Proof-Based Approach to Hybrid Systems Development: Dynamic Logic and Event-B

Guillaume Dupont, Yamine Aït-Ameur, Marc Pantel, Neeraj Kumar Singh

▶ **To cite this version:**

Guillaume Dupont, Yamine Aït-Ameur, Marc Pantel, Neeraj Kumar Singh. Proof-Based Approach to Hybrid Systems Development: Dynamic Logic and Event-B. 6th International Conference Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ 2018), University of Southampton, Jun 2018, Southampton, United Kingdom. pp.155-170. hal-02450998

HAL Id: hal-02450998
https://hal.science/hal-02450998

Submitted on 23 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Open Archive Toulouse Archive Ouverte

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible

This is an author's version published in:
http://oatao.univ-toulouse.fr/24887

Official URL
DOI : https://doi.org/10.1007/978-3-319-91271-4_11

To cite this version: Dupont, Guillaume and Ait Ameur, Yamine and Pantel, Marc and Singh, Neeraj *Proof-Based Approach to Hybrid Systems Development: Dynamic Logic and Event-B.* (2018) In: International Conference Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ 2018), 5 June 2018 - 8 June 2018 (Southampton, United Kingdom).

Any correspondence concerning this service should be sent to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr

Proof-Based Approach to Hybrid Systems Development: Dynamic Logic and Event-B

Guillaume Dupont(✉), Yamine Aït-Ameur, Marc Pantel, and Neeraj Kumar Singh

INPT-ENSEEIHT/IRIT, University of Toulouse, Toulouse, France
{guillaume.dupont,yamine,marc.pantel,nsingh}@enseeiht.fr

Abstract. The design of hybrid systems controllers requires one to handle both discrete and continuous functionalities in a single development framework. In this paper, we propose the design and verification of such controllers using a *correct-by-construction* approach. We use proof-based formal methods to model and verify the required safety properties of the given controllers. Both Event-B with Rodin, and hybrid programs and dynamic differential logic with KeYmaera are experimented on a common case study related to the modelling of a car controller. Finally, we discuss the lessons learnt from these experiments and draw the first steps towards a generic method for modelling hybrid systems in Event-B.

Keywords: Hybrid systems · Event-B · Hybrid programs
Differential dynamic logic · Proofs · Refinement

1 Introduction

Hybrid systems are present in many safety-critical applications. Thus, the formal verification of hybrid systems is a key issue in system engineering. Contrary to classical discrete systems, the formal specification and verification of hybrid systems require taking into account continuous features like differential equations to characterise plant behaviours and the appropriate logic and proof system. Several research works addressed the formal verification of hybrid systems [4]. Hybrid model-checking, proof based approaches, program analysis and simulation have been proposed. These approaches consider a hybrid system as the tight integration of discrete and concrete features defining models for controllers and for the plant to be controlled.

However, addressing formal verification at the model level allows for the abstraction of some implementation details, and in particular floating point arithmetic. The verification of the defined models and the synthesis of controllers guarantees the satisfaction of system requirements independently of any implementation. Only implementation requirements like floating point computation will remain to be addressed once code is obtained from the verified models.

https://doi.org/10.1007/978-3-319-91271-4_11

Our paper deals with correct-by-construction approaches with refinement and proof-based techniques. We propose to handle the integration of continuous and discrete behaviours in Event-B [1] and in the Rodin [2] platform to develop hybrid systems models. This approach requires the modelling of continuous mathematical concepts which are not currently available in Event-B. For this purpose we use the *Theory* plug-in developed for Rodin [13] to define a theory for continuous functions and differential equations that we need to handle in our Event-B models.

In order to position our work, we select the approach of Platzer as a second formal development technique. This choice is motivated by the defined proof-based approach and the availability of a tool. Hence, we show how dynamic differential logic [15] and KeYmaeara [16] are set up for the same objective as ours. A case study of a car controller, borrowed from Platzer's work, is used to illustrate both approaches. As a second step, we present a generalisation of our approach with Event-B in order to reduce the effort needed for feasibility proofs during model instantiation.

This paper is organised as follows. The next section presents the case study we have chosen to illustrate our approach. Event-B and dynamic differential logic are summarised in Sect. 3 and their use for the development of the case study is described in Sect. 4. The methodological lessons learnt from these developments are discussed in Sect. 5. Finally, the last section is devoted to concluding remarks and a research agenda.

2 Case-Study

The chosen case study deals with a *stop sign controller* proposed by Quesel et al. in [17] (Sect. 5.3). It consists in modelling a car controller that automatically stops a car at a stop-sign (SP).

Behavioural Requirements. The car is modelled through its horizontal position and behaves according to three modes, each of which corresponds to a particular *acceleration*. The given accelerations are constant and, as a matter of simplification, wrap every potential physical phenomena (air and road friction for instance). These modes are defined as follows.

- **Accelerating:** the car increases its velocity. In this case, the associated acceleration, denoted by A, is positive.
- **Braking:** the velocity of the car decreases. In this case, the associated acceleration is $-B$, where B denotes the braking power (positive).
- **Stabilizing:** the velocity of the car does not change. In this case, the associated acceleration is 0.

The system is modelled by its position (p), velocity (v) and acceleration (a), which evolve according to the differential equation: $\dot{p} = v, \dot{v} = a$, where the dot stands for *time derivative*.

At initialisation, the system is in *stabilizing* mode. The car is given an arbitrary initial position and velocity denoted as p_0 and v_0, respectively.

Safety Requirements. The system shall observe two invariant properties.

- **SAF1.** The velocity of the car cannot be negative.
- **SAF2.** The position of the car never exceeds the stop sign position SP.

Note that the two safety requirements are of different nature. **SAF1** has a purely physical origin whereas **SAF2** is a behavioural system requirement.

3 Two Proof-Based Methods

To address the case study presented in Sect. 2 we considered two different approaches. A first one is based on differential logic (d$\mathcal{L}$) and KeYmaera to express and prove an hybrid controller, and the second one uses Event-B and Rodin to express the system using events and invariants.

3.1 Hybrid-Programs/Dynamic Logic with KeYmaera

The seminal work of [15] led to the definition of a rigorous method to model controllers for hybrid systems integrating both continuous and discrete behaviours. The approach revolves around three components: hybrid programs to model system behaviours, differential dynamic logic d$\mathcal{L}$ to specify properties and the KeYmaera tool that supports system behaviour specification and verification using a theorem prover for d$\mathcal{L}$. Below we give the required information to understand the development conducted in this paper. More details can be found in the abundant bibliography published by the authors.

Modelling: Hybrid Programs. According to Platzer [15], hybrid programs (HP) define a program notation for hybrid systems. These HP offer a structural decomposition to support d$\mathcal{L}$ reasoning. Additionally to classical programs, HP support the definition of variables that evolve along a differential equation. Some basic constructs of such programs are discrete assignments (`:=`), sequential composition (`;`), choice ($\cup$), state assertion or condition (`?H`), iteration (`*`) and continuous evolution of a continuous variable along differential equation (`x'=t & H`) in an evolution domain H (optional).

Property Specification and Verification: Differential Dynamic Logic. Differential dynamic logic d$\mathcal{L}$ is a first order logic with built-in statements dealing with hybrid systems. Similarly to first order logic which supports reasoning on classical programs using weakest precondition or substitution calculi, d$\mathcal{L}$ supports reasoning on hybrid programs. Operators of first order logic together with the modalities [] and $\langle\rangle$ are defined in d$\mathcal{L}$. $[\alpha]\,\phi$ and $\langle\alpha\rangle\,\phi$ assert respectively that ϕ holds after all runs and after at least one run of the HP α.

For example, $Init \longrightarrow [plant]\,(req)$ defines an uncontrolled system where $plant$ is a HP, $Init$ is the d$\mathcal{L}$ predicate characterising the initial state and req is a d$\mathcal{L}$ predicate defining a safety property. $Init \longrightarrow [(ctrl; plant)^*]\,(req)$

defines another system where the HP is made of instantaneous control events *ctrl* sequentially composed with the *plant* HP with a possible modification of its behaviour. The definitions of *ctrl* and *plant* are built using the constructs of HP, and *req* is again a safety property. We will use this template to model our case study.

Tool: KeYmaera. KeYmaera [16] is the theorem prover associated with differential logic. It supports proof of properties of hybrid programs. Additionally to the classical proof rules associated to first order logic, KeYmaera implements a set of specific proof rules defined for $\mathtt{d}\mathcal{L}$, including differential invariants, differential auxiliary and ODE-related tactics. In particular, differential invariants give an induction proof principle on differential equations.

3.2 Event-B with Rodin

Event-B [1] is a *correct-by-construction* approach to design an abstract model and a series of refined models for developing any large and complex system.

Modelling: Event-B Machines. The Event-B language uses set theory and first order logic. It has two main components, *context* and *machine*, to characterise systems. A *context* describes the static structure of a system using *carrier sets* s, *constants* c, *axioms* $A(s, c)$ and *theorems* $T_c(s, c)$, and a *machine* describes the dynamic structure of a system using *variables* v, *invariants* $I(s, c, v)$, *theorems* $T_m(s, c, v)$, *variants* $V(s, c, v)$ and *events* evt. A list of events can be used to model possible system behaviour to modify the state variables by providing appropriate *guards* in a *machine*. A set of *invariants* and *theorems* can be used to represent relevant properties to check the correctness of the formalized behaviour. To define the convergence properties, *variants* can be used.

Refinement of Event-B Models. Refinement decomposes a model (thus a transition system) into another transition system containing more design decisions while moving from an abstract level to a less abstract one. It supports the modelling of a system gradually by introducing safety properties at various refinement levels. New variables and new events may be introduced. These refinements preserve the relation between the refining model and the refined one while introducing new events and variables to specify more concrete behaviour of the system. The defined abstract and concrete state variables are linked by introducing *gluing invariants*.

Property Verification: Proof Obligations (PO). To verify the correctness of an Event-B model (machine or refinement) the generated POs (issued from the calculus of substitutions) need to be proved. A proof system allows to prove the POs. The main proof obligations are listed in Table 1, in which the prime notation is used to denote the value of a variable after an event is triggered.

These POs require to demonstrate that the theorems hold, each event preserves the invariant (inductive), each event can be triggered (feasibility) and if a variant is declared, it shall decrease.

Table 1. Proof obligations

Theorems	$A(s,c) \Rightarrow T_c(s,c)$ $A(s,c) \wedge I(s,c,v) \Rightarrow T_m(s,c,v)$
Invariant preservation	$A(s,c) \wedge I(s,c,v) \wedge G(s,c,v,x) \wedge BA(s,c,v,x,v') \Rightarrow I(s,c,v')$
Event feasibility	$A(s,c) \wedge I(s,c,v) \wedge G(s,c,v,x) \Rightarrow \exists v'.BA(s,c,v,x,v')$
Variant progress	$A(s,c) \wedge I(s,c,v) \wedge G(s,c,v,x) \wedge BA(s,c,v,x,v') \Rightarrow V(s,c,v') < V(s,c,v)$

Regarding refinement, two more relevant proof obligations need to be discharged. First, the simulation PO to show that the new modified action in the refined event is not contradictory to the abstract action and the concrete event simulates the corresponding abstract event. Second, in the refined events, we can strengthen the abstract guards to specify more concrete conditions. More details on proof obligations can be found in [1].

Tool: Rodin Platform. Rodin [2] is an open source tool based on the Eclipse framework for developing Event-B models. It is a collection of different tools including project management, model development, refinement and proof assistance, model checking, and code generation.

The Theory Plug-In. A recent development of the Event-B language allows to extend it with theories [3] similar to algebraic specifications. In the Rodin Platform, this development is provided by the *Theory* plug-in [13]. In our work, we extend the theory of *Reals*, written by Abrial and Butler[1], for developing the required theories for modelling hybrid systems. In particular, all the relevant definitions, theorems and proof rules related to continuous functions, Ordinary Differential Equations (ODEs), Cauchy-Lipschitz conditions, etc. are defined in the developed theories.

4 Development of the Case Study

In this section, we describe how the approaches presented in Sect. 3 can be used to address the case study exposed in Sect. 2. As for the section presenting the tool, we first describe what has been done by Quesel et al. to design a solution using $\mathsf{d}\mathcal{L}$ and KeYmaera, and then we move on to how we dealt with this problem using Event-B and Rodin.

4.1 HP/dL/KeYmaera

Model. Table 2 shows the $\mathsf{d}\mathcal{L}$ formula (Eq. (1)) specifying the behaviour and requirements of the system described in the case study of Sect. 2. In Eq. (2), an initial condition is defined for velocity v, acceleration A and breaking power B. It also describes the *safe* condition which defines the safety envelope (or evolution domain) for the car regarding the stopping point SP.

[1] http://wiki.event-b.org/index.php/Theory_Plug-in#Standard_Library.

Equation (1) states that given the initial condition and after any run of non-deterministic iteration composing sequentially the controller (4) and the plant (5) hybrid programs, the safety requirement *req* (6) stating that the position is always ([ ]) before the stopping point SP. Finally two equations define controller and plant. Equation (4) models the control.

First, when the *safe* condition holds, the acceleration is unchanged, otherwise it is set to $-B$ for breaking. Second, Eq. (5) sets up the ODEs associated to the position and velocity with respect to the reachability of the stopping point SP.

Table 2. Hybrid program for the self-driven car

$init \rightarrow$	$[(ctrl; plant)^*]\,(req)$	(1)
$init \equiv$	$v \geq 0 \wedge A > 0 \wedge B > 0 \wedge safe$	(2)
$safe \equiv$	$p + \frac{v^2}{2B} < SP$	(3)
$ctrl \equiv$	$(?safe; a := A) \cup (?v = 0; a := 0) \cup (a := -B)$	(4)
$plant \equiv$	$(p' = v, v' = a \& v \geq 0 \wedge p + \frac{v^2}{2B} \leq SP)$	(5)
	$\cup (p' = v, v' = a \& v \geq 0 \wedge p + \frac{v^2}{2B} \geq SP)$	
$req \equiv$	$p \leq SP$	(6)

Property Verification. The hybrid program of Table 2 is given to the KeYmaera prover. The user must then prove the global formula (1), and the proof is conducted by applying inference rules in a natural deduction style. Similarly to other provers, several proof rules and tactics are available. Figure 1 shows an extract of the proof tree associated to the $\mathtt{d}\mathcal{L}$ formula $init \rightarrow [(ctrl; plant)^*]\,(req)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\cdots}{init \vdash J}\,\text{QE} \quad \dfrac{\cdots}{J \vdash [(ctrl; plant)]\,J}\,\text{ODE} \quad \dfrac{\cdots}{J \vdash req}\,\text{QE}}{\vdash J \rightarrow [(ctrl; plant)^*]\,(req)}\,\text{loop}}{init \vdash [(ctrl; plant)^*]\,(req)}\,\rightarrow R}{\vdash init \rightarrow [(ctrl; plant)^*]\,(req)}$$

Fig. 1. Example of KeYmaera proof tree ($J \equiv v \geq 0 \wedge p + \frac{v^2}{2B} \leq SP$)

The power of the KeYmaera prover resides in the availability of several proof rules and tactics dealing with continuous aspects, ODEs and induction using differential equations (*loop* proof rule in Fig. 1).

4.2 Event-B/Rodin

Model. To build our Event-B model of the case study, our approach encodes a classical hybrid automaton [4,5] where transitions are *events* and states are simply stored as variables. Similar to the approach of [8], two types of variables are considered: (1) control variables (discrete) used for the controller (e.g. to record mode changes) and (2) variables (continuous) recording continuous state of the plant (e.g. to record the physical state of the plant). However, this is not enough to address all the complexity of hybrid systems. Namely, nothing is done to convey the "internal" evolution of the system (the time-step transitions), to handle the changes of the continuous variables with respect to time. So, additionally, we introduce events to reflect the overall continuous *progress* of the system.

Last, the core Event-B modelling language is not equipped with the formal material related to the definition of continuous mathematics required to model the physics of the controlled plant. To overcome this drawback, instead of re-designing a language, we used the so-called Event-B *theories*. Several theories have thus been defined and used for the development of the case study. The remainder of this section describes the whole Event-B development which can be downloaded from yamine.perso.enseeiht.fr/HS_eventb_models.pdf.

The *Derivative* Global Context to Manipulate Continuous Functions. Concept such as continuous functions, derivative, differential equations etc. required to model the physics of a plant are introduced in a generic context `Derivative` holding various (axiomatic) mathematical definitions. In particular, it gives the sets of continuous, once- and twice- differentiable functions as well as a basic "derivation" operator.

Beside that, we also defined a weak and simple version of the *Cauchy-Lipschitz theorem*, in order to be able to express the condition of existence of a solution to the given differential equation. Observe that the derivation operator D is used to define *time* derivation operation of the form $\frac{\mathrm{d}}{\mathrm{d}t}$.

CONTEXT Derivative
...
AXIOMS
axm1: $D \in ((\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R}))$ -- *derivative operator*
axm2: $\mathcal{C}^0(\mathbb{R}^+) \subset (\mathbb{R}^+ \to \mathbb{R})$ -- *continuous functions*
axm3: $\mathcal{D}^1(\mathbb{R}^+) \subset (\mathbb{R}^+ \to \mathbb{R})$ -- *once-differentiable functions*
axm4: $\mathcal{D}^2(\mathbb{R}^+) \subset (\mathbb{R}^+ \to \mathbb{R})$ -- *twice-differentiable functions*
axm5: $\mathcal{D}^1(\mathbb{R}^+) \subset \mathcal{C}^0(\mathbb{R}^+)$
axm6: $\mathcal{D}^2(\mathbb{R}^+) \subset \mathcal{D}^1(\mathbb{R}^+)$
...
cauchy_lipschitz:
$\forall f, t_0, x_0 \cdot f \in (\mathbb{R}^+ \to \mathbb{R}) \land f \in \mathcal{C}^0(\mathbb{R}^+) \land t_0 \in \mathbb{R}^+ \land x_0 \in \mathbb{R}$
$\Rightarrow \exists x \cdot x \in (\mathbb{R}^+ \to \mathbb{R}) \land D(x) = f \circ x \land x(t_0) = x_0$
...

The *Car_Context_C1* Context for Car Behaviours. It extends *Derivative* and declares the required concepts needed to build the Event-B model of the studied system. The constants defining the states of the controller (`acceleration`, `braking`, `stabilizing`, `nearing_stop` and `stopped`) are introduced together with A (acceleration), B (breaking power), v_0 (initial velocity) and SP (stopping point) used in the differential equations throughout the model.

This context also introduces the particular structure *Plant* for the characteristics of the plant (i.e.: the car). It is a 7-tuple with a differential equation (with its initial condition) for a (acceleration), v (velocity) and p (position) functions on time. They represent the state of the plant. k denotes the constant value of the acceleration, t_i, v_i and p_i represent the initial condition ($v(t_i) = v_i$ and $p(t_i) = p_i$). `axm10` defines the *Plant* structure, which holds every properties the elements of the model should satisfy with regards to the plant behaviour (type of the functions and differential equation). It also enforces **SAF2** (by indicating that whenever the velocity v becomes 0, the acceleration becomes 0 as well). Last, `CL_extension` entails the Cauchy-Lipschitz condition for this specific plant.

CONTEXT $Car\_Context\_C1$
EXTENDS $Derivative$
SETS $STATES$
CONSTANTS $B, A, v_0, SP, Plant$
AXIOMS
axm1: $partition(STATES, \{accelerating\}, \{braking\}, \{stabilizing\}, \{nearing\_stop\}, \{stopped\})$
axm2_9: $A \in \mathbb{R} \land B \in \mathbb{R} \land 0 < A \land 0 < B \land SP \in \mathbb{R} \land 0 < SP \land v_0 \in \mathbb{R} \land 0 \leq v_0$
axm10: $\forall a, v, p, k, t_i, v_i, p_i \cdot a \in \mathbb{R}^+ \rightarrow \mathbb{R} \land v \in \mathbb{R}^+ \rightarrow \mathbb{R} \land p \in \mathbb{R}^+ \rightarrow \mathbb{R} \land k \in \mathbb{R} \land t_i \in \mathbb{R}^+ \land v_i \in \mathbb{R} \land p_i \in \mathbb{R} \land (a, v, p, k, t_i, v_i, p_i) \in Plant$
$\Leftrightarrow (v \in \mathcal{D}^1 \land p \in \mathcal{D}^2 \land D(v) = a \land D(p) = v \land v(t_i) = v_i \land p(t_i) = p_i \land$
$(\exists t_0 \cdot t_0 \in \mathbb{R}^+ \land v(t_0) = 0$
$\Rightarrow (\forall t \cdot t \in \mathbb{R}^+ \Rightarrow ((t < t_0 \Rightarrow a(t) = k) \land (t \geq t_0 \Rightarrow a(t) = 0)))) \land$
$(\forall t_0 \cdot t_0 \in \mathbb{R}^+ \land v(t_0) \neq 0 \Rightarrow (\forall t \cdot t \in \mathbb{R}^+ \Rightarrow a(t) = k))$
$)$
CL_extension: $\forall k, t_i, v_i, p_i \cdot k \in \mathbb{R} \land t_i \in \mathbb{R}^+ \land v_i \in \mathbb{R} \land p_i \in \mathbb{R} \Rightarrow$
$(\exists a, v, p \cdot a \in \mathbb{R}^+ \rightarrow \mathbb{R} \land v \in \mathbb{R}^+ \rightarrow \mathbb{R} \land p \in \mathbb{R}^+ \rightarrow \mathbb{R} \land (a, v, p, k, t_i, v_i, p_i) \in Plant)$
END

The *Plant* structure is handled as a whole in the Event-B model. From a methodological point of view, it shall be defined for each modelled plant.

$inv1\_2 : t \in \mathbb{R}^+ \land current\_state \in state$
$inv3\_5 : a \in \mathbb{R}^+ \rightarrow \mathbb{R} \land v \in \mathbb{R}^+ \rightarrow \mathbb{R} \land p \in \mathbb{R}^+ \rightarrow \mathbb{R}$
$inv6\_7 : \forall t0 \in \mathbb{R}^+ : p(t0) \geq 0 \land \forall t0 \in \mathbb{R}^+ : p(t0) \leq SP$
$inv8\_11 : v \in \mathcal{D}^1 \land p \in \mathcal{D}^2 \land D(v) = a \land D(p) = v$

Variables and Invariants. We use the relevant contexts and model the system's state with five variables. A read only variable t for time is introduced. *current_state* defines the current state of the controller, and the three variables are associated to the controlled plant (p position, v speed and a acceleration).

Initialisation. The initialisation event defines the initial state and the starting read time value as well as the initial values of each variables. `act3` defines, using the *Plant* structure, the initial conditions and the differential equation governing the evolution of the variables p, v and a. It sets the acceleration a to 0 and initialises the position and velocity with p_0 and v_0.

INITIALISATION $\widehat{=}$
THEN
act1 : $current\_state := stable$
act2 : $t := 0$
act3 : $a, v, p : | a' \in \mathbb{R}^+ \rightarrow \mathbb{R} \land v' \in \mathbb{R}^+ \rightarrow \mathbb{R} \land p' \in \mathbb{R}^+ \rightarrow \mathbb{R} \land (a', v', p', 0, t, v_0, p_0) \in Plant$
END

Progress $\widehat{=}$
THEN
act1 : $t : | t' \in \mathbb{R}^+ \land t < t'$
END

Time Handling: The *Progress* Event. In order to model the progress of time *independently* of any other event (i.e. for the other events, time can only be read), the `Progress` event is introduced. This event occurs continuously in parallel with the other model events. A before-after predicate describes strictly increasing time using a positive real variable t.

Behaviour of the Plant. The remainder of the model contains two categories of events: sensing and actuating events. Sensing events are split into command events (user or driver orders) and the actual sensing (coming from the environment through sensors) events. To keep the paper at a reasonable length, we only describe one event of each category. The whole model is available on yamine.perso.enseeiht.fr/HS_eventb_models.pdf.

Command-Sensing Events. The command-sensing events observe, through sensing, the state of the car (plant) and trigger state changes on the controller (state-transition system). For example, under the condition that the velocity is positive, the `ctrl_sense_usr_input_stabilize` records that the driver decided to stabilize her speed.

ctrl_sense_usr_input_stabilize $\widehat{=}$
WHERE
grd1 : $p(t) + \frac{v(t)^2}{2B} < SP$
THEN
act1 : $current_state := stabilizing$
END
ctrl_sense_usr_input_accel $\widehat{=}$ ...
ctrl_sense_usr_input_brake $\widehat{=}$ ...

ctrl_sense_near_stop $\widehat{=}$
WHERE
grd1 : $p(t) + \frac{v(t)^2}{2 \times B} \geq S$
THEN
act1 : $current\_state := nearing\_stop$
END
ctrl_sense_stopping $\widehat{=}$ ...

ctrl_actuate_stabilize $\widehat{=}$
WHERE
grd1 : $current\_state \in \{stabilizing, stopped\}$
THEN
act1 : $a, v, p : |a' \in \mathbb{R}^+ \to \mathbb{R} \land v' \in \mathbb{R}^+ \to \mathbb{R} \land$
$p' \in \mathbb{R}^+ \to \mathbb{R} \land$
$(a', v', p', 0, t, v(t), p(t)) \in Plant$
END
ctrl_actuate_accelerating $\widehat{=}$ ...
ctrl_actuate_brake $\widehat{=}$ ...

Control-Sensing Events. These events are triggered when information from the external environment (typically: coming from sensors) is available. For example, the `ctrl_sense_near_stop` event is triggered when the stop sign needs to be taken into account. The physics of the car provides the model with the relevant trigger condition, used as a guard (`grd1`).

Control-Actuating Events. Whenever the controller changes state, it sets the right actuation on the car (plant). Using a before-after predicate, it changes the differential equation characterizing the plant behaviour (car) to a new one (change of acceleration a), ensuring that the past behaviour is preserved. This change occurs at the current time and holds until the next actuation. For example, the `ctrl_actuate_stabilze` event modifies the variables a, v and p in `act1` when the controller is in either `stabilizing` or `stopped` mode.

Property Verification. The theory defined for continuous features and ODEs generated several proof obligations, in particular those related to the theorems and thus to the proof rules. Then, other proof obligations are generated from the Event-B model. Due to our extensive use of the theory plug-in, most of these proofs have numerous manual steps, particularly the ones related to the continuous features. Even simple proofs, such as well-definedness, need to access real type operators via interactive theorem instantiation.

Obtained Results. The Event-B development of Sect. 4.2 shows that it is possible to model both continuous and discrete behaviours, using an event-based modelling style, within Event-B on the Rodin platform. It also shows that Event-B can handle modelling of hybrid systems modelled by hybrid automata.

5 A Development Method for Hybrid Systems

Taking the development carried out in Sect. 4.2 one step further, we present, in this section, the methodological lessons learnt from this development.

5.1 The Approach

The development of Sect. 4.2 is a *direct* formalisation of the requirements presented in the case study. However, when observing how the events are defined in the Event-B machine, one can identify a set of methodological rules that help to produce such models in a systematic way.

Required Theories. First, the global set of axioms and theorems, CONTEXT `Derivative`, related to the manipulation of continuous functions (derivation, Lipshitz condition, etc.) is used for all the Event-B developments. Second, the specific context with all the concepts needed to model the specific case study shall be introduced. This context defines the control states of the mode automaton associated to the control together with the continuous functions associated to the plant to be controlled. It also defines the global structure representing both continuous and discrete elements manipulated by the behavioural model through variables and events (definition of the *Plant* 7-tuple structure). Regarding the case study developed in this paper, this context corresponds to the `Car_Context_C1` context.

$x_s : Ctrl\_State$
$x_p : Plant\_State$
INVARIANT
Inv : $Inv\_Exp(x_s, x_p)$

INITIALISATION
$x_s, x_p : |Init\_Pred(x_s, x_p, x'_p, x'_s)$

Modeling Hybrid Systems. The next steps concern the design of the model itself using an Event-B machine. First, state variables are declared. x_s and x_p are the variables associated to the controller and to the plant respectively. They shall conform to the invariant defined by the Inv_Exp predicate. They are initialised with initial conditions defined by the predicate $Init\_Pred$. Two categories of events are needed to handle sensing and actuation. They are defined by two templates. `CTRL_Sense` events for the sensing events (user commands or plant sensing) that may modify the state ($Exp\_Next\_for\_s$ before-after predicate) of the controller while `CTRL_Actuate` defines the actuation events to modify ($Exp\_Next\_for\_p$ before-after predicate) the plant behaviour.

EVENT CTRL_Sense
WHEN
grd : $Cond\_Exp\_p(x_p)$
THEN
act : $x_s : |Exp\_Next\_for\_s(x_s, x_p)$
END

EVENT CTRL_Actuate
WHEN
grd : $Cond\_Exp\_s(x_s)$
THEN
act : $x_p : |Exp\_Next\_for\_p(x_p, x_s)$
END

The steps described above have been followed, in Sect. 4.2, to develop the Event-B models of the case study.

5.2 Deep Modelling: Generalisation

As mentioned previously, the approach described in the previous section sets up some methodological steps without any restriction on the resulting development. Encoding the previous steps in a generic deep Event-B model makes it possible to introduce more oversight and rigour into the design of models for hybrid systems.

In this section, we present a generalisation of the previous approach. We propose a generic Event-B model that explicitly formalises the different methodological steps defined in the previous section. We also show that a particular system can be modelled by instantiating this generic model and supplying witnesses.

A Theory for ODEs. Continuous functions, ODEs together with their relevant properties are defined with an Event-B theory. Listing 1.1 shows an extract of the theory of ODEs with *ode* as a constructor for differential equations. The operators $solutionOf$, $Solvable$ and $CauchyLipschitzCondition$ define predicates that states respectively that a given function is a solution of an ODE (with initial conditions), that an ODE admits a solution and that the given equation fulfill a Cauchy-Lipschitz condition. The *bind* operator returns an expression that links generic plant variables to a pair of variables associated to a particular plant. It has been introduced to ease instantiation.

Listing 1.1. Elements of the differential equations theory

TYPE PARAMETERS E, F, G
DATATYPES
DE(F)
Constructors
ode$(f : \mathcal{P}(\mathbb{R}^+ \times F \times F),\ f_0 : F,\ t_0 : \mathbb{R}^+)$
OPERATORS
solutionOf <predicate> (eta : $\mathbb{R}^+ \to F$, eq : $DE(F)$)
WHEN eq $\equiv$ ode(f, f_0, t_0) **THEN**
$eta \in \mathbb{R}^+ \to F \land eta \in \mathcal{D}^1(\mathbb{R}^+, F) \land D(eta) = f(eta) \land eta(t_0) = f_0$
Solvable <predicate> (eq : $DE(F)$)$\exists x \cdot x \in (\mathbb{R}^+ \to F) \land (x$ solutionOf $eq)$
CauchyLipschitzCondition <predicate> (eq : $DE(F)$)
WHEN eq $\equiv$ ode(f, f_0, t_0) **THEN**
$f_0 \in F \land t_0 \in \mathbb{R}^+ \land f \in (\mathbb{R}^+ \times F \to F) \land f \in \mathcal{C}^0(\mathbb{R}^+ \times F, F) \land$
$(\forall t^* \cdot t^* \in \mathbb{R}^+ \Rightarrow lispchitzContinuous(F, F, (\lambda y \cdot y \in F \mid f(t^*, y)))$
bind <expression> $(A : \mathcal{P}(E),\ B : \mathcal{P}(F),\ C : \mathcal{P}(G),\ f_{ab} : A \to B,\ f_{ac} : A \to C)$
$(\lambda x \cdot x \in A \mid (f_{ab}(x), g_{ac}(x)))$
...
AXIOMATIC DEFINITIONS
$\mathcal{C}^0,\ \mathcal{C}^n,\ \mathcal{D}^1,\ \mathcal{D}^n,$
...
lipschitzContinuous <predicate> $(A : \mathcal{P}(E),\ B : \mathcal{P}(F),\ f_{ab} : A \to B)$
...
THEOREMS
CauchyLipschitz: $\forall eq \cdot eq \in DE(F) \land CauchyLipschitzCondition(eq) \Rightarrow Solvable(eq)$

A Generic Model. The following model elements defined in `MACHINE System` gives a generalisation of the approach. We consider that the plant variable x_p

belongs to $S = \mathbb{R} \times \mathbb{R}$. It is indexed on time ($x_p \in \mathbb{R}^+ \rightarrow S$) and evolves according to any ODE e in the `actuate` event. The controller models state transitions belonging to the set of states `STATES` using the `Transition` event.

MACHINE System
...
INVARIANTS
inv1 : $t \in \mathbb{R}^+$
inv2 : $x_p \in \mathbb{R}^+ \rightarrow S$
EVENTS
INITIALISATION $\hat{=}$
THEN
act1 : $t := 0$
act2 : $x_p :\in \mathbb{R}^+ \rightarrow S$
act3 :
$current\_state :\in STATES$
END
Progress $\hat{=}$
THEN
act1 : $t : \mid t' \in \mathbb{R}^+ \wedge t' > t$
END

Actuate $\hat{=}$
ANY e, s
WHERE
grd1 : $e \in DE(S)$
grd2 : $Solvable(e)$
grd3 : $s = current\_state$
THEN
act1 : $x_p : \mid x'_p \in \mathbb{R}^+ \rightarrow S \wedge$
$(x'_p$ solutionOf $e)$
END
Transition $\hat{=}$
ANY s
WHERE
grd1 : $s \in STATES$
THEN
act1 : $current\_state := s$
END

Instantiation of the Generic Model: Two Steps. To get the specific model associated to the system corresponding to the case study of Sect. 2, two steps are required.

The first step consists of defining the Event-B context for the relevant information of the system by introducing acceleration, velocity etc. and the different ODEs describing plant evolution. It uses the constructors defined in the theory presented in Listing 1.1. The `CONTEXT Car_C0` shows such instantiation.

CONTEXT Car_C0
CONSTANTS $B, A, v_0, SP, Plant$
AXIOMS
axm1: $partition(STATES, \{stabilizing\}, \{braking\}, \{accelerating\},$
$\{nearing\_stop\}, \{stopped\}$
...
axm12: $f_{stable} \in \mathbb{R}^+ \times S \rightarrow S$
axm13: $f_{stable} = (\lambda t, (p, v) \cdot t \in \mathbb{R}^+ \wedge (p, v) \in S | (v, 0))$
...
axm16: $\forall t_0 \cdot t_0 \in \mathbb{R}^+ \Rightarrow lipschitzContinuous(S, S, (\lambda x \cdot f_{stable}(t_0, x)))$
...

The second step consists of supplying witnesses to the actuating and sensing events. The `MACHINE Car_M0` shows a witness for the plant variable x and v evolving according to an ODE for function f_{stabe}. It also shows an actuating event `ctrl_actuate_stabilize` where the actuation sets the plant variables to evolve according to the defined ODE.

MACHINE Car_M0
REFINES $ControlledSystem$
...
INVARIANTS
inv1 : $v \in \mathbb{R}^+ \rightarrow \mathbb{R}$
inv2 : $x \in \mathbb{R}^+ \rightarrow \mathbb{R}$
inv3 : $triggers \subseteq STATES$
inv4 : $x_p = bind(\mathbb{R}^+, \mathbb{R}, \mathbb{R}, v, x)$
EVENTS
INITIALISATION$\widehat{=}$
WITH
$x'_p : x'_p = bind(\mathbb{R}^+, \mathbb{R}, \mathbb{R}, v', x')$
THEN
act1 : $t := 0$
act2 : $current\_state := stabilizing$
act3 : $v, x :|$
$x' \in \mathbb{R}^+ \rightarrow \mathbb{R} \wedge v' \in \mathbb{R}^+ \rightarrow \mathbb{R} \wedge$
$bind(\mathbb{R}^+, \mathbb{R}, \mathbb{R}, v', x')$
solutionOf $ode(f_{stable}, (v_0, 0), 0)$
END

ctrl_actuate_stabilize $\widehat{=}$
REFINES $Actuate$
WHERE
grd1 : $current\_state \in \{stabilizing, stopped\}$
WITH
$e : e = ode(f_{stable}, (v(t) \mapsto x(t)), t)$
$s : s = stabilizing$
$x'_p : x'_p = bind(\mathbb{R}^+, \mathbb{R}, \mathbb{R}, v', x')$
THEN
act1 : $v, x :|$
$x' \in \mathbb{R}^+ \rightarrow \mathbb{R} \wedge v' \in \mathbb{R}^+ \rightarrow \mathbb{R} \wedge$
$bind(\mathbb{R}^+, \mathbb{R}, \mathbb{R}, v', x')$
solutionOf $ode(f_{stable}, (v(t) \mapsto x(t)), t)$
END

The previous models are extracts of a generic development that can be downloaded from yamine.perso.enseeiht.fr/HS_eventb_models.pdf.

5.3 About KeYmaera and Rodin: Assessments

The case study presented in Sect. 2 has been developed in both differential dynamic logic with KeYmaera and Event-B with Rodin. In both cases, the model of the system has been designed and the requirements have been proved. However, we have observed several differences in the use of these two modelling techniques.

KeYmaera supports the definition and verification of hybrid systems models expressed using $\mathsf{d}\mathcal{L}$ and hybrid programs. The logic is built-in and fixed once and for all and the proof rules are hard-coded into the KeYmaera prover. The advantage of such an approach is the availability of very powerful proof rules associated to the manipulation of differential equations (see Fig. 1), and in particular the differential invariant rule that defines an inductive proof schema (*loop* rule in Fig. 1) for differential equations.

To model hybrid systems in Event-B, we define the operational behaviour using the events. Invariants and other properties are defined at the same time. The model is seen as a set of events that perform either sensing or actuations. Proof by induction is obtained by proving invariant preservation by each event while KeYmaera advocates proof of invariant preservation on the whole hybrid program without a possibility of decomposing this hybrid program (as Event-B does for events). The specific proof rules for ODEs (like Cauchy-Lipschitz theorem) need to be added in the defined theories while they are built-in in KeYmaera. This task is cumbersome but should only be done once.

Our experiments with Event-B have been conducted in two manners. First, we have encoded directly the case study as an Event-B model in the spirit of $\mathsf{d}\mathcal{L}$ and KeYmaera. Contrary to KeYmaera, this process requires the definition of all the material related to the manipulation of continuous functions and ODEs.

Compared to KeYmaera, this process may be time-consuming due to the important proof effort just to set up the different functions and ODEs.

Secondly, we have developed an abstract model that generalises the theory of hybrid controllers. This model is designed and proved once and for all. It may be instantiated, using Event-B refinement, for specific cases by providing witnesses and proof of existence. Indeed, these feasibility proof obligations convey a technical point of differential equation theory; that is: the existence of solutions to a given problem. However, *KeYmaera* proofs seem to rely on the *ad hoc* existence of those solutions.

The definition of the generic model makes explicit definitions of all the concepts manipulated by the model. These definitions can be customised for specific kinds of controllers and ODEs (for example, we can add more constraints on ODEs to admit only linear ones). The Rodin theory plug-in helps to define the proof rules associated to the use of these definitions.

Finally, some issues still need to be solved. For example, the difficulty to define inductive Cartesian products ($S \times \cdots \times S$ or S^n for an arbitrary n) to define vectors of state variables. We have to use an inductive structure for this purpose and thus rewrite the *bind* generic operator. Secondly, the definition of a condition to assert the non-zeno property of the system described by ODEs must be addressed. This can be done by adding a condition on the existence of a piecewise decomposition of an ODE in a finite set of arbitrary intervals. KeYmaera makes this assumption but does not make it explicit.

Other Proof-Based Approaches. Here we briefly review three main contributions in formal modelling and proof based verification of hybrid systems.

As described above, differential dynamic logic using KeYmaera and KeYmaeraX tool suite have been used to model several cases of hybrid systems.

Additionally to the approach presented in this paper, other Event-B contributions can be mentioned. [12,18] use Event-B and the Rodin Platform [14] to model hybrid systems in a closed-loop model. Time is explicitly modelled using a specific state variable. The authors consider continuous functions and they define discrete and continuous transitions preserving invariants which characterise the correct behaviour of the described hybrid system.

[9] proposes the Hybrid Event-B extension of Event-B with built-in concepts for continuous behaviours, differential equations discrete and continuous (pliant) events. Several hybrid systems models (e.g. [8]) have been developed. A similar approach has been proposed to define continuous abstract state machines in [10]. For both approaches, there is no available supporting tool.

Last, we mention the work of [11] using a proof-based approach with COQ for the analysis of C hybrid systems programs. The approach uses formal annotations on discrete and continuous program elements as input to a set of provers.

Finally, we recall that several other approaches based on model checking of hybrid systems modelled as hybrid automata [5]. Due to space limitations these approaches are not discussed in this paper.

6 Conclusion and Future Work

The formal development of hybrid systems requires modelling of both discrete and continuous behaviours. In this paper, we have presented two formal developments of a case study related to a stop-sign controller of a car. The first one is based on differential dynamic logic with KeYmaera and the second one uses Event-B and Rodin. Both approaches proved useful to model such a system and refer to interactive proofs involving proof rules on differential equations. Besides, using the theory plug-in to extend Rodin's capabilities, Event-B proved to be well adapted for generalisation.

This work opens several research paths. First the generalisation we have presented in Sect. 5.2 can be improved in order to formally handle more features like invariants, guards or different kinds of ODEs. Offering such a possibility will allow us to produce generic templates of hybrid models that correspond to different types of hybrid systems (for example a non-linear system, or a polyhedral invariant). The ultimate objective is to hide the development details through the definition of development/proof patterns. Second, following our preliminary results in [6,7], the synthesis of a discrete controller from hybrid models similar to those presented in this paper is also a key issue. Last, we will address the simulation aspect related to the modelling of hybrid systems.

Finally, we advocate that Event-B together with the plug-in associated to powerful provers will allow us to achieve these goals.

Acknowledgment. We would like to thank Dr Thai Son Hoang for his help regarding Rodin's Theory plug-in.

References

1. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
2. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. Int. J. Softw. Tools Technol. Transfer **12**(6), 447–466 (2010)
3. Abrial, J.R., Butler, M., Hallerstede, S., Leuschel, M., Schmalz, M., Voisin, L.: Proposals for mathematical extensions for Event-B. Technical report (2009). http://deploy-eprints.ecs.soton.ac.uk/216/
4. Alur, R.: Formal verification of hybrid systems. In: Proceedings of the Ninth ACM International Conference on Embedded Software, EMSOFT 2011, pp. 273–278. ACM, New York (2011)
5. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.-H.: Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) HS 1991-1992. LNCS, vol. 736, pp. 209–229. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57318-6_30
6. Babin, G., Aït-Ameur, Y., Singh, N.K., Pantel, M.: Handling continuous functions in hybrid systems reconfigurations: a formal Event-B development. In: Butler, M., Schewe, K.-D., Mashkoor, A., Biro, M. (eds.) ABZ 2016. LNCS, vol. 9675, pp. 290–296. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33600-8_23

7. Babin, G., Aït-Ameur, Y., Singh, N.K., Pantel, M.: A system substitution mechanism for hybrid systems in Event-B. In: Ogata, K., Lawford, M., Liu, S. (eds.) ICFEM 2016. LNCS, vol. 10009, pp. 106–121. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47846-3_8
8. Banach, R.: Formal refinement and partitioning of a fuel pump system for small aircraft in hybrid Event-B. In: 2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE), pp. 65–72, July 2016
9. Banach, R., Butler, M., Qin, S., Verma, N., Zhu, H.: Core hybrid Event-B I: single hybrid Event-B machines. Sci. Comput. Program. **105**, 92–123 (2015)
10. Banach, R., Zhu, H., Su, W., Wu, X.: ASM, controller synthesis, and complete refinement. Sci. Comput. Program. **94**, 109–129 (2014)
11. Boldo, S., Clément, F., Filliâtre, J.C., Mayero, M., Melquiond, G., Weis, P.: Trusting computations: a mechanized proof from partial differential equations to actual program. Comput. Math. Appl. **68**(3), 325–352 (2014)
12. Butler, M., Abrial, J.R., Banach, R.: Modelling and refining hybrid systems in Event-B and Rodin. In: Petre, L., Sekerinski, E. (eds.) From Action Systems to Distributed Systems: The Refinement Approach. Computer and Information Science Series, Chap. 3, pp. 29–42. Chapman and Hall/CRC (2016)
13. Butler, M., Maamria, I.: Practical theory extension in Event-B. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods. LNCS, vol. 8051, pp. 67–81. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39698-4_5
14. Jastram, M.: Rodin User's Handbook, October 2013. http://handbook.event-b.org
15. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. **41**(2), 143–189 (2008)
16. Platzer, A., Quesel, J.-D.: KeYmaera: a hybrid theorem prover for hybrid systems (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 171–178. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71070-7_15
17. Quesel, J.D., Mitsch, S., Loos, S., Aréchiga, N., Platzer, A.: How to model and prove hybrid systems with KeYmaera: a tutorial on safety. Int. J. Softw. Tools Technol. Transfer **18**(1), 67–91 (2016)
18. Su, W., Abrial, J.R., Zhu, H.: Formalizing hybrid systems with Event-B and the rodin platform. Sci. Comput. Program. **94**(Part 2), 164–202 (2014). Abstract State Machines, Alloy, B, VDM, and Z