Modeling of a Speed Control System using Event-B *

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

[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

Abstract. The present paper presents our proposal of an Event-B model of a speed control system available in some German cars, a part of the case study provided in the ABZ2020 conference that will take place in the Ulm city. The case study describes how the system regulates the current speed of a car according to a set criteria like the speed desired by the driver, the position of a possible preceding vehicle but also a given speed limit that the driver must not exceed. For that purpose, this controller reads different information form the available sensors (key state, desired speed, etc.) and takes the adequate actions by acting on the actuators of the car's speed according to the read information. To formally model this system, we adopt a stepwise refinement approach with the Event-B method. We consider most 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 contained in the first versions of the case study.

Keywords: Speed control system, Event-B method, Refinement, Verification

1 Introduction

The case study, proposed in the context of the ABZ2020 conference that will take place in the city of Ulm, is composed of two parts: *Adaptive Exterior Light and Speed Control Systems* that equipped some German cars. Since the whole case study is quite lengthy/complex and the two parts are only loosely coupled as stated in the description document, we chose to handle each part in a separate paper. The present paper deals with the speed control system whereas a companion paper considers the adaptive exterior light system [7].

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

The goal of the speed control system is to regulate the current speed of a car according to a set of criteria like the speed desired by the driver, the position of a possible preceding vehicle but also a given speed limit that the driver must not exceed. The system can behave according to two options: the first one regulates the speed independently on the any preceding vehicle, the second component takes into account the position of a possible preceding vehicle by maintaining a security distance. The driver has the possibility to choose which option to activate at a given moment. Like a controller, in both options, the system reads different informations from the available sensors (key state, desired speed, the preceding vehicle position, etc.) and takes the adequate actions by acting on the actuators of the car's speed according to the read information.

The present paper describes the formal modeling of the speed control system using the Event-B method and its refinement technique that permits to master the complexity of a system by gradually introducing its different elements/characteristics. Proposed by Abrial as a successor of the B method [1], the Event-B method [2] permits to model discrete systems using mathematical notations. 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. The dynamic part is included in a machine that defines variables and a set of events. The possible values that the variables can hold are specified by an invariant using a first-order formula on the state variables. The different machines composing an Event-B specification are related with a refinement relation whereas the contexts are linked with an extension link (extends). Each refinement adds new information to a model; these could be new state variables, new events or new properties. Event-B refinement allows for guard strengthening, nondeterminism reduction, and new events introduction. New events of a model M' that refines a mode M are considered to refine a *skip* event of M, hence they cannot modify a variable of M. Therefore, all events that need to modify a variable v must be defined in the same model where v is first introduced. The correctness of an Event-B model is ensured by proof obligations that verify that the invariant is preserved by each event and that the refinement preserves the properties of the system.

The development of our Event-B models has been done under the Rodin platform [3] that provides editors, provers and several other plugins for various tasks like animation and model checking with ProB [5]. We use ProB in order to animate the built models with two purposes: exhibiting the problematic scenarios that violate the invariant prior to the hard/long proof phase, but also validating the specification by playing the provided scenarios in order to be sure that we have specified the right system.

The rest of this paper is structured as follows. Section 2 describes our modelling strategy. Section 3 describes our model in more details. Section 4 describes the validation and verification of our model. 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. We conclude in 6.

2 Modelling Strategy

The speed control system subject of this paper can be seen as a control system that interacts with its environment through a set of sensors, which provide it with information about the state of the physical elements, and a set of actuators that are used to transmit the adequate orders to these elements. In this paper, we use the concepts described in [10]. A sensor measures the value of some environment elements 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. 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. In this case study, we use Event-B state variables to represent both environment and controller variables. We do not model sensor or actuator failures.

A well-known architecture of a control system is a control loop that reads all input variables at once, at a given moment, and then computes all output variables in the same iteration. But, it can be also viewed as a continuous system that can be interrupted by any change in the environment represented by a new value sent by a sensor. In this paper, we see the controller as a distributed system; each sub-system is associated to a given sensor. In that case, the system reacts to each single modification of the sensor. This approach can be seen as 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 Model Details

This section briefly describes the main modeling elements that characterize our specification. The complete archive of the Event-B project is available in [6]. Our model contains 4 contexts and 4 machines/refinements. Table 1 relates the components of our model with the requirements listed in [4]. As one can remark, some requirements are modelled as invariants whereas others are dealt with in the adequate events. We chose to do not model some requirements as invariants because this would make the modeling and the proof activities more complex and difficult. Requirement SCS-41 for example: "*... the self-test of the radar*

system is restarted every 10 min" is modeled by a variable *nextTest* that is set to the current time plus 10 minutes in the events that represent the movement of the key and the progression of the time because the self-test of the radar system should be performed at the start (the key is in the ignition position) and then every 10 minutes. Modeling this requirement as an invariant would require the introduction of two extra variables to store the moments of two consecutive self-test activities, then we have to state that 10 minutes should be elapsed between these two moments.

Requirements [4]	**Component**	**Invariant/Event**
SCS-1, SCS-31	M2	i$inv4$
SCS-2, SCS-31	M2	i$inv5$
SCS-3, SCS-12, SCS-13, SCS-16, SCS-17, SCS-31	M2	$inv6$ and $inv7$
SCS-4, SCS-19, SCS-31	M2	i$inv8$
SCS-5, SCS-19, SCS-31	M2	i$inv9$
SCS-6, SCS-19, SCS-31	M2	i$inv10$ and $inv11$
SCS-7, SCS-19, SCS-31	M2	$inv12$
SCS-8, SCS-19, SCS-31	M2	$inv13$
SCS-9, SCS-31	M2	$inv14$
SCS-10, SCS-31	M2	$inv15$
SCS-11, SCS-31	M2	$inv16$
SCS-15	M3	$inv13$
SCS-20	M3	$inv12$
SCS-21		not covered
SCS-22	M3	$inv11$
SCS-23	M3	$inv14$, $inv15$ and $inv16$
SCS-24	M3	$inv17$
SCS-25	M3	$inv19$
SCS-26	M3	$inv20$
SCS-27-SCS-28		not covered
SCS-29	M3	Event moveSpeedLimiterSwitch
SCS-30		not covered since it is related to the interface appearance
SCS-32, SCS-33, SCS-34	M2	$inv21$
SCS-35	M2	$inv26$ and moveSpeedLimiterSwitch
SCS-36 to SCS-39	M2	$inv24$
SCS-40 and SCS-41	M2	Event moveKey and progress
SCS-42	M3	$inv13$

Table 1. Cross-reference between the components of our model and the requirements of [4]

Machine $M0$ models the current speed of the studied car independently from any preceding vehicle and also without giving any condition on its evolution. This machine defines the following unique invariant:

$$currentSpeed \in rangeSpeed$$

where $rangeSpeed$ denotes a constant defined in the context $C0$ to set the range values for the speed ($rangeSpeed = 0..5000$). Machine $M0$ defines a unique event $updateVehiculeSpeed$ to set the current speed of the car as follows:

Event updateVehiculeSpeed ⟨ordinary⟩ $\widehat{=}$
any
val
where
grd1: $val \in rangeSpeed$
then
act1: $currentSpeed := val$
end

Machine $M1$ introduces the physical elements that are manipulated by the driver and that have an impact on the current speed of the car. These elements include gas/brake pedal, key, cruise control lever, etc. Machine $M1$ describes how the position of each of these elements evolves depending on its current position. In this same machine, we also introduce the event progress that makes the current time progressing. Machine $M2$ models the desired speed together with the activation of the normal/adaptive cruise control and also the traffic sign detection that has an impact on the value of the desired speed according to the requirements (SCS-36,SCS-39). It is worth noting that some events, like that related to the traffic sign detection, are introduced in $M1$ even if this aspect is really dealt with in Machine $M2$. Indeed, these events need to modify some variables that are introduced in $M1$ and, as noted before, a new event cannot modify a variable defined in a previous refinement level. Machine $M3$ specifies the different aspects that depend on or impact the desired/current speed like speed dependent safety distance that also depends on the speed of the preceding vehicle but also the faults that can happen on the radar system. The main elements of these Event-B components are described hereafter.

3.1 Machine M1: physical elements

This machine refines Machine M0 by introducing the different elements that impact the current speed of the car. This includes the physical elements that the driver manipulates but also the time progression since it makes some variables evolve like the desired speed. For that purpose, several variables/invariants are introduced to model how the position of the physical elements evolves depending on its current position. In this paper, we give details about the cruise control lever modeled by the variable $SCSLeverUD$ and its typing invariant: $SCSLeverUD \in SCSLeverPositions$ where $SCSLeverPositions$ is a given set defined in Context $C1$ that extends $C0$:

$partition(SCSLeverPositions, Upward, Downward,$
$\qquad\qquad\qquad\qquad \{Backward\}, \{Forward\}, \{Neutral\})$
$partition(Upward, \{Upward5\}, \{Upward7\})$
$partition(Downward, \{Downward5\}, \{Downward7\})$

For each of these elements, invariants are defined in Machine $M1$ to specify the authorized position changes together with the event that models them. The following invariant states that the cruise control level cannot directly move from an *Upward* position to a *Downward* position bypassing the *Neutral* position. As we can remark, the above invariant uses an extra variable $SCSLeverUDP$ to model the previous position of the cruise control level. In the next section, we show that this kind of variables are also relevant for modeling some requirements that need to make reference to the current and previous states of the system.

$$
\begin{array}{l}
SCSLeverUDP \neq Neutral \implies \\
\quad SCSLeverUD = SCSLeverUDP \\
\qquad \lor \\
\quad (SCSLeverUDP \in Upward \land SCSLeverUD \in Upward) \\
\qquad \lor \\
\quad (SCSLeverUDP \in Downward \land SCSLeverUD \in Downward) \\
\qquad \lor \\
\quad SCSLeverUD = Neutral
\end{array}
$$

Machine $M1$ defines a set of events, among others, event moveSCSLeverUD that models the cruise control level movements:

Event moveSCSLeverUD ⟨ordinary⟩ $\widehat{=}$
any
valSCS
where
grd1: $valSCS \in Upward \cup Downward \cup \{Neutral\}$
grd2: $SCSLeverUD \neq Neutral \implies$
$(SCSLeverUD \in Upward \land valSCS \in Upward)$
$\lor$
$(SCSLeverUD \in Downward \land valSCS \in Downward)$
then
act1: $SCSLeverUD := valSCS$
act2: $SCSLeverUDP := SCSLeverUD$
end

Guard $grd2$ permits to make the invariant preserved after the observation of this event. In addition, as stated before, an event progress making a variable $currentTime$ increases by a tenth of a second is defined in $M1$ to model the time progression. We choose a progression step of a tenth of a second because some data in the case study are with 0.1 precision.

3.2 Machine M2: desired speed

This machine describes how the desired speed evolves according to the requirements (SCS-1 to SCS-12) by moving the cruise control level into different positions. We also model the activation of the normal/adaptive cruise control as described in the document. In addition, we specify the speed limit requirements (SCS-29 to SCS-34) because the calculation of the current speed must respect such a limit.

Mainly, this machine introduces some additional variables to model the desired speed (*desiredSpeed*) and the normal/adaptive cruise control (*normContr* and *adapContr*) with their associated variables to represent their previous values. Moreover, as the evolution of the desired speed depends on the time, we store the last time ($lastTimeSCSLeverUD$) when the cruise control level has been in the Up/down positions. Thus requirements SCS-4 and SCS-7 are modeled as follows. Requirement SCS-4 specifies that, while the cruise control is activated, the desired speed increases by 1 the first time the cruise control level is put in position $Upward5$ whereas Requirement SCS-7 states that the desired speed continues to increase by 1 by each second as long as the cruise control level stays in that position for more than 2 seconds. Variable *lastdesiredSpeed* represents the desired speed when the lever has been moved into a given position.

$$\begin{array}{c} SCSLeverUDP \neq Upward5 \land SCSLeverUD = Upward5 \\ \land \\ (adapContrP = TRUE \lor normContrP = TRUE) \\ \Longrightarrow \\ desiredSpeed = min(\{200, desiredSpeedP + 1\}) \end{array}$$

and

$$\begin{array}{c} (normContr = TRUE \lor adapContr = TRUE) \land SCSLeverUD = Upward5 \\ \land \\ currentTime - lastTimeSCSLeverUD \geq 20 \\ \Longrightarrow \\ desiredSpeed = \\ min(\{200, lastdesiredSpeed + (currentTime - lastTimeSCSLeverUD - 10) \div 10\}) \end{array}$$

Let us give more explanation about the last invariant. Expression ($currentTime - lastTimeSCSLeverUD - 10$) permits to update the desired speed immediately after 2 seconds, this is why we subtract 10 units of time and not 20. As stated before, as we chose a progression step of tenth of a second, we must divide by 10 each data related to the time. To make these invariants preserved, we have refined the moveSCSLeverUD event according to Requirement SCS-4 but also the progress event with respect to Requirement SCS-7. Event progress for instance is refined by adding the following guard that calculates the new desired speed:

despeed=
{TRUE ↦
{TRUE ↦

$$
\begin{array}{l}
\quad \{\text{TRUE} \mapsto min(\{200, lastdesiredSpeed + \\
\qquad\qquad ((currentTime + 1 - lastTimeSCSLeverUD - 10) \div 10)\}), \\
\quad\ \text{FALSE} \mapsto min(\{200, lastdesiredSpeed + \\
\qquad\qquad ((currentTime + 1 - lastTimeSCSLeverUD) \div 2)\}) \\
\quad\} (\mathbf{bool}(SCSLeverUD = Upward5)), \\
\text{FALSE} \mapsto \\
\quad \{\text{TRUE} \mapsto max(\{10, lastdesiredSpeed - \\
\qquad\qquad (currentTime + 1 - lastTimeSCSLeverUD - 10) \div 10\}), \\
\quad\ \text{FALSE} \mapsto max(\{10, lastdesiredSpeed - \\
\qquad\qquad ((currentTime + 1 - lastTimeSCSLeverUD) \div 2)\}) \\
\quad\}(\mathbf{bool}(SCSLeverUD = Downward5)) \\
\}(\mathbf{bool}(SCSLeverUD \in Upward)), \\
FALSE \mapsto desiredSpeed \\
\}(\mathbf{bool}(currentTime + 1 - lastTimeSCSLeverUD \geq 20))
\end{array}
$$

The above guard distinguishes different cases according to the position of the control lever and the time elapsed since its last position change($currentTime + 1 - lastTimeSCSLeverUD \geq 20$). The term ($currentTime + 1$) denotes the after-value of $currentTime$ when Event progress is observed.

Let us note that the Event-B method and its underlying language is not well-adapted to model the evolution of the speed vehicle according to its acceleration/speed and the time passing. Indeed, since the language does not support real numbers, we model the current speed as an integer amount that evolves according to the usual equation ($V = \gamma \times t + V_p$) where the γ represents the acceleration/deceleration of the vehicle, ($t = 1$) the time progression and V_p the previous speed. As our time progression is by a tenth of a second, the progression of the speed is very small, that is, less than one kilometer. This progression can not be taken into account using the B language. To overcome such a limit, we proceed as follows. We do not include the increasing/decreasing of the current speed in the event that makes the time progress but we introduce a new event setSpeed that set the current speed to a given value. This also permits to play and produces the scenarios provided in the case study.

3.3 Machine M3: other elements

In this level, we model the different aspects that depend on or impact the desired/current speed, like speed dependent safety distance and the speed of the preceding vehicle. Moreover, we model the faults that can happen on the radar system. Machine $M3$ introduces two new events turnHead and VehicHeadDetect to model respectively the selection of a safety level by turning the cruise control lever head and the detection of a preceding vehicle by catching its speed that is relevant for determining the speed dependent safety distance and also to make the system decelerates if it is necessary. Event VehicHeadDetect for instance is specified as follows:

Event VehicHeadDetect ⟨ordinary⟩ $\widehat{=}$

any
val stv brk secdis speh
where
grd1: $val \in rangeRadarSensorValues$
grd2: $rangeRadarState = FALSE \Leftrightarrow val = 255$
grd3: $speh \in rangeSpeed$
grd4: $speh \leq 200 \wedge speedOfHead > speh \wedge speh \neq 0 \wedge adapContr = TRUE \wedge val \notin \{0, 255\} \implies secdis = 25 \times currentSpeed \div 360$
grd5: $speh = 0 \wedge currentSpeed = 0 \wedge adapContr = TRUE \wedge val \notin \{0, 255\} \implies secdis = 2$
grd6: $speedOfHead < speh \wedge speh \neq 0 \wedge speh \leq 200 \wedge adapContr = TRUE \wedge val \notin \{0, 255\} \implies secdis = 30 \times currentSpeed \div 360$
grd7: $speh > 200 \wedge adapContr = TRUE \implies secdis = safetyDistance \times currentSpeed \div 360$
grd8: ...
then
act1: $rangeRadarSensor := val$
act2: $speedOfHead := speh$
act3: securedistanceToHead:= secdis
act4: ...
end

Event parameter *val* represents the distance between the studied car and a possible preceding vehicle as provided by the radar. Guard *grd*2 states that such a value should be equal to 255 if the radar system is not ready. Guards *grd*4-*grd*7 permits to calculated the new value for the speed dependent safety distance according to the requirements SCS-23 and SCS-24 with the event parameter *speh* denoting the speed of the preceding vehicle.

Already existing events of $M2$ are refined in $M3$ in similar way by calculating the value of the different variables. For instance, the desired speed should be updated when a traffic sign is detected, the speed dependent safety distance is updated when the current speed is modified or the speed of a preceding vehicle changes. More details can be found in [6].

4 Validation and Verification

To ensure the correctness and validate the built Event-B models, we have proceeded into three steps detailed hereafter.

4.1 Model checking of the specification

We used the ProB tool as a model checker in order to ensure that all the invariants of each machine are preserved after the observation of each event, that is, there is no sequence of events that makes an invariant not satisfied. Basically, when an invariant becomes violated, ProB exhibits such a sequence of events that, starting from a valid initial state of the machine, leading to a state that violates the related invariant. Such specification errors can be due to a guard/action missing, to an incorrect specification of the invariant but sometimes also to an incorrect property, that is the system really does not satisfy the property. Let us note that even if no invariant violation is found by the tool, there may still exist scenarios that violate the invariant that the tool cannot find due to their complexity or/and the timeout on the model checking process. This is why a proof phase should be performed to ensure that the specification is invariant-violation free.

4.2 Validation with scenarios

This step aims at verifying that we have built the right model whose behaviors conform to the desired ones as described by the scenarios of the specification document. For that purpose, the animation capability of ProB is used to play the different scenarios provided in the case study. This step allows us to point out some flaws/ambiguities in the initial release of the description document. For instance, the initial examples provided to illustrate the requirements SCS-5-SCS-9 were incorrect with respect to the requirements. In addition, in some place like SCS-7-SCS-9, the term "target speed" is used instead of "desired speed", etc. All these aspects have been discussed with the case study authors because we are not specialists of the domain. Let us note that we have faced some difficulties to play the provided scenarios since no information is provided on how the controller calculates the acceleration at each step. So, we have made our best to "simulate" these values without any representation about their suitability, reliability.

4.3 Proof of the specification

This last phase aims at ensuring the correctness of the specification by discharging all the proof obligations generated by Rodin to prove that the invariants are preserved by each event, but also that the guard of each refined event is stronger than that of the abstract one. Figure 1 provides the proof statistics of the case study: 579 proof obligations have been generated, of which 60% (345) 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 even if we had to add some theorems on min/max operators (a min/max of a finite set is an element of the set, etc) but also on the transitivity property of the comparison operator ($\geq$, $\leq$, etc.).

Element Name	Total	Auto	Manual	Reviewed	Undischarged
SCS_012020	**579**	**345**	**234**	**0**	**0**
C0	0	0	0	0	0
C1	0	0	0	0	0
C2	18	15	3	0	0
C3	0	0	0	0	0
M0	2	2	0	0	0
M1	78	72	6	0	0
M2	215	104	111	0	0
M3	266	152	114	0	0

Fig. 1. RODIN proof statistics of the case study

5 Other Points

This section reports on some points about the choices made during the Event-B modeling of the speed control system.

5.1 Feedback on the specification document

The formal modeling of the specification document [4] lead us to question ourselves about the semantics of some requirement and identify a number of ambiguities and some contradictions with the test scenarios provided. Being not specialist of the domain, we have communicated these to the authors of the requirements document, and a number of revisions were produced, following our comments. Our discussion and exchange lead to the modification/revision of a set of requirements to make them clearer and consistent. A detailed list of these elements are described in the last version (*i.e.*, 1.17) of the requirements document. We have mainly clarified/corrected the examples provided for the requirements SCS-5, SCS-7, SCS-8 and SCS-9. We have also adjusted the maximum acceleration and deceleration values in SCS-20, SCS-22 and SCS-23. Finally to make the modeling easier and after a discussion with the case study authors, the signal *SCSLever* has been splitted into signals *SCSLeverForthBack* and *SCSLeverUpDown* with their corresponding positions (states) and the possible transitions between them.

5.2 Modeling temporal properties

As stated before, a number of requirements refer to the current and previous state of an element. In order to be able to verify these requirements using a proof strategy, we modeled them as invariants by introducing two variables for each element to store their current and previous values. The obtained specification

is quite cumbersome especially that we have to add for each event that does not modify a variable that its previous value is equal to its current value. We think that it would be interesting to investigate existing tools/approach that could help us specify this kind of properties in a simpler manner. An example of such tools is the Event-B State machines plugin[3] that produces 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. This makes difficult to trace and justify the usefulness of the generated guards.

6 Conclusion

This paper presents a formal modeling proposal of a speed control system using the Event-B method. We have modelled most of requirements that permits us to point out some ambiguities in the requirements that we have discussed and clarified with the case study authors by rephrasing them. These ambiguities have been discovered during during different development phases: formalization, proof and validation using the provided scenarios. This experience has affirmed that the formal modeling of a system helps the software users detect error in early development phase that makes its correction cheaper.

The main difficulty when modeling the speed control system is to determine the order in which elements should be introduced during the refinement especially that many elements are interdependent. Due to time constraints, we were unfortunately not able to explore the different decomposition plugins of Rodin that might produce smaller specification parts that would be easier to understand and maintain. We plan to explore some decomposition techniques as future work. The work presented in this paper can also be extended by considering the remaining requirements that need more clarifications. Requirement SCS-21 for instance needs more information on how the system can deduce that deceleration of $3m/s^2$ is insufficient to prevent a collision without having any information about the acceleration of the preceding vehicle. Also, we think that more information should be provided on the internal variables like $setVehicleSpeed$ that represents the automatic acceleration of the system in order to able to build a more complete system. Finally through the different case studies proposed in the ABZ conference [9,8], we are now convinced of the need to improve the Event-B language to make it supports the real numbers as basic types. Its prover should be also extended to include more rules on arithmetic and set theories.

Acknowledgements 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.

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

References

1. Abrial, J.: The B-book - assigning programs to meanings. Cambridge University Press (1996)
2. Abrial, J.: Modeling in Event-B. Cambridge University Press (2010)
3. Consortium, E.B.: http://www.event-b.org/
4. Houdek, F., Raschke, A.: Adaptive Exterior Light and Speed Control System. https://abz2020.uni-ulm.de/case-study#Specification-Document (November 2019)
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.: Modeling of a Speed Control System using Event-B. http://www-public.imtbs-tsp.eu/~mammar_a/SpeedControl.html (January 2020)
7. 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)
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. Parnas, D.L., Madey, J.: Functional Documents for Computer Systems. Science of Computer Programming **25**(1), 41–61 (1995)