, 15(2):337–356, April 1993.

[12] M. Neovius. Formal Stepwise Development of an In-House Temperature Control System. Technical Report 1078, TUCS - Turku Centre for Computer Science, May 2013. http://tucs.fi/publications/view/?pub_id=tNeovius_Mats13a.

[13] M. Waldén and K. Sere. Reasoning About Action Systems Using the B-Method. *Formal Methods in Systems Design*, 13:5–35, 1998.

List of Figures

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 A, 20520 TURKU, Finland | www.tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics

Turku School of Economics
- Institute of Information Systems Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

ISBN 978-952-12-2894-0
ISSN 1239-1891