Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

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

Amel Mammar[1], Marc Frappier[2], Régine Laleau[3]

[1]SAMOVAR, Institut Polytechnique de Paris, Télécom SudParis, France
[2]Laboratoire GRIF, Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec, Canada
[3]Université Paris-Est, LACL, UPEC, IUT Sénart Fontainebleau, France

Received: date / Revised version: date

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.

Key words: 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 objective of the exterior light system subject is 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 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.

Using the EVENT-B method and its associated tools, the models have been entirely developed by the first author who has been involved in the formal specification and verification of railway interlocking systems with the collaboration of Thales and RATP. A good experience has also been gained from the development of the previous ABZ case studies. During the development of these models, she very frequently exchanges with Frank Houdek in order to clarify some ambiguous informal descriptions but also to fix some errors detected during the animation and/or the proof phases. During the paper writing, performed by the other authors, the adopted choices/modelings have been discussed to make them clearer.

Send offprint requests to: Amel Mammar, E-mail: amel.mammar@telecom-SudParis.eu

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

1.1 Event-B method

Event-B [2] is the successor of the B method [1] permitting to model discrete systems using mathematical notations. The complexity of a system is mastered thanks to the refinement concept that allows to gradually introduce the different parts that constitute the system starting from an abstract model to a more concrete one. An Event-B specification is made of two elements: *context* and *machine*. A context describes the static part of an Event-B specification; it consists of constants and sets (user-defined types) together with axioms that specify their properties:

CONTEXT
    Cont
Sets
    S
Constants
    C
Axioms
    A
END

The dynamic part of an Event-B specification is included in a machine that defines variables V and a set of events E. The possible values that the variables hold are restricted using an invariant, denoted Inv, written using a first-order predicate on the state variables:

MACHINE
    Name
SEES
    Cont
Variables
    V
Invariants
    Inv
Events
    E

Each event has the following form:

ANY
    X
WHEN
    G
THEN
    Act
END

This event can be executed if it is enabled, i.e. all the conditions G, named guards, prior to its execution hold. Among all enabled events, only one is executed. In this case, substitutions *Act*, called actions, are applied over variables. In this paper, we restrict ourselves to the *becomes equal* substitution, denoted by $(x := e)$.

The execution of each event should maintain the invariant. To this aim, proof obligations are generated. For each event, we have to establish that:

$$\forall S, C, X.\ (A \wedge G \wedge Inv \Rightarrow [Act]Inv)$$

where $[Act]Inv$ gives the weakest constraint on the *before* state such that the execution of Act leads to an *after* state satisfying Inv.

Refinement is a process of enriching or modifying a model in order to augment the functionality being modeled, or/and explain how some purposes are achieved. Both Event-B elements *context* and *machine* can be refined. A context can be extended by defining new sets S_r and/or constants C_r together with new axioms A_r. A machine is refined by adding new variables and/or replacing existing variables by new ones V_r that are typed with an additional invariant Inv_r. New events can also be introduced to implicitly refine a **skip** event. In this paper, the refined events have the same form:

ANY
    X_r
WHEN
    G_r
THEN
    Act_r
END

To prove that a refinement is correct, we have to establish the following two proof obligations:

– *guard refinement*: the guard of the refined event should be stronger than the guard of the abstract one:

$$\forall (S, C, S_r, C_r, V, V_r, X, X_r).\\ (A \wedge A_r \wedge Inv \wedge Inv_r \Rightarrow (G_r \Rightarrow G))$$

– *Simulation*: the effect of the refined action should be stronger than the effect of the abstract one:

$$\forall (S, C, S_r, C_r, V, V_r, X, X_r).\\ (A \wedge A_r \wedge Inv \wedge Inv_r \wedge [Act_r]Inv_r \Rightarrow [Act]Inv)$$

To discharge the proof obligations, the Rodin platform[1] offers an automatic prover but also the possibility to use external provers as plugins, like the SMT and Atelier B provers that we use in this work. Both provers can be used either in automatic or interactive modes to discharge the proof obligations.

[1] `http://www.event-b.org/install.html`

1.2 *The* ProB *model checker*

ProB [12] is an animator and explicit automatic model checker, originally developed for the verification and validation of software development based on the B language. Developed at the University of Düsseldorf starting from 2003, ProB[2] implements an automatic model checking technique to check LTL (linear temporal logic) [22] and CTL (Computational Tree Logic) [5] properties against a B specification. The core of ProB is written in a logical programming language called Prolog; its purpose is to be a comprehensive tool in the area of formal verification methods. Its main functionalities can be summarized up as follow:

1. ProB can find a sequence of operations that, starting from a valid initial state of the machine, moves the machine into a state that violates its invariant,
2. giving a valid state, ProB ProB can exhibit the operation that make the invariant violated,
3. ProB allows the animation of the B/EventB specification to permit the user play different scenarios from a given starting state that satisfies the invariant. Through a graphical user interface implemented in Tcl/Tk, the animator provides the user with: (i) the current state, (2) the history of the operation executions that has led to the current state and (3) a list of all the enabled operations, along with proper argument instantiations. In this way, the user does not have to guess the right values for the operation arguments.
4. ProB supports the model checking of the LTL and CTL assertions.

1.3 *Contributions*

The development of the Event-B models provided in [16] took about two months. Since we had already modeled all the features of the case study in preparation for the first paper published at the ABZ'20 conference [15], this paper essentially provides a more detailed account of our model and its development. We have slightly improved our model following comments received from attendees at the conference regarding the modeling of the key/switch behaviors. The main additional contributions of this paper are as follows:

- A detailed description of the errors and ambiguities identified in the specification document.
- A comparison with similar approaches, presented at ABZ'20 conference, for the formal modeling of the case study
- A detailed presentation of our approach to deal with the timed aspects.

[2] `https://prob.hhu.de/`

1.4 *The structure of the paper*

The rest of this paper is structured as follows. Section 2 presents our modelling strategy. Section 3 describes our model in more details. The validation and verification of our model are discussed in Section 4. Section 5 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. Section 6 compares our model with other solutions of this case study. We conclude in Section 7.

2 Requirements and modelling strategy

We reuse the terminology introduced in [21]. A control system interacts with its environment using sensors and actuators. Fig 1 illustrates the structure of the interaction between the controller interacting and its environment, and the variables used to represent them. 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.

2.1 *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.

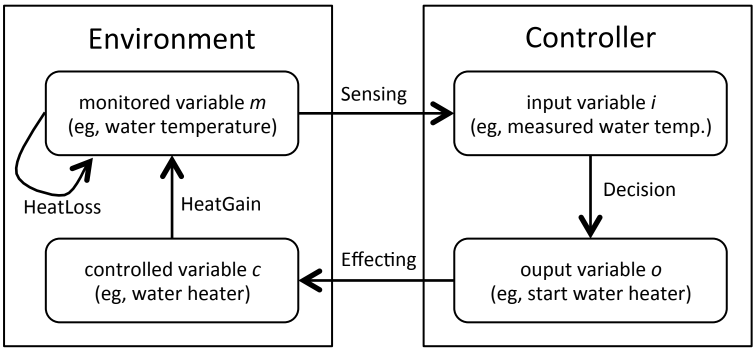

Fig. 1. Environment and controller variables

2.2 Model Structure

As depicted in Figure 2, the specification is structured into five refinements steps (five contexts and six machines). At the most abstract level we introduce various kinds of lights controlled 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 [8].

Machine M0 in Fig. 3 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.

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.

2.3 Formalization of the Requirements

Table 1 relates the components of our model with the requirements listed in [8]. 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.

2.4 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:

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

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

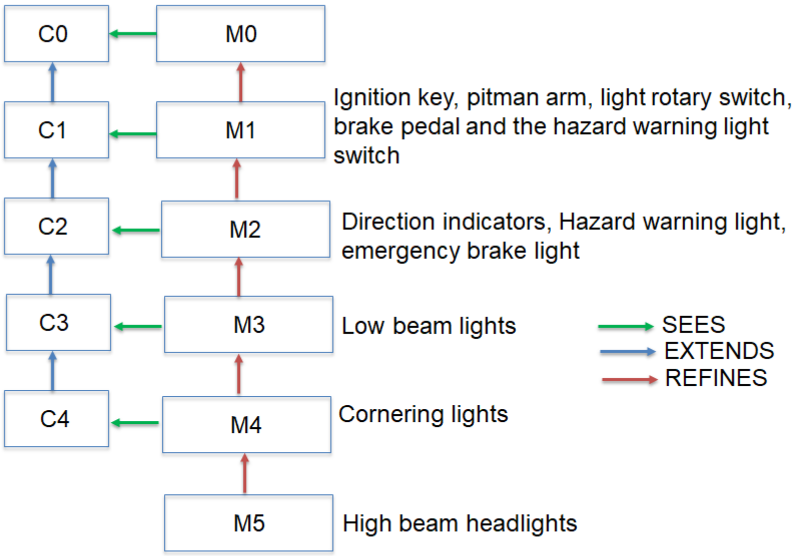

Fig. 2. Event-B structure of the project

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: $headingState := headingState \Leftarrow hl$
  end
END

Fig. 3. Machine M0

in [17], 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)

$$ELS16 = TRUE \land ELS16P = FALSE$$
$$\Rightarrow$$
$$keyState \neq KeyInIgnitionOnPosition \land$$
$$keyStateP \neq KeyInIgnitionOnPosition \land$$
$$lightSwitch = Auto \land lightSwitchP \neq Auto$$

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.

3 Model Details

In this section, we describe some specific ways of modelling that characterize our specification. The complete archive of the Event-B project is available in [16].

3.1 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:

Requirements [8]	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 [8]

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

In the context C1, we also define a constant *KeyMoves* to represent the authorized transitions for a key:

$$KeyMoves = \{NoKeyInserted \mapsto KeyInserted, KeyInserted \mapsto KeyInIgnitionOnPosition, KeyInIgnitionOnPosition \mapsto KeyInserted, KeyInserted \mapsto NoKeyInserted\}$$

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 Invariant inv2:

$$keyStateP \mapsto keyState \in KeyMoves \lor keyStateP = keyState$$

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: $hl \in LowBeams \cup tailLamps \cup directionIndicators \cup \{corneringLightLeft, corneringLightRight\} \nrightarrow LigntnessLevel$
grd2: $(keyState \mapsto valkey \in KeyMoves)$
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`. It uses an idiom to mimic a conditional **if** c **then** $x := v1$ **else** $x := v2$ construct, because the EVENT-B notation does not provide a conditional statement for actions. This idiom has the form

$$x := \{TRUE \mapsto v1, FALSE \mapsto v2\}(bool(c))$$

The term $\{TRUE \mapsto v1, FALSE \mapsto v2\}$ denotes a function, so it is evaluated at point $bool(c)$. Operator $bool(c)$ evaluates formula c and returns a result of the predefined set $BOOL = \{TRUE, FALSE\}$.

Event insertKey $\widehat{=}$
refines moveKey
  any
    hl
  where
    grd1: $hl \in LowBeams \cup tailLamps \cup directionIndicators \rightarrow LigntnessLevel$
    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.

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

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

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:

$$(... \vee dayTimeLightCont = TRUE) \wedge ... \wedge ELS16 = FALSE \wedge ... \Rightarrow headingState[LowBeams] = 100$$

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

3.3 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* ($currentTime \in N$) 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: $hl \in LowBeams \cup tailLamps \cup directionIndicators \cup \{corneringLightLeft, corneringLightRight\} \nrightarrow LigntnessLevel$
    grd2: $step \in N1$
  then
    act1: $headingState := headingState \lhd\!\!- hl$
    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. To model the timed part of the requirement ELS-18: *If the light rotary switch is in position Auto and the ignition is On, the low beam headlights are activated as soon as the exterior brightness is lower than a threshold of 200 lx. If the exterior brightness exceeds a threshold of 250 lx, ... In any case, the low beam headlights remain active at least for 3 seconds.*, we have defined the deadline variable *threeSecondsLater* which is updated when the following condition are fulfilled:

1. light rotary switch is in position Auto
2. the ignition is On,
3. the exterior brightness is lower than a threshold of 200

In that case, the requirement ELS-18 is modelled by the following invariant:

$$daytimeLights = FALSE \land brightnessSensor > 250 \land threeSecondsLater \neq 0 \land lightSwitch = Auto \land keyState = KeyInIgnitionOnPosition \Rightarrow headingState[LowBeams] = \{100\}$$

To satisfy this behaviour, the event putIgnitionOn is refined by adding the following action to set the deadline variable to the desired value:

$$threeSecondsLater := \{TRUE \mapsto currentTime + 30, FALSE \mapsto 0\}(\mathbf{bool}(brightnessSensor < 200 \land lightSwitch = Auto...))$$

Moreover, the event progress is refined by adding the following elements:

– a guard to prohibit time progression beyoud the deadline:

$$threeSecondsLater \neq 0 \Rightarrow currentTime + step \leq threeSecondsLater$$

– an action that resets the deadline *threeSecondsLater* if the three seconds are elapsed:

$$threeSecondsLater := \{TRUE \mapsto 0, FALSE \mapsto threeSecondsLater\}(\mathbf{bool}(currentTime + step = threeSecondsLater))$$

3.4 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.

4 Validation and Verification

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

4.1 Model checking of the specification

In this step, ProB is used as a model checker to ensure that the specification is free of invariant violation for trivial scenarios. 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

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

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. In an initial version of the event moveKey, the action act2 has been omitted causing the violation of the invariant inv2 for the trace execution depicted by Table 3. Indeed, the values of the variables *keyStateP* and *keyState* are different and the tuple $NoKeyInserted \mapsto KeyInIgnitionOnPosition$ does not belong to the set *KeyMoves* that represents the behavior of the key.

4.2 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 5 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 defined the event parameter *hl* as a partial function on the set of all the lights. The number fo such partial functions being very large, ProB could not terminate in a reasonable time. 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.

4.3 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* 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 4 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: $lowBeamRight \in dom(hl)$

Step	Event	keyStateP	keyState
1	Initialisation	*NoKeyInserted*	*NoKeyInserted*
2	moveKey	*NoKeyInserted*	*KeyInserted*
3	moveKey	*NoKeyInserted*	*KeyInIgnitionOnPosition*

Table 3. Execution trace violating an invariant

$\Rightarrow$
$hl(lowBeamRight) \in 0..100$

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

5 Other Points

5.1 Feedback on the requirements document

The formal modeling of the requirements document [8] 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. Table 4 gives the main modifications we made on the first release of the requirement 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 split into signals *pitmanArmForthBack* and *pitmanArmUpDown* with their corresponding positions (states) and the possible transitions between them.

5.2 Modeling temporal properties

Dealing with previous values to prove temporal properties turned out to be a significant burden. To improve and 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[3]. 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.

5.3 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 [23] or modularization [9], 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 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.

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

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. 4. Rodin proof statistics of the case study

5.4 Dealing with variable requirements

The requirement document of the case study includes the following three variability points:

- *driverPosition*: it states weather the vehicle is configured for left-hand or right-hand traffic.
- *armoredVehicle* indicates, if the current car is an armored vehicle or not.
- *marketCode parameter* specifies the market for which the car is to be built(001 = USA, 002 = Canada, 003 = EU).

In this case study, these variability elements induce that some functionalities are only available for specific values of these elements. For instance, the darkness switch being only available on armored cars, the requirements ELS-21 and ELS-24 make sense only for this kind of vehicles. Similarly, tail lamps are only used as rear direction indicator on USA and Canadian cars. Moreover, from the requirement document, we have not identify any element that would be impacted by the position driver. This is why we did not consider that in the formal modelling of the case study.

In Event-B, we defined two constants: *armoredVehicle* in the context C1 (*armoredVehicle*∈ `BOOL`) and *marketCode* in the context C2 (*marketCode*∈ {1, 2, 3}). Then, we have expressed the invariants corresponding to the related requirement by including conditions on the values of these constants. For instance, to specify that the darkness switch is only available for armored cars, we define an invariant that makes the variable *darknessModeSwitchOn* always false if the constant *armoredVehicle* is false:

$$armoredVehicle = FALSE \\ \Rightarrow \\ darknessModeSwitchOn = FALSE$$

Moreover, we included the guard ($armoredVehicle = TRUE$) in the event moveDarknessSwitch that models the actions on the darkness switch. Similarly, we model the flashing of the tail lamps as a partial function by stating that its domain is empty for European cars:

$$tailLampsFlash \in tailLamps \nrightarrow BOOL \land \\ (marketCode \in \{1, 2\} \Rightarrow dom(tailLampsFlash) = tailLamps \land \\ marketCode = 3 \Rightarrow dom(tailLampsFlash) = \varnothing)$$

6 Comparison

In the context of the ABZ conference, this case study has been dealt with using different approaches/techniques. In [11], a low level modelling using MISRA C, a programming language close to C, is presented. The requirement and the behavior of the system is directly coded in MISRA C, then the verification is performed in two steps. In a first step, simple requirements, related to single elements, are verified as unit test, then the CBMC model checker[6] is used to verify complex requirements that relate several elements. The authors report on some flaws/ambiguities but did not state how they dealt with them. Moreover, even if this approach has the advantage of directly producing the executable code, its correctness cannot be guaranteed since model checking does not ensure the absence of bugs.

In [3], a refinement-based approach, very similar to ours, using ASM [4] is presented. The modelling starts with a very abstract ASM which is then gradually refined by introducing more details. The validation of the developed models is carried out by animating them with the provided scenarios. The verification of the requirement is performed by applying a model-checking technique, using NuSMV, on the corresponding CTL/LTL formulas. As stated by the authors, since model-checking is only effective on finite state space, the domain of values have been restricted to be finite. As for the previous approach, model checking can not ensure that the specification is error-free.

In [7], Electrum [14], a formal language close to Alloy [10], is used for the modelling of the automotive light. The structural aspect of the system are modelled as signatures whereas its behavior is represented by predicates setting the output element according to the inputs of the system. The validation and the verification of the built

Version	Requiement	Initial specificaiton	Corrected specificaiton
1.2	ELS-8	...the hazard warning light switch is **released**	The term *released* being ambiguous, the term has been replaced by *pressed*.
1.2	ELS-12	When hazard warning is deactivated and the pit arm is in position “direction blinking left” or “direction blinking right”, the direction blinking cycle should be started.	The condition **ignition is On** is added.
1.2	ELS-14	If the ignition is *On*, the driver activates the low beam headlights by turning the light rotary switch to position *On*.	If the ignition is *On* and the light rotary switch is in the position *On*, then low beam headlights are activated.
1.2	ELS-15	If the ignition is off and the driver turns the light rotary switch to position *On*, the low beam headlights are activated with 50% (sidelight).	While the ignition is in position *KeyInserted*: if the light rotary switch is turned to the position *On*, the low beam headlights are activated with 50% (to save power). With additionally activated ambient light, ambient light control (Req. ELS-19) has priority over Req. ELS-15.
1.2	ELS-16	If the ignition is off and the driver turns the light rotary switch to position *Auto*, the low beam headlights remain off or are deactivated (depending on the previous state).	If the ignition is **already** off . . . (depending on the previous state). If ambient light is active, (see Req. ELS-19) ambient light delays the deactivation of the low beam headlamps.
1.2	ELS-19 ELS-20	With activated ambient light, the low beam headlights are activated as soon as at least one door of the vehicle is opened and the exterior brightness outside the vehicle is threshold 200 lx. The low beam headlights are deactivated lower than the as soon as all vehicle doors closed again. With activated ambient light, the low beam headlights are activated as soon as the engine is switched off and the ignition key is pulled out of the ignition lock. The low beam headlights (as ambient light) are deactivated as soon as none of the following actions occur within the next 30 seconds: (1) Opening or closing a door, (2) Insertion or removal of the ignition key.	Ambient light prolongs (keeps low beam headlamps at 100% if they have been active before) the activation of low beam headlamps(as ambient light) if ambient light has been activated, engine has been stopped (i.e. keyState changes from KeyInIgnitionOnPosition to NoKeyInserted or KeyInserted) and the exterior brightness outside the vehicle is lower than the threshold 200 lx. In this case, the low beam headlamps remain active or are activated. The low beam headlights are deactivated or parking light is activated (see Req.ELS-28) after 30 seconds. This time interval is reset by (1) Opening or closing a door, (2) Insertion or removal of the ignition key.
1.2	ELS-17	With activated daytime running light, the low beam headlights are activated after starting the engine. The daytime running light remains active as long as the ignition key is in the ignition lock(i.e. KeyInserted or KeyInIgnitionOnPosition). With activated ambient light, the low beam headlights remain active according to Req. ELS-20.	With activated daytime running light, . . . (i.e.. . . or KeyInIgnitionOnPosition). **With additionally activated ambient light, ambient light control (Req. ELS-19) has priority over daytime running light.**
1.2	ELS-18	If the light rotary switch is in position Auto, the low beam headlights are activated as soon as the exterior brightness is lower than a threshold of 200 lx. If the exterior brightness exceeds a threshold of 250 lx, the low beam headlights are deactivated. In any case, the low beam headlights remain active at least for 3 seconds.	If the light rotary switch is in position Auto **and the ignition is *On***, the low beam. . . at least for 3 seconds. .
1.2	ELS-28	The parking light is the low beam and the tail lamp on the left or right side of the vehicle to illuminate the vehicle if it is parked on a dark road at night. The parking light is activated, if the key is not inserted, the light switch is in position *On*, and the pitman arm is engaged in position left or right (②/③). To save battery charge, the parking light is activated with only 10% brightness of the normal low beam lamp and tail lamp.	Adding the requirement: An active ambient light (see Req. ELS-19) delays parking light.
1.2	ELS-34	If the camera recognizes the lights of an advancing vehicle, an activated high beam headlight is reduced to low beam headlight within 0.5 seconds by reducing the area of illumination to 65 meters by an adjustment of the headlight position as well as by reduction of the luminous strength.	Adding a precision on the reduction amount : **by reduction of the luminous strength to 30%.**

Table 4: Requirement clarification and update

specification is achieved into two steps. During the validation phase, the authors first define a number of scenarios to check requirements related to simple behaviours in order to rapidly detect some obvious consistencies, Then to check more complex scenarios, like those provided in the case study description document, a validator has been implemented. This validator permits to check whether there exists a valid trace that produces given outputs from specific values of inputs. The validator is also used to animate the model on a set of inputs and to produce outputs that are validated by the domain experts. In a last step, the requirements, as described in the document, are modelled as assertions to be checked on the developed models. As stated by the authors, these different validation/verification steps have permitted to detect some flaws and ambiguities reported to the case study chair. These flaws/ambiguities include the need for requirement prioritisation and infeasible scenarios. Due to limitations of ELECTRUM (representation of concrete integer values and time), the time requirements and those involving arithmetic calculation have not been considered.

Classical B and EVENT-B have been used in [13] to model a subset of the same case study (blinking lamps and Pitman controller). Classical B is used to take advantage of its specification modularization capabilities, and EVENT-B is used to take advantage of its stronger proving environment. The proposed approach proceeds into three steps: (1) Modeling independently the behaviors of the different elements with operations defined in separate machines; (2) Defining a new machine to relate dependent elements; this new machine includes the machines corresponding to these elements and defines operations that calls that operations defined in the included machines; (3) Translating the obtained B specification into EVENT-B for verification purpose. In this paper, the authors model time as we did in [20]. The approach also permitted to detect some inconsistencies during the model-checking of the specification using PROB, for which the authors propose some corrections.

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 % [19], [18].

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 right from 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.R.: The B-book - Assigning Programs to Meanings. Cambridge University Press (1996)
2. Abrial, J.: Modeling in Event-B. Cambridge University Press (2010)
3. Arcaini, P., Bonfanti, S., Gargantini, A., Riccobene, E., Scandurra, P.: Modelling an Automotive Software-Intensive System with Adaptive Features Using ASMETA. In: Raschke, A., Méry, D., Houdek, F. (eds.) Rig-

orous State-Based Methods - 7th International Conference, ABZ 2020, Ulm, Germany, May 27-29, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12071, pp. 302–317. Springer (2020)
4. Börger, E., Stärk, R.F.: Abstract State Machines. A Method for High-Level System Design and Analysis. Springer (2003)
5. Clarke, E.M., Emerson, E.A.: Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking - History, Achievements, Perspectives. Lecture Notes in Computer Science, vol. 5000, pp. 196–215. Springer (2008)
6. Clarke, E.M., Kroening, D., Lerda, F.: A Tool for Checking ANSI-C Programs. In: Jensen, K., Podelski, A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2988, pp. 168–176. Springer (2004)
7. Cunha, A., Macedo, N., Liu, C.: Validating Multiple Variants of an Automotive Light System with Electrum. In: Raschke, A., Méry, D., Houdek, F. (eds.) Rigorous State-Based Methods - 7th International Conference, ABZ 2020, Ulm, Germany, May 27-29, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12071, pp. 318–334. Springer (2020)
8. Houdek, F., Raschke, A.: Adaptive Exterior Light and Speed Control System. `https://abz2020.uni-ulm.de/case-study#Specification-Document` (November 2019)
9. 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. vol. 5977, pp. 174–188. Springer, Berlin (2010)
10. Jackson, D.: Software Abstractions - Logic, Language, and Analysis. MIT Press (2006)
11. Krings, S., Körner, P., Dunkelau, J., Rutenkolk, C.: A Verified Low-Level Implementation of the Adaptive Exterior Light and Speed Control System. In: Raschke, A., Méry, D., Houdek, F. (eds.) Rigorous State-Based Methods - 7th International Conference, ABZ 2020, Ulm, Germany, May 27-29, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12071, pp. 382–397. Springer (2020)
12. Leuschel, M., Butler, M.J.: ProB: An Automated Analysis Toolset for the B Method. International Journal on Software Tools for Technology Transfer 10(2), 185–203 (2008)
13. Leuschel, M., Mutz, M., Werth, M.: Modelling and Validating an Automotive System in Classical B and Event-B. In: Raschke, A., Méry, D., Houdek, F. (eds.) Rigorous State-Based Methods - 7th International Conference, ABZ 2020, Ulm, Germany, May 27-29, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12071, pp. 335–350. Springer (2020)
14. Macedo, N., Brunel, J., Chemouil, D., Cunha, A., Kuperberg, D.: Lightweight specification and analysis of dynamic systems with rich configurations. In: Zimmermann, T., Cleland-Huang, J., Su, Z. (eds.) Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016. pp. 373–383. ACM (2016)
15. Mammar, A., Frappier, M., Laleau, R.: An Event-B Model of an Automotive Adaptive Exterior Light System. In: Raschke, A., Méry, D., Houdek, F. (eds.) Rigorous State-Based Methods - 7th International Conference, ABZ 2020, Ulm, Germany, May 27-29, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12071, pp. 351–366. Springer (2020)
16. 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/STTT2022/LightControlSystem.html` (May 2022)
17. 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)
18. 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)
19. 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)
20. Mammar, A., Laleau, R.: Modeling a landing gear system in event-b. Int. J. Softw. Tools Technol. Transf. 19(2), 167–186 (2017)
21. Parnas, D.L., Madey, J.: Functional Documents for Computer Systems. Science of Computer Programming 25(1), 41–61 (1995)
22. Pnueli, A.: The Temporal Logic of Programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence. pp. 46–57. IEEE Computer Society (1977)
23. Silva, R., Pascal, C., Hoang, T.S., Butler, M.J.: Decomposition tool for Event-B. Software: Practice and Experience 41(2), 199–208 (2011)