An Event-B Model of an Automotive Adaptive Exterior Light System*

Amel Mammar[1][0000−0003−0016−6898], Marc Frappier[2][0000−0002−4402−2514], and Régine Laleau[3]

[1] SAMOVAR, Institut Polytechnique de Paris, Télécom SudParis, France
amel.mammar@telecom-sudparis.eu
[2] Laboratoire GRIF, Département d'informatique, Faculté des sciences
Université de Sherbrooke, Québec, Canada
marc.frappier@usherbrooke.ca
[3] Université Paris-Est Créteil, LACL, Créteil, France
laleau@u-pec.fr

Abstract. This paper introduces an Event-B formal model of the adaptive exterior light system for cars, a case study proposed in the context of the ABZ2020 conference. The system describes the different provided lights and the conditions under which they are switched on/off in order to improve the visibility of the driver without dazzling the oncoming ones. The system can be viewed as a lights controller that reads different information form the available sensors (key state, exterior luminosity, etc.) and takes the adequate actions by acting on the actuators of the lights in order to ensure a good visibility for the driver according to the information read. Our model is built using stepwise refinement with the Event-B method. We consider all the features of the case study, all proof obligations have been discharged using the Rodin provers. Our model has been validated using ProB by applying the different provided scenarios. This validation has permitted us to point out and correct some mistakes, ambiguities and oversights in the first versions of the case study.

Keywords: Adaptive Exterior Light System, Event-B method, Refinement, Verification

1 Introduction

This paper presents a formal system model of an adaptive exterior light system (ELS) for a car. This system has been proposed as a case study for the ABZ2020 conference. We use Event-B to construct and represent this formal model.

The exterior light system subject of this case study has objective to adapt the brightness of the different lights with respect to the status of the car but also the oncoming ones. For that purpose, the cars are equipped with different lights that can be switched on/off under specific conditions. In this paper, we

* This work was supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada).

stress more on the modeling of low beams, tail lamps and direction indicators. Roughly speaking, the low beams illuminate the road when the vehicle is running or vehicle surrounding while leaving the car during darkness; tail lamps permit to illuminate the vehicle if it is parked on a dark road at night, whereas the direction indicators allow to inform the following vehicle that the car will turn on the right/left. To control these exterior lights, the driver acts on the different physical elements like the key, the hazard switch etc. The position of the key (NoKeyInserted, KeyInserted, KeyInIgnitionOnPosition) is transmitted to the controller of the lights via the sensor keyState. Similarly, the hazard warning switch, with two positions (On/Off), permits to make both director indicators flashing at the same time.

The rest of this paper is structured as follows. After a brief overview of the Event-B method provided in Section 2, Section 3 presents our modelling strategy. Section 4 describes our model in more details. The validation and verification of our model are discussed in Section 5. Section 6 identifies the weaknesses of the requirements document provided for the case study, and the adequacy of the Event-B method for constructing a model of this case study. We conclude in Section 7.

2 Event-B Method

Event-B is a formal system modeling notation [1] proposed by Abrial. It allows for the stepwise construction of models using refinement. It is inspired from actions systems originally proposed by Back [2] and extended by several others. An Event-B model is made of components of two types: machine and context. A machine consists of events that modify state variables. An event has a set of guards and actions. When the guards hold, the event can be triggered; its actions can then modify the system state. A machine has state invariants that can be proved by discharging proof obligations.

A machine can refine another machine by replacing or adding state variables and by adding new events. System elements can be gradually taken into account through refinement. Existing variables can also be replaced by new variables; to show behavior preservation, a gluing invariant must relate the old variables to the new variables. An event refines an existing event by reducing its nondeterminacy, by strengthening its guards and/or choosing a value v, called *witness*, for a parameter p of the event. In that case, we should replace each occurrence of p by the value v in the guard and the substitution of the event. New events implicitly refine a skip event, so they cannot modify existing variables; they can only modify the added (new) variables. System constants are specified in contexts. A context can extend another context. Invariants are preserved by refinement, so invariants are introduced at the most appropriate step that simplifies their proof.

Event-B is supported by the Rodin platform [10], an Eclipse-based tool that provides editors, provers and several other plugins for various tasks (*e.g.*, animation and model checking with ProB [5], integration with UML class dia-

grams and state machines with UML-B[13]). In this paper, we have used Rodin with the ProB and AtelierB provers plugins. We did not use any other plugin.

In this paper, we report on the use of this formal method for the modeling and the verification of the the automotive adaptive exterior light system whose behavior is briefly described in the introduction. The chose of this formal method can be justified by the refinement technique it provides to gradually introduce the details of the system and also its different available support tools for modeling, animating and proving a specification.

3 Modelling Strategy

We reuse the terminology introduced in [11]. A control system interacts with its environment using sensors and actuators. A sensor measures the value of some environment characteristic m, called a *monitored* variable (*e.g.*, the state of the ignition key), and provides this measure (*e.g.*, whether the key is inserted or not) to the software controller as an *input* variable i. In a perfect world, we have $m = i$, but a sensor may fail. The software controller can influence the environment by sending commands, called *output* variable o to actuators. An actuator influences the value of some characteristics of the environment, call a *controlled* variable c. Variables m and c are called *environment variables*. Variables i and o are called *controller variables*. Finally, a controller has its own internal state variables to perform computations. We use Event-B state variables to represent environment (*i.e.*, *monitored* and *controlled*) variables, and controller variables. We do not model sensor/actuator failures.

3.1 Control Abstraction

A typical implementation of a control system such as the ELS is either a control loop that reads all input variables at once and then computes all output variables in the same iteration, or it can be driven by interruption triggered when a sensor provides a new value. The body of a control loop represents a single event and state transition. This allows for the definition of priorities between input variable changes. In our model, we use a more abstract approach, as it is common in the Event-B style of system modeling. We define one event for each input variable change, which allows for a more modular specification that is easier to prove. This is closer to an interrupt-driven control system. Our Event-B abstraction is also a reasonable abstraction for a control loop, considering that in most cases, a single input variable changes between two control loop iterations. The control loop can be derived from our specification by merging all events and defining priorities between events.

3.2 Modeling Structuration

The specification is structured into five refinements steps (five contexts and six machines). At the most abstract level we introduce various kinds of lights con-

trolled by the system. They are declared as constants in Context C0. The considered lights are: the direction indicators (left or right), the low beam headlights (left and right), the tail lamp (left and right), the reverse light (that indicates that the vehicle will move backwards), the brake lights and the cornering lights (that illuminate the cornering area separately when turning left or right). The high beam headlights are considered in Context C4 and Machine M5 since their behavior is different from the other lights, as it can be adaptive. Constant *LigntnessLevel* indicates the high beam light range, as specified in the requirement document [3].

Machine M0 in Fig. 1 contains a unique variable *headingState* that associates a level of brightness to each light declared in Context C0, and a unique event headLightSet that assigns an arbitrary level of brightness to these lights.

MACHINE M0
SEES C0
VARIABLES
headingState
INVARIANTS
inv1: $headingState \in HeadLights \rightarrow LigntnessLevel$
EVENTS
Initialisation
begin
act1: $headingState := HeadLights \times \{0\}$
end
Event headLightSet $\widehat{=}$
any
hl
where
grd1: $hl \in HeadLights \nrightarrow LigntnessLevel$
then
act1: $headingState := headingState \lhd\!\!\!+ hl$
end
END

Fig. 1. Machine M0

The first refinement, Machine M1 and Context C1, introduces the elements that the car driver can control and that can have an impact on the state of the lights declared in Context C0, namely the ignition key, the pitman arm, the light rotary switch, the brake pedal and the hazard warning light switch. For each of these elements, there is one event that refines headLightSet and that arbitrary modifies the lights impacted by this element.

Each of the subsequent refinements describes the behavior of particular lights. The choice of the lights taken into account in the refinements is arbitrary. Machine M2 and Context C2 consider the direction indicators, the hazard warning light and the emergency brake light. Machine M3 and Context C3 consider the low beam lights. Machine M4 considers the cornering lights and Machine M5 and Context C4 consider the high beam headlights.

3.3 Formalization of the Requirements

Table 1 relates the components of our model with the requirements listed in [3]. As one can remark, some requirements are specified as invariant whereas others are only considered in the related events. Requirement ELS-10 for instance stating the duration of a flashing cycle does not correspond to an invariant but it is considered in the event flashingDark that makes the current time progress by a unit of time. Specifying such requirements as an invariant would require the introduction of two extra variables to store the starting and the ending moment of the cycle to set that the difference should be equal to a unit of time. Roughly speaking, a timed requirement, an action duration more precisely, is modelled as an event if there is no other requirement that refers to such a duration otherwise an invariant is associated with it. Moreover, let us note that $M3$ is the refinement with the most invariants number because it models several interrelated lights, that is the low beams, the tail lamps, the parking lights etc.

3.4 Modeling of Temporal Requirements

Some properties of the requirements depend on two consecutive states. For example, requirement ELS-16 applies only when the rotary switch is turned to `Auto` while the ignition is already `Off`. This requirement can be expressed using an LTL formula as follows:

$$\begin{array}{l} \mathsf{G} \quad ((keyState \neq KeyInIgnitionOnPosition \wedge lightSwitch \neq Auto) \\ \quad \Rightarrow \\ \qquad \mathsf{X}\ (lightSwitch = Auto \Rightarrow headingState[LowBeams] = 0)) \end{array}$$

Unfortunately Event-B does not support the expression of LTL formula as part of the specification even if the ProB model-checker can check LTL formulas on an Event-B specification with a finite state space, but it does not terminate for our model on such properties, because of the size of the state space. On the other hand, a proof-based approach for temporal formulas is proposed in [7], but it generates a large number of proof obligations for a model of this size. Thus, we have chosen to express these properties as invariants by adding an extra variable to store the previous value of a state variable that is needed in a two-consecutive-state property. For example, to express ELS-16 as an invariant, we have to say that: (1) the current and previous states of the ignition are not equal to `On`, (2) the previous state of the switch is different from `Auto`, and (3) the current state of the switch is equal to `Auto`, which is represented by the following invariant (Machine M3, Invariant inv18)

$$\begin{array}{l} \quad ELS16 = TRUE \wedge ELS16P = FALSE \\ \Rightarrow \\ \quad keyState \neq KeyInIgnitionOnPosition \wedge \\ \quad keyStateP \neq KeyInIgnitionOnPosition \wedge \\ \quad lightSwitch = Auto \wedge lightSwitchP \neq Auto \end{array}$$

Requirements [3]	**Component**	**Invariant/Event**
ELS-1, ELS-2, ELS-4, ELS-23	M2	inv5, inv7
ELS-3		movePitmanUD
ELS-5, ELS-23	M2	inv8
ELS-6	M3	inv10
ELS-7	M2	movePitmanUD
ELS-8	M2	inv6, inv8
ELS-10	M2	flashingDark
ELS-11 to ELS-13	M2	movePitmanUD
ELS-14	M3	inv2
ELS-15	M3	inv3
ELS-16	M3	inv4
ELS-17	M3	inv5
ELS-18	M3	inv6,7,8,9
ELS-19	M3	inv10
ELS-21	M3	inv3-5, inv10,inv14
ELS-22	M3	inv11,12,13
ELS-24,25,26,27	M4	inv2-inv13
ELS-28	M3	inv14
ELS-29		all invariants defining the brightness level
ELS-30, ELS-31	M5	inv3,5
ELS-32..38	M5	inv6-11
ELS-39	M2	inv12,13
ELS-40	M2	inv14
ELS-41	M1	inv12,13
ELS-42	M5	inv4
ELS-43...49	M5	inv6-11

Table 1. Cross-reference between the components of our model and the requirements of [3]

Variable $ELS16$ represent the satisfaction of the conditions of ELS-16 and it is maintained by event moveSwitchAuto representing the state change of the rotary switch to position `Auto`. Variable $ELS16P$ represents its previous value. It conditions the invariant to the state change of the rotary switch.

These extra variables storing previous values must obviously be maintained in the events that change the value of the corresponding variable, but also in events that rely on the previous value for making a decision, even if they do not modify the corresponding variable.

4 Model Details

In this section, we briefly describe some specific ways of modelling that characterize our specification. The complete archive of the Event-B project is available in [6].

4.1 Modeling Complex User Interface Elements

There are elements manipulated by the car driver that have several positions and that control several lights depending on their positions. This is the case of the key and the light rotary switch. For each of these elements, the position it can take depends on the current position and thus can be described by a state-transition diagram. In the more abstract levels, we have chosen to gather all the possible transitions into a single event because at these levels the invariants do not depend on a specific position.

Let us take the case of the key. In Context C1, set *keyStates* describes all the states of the key:

$$partition(keyStates, \{NoKeyInserted\}, \{KeyInserted\}, \{KeyInIgnitionOnPosition\})$$

In Machine M1, Variable *keyState* represents the current state of the key, Variable *keyStateP* contains the previous state of the key and the authorized transitions are specified in Invariants inv2,inv3:

$$\begin{array}{ll} & keyState = NoKeyInserted \\ \Rightarrow & keyStateP = NoKeyInserted \lor keyStateP = KeyInserted \end{array}$$

$$\begin{array}{ll} & keyState = KeyInIgnitionOnPosition \\ \Rightarrow & keyStateP = KeyInIgnitionOnPosition \lor keyStateP = KeyInserted \end{array}$$

Event moveKey specifies the new state of the key according to its previous state and restricts the value of the event parameter *hl* to the lights controlled by the key.

Event moveKey $\widehat{=}$
refines headLightSet
  any

hl,valkey
where
grd1: $hl \in \quad LowBeams \cup tailLamps \cup directionIndicators \cup \{corneringLightLeft, corneringLightRight\} \nrightarrow LigntnessLevel$
grd2: $(keyState = NoKeyInserted \Rightarrow valkey = KeyInserted) \wedge (keyState = KeyInserted \Rightarrow valkey \in \{NoKeyInserted, KeyInIgnitionOnPosition\}) \wedge (keyState = KeyInIgnitionOnPosition \Rightarrow valkey = KeyInserted)$
then
act1: $headingState := headingState \lhd\!\!+ hl$
act2: $keyState := valkey$
act3: $keyStateP := keyState$
act4: $pitmanArmUDP := pitmanArmUD$
end

In Machine M2, Event moveKey is refined to specify the behavior of the direction indicator and the tail lamps according to the position of the key and the position of the hazard warning switch.

In Machine M3, we have split Event moveKey into four events (*i.e.*, insertKey, insertKeyputIgnitionOn, insertKeyputIgnitionOff, removeKey) to be more precise on the state of the lights according to the position of the key.

Let us take the two events insertKey and insertKeyputIgnitionOn. In Event insertKey, Action act4 specifies that if the hazard warning switch is not activated then the direction indicator is `off`, otherwise it is `on` and the two flashing lights are `on`.

Event insertKey $\widehat{=}$
refines moveKey
any
hl
where
grd1: $hl \in \quad LowBeams \cup tailLamps \cup directionIndicators \rightarrow LigntnessLevel$
grd2: $keyState = NoKeyInserted$
grd3: ...
grd4: $hazardWarningSwitchOn = FALSE \Rightarrow (directionIndicators) \times \{0\} \subseteq hl$
...
with
valkey: valkey= keyInserted
then
act1: $headingState := headingState \lhd\!\!+ hl$
act2: $keyState := KeyInserted$
act3: $keyStateP := keyState$
act4: $direcIndFlash :=$
$\{TRUE \mapsto \{blinkRight \mapsto FALSE, blinkLeft \mapsto FALSE\},$

$FALSE \mapsto directionIndicators \times \{TRUE\}$
$\}(bool(hazardWarningSwitchOn = FALSE))$
...
end

In Event putIgnitionOn, Action act4 specifies that if the hazard warning switch is not activated then the direction indicator is activated to the left or right according to the position of the pitman arm, otherwise it is `on` and the two flashing lights are `on`.

Event putIgnitionOn $\widehat{=}$
refines moveKey
any
hl
where
grd1: $hl \in LowBeams \cup tailLamps \cup directionIndicators \rightarrow LigntnessLevel$
...
with
valkey: valkey= KeyInIgnitionOnPosition
then
act1: $headingState := headingState \lhd hl$
act2: $keyState := KeyInIgnitionOnPosition$
act3: $keyStateP := keyState$
act4: $direcIndFlash :=$
$\{TRUE \mapsto \{blinkRight \mapsto bool(pitmanArmUD \in Upward),$
$blinkLeft \mapsto bool(pitmanArmUD \in Downward)\},$
$FALSE \mapsto directionIndicators \times \{TRUE\}$
$\}(bool(hazardWarningSwitchOn = FALSE))$
...
end

We have applied the same modeling process to the Light Rotary Switch.

Splitting the event makes the proof obligations easier to discharge even if more proof obligations are generated.

4.2 Managing Priorities between Requirements

Some requirements can be in conflict because they have common system states with different transitions. This is the case for Requirements ELS-16 and ELS-17. On one hand, ELS-16 states that if the key state is `inserted` then the low beam headlights are `off`. This is specified in Invariant inv4 of Machine M3:

$$ELS16 = TRUE \wedge ... \Rightarrow headingState[LowBeams] = 0$$

where Variable *ELS16* is `TRUE` if the key state is `inserted`.

On the other hand, ELS-17 states that if the daytime running light is activated then the low beam headlights are activated after starting the engine and

remain activated as long as the key is not removed, that is, either the key position is `inserted` or the ignition is `on`.

We have detected the conflict when we have animated the specification. The solution is to prioritize the requirements. After discussing with the case study authors, a priority for ELS-16 over ELS-17 has been set; this is specified in Invariant inv5 of Machine M3 that translates ELS-17:

$$\begin{array}{ll} & (... \lor dayTimeLightCont = TRUE) \land ... \land ELS16 = FALSE \land ... \\ \Rightarrow & headingState[LowBeams] = 100 \end{array}$$

where Variable *dayTimeLightCont* is `true` if the daytime running light is activated.

4.3 Modeling Time Duration

In Event-B, a specification of requirements that involves time duration requires to explicitly model time. In this case study, time can trigger changes on the state of lights (*e.g.* Requirements ELS-18, 19, 24, ... specify time intervals where particular lights have to be activated or not). A variable *currentTime* has been introduced in Machine M1 to model the time progression together with Event progress that increments this variable by an arbitrary positive number (Action act2). Action act1 specifies the lights whose state can be modified by a time progress.

Event progress $\widehat{=}$
refines headLightSet
  any
    hl
    step
  where
    grd1: $hl \in LowBeams \cup tailLamps \cup directionIndicators \cup \{corneringLightLeft, corneringLightRight\} \rightarrow\!\!\!\!\!\!\rightarrow LigntnessLevel$
    grd2: $step \in N1$
  then
    act1: $headingState := headingState \lhd\!\!\!\!\!\!\rightarrow hl$
    act2: $currentTime := currentTime + step$
      ...
  end

Event progress is refined in Machines M3, M4, M5 by detailing how each kind of lights is impacted. For instance, in M3, the exterior brightness (ELS-18) and the ambient light (ELS-19) imply to activate the low beam headlights for a given time interval.

4.4 Model Statistics

Table 2 describes the size of the model. Since Rodin does not use text files to store models, there are various ways of counting the lines of code (LOC)

of a model. Moreover, code is inherited when refinement and event extension is used. Lines of code are computed using the Camille editor representation of the Event-B model, which does not count inherited LOC through event extension and puts all variables on the same line. Total LOC, which includes inherited LOC, is provided within "( )", and computed using the pretty printer of the Rodin Event-B Machine Editor. Comments are excluded. Since we do not use data refinement (*i.e.*, no variable is replaced through refinement), we provide the total number of variables for each machine along with the number of new variables (*i.e.*, introduced in a refinement) enclosed by "( )". Invariants are specific to each machine. Since some events are renamed by refinement, we provide the total and new events introduced in each machine.

Component	**Size in LOC (Extended)**	**Constants / Variables Total (New)**	**Axioms / Invariants New**	**Events Total (New)**
C0	15	(17)	7	
C1	15	(17)	7	
C2	8	(2)	2	
C3	10	(2)	2	
C4	16	1	10	
M0	21 (28)	1 (1)	1	1
M1	215 (320)	15 (14)	13	12 (11)
M2	382 (691)	25 (10)	18	14 (2)
M3	908 (1619)	37 (12)	36	19 (5)
M4	885 (2377)	50 (13)	15	20 (1)
M5	416 (2694)	61 (11)	15	23 (3)
Total	2875		126	

Table 2. Model size

5 Validation and Verification

To verify and validate the Event-B models presented in the previous sections, we have proceeded into three steps detailed hereafter.

5.1 Model checking of the specification

In this step, the ProB tool is used as a model checker in order to ensure that the specification is invariant violation-free, that is, there is no trivial scenario that violates the invariants. From a practical point of view, ProB can find a sequence of events that, starting from a valid initial state of the machine, leads to a state that violates its invariant. Such scenarios (or counterexamples) may result from a guard/action missing but also from an incorrect invariant. This

step permits us to fix trivial bugs before the proof phase that can be very long and hard. It is worth noting that even if the tool does not find any invariant violation, it does not mean that the specification is correct. Indeed, there may be a scenario that the tool fails to find for different reasons like a timeout on the model checking process. In the present case study, the model checking step permits us to detect missing actions in particular those related to the variables representing the previous state of an element. Indeed, this makes the invariants depending on such variables violated as they should be verified only when the current and the previous values of these variables are different.

5.2 Validation with scenarios

The goal of this phase is to be sure that the specification satisfies the requirements. To this aim, we used the animation capability of ProB and played the different scenarios provided with the case study. This step permits us to exhibit several flaws/ambiguities in the initial release of the description documents (see Section 6 for more details). As examples of such flaws, we can cite the lack of prioritization between some requirements like ELS-16 and ELS-17 that share the same activation conditions when the *daytime running light* option is activated with the ignition in the `Off` position and the driver turns the switch in the `Auto` position. To correct these flaws/ambiguities, we have discussed with the case study authors because we are not specialists of the domain. For the above particular example, a priority is given to ELS-16 over ELS-17. It is worth noting that such flaws/ambiguities can not be detected in the model checking phase because they make the guard of some events unsatisfied, thus the event is not enabled and the invariant is thus not violated. Let us note that we had some problems to animate the first version of our models where we have kept the event parameter *hl* as a partial function on the set of all the lights. Indeed in that case, ProB checks all the possible partial functions on these lights which leads to a timeout. To overcome this issue, we have replaced each partial function by a more restrictive total function on the right domain, that is, the lights whose state actually changes after the execution of the event.

5.3 Proof of the specification

It is the last step, whose goal is to ensure the correctness of the specification by discharging proof obligations generated by Rodin. These proof obligations aim at proving invariant preservation by each event, but also to ensure that the guard of each refined event is stronger than that of the abstract event. These guard strengthening refinement proof obligations ensure that event parameters like *hl* mentioned above are properly refined. For instance, *hl* is defined as a partial function in the abstract event `headLightSet`; it is refined using total functions by giving its value for each refining event. So, we have to ensure that these values satisfy the initial guard. Figure 2 provides the proof statistics of the case study: 1643 proof obligations have been generated, of which 23% (385) were automatically proved by the various provers. The remaining proof obligations were

discharged interactively since they needed the use of external provers like the Mono Lemma prover that has shown to be very useful for arithmetic formulas. In addition, we have added some theorems on min/max operators (a min/max of a finite set is an element of the set, etc).

Let us note that the results of this phase has especially impacted some modeling choices. For instance, to speed up the proof phase, we have included in the guards some properties tagged as theorems in order to prove them only once and reuse them in all the proofs that need them for that event. This is the case of Guards grd9,grd10 of insertKey in Machine M3 that state:

grd9: $lowBeamRight \in dom(hl) \Rightarrow hl(lowBeamRight) \in 0..100$

grd10: $lowBeamLeft \in dom(hl) \Rightarrow hl(lowBeamLeft) \in 0..100$

Element Name	Total	Auto	Manual	Reviewed	Undischarged
ELS_1112	**1643**	**385**	**1258**	**0**	**0**
C0	0	0	0	0	0
C1	0	0	0	0	0
C2	0	0	0	0	0
C3	0	0	0	0	0
C4	14	9	5	0	0
M0	2	1	1	0	0
M1	88	55	33	0	0
M2	206	25	181	0	0
M3	738	131	607	0	0
M4	402	128	274	0	0
M5	193	36	157	0	0

Fig. 2. RODIN proof statistics of the case study

6 Other Points

6.1 Feedback on the requirements document

The formal modeling of the requirements document [3] lead us to identify a number of ambiguities and some contradictions with the test scenarios provided. We have communicated these to the authors of the requirements document, and a number of revisions were produced, following our comments. Our comments induced 9 of the 17 versions produced after the publication of the initial version of the requirements document. These modifications impacted 18 of the 49 requirements of the Exterior Light System. A detailed list of these elements are described in the last version (*i.e.*, 1.17) of the requirements document. We have mainly rephrased some requirements for which the applicability conditions should hold at different time points. For instance, in requirement ELS-16, the

condition "the switch in position `Auto`" should happen after the condition "the ignition is already `Off`". Moreover, we have defined priorities between requirements to make the specification deterministic: ELS-16 has priority over ELS-17, ELS-19 has priority over ELS-17, etc. We have also rephrased some sentences to clarify them. For instance in the first version of the document, the word "released" was used with the meaning "button pushed" in some places and with the meaning "button not pushed" in some others. To remove this ambiguity, we have replaced it with the terms "active" and "not active". Finally to make the modeling easier and after a discussion with the case study authors, the signal *pitmanArm* has been splitted into signals *pitmanArmForthBack* and *pitmanArmUpDown* with their corresponding positions (states) and the possible transitions between them.

6.2 Modeling temporal properties

Dealing with previous values to prove temporal properties turned out to be a significant burden. To improve/facilitate the specification of such kind of properties, which are probably very common in control systems, it would be interesting to study how they could be handled in Rodin or in some other plugin like the Event-B State machines plugin[4]. This plugin permits to generate Event-B events from a state machine including their guards that specify the requirements modeled by the state machine but without producing the related invariants. In that case, it becomes difficult to trace and justify the usefulness of the generated guards.

6.3 Identifying a refinement strategy

The crux in defining the structure of the Event-B model was to define the requirements elements to include at each refinement level. Recall that once a variable is introduced in a model, it cannot be modified by new events of subsequent refinements. Thus, when a variable is introduced, each event that needs to update it must be also introduced. In this case study, there are several dependencies between requirements elements. As many lights mutually rely on the same sensors and are correlated in terms of behavior, we have defined a single event, in the first machine, to model the light state changes and refined it according to the different actuators/sensors. But, we think that it would be interesting to look deeper into the existing structuring approaches for Event-B: decomposition [12] or modularization [4], in order to structure the specification into smaller logical units to make the proofs easier. A refactoring tool based on the read/update dependencies between events and state variables would be nice. It could help in finding an optimal decomposition based on the connected components of a dependency graph for a given machine. Building such a graph from the requirements is not easy, as one typically needs to formalize the requirements to precisely understand which variables are needed and where. So, the specifier

[4] http://wiki.event-b.org/index.php/Event-B_State machines

typically finds the ideal refinement structure only after creating a potentially non optimal refinement structure. Often a lot of effort has been invested in creating this first model, and there is no resource left to do a refactoring to obtain a better model. By better, we mean a model whose refinement decomposition would yield easier proofs for the same set of properties.

7 Conclusion

We have presented an Event-B model for the ELS case study. Our model takes into account all of the requirements. The model was verified by proving a large number of properties (98 invariants) and by simulation using ProB. Temporal properties involving two consecutive states were proved using variables storing previous state values. Due to the model size (61 state variables), ProB was unable to verify invariant or temporal properties. The proof effort was quite significant: 1258 proofs obligation (76 %) had to be manually discharged. The last Event-B machine is quite large (2 694 LOC), which denotes that the case study was an interesting modeling and verification challenge. The Rodin provers were less efficient than in previous ABZ case studies, where the manual proofs ratio was closer to 30 % [9], [8].

The formalization lead us to identify several small ambiguities in the requirements. They have been discussed with the case study authors as they were discovered, which lead to 9 out of the 17 revisions of the case study text that were published during the modeling process. This shows that formalization is an effective technique to discover defects early in the software development process. It is well-known in the software engineering literature that the earlier a defect is found, the cheaper it is to fix it.

Determining the best refinement strategy remains a challenge in Event-B. We fell short of time to try out the model decomposition plugins available in Rodin. They might have been useful in decomposing the specification into smaller, more manageable parts. This case study is of a different nature than the previous ones in the ABZ conference series (*i.e.*, 2014 Landing gear, 2016 Hemodialysis, 2018 ERTMS). Its elements are more tightly coupled, which made it more difficult to find an appropriate refinement strategy. It contains more properties to prove than the previous ones, but they are more localized properties (*i.e.*, each property referring to a small number of events on at most two consecutive states) that do not depend on the relationship between monitored variables and controlled variables. However, we really think that the Event-B method must include modularization clauses as native structuring mechanisms like those of the B method that permit to have a modular specification since the first phases of the development. This will make Event-B more suitable for the development of big and complex systems. For comparison, in the ERTMS case study, we had to build a relationship between the real (actual) positions of the trains and the controller view of the train positions to prove safety properties. There were no such issues in the ELS case study.

Acknowledgments The authors would like to thank the case study authors, and Frank Houdek in particular, for his responsiveness and useful feedback during the modeling process when questions were raised or when ambiguities were found. The authors would also like to thank Michael Leuschel for his quick feedback on using ProB for this large case study.

References

1. Abrial, J.: Modeling in Event-B. Cambridge University Press (2010)
2. Back, R.J.R., Sere, K.: Stepwise refinement of action systems. In: van de Snepscheut, J.L.A. (ed.) Mathematics of Program Construction. pp. 115–138. Springer Berlin Heidelberg, Berlin, Heidelberg (1989)
3. Houdek, F., Raschke, A.: Adaptive Exterior Light and Speed Control System. https://abz2020.uni-ulm.de/case-study#Specification-Document (November 2019)
4. Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A.B., Varpaaniemi, K., Ilic, D., Latvala, T.: Supporting reuse in Event-B development: Modularisation approach. In: ASM. Lecture Notes in Computer Science, vol. 5977, pp. 174–188. Springer (2010)
5. Leuschel, M., Bendisposto, J., Dobrikov, I., Krings, S., Plagge, D.: From Animation to Data Validation: The ProB Constraint Solver 10 Years On. In: Boulanger, J.L. (ed.) Formal Methods Applied to Complex Systems: Implementation of the B Method, chap. 14, pp. 427–446. Wiley ISTE, Hoboken, NJ (2014)
6. Mammar, A., Frappier, M., Laleau, R.: An Event-B Model of an Automotive Adaptive Exterior Light System. Available at http://www-public.imtbs-tsp.eu/~mammar_a/LightControlSystem.html and http://info.usherbrooke.ca/mfrappier/abz2020-ELS-Case-Study/ (January 2020)
7. Mammar, A., Frappier, M.: Proof-based verification approaches for dynamic properties: application to the information system domain. Formal Asp. Comput. **27**(2), 335–374 (2015)
8. Mammar, A., Frappier, M., Fotso, S.J.T., Laleau, R.: An Event-B model of the hybrid ERTMS/ETCS level 3 standard. In: Abstract State Machines, Alloy, B, TLA, VDM, and Z - 6th International Conference, ABZ 2018. Lecture Notes in Computer Science, vol. 10817, pp. 353–366. Springer (2018)
9. Mammar, A., Laleau, R.: Modeling a landing gear system in Event-B. In: ABZ 2014: The Landing Gear Case Study - Case Study Track, Held at the 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z. Communications in Computer and Information Science, vol. 433, pp. 80–94. Springer (2014)
10. Event-B Consortium: http://www.event-b.org/
11. Parnas, D.L., Madey, J.: Functional Documents for Computer Systems. Science of Computer Programming **25**(1), 41–61 (1995)
12. Silva, R., Pascal, C., Hoang, T.S., Butler, M.J.: Decomposition tool for Event-B. Softw., Pract. Exper. **41**(2), 199–208 (2011)
13. Snook, C.F., Butler, M.J.: UML-B: A plug-in for the event-b tool set. In: Börger, E., Butler, M.J., Bowen, J.P., Boca, P. (eds.) Abstract State Machines, B and Z, First International Conference, ABZ 2008, London, UK, September 16-18, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5238, p. 344. Springer (2008), https://doi.org/10.1007/978-3-540-87603-8_32