Modelling Hybrid Programs with Event-B

Meryem Afendi[1](✉), Régine Laleau[1], and Amel Mammar[2]

[1] Université Paris-Est Créteil, LACL, Créteil, France
{meryem.afendi,laleau}@u-pec.fr
[2] SAMOVAR, Institut Polytechnique de Paris, Télécom SudParis, Évry, France
amel.mammar@telecom-sudparis.eu

Abstract. Hybrid systems are one of the most common mathematical models for Cyber-Physical Systems (CPSs). They combine discrete dynamics represented by state machines or finite automata with continuous behaviors represented by differential equations. The measurement of continuous behaviors is performed by sensors. When these sensors have a continuous access to these measurements, we call such model an *Event-Triggered* model. The properties of this model are easier to prove, while its implementation is difficult in practice. Therefore, it is preferable to introduce a more realistic model, called *Time-Triggered* model, where the sensors take periodic measurements. Contrary to *Event-Triggered* models, *Time-Triggered* models are much easier to implement, but much more difficult to verify. Based on the differential refinement logic (d$\mathcal{RL}$), a dynamic logic for refinement relations on hybrid systems, it is possible to prove that a *Time-Triggered* model refines an *Event-Triggered* model. The major limitation of such logic is that it is not supported by any prover. In this paper, we propose a correct-by-construction approach that implements the reasoning on hybrid programs particularly the reasoning of d$\mathcal{RL}$ in Event-B to take advantage of its associated tools.

Keywords: Cyber-Physical Systems · Hybrid systems · Event-B · Refinement · Differential refinement logic

1 Introduction

Recent progress in the industrial sector have allowed the development of a new production model based on digital network architectures to give birth to a fourth industrial revolution ("industry 4.0" or "industry of the future"). Cyber Physical Systems (CPSs) [2] are one of the main technologies in this industry and form the basis of future technologies. The domain of these systems has rapidly become a source of innovation with applications in many sectors: health, transport, smart grid, etc. This type of systems allows to connect the discrete virtual world and the continuous physical world via a network of sensors and actuators. One of

This work was supported in part by the DISCONT project [1] funded by the French National Research Agency (ANR).

the most common architectures in CPSs is a discrete software controller that represents the computation part and controls the physical part through a loop involving sensors and actuators.

A common mathematical model for CPSs is that of hybrid systems that combine discrete behavior represented by state machines or finite automata with continuous behavior described by differential equations. The development of techniques and tools to effectively design hybrid systems has drawn the attention of many researchers. Traditional approaches are based on simulation tools like Matlab/Simulink or Stateflow. Since these tools are time-consuming and produce results tainted with uncertainty, traditional approaches can be very expensive and difficult to apply. To overcome these limitations, several formal approaches have been proposed. These approaches can be grouped into two categories: *model-checking-based* approaches and *proof-based* approaches. Model-checking-based approaches use hybrid automata to model hybrid systems and algorithmic analysis methods to prove their safety. They are based on the calculation of the set of reachable states for hybrid automata. These approaches suffer from the classical problems related to the state space explosion and boundedness of the considered variables issues. Proof-based approaches use deductive verification to prove the properties of hybrid systems. One of the strong points of these approaches is that they support the description of any kind of hybrid systems. However, they require significant effort and a high expertise in modelling and proof phases. This is the main reason why these approaches do not yet scale to industrial applications.

In CPSs, the measurement of continuous behaviors is performed by sensors. Ideally sensors have a continuous access to these measurements, this can be captured by an abstract model of CPSs, called *Event-Triggered* system by Kopetz in [3]. However, implementing such models is difficult in practice. Therefore, it is preferable to introduce a more realistic model, called *Time-Triggered* system in [3], where the sensors take periodic measurements. Contrary to *Event-Triggered* models, properties on *Time-Triggered* models are difficult to verify. Platzer et al. [4,5] use this approach to model hybrid systems. They have proved that a *Time-Triggered* model can be a refinement of an *Event-Triggered* model, by using an extension of the differential dynamic logic (d$\mathcal{L}$), called the differential refinement logic (dR$\mathcal{L}$). However dR$\mathcal{L}$ is not supported by any prover and dR$\mathcal{L}$ formulas can only be manually proved, which heavily restricts its use, especially in an industrial context. In this paper we propose an approach to model *Event-Triggered* systems and *Time-Triggered* systems in Event-B to take advantage of its well-defined refinement process and of its support tools. We also reused the work proposed in [6] that allows to model differential equations in Event-B.

This paper is organised as follows. Section 2 briefly describes d$\mathcal{L}$, dR$\mathcal{L}$ and Event-B. Section 3 presents a state of the art of some proof based-approaches for CPS modelling. Section 4 presents *Event* and *Time-Triggered* systems and their modelling in dR$\mathcal{L}$. Section 5 then introduces our proposed approach and discusses the difference between modelling *Event* and *Time-Triggered* systems in dR$\mathcal{L}$ and Event-B. Finally, Sect. 6 concludes and presents some future work.

2 Background

2.1 Differential Dynamic Logic d$\mathcal{L}$

This section describes a first-order dynamic logic in the domain of real ($\mathbb{R}$) introduced by A. Platzer to specify hybrid systems and verify their correctness using its associated proof calculus [4]. d$\mathcal{L}$ formulas are built using logical symbols of first-order logic and the modalities $[\ ]$ (Box-modality) and $\langle\ \rangle$ (Diamond-modality). Formula $[\alpha]\phi$ is true iff after all runs of the hybrid program α, formula ϕ holds. $\langle\alpha\rangle\phi$ is true iff there is at least one run of the hybrid program α, after which formula ϕ holds. The major advantage of d$\mathcal{L}$ is its ability to handle differential equations, even those with non-polynomial solutions. Moreover, d$\mathcal{L}$ and its associated proof calculus are supported by two automatic formal verification tools, KeYmaera [7] and its successor KeYmaera X [8].

In d$\mathcal{L}$, hybrid systems are given operationally as hybrid programs (HPs). These latter describe both discrete and continuous behaviors of hybrid systems using sequential composition (;), non-deterministic choice ($\cup$), non-deterministic repetition ($*$), discrete assignments (:=), continuous evolution ($'$), etc. Most HPs are defined using the notation, $(ctrl; plant)^*$, where *ctrl* denotes the execution of the controller (discrete evolution), followed by the physical part *plant* (continuous evolution). This sequence is non-deterministically repeated as denoted with the star (*).

Finally, in order to establish a safety property, *safeReq*, for a system, a typical formula expressing safety relative to initial conditions needs to be proved, $init \rightarrow [(ctrl; plant)^*](safeReq)$ that means: if the initial conditions (*init*) hold, then, after all runs of the hybrid program *safeReq* holds.

2.2 Differential Refinement Logic dR$\mathcal{L}$

dR$\mathcal{L}$ is a logic with first-class support for refinement relations on hybrid systems [5]. It extends d$\mathcal{L}$ by introducing a refinement operator ($\leq$) for HPs. In addition to d$\mathcal{L}$ formulas, dR$\mathcal{L}$ introduces formulas of the form $\alpha \leq \beta$, α refines β, with α and β denoting HPs. According to [5], formula $\alpha \leq \beta$ is true in a state s iff all states reachable from s by following the transitions of α could also be reached from state s by following transitions of β.

dR$\mathcal{L}$ preserves the safety properties of refined hybrid programs by showing that if $\alpha \leq \beta$ and $[\beta]\phi$, then the formula ϕ is true in all states reachable from s by following the transitions of α ($[\alpha]\phi$). There is a similar rule for diamond modalities ($\langle\ \rangle$), which states that if α refines β, and there is at least one transition on α to a state where ϕ is true, then $\langle\beta\rangle\phi$ is true. Moreover, dR$\mathcal{L}$ establishes that a *Time-Triggered* system refines an *Event-Triggered* system using its associated proof calculus (Sect. 4).

2.3 Event-B

Event-B [9] is a formal method based on set theory, first-order logic and predicate logic. An Event-B model is composed of a set of machines and contexts. An

Event-B context consists of sets and constants with their axioms. An Event-B machine represents the dynamic behavior of a given system, and it may see one or more Event-B contexts. To any Event-B model, a set of proof obligations (POs) is associated. These POs must be proved to verify the correctness of a given Event-B model. They can be automatically generated using for example the Rodin platform [10], which is an Eclipse-based IDE for Event-B. This platform allows to add new features as Eclipse plug-ins. For example, the Theory plug-in [11] is a Rodin extension that allows to define new data types like $REAL$, new operators, etc. Event-B has a key feature that consists in using abstract modelling to represent the abstract behavior of a given system and the refinement to demonstrate compliance between an abstract model and a concrete one.

3 State of the Art

In this section, we focus on proof-based approaches defined to specify and verify hybrid systems using Event-B, the d$\mathcal{RL}$ approach will be discussed in Sect. 4. We briefly describe three main approaches. The approach presented in [12] proposes a new formal method, called Hybrid Event-B, to add continuous aspects to traditional discrete Event-B. It defines two kinds of events: *mode events* and *pliant events*. *mode events* represent the traditional discrete Event-B events. *pliant events* specify the continuous evolution of continuous measurements. As d$\mathcal{RL}$, Hybrid Event-B is not supported by any prover.

The authors of [13] propose an approach supported by the Rodin toolset to model hybrid systems using continuous functions over real intervals. Preserving the properties of these functions is the key for ensuring the correction of refined machines. This approach uses the Event-B refinement to reduce the non-determinism in continuous behaviors and introduce periodic control.

Finally, the approach proposed by Dupont et al. in [6] uses the Theory plug-in of Event-B to define theories that handle continuous aspects of hybrid systems. The behavior of CPSs is specified by the following three Event-B models:

– ***System* model** is used to describe the continuous evolution of the physical part of a hybrid system. Its machine contains two events:
  - the *Progress* event models the continuous evolution of time by using a positive real variable $t \in (TIME = RRealPlus)$[1]. The lt symbol corresponds to the operator ($<$) in the *RReal* theory [6].

    Progress
    THEN act1: $t \; :\mid t' \in TIME \wedge (t \mapsto t' \in lt)$ **END**

  - the *Behave* event models the physical part's evolution represented by the physical state variable $plantV$. While modelling a car, $plantV$ will be replaced by the car's position p and the car's velocity v. $plantV$ evolves according to the differential equation $e \in DE(S = RReal * RReal)$[2]

[1] *RRealPlus* represents $\mathbb{R}^+$ in the *RReal* theory developped by Dupont et al.
[2] *RReal* represents $\mathbb{R}$ in the *RReal* theory developped by Dupont et al.

defined as a parameter of the *Progress* event, where $DE(S)$ is a set of differential equations defined on S. This differential equation must have a solution in the interval $[t, \infty[$ that is represented by Guard $grd2$.

```
Behave
ANY  e
WHERE
    grd1: $e \in DE(S)$
    grd2: $Solvable(Closed2Infinity(t), e)$
THEN
    act1: $plantV \ :\mid plantV' \in TIME \rightarrow S \ \wedge$
          $AppendSolutionBAP(e,\ TIME,\ Closed2Open(Rzero,\ t),$
                         $Closed2Infinity(t),\ plantV,\ plantV')$
END
```

- ***State_System* model** refines the previous model by adding the evolution of the discrete part (the controller). It introduces a new variable named *x_s* to model the possible states of the controller. It also introduces a new event, named *Transition*, to update the controller's state by assigning a non deterministic value to *x_s*. The possible values that can be assigned to this variable are defined in the associated context as elements of a set *STATES* defined in the same context.

- ***Controlled_System* model** refines the *State_System* model by adding two new events that allow the interaction between the physical part and the discrete part:
  - *Sense* event allows to modify the controller's state according to the physical part's state. It introduces a parameter p which depends on *x_s*, t and $plantV(t)$ ($p \in P(STATES \times TIME \times S)$). This parameter allows to define the system safety envelope according to its discrete state.
  - *Actuate* event refines the *Behave* event by adding a constraint on the controller's state. This constraint is represented by the following formulas: $s \subseteq STATES$ and $x\_s \in STATES$.

4 Event and Time-Triggered Systems

In order to design a model that better corresponds to real CPSs it is preferable to start with an abstract one, called *Event-Triggered* model, where the controller interrupts the physical part when a particular event occurs, and then introduce a more realistic model, called *Time-Triggered* model, where the controller interrupts periodically the physical part [3]. *Event-Triggered* models describe an ideal behavior where the time is continuous and the sensors have continuous access to continuous measurements which is not always possible in practice. *Time-Triggered* models describe a more realistic behavior where the sensors take periodic measurements. Therefore, the controller of a *Time-Triggered* system must make a choice that will be safe until the next sensor's update, which makes this type of systems difficult to prove compared to *Event-Triggered* systems.

d$\mathcal{RL}$ allows to specify and prove that a *Time-Triggered* system refines an *Event-Triggered* system. It introduces two generic templates [5], *Model 1* and *Model 2*, to model and prove these two types of systems. The control part of these two generic templates has only two modes: the *normal* mode which is triggered if the system safety envelope, denoted *safe*, is satisfied, and the *evade* mode which is triggered otherwise. As already mentioned, the major limitation of d$\mathcal{RL}$ is that it is not supported by any prover. This limitation represents a strong restriction on its application to more complex hybrid systems. This paper proposes to deal with this restriction through the use of Event-B and its support tools.

4.1 Event-Triggered Model

Model 1: Event-triggered Generic Model

$$
\begin{aligned}
event^* &\equiv (ctrl_{Ev}; plant_{Ev})^* && \textbf{(1.1)}\\
ctrl_{Ev} &\equiv (ctrlV := evade\_value) \cup (ctrlV := *; ?safe(plantV)) && \textbf{(1.2)}\\
plant_{Ev} &\equiv t := 0;\ \ plantV_0 := plantV;\\
&\quad (plantV' = f\_evol(ctrlV), t' = 1\ \ \&\ \ evt\_trig(plantV)\\
&\quad \wedge dom\_evol(plantV)) && \textbf{(1.3)}\\
&\quad \cup (plantV' = f\_evol(ctrlV), t' = 1\ \ \&\ \ \sim evt\_trig(plantV)\\
&\quad \wedge dom\_evol(plantV)) && \textbf{(1.4)}
\end{aligned}
$$

where:

$ctrlV$: the control variable (acceleration in the case of a car).
$plantV$: the state variable of the system.
$safe(plantV)$: defines the system safety envelope. It is calculated from the safety requirement that the system must satisfy.
$plantV' = f\_evol(ctrlV)$: defines the system ODE that describe the continuous evolution of the system.
$evt\_trig(plantV)$: the predicate that defines the boundary of the safety envelope. When this latter becomes false, the controller triggers the *evade* mode. It must define a closed domain.
$\sim evt\_trig(plantV)$: topological closure of the complement of $evt\_trig(plantV)$.
$dom\_evol(plantV)$: defines the evolution domain of the system. It is a set of constraints on the state variable.
$plantV_0$: represents the initial value of $plantV$.

N.B: the variables t and $plantV_0$ have no effect on the state of this model. They will be used in the second model.

Model 1 represents the generic model associated with a controller triggered by events. When the formula *safe* is satisfied, the system can evolve continuously according to formula **(1.3)** until it reaches the boundary of the domain $evt\_trig(plantV)$. Once the system reaches the boundary of this domain, the controller must then switch to the *evade* mode by affecting a deterministic value *evade_value* to the control variable ($ctrlV$). After the switch of the controller to the *evade* mode, the system no longer satisfies the formula *safe*. Therefore, it can no longer evolve in the domain $evt\_trig(plantV)$ that's why dR$\mathcal{L}$ defines formula **(1.4)**. This latter allows the system to evolve continuously when it is in the *evade* mode. To prove the safety of this model, dR$\mathcal{L}$ provides the following proof rule where Γ represents other assumptions not affected by the program *event*:

$$evt\_trig(plantV) \wedge \Gamma \vdash [event](evt\_trig(plantV) \wedge \Gamma)$$

This proof states that *Model 1* is safe if its associated hybrid program *event* always satisfies the loop invariant $evt\_trig(plantV)$ which includes the formula $safe(plantV)$.

4.2 Time-Triggered Model

Model 2: Time-triggered generic model

$$\begin{aligned} time^* \equiv\ & (ctrl_t; plant_t)^* && \textbf{(2.1)} \\ ctrl_t \equiv\ & (ctrlV := evade\_value) \\ & \cup\ (ctrlV := *; ?safe_\epsilon(plantV, ctrlV)) && \textbf{(2.2)} \\ plant_t \equiv\ & t := 0;\ plantV_0 := plantV; (plantV' = f\_evol(ctrlV), \\ & t' = 1\ \ \&\ \ t \leq \epsilon\ \wedge\ dom\_evol(plantV)) && \textbf{(2.3)} \end{aligned}$$

where

ϵ: maximum time between two sensors updates.
t: allows to know if the duration ϵ is reached or not.

Model 2 represents the generic model associated with a *Time-Triggered* system. The controller of such system reacts at least every ϵ seconds. To express this constraint, dR$\mathcal{L}$ replaces formulas **(1.3)** and **(1.4)** by a single one **(2.3)**. Formula *safe* is also replaced by formula $safe_\epsilon$, which depends on both the current choice of $ctrlV$ and the time duration ϵ, in addition to the current state $plantV$, in order to guarantee that the controller will make a choice that will be safe for up to ϵ time. To prove that *Model 2* satisfies a safety property ϕ, dR$\mathcal{L}$ has introduced the following proof rule ($[\leq]$) where Δ represents other obligations in the context not affected by the proof rule.

$$\frac{\Gamma \vdash [event^*]\phi, \Delta \quad \Gamma \vdash (time^* \leq event^*), \Delta}{\Gamma \vdash [time^*]\phi, \Delta}[\leq]$$

This proof consists of two sub-goals: the first one is to prove that *Model 1* satisfies the system safety property ϕ, and the second aims at verifying that *Model 2* refines *Model 1*.

4.3 Time-Triggered Model Refines Event-Triggered Model

To prove that a *Time-Triggered* system refines an *Event-Triggered* system, dR$\mathcal{L}$ provides three proof obligations:

– **PO 1:** $evt\_trig(plantV) \wedge \Gamma \wedge safe_{\epsilon}(plantV, \bar{ctrlV}) \vdash safe(plantV)$

  where:

  $\bar{ctrlV}$: represents a non-deterministic choice of the control variable.

  This proof expresses that the safety envelope of *Model 2* implies that of *Model 1*, which means that the discrete controller refines the continuous one.

– **PO 2:** $evt\_trig(\bar{plantV}_0) \ \wedge \ \Gamma \ \wedge \ safe_{\epsilon}(\bar{plantV}_0, \bar{ctrlV}) \ \wedge \ 0 \leq t \leq \epsilon$
  $\wedge \ dom\_evol(\bar{plantV}) \wedge \bar{plantV} = S_{\bar{plantV}_0, \bar{ctrlV}}(t) \vdash evt\_trig(\bar{plantV})$

  where:

  $\bar{plantV}_0$: set of physical state variables values at instant $t = 0$.

  $\bar{plantV}$: set of physical state variables values at instant t.

  $S_{\bar{plantV}_0, \bar{ctrlV}}(t)$: solutions of the ODE associated with $plantV_0$, given a control variable choice $\bar{ctrlV}$.

  This proof expresses that the non-deterministic choice of $ctrlV := *$ expressed by $\bar{ctrlV}$ guarantees that the system will not cross the boundary of $evt\_trig(plantV)$ within time ϵ.

– **PO 3:** $evt\_trig(\bar{plantV}_0) \ \wedge \ \Gamma \wedge 0 \leq t \leq \epsilon \ \wedge \ dom\_evol(\bar{plantV})$
  $\wedge \ \bar{plantV} = S_{\bar{plantV}_0, evade\_value}(t) \vdash evt\_trig(\bar{plantV})$

  This proof is similar to the previous one except that here the control choice is deterministic $ctrlV := evade\_value$.

5 Modelling Hybrid Programs with Event-B

The objective of the DISCONT project [1] is to elaborate a correct-by-construction method, based on Event-B, to specify hybrid systems models. Two approaches are considered. The first one, developed by Dupont et al. [6], is based on a translation of hybrid automata in Event-B extended by theories that handle differential equations and continuous functions (derivation, Lipschitz condition, etc.). In our approach we propose to model the high-level structure of hybrid programs, (ctrl;plant)*, in Event-B, and more precisely the generic templates defined for modelling *Event* and *Time-Triggered* systems in dR$\mathcal{L}$.

One of our objectives is to use the Event-B refinement and its associated tools to demonstrate the compliance between these two models, and also compare the refinement proof obligations generated in Event-B with those provided by dR$\mathcal{L}$. The approach consists of three models as depicted in Fig. 1 where *System_M* and *System_Ctx* are those of [6]. We also reuse the Event-B theories that handle continuous aspects of hybrid systems. The whole models can be downloaded from: https://cloud.lacl.fr/index.php/s/K75Lt28ApPbkY7z.

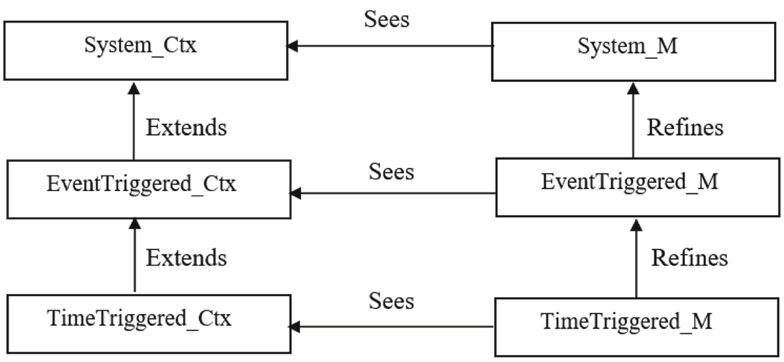

Fig. 1. Structure of the Event-B specification.

5.1 Event and Time-Triggered in Event-B

Event-Triggered Model is a generic model designed to specify and prove *Event-Triggered* systems in Event-B. It is based on the first generic template of dR$\mathcal{L}$, *Model 1*. As we mentioned above, dR$\mathcal{L}$ models an *Event-triggered* system by adding the constraint $evt\_trig(plantV)$ to the system evolution domain. Since Event-B models the transitions between discrete states as events, we do not need to use this constraint in Event-B. Moreover, through the use of Event-B, we can see the different transitions of a given system.

The *Event-Triggered* model is composed of an Event-B context named *Event-Triggered_Ctx* and an Event-B machine named *EventTriggered_M*. *EventTriggered_Ctx* defines a set of constants and axioms required to model an *Event-Triggered* system, such as the formula *safe* that represents the safety envelope for the modeled system. As in dR$\mathcal{L}$, the formula *safe* depends on the current physical state variable as well as the control variable since it may contain some limits on how this latter may be set. The domain of this formula must be included in that of $evt\_trig(plantV)$ formula. Moreover, *safe* must be initially satisfied. In that case, proving the safety of an *Event-Triggered* model consists in ensuring that the specific choice of the *evade* mode is safe. Machine *EventTriggered_M* refines that of the abstract model *System* by adding two new variables:

- *ctrlV* represents the control variable and belongs to *RReal*. The current value of this variable corresponds to the current controller's state.
- *exec* is a flag used to model the alternation between the control part and the physical part as represented in the high-level structure of hybrid programs, $(ctrl; plant)^*$. Therefore, *exec* can take two values *ctrl* and *plant*. In Event-B, the time must be explicitly handled. To be sure that this explicit time will only be updated after the execution of the controller and the physical part, we added a third value, *prg*, to *exec*. Moreover, we refined the *Progress* event of the Machine *System_M* (page 4) to add the constraint $exec = prg$ as a guard and $exec := ctrl$ as an action. Therefore, our model follows the following structure: $init; (ctrl; plant; prg)^*$, where *init* represents the *INITIALISATION* event.

To model the evolution of the physical part, we have defined the *Plant* event. This latter refines the *Behave* event by replacing the abstract differential equation e with that defined for a function denoted f_evol_plantV in order to model $plantV' = f\_evol(ctrlV)$. The function f_evol_plantV describes the evolution of the state variable *plantV* according to the system discrete state.

Plant
REFINES
  Behave
WHERE
  grd1: $ode(f\_evol\_plantV(ctrlV),\ plantV(t),\ t) \in DE(S)$
  grd2: $Solvable(Closed2Infinity(t), ode(f\_evol\_plantV(ctrlV), plantV(t), t))$
  grd3: $exec = plant$
WITH
  e: $e = ode(f\_evol\_plantV(ctrlV),\ plantV(t),\ t)$
THEN
  act1: $plantV :| plantV' \in (TIME \to S) \wedge$
    $AppendSolutionBAP(ode(f\_evol\_plantV(ctrlV),\ plantV(t),\ t),$
    $TIME, Closed2Open(Rzero, t), Closed2Infinity(t), plantV, plantV')$
  act2: $exec := prg$
END

Regarding the evolution of the control part, we have added two new events:

- *Ctrl_normal* event representing the *normal* mode. It is triggered when it is the controller's turn $(exec = ctrl)$ and the formula *safe* is true. It assigns a non-deterministic value, defined in the *ANY* clause, to the control variable *ctrlV* and gives the turn to the physical part $(exec := plant)$.
- *Ctrl_evade* event representing the *evade* mode. It assigns the parameter *evade_value* to the control variable *ctrlV* and gives the turn to the physical part $(exec := plant)$. This event can be triggered even if the system has not yet reached the boundary of $evt\_trig(plantV)$, i.e. the system still satisfies the formula *safe*. However, we keep the guarantee that it will be triggered exactly when the system reaches the boundary of $evt\_trig(plantV)$ since the controller is continuous.

```
Ctrl_normal
ANY  $nrml\_value$
WHERE
    grd1: $nrml\_value \in RReal$
    grd2: $exec = ctrl$
    grd3: $safe(plantV(t) \mapsto$
              $nrml\_value) = TRUE$
THEN
    act1: $ctrlV := nrml\_value$
    act2: $exec := plant$
END
```

```
Ctrl_evade
ANY  $evade\_value$
WHERE
    grd1: $exec = ctrl$
    grd2: $evade\_value \in RReal$
THEN
    act1: $ctrlV := evade\_value$
    act2: $exec := plant$
END
```

Time-Triggered Model refines the previous model to get a system corresponding to that described by *Model 2*. As mentioned in the previous section, the sensors of a *Time-Triggered* system take periodic measurements of physical state variables and its controller executes each time those sensors updates are taken. Moreover, the longest time between sensors updates is bounded by a symbolic duration named ϵ. Therefore, the controller can execute at least every ϵ time. For this purpose, we have calibrated a new variable named d (variable t in dR$\mathcal{L}$) to know whether the duration ϵ is reached or not. This variable is reset (set to *Rzero*) before each execution of the physical part and evolves according to a function f_evol_d defined in the associated context. We have also added the constraint $d(t') \leq \epsilon$ to the first action of the *Progress* event to be sure that the sensors updates occurs at least every ϵ. Since the controller of a *Time-Triggered* system must make a choice that will be safe for up to ϵ time, we defined a new safety envelope named *safeEpsilon* ($safe_\epsilon(plantV, ctrlV)$ in dR$\mathcal{L}$). As in dR$\mathcal{L}$, we have replaced *safe* with *safeEpsilon* by defining a new event named *Ctrl_normal_time*. This latter refines the *Ctrl_normal* event and is triggered when a given value, *nrml_value*, satisfies the formula *safeEpsilon*. In that case, we assign this value to *ctrlV* and give the turn to the physical part.

5.2 Application

To apply our approach to a concrete system, we define two concrete models, *Concrete_System_Event-triggered* model and *Concrete_System_Time-triggered* model. The first model refines the *Event-Triggered* model through replacing *plantV* by the system physical state variables, defining the system safety properties as invariants in addition to the associated evolution function f_evol_plantV, then the formula *safe* is instanciated to define the system safety envelope. The second model, can either refine the first one or the *Time-Triggered* model. If we choose the first alternative, the refined model will then inherit the system safety properties but on the other hand we must add the notion of control period *epsilon*.

5.3 Proof of Refinement

In Event-B, two proof obligations are generated to prove that a concrete Event-B machine refines an abstract one:

- *Guard strengthening (GRD):* ensures that a concrete guard is stronger than the corresponding abstract one.
- *Action simulation (SIM):* ensures that each concrete action is not contradictory to the corresponding abstract one.

As mentioned earlier, we replaced the safety envelope formula *safe* by the formula *safeEpsilon* in the *Ctrl_normal_time* event. In this case, the following *Guard strengthening (GRD)* proof obligation has been generated:

$$(exec = ctrl \land safeEpsilon(plantV(t) \mapsto nrml\_value) = TRUE) \\ \Rightarrow safe(plantV(t)) = TRUE$$

To prove that the concrete machine, *TimeTriggered_M*, refines the abstract one, *EventTriggered_M*, we must prove that, during a control period ϵ, the safety formula *safeEpsilon*, defined in the concrete model, implies the safety formula *safe* defined in the abstract one. This proof is similar to the *PO 1* provided by d$\mathcal{RL}$. *PO 2* and *PO 3* are not generated as refinement POs by the proof obligation generator of Event-B, though they are needed to prove the refinement relation between our two generic models, *Time-Triggered* model and *Event-Triggered* model. Therefore, they must be added manually as Event-B proof obligations. Since we model the evolution of the physical state variables using a single event, *Plant* in the *Event-Triggered* model and *Plant_time* in the *Time-Triggered* model, we will then replace the equations of the d$\mathcal{RL}$ POs, $\bar{plantV} = S_{\bar{plantV_0},\bar{ctrlV}}(t)$ and $\bar{plantV} = S_{\bar{plantV_0},evade\_value}(t)$ by the guard $exec = prg$. *init* represents the initial conditions of the modeled system and *plantV(t0)* represents the initial value of the physical state variable *plantV*. In Event-B, the proof obligations are as follows:

- PO 2:

$$safeEpsilon(plantV(t0) \mapsto nrml\_value) = TRUE \land evt\_trig(plantV(t0)) \\ \land init \land (Rzero \mapsto t \in leq) \land (t \mapsto epsilon \in leq) \land exec = prg \\ \Rightarrow evt\_trig(plantV)$$

- PO 3:

$$evt\_trig(plantV(t0)) \land init \land (Rzero \mapsto t \in leq) \land (t \mapsto epsilon \in leq) \\ \land exec = prg \Rightarrow evt\_trig(plantV)$$

These two proof goals are based on the safety envelope of the system and the choices of the control variable. When the safety envelope of the system is satisfied, the controller can non-deterministically choose between the *normal* mode or the *evade* mode. In the case of a *Event-Triggered* system, we have the guarantee that the controller is able to switch to the *evade* mode exactly when the safety envelope is no longer satisfied. While in a *Time-Triggered* system, we must prove that *nrml_value* and *evade_value* guarantee that the system will not exceed the domain of the safety envelope within time ϵ.

5.4 Case Study

To validate our approach, we chose the *Stop Sign* case study [14] which deals with a stop sign controller whose objective is to ensure the stopping of a car before a stop signal SP. The control strategy is to adjust the velocity of the car by accelerating or braking, without ever backing down. The continuous behavior of this system is modeled by the position and the velocity of the car specified respectively by the state variables p and v, as well as its acceleration represented by the control variable $ctrlV$. This continuous behavior evolves according to linear differential equations, $p' = v, v' = ctrlV \equiv (\frac{dp}{dt} = v, \frac{dv}{dt} = ctrlV)$, which describe the evolution of the position and the velocity over time. The system can behave according to the following two discrete states: State *accelerate* and State *braking*. State *accelerate* is triggered when the car is very far from the stop signal SP. In this case, the car velocity can evolve according to a non-deterministic value assigned to the control variable $ctrlV$. This value must never exceed the physical limits of the car expressed by A (maximum limit of acceleration) and B (maximum limit of braking). State *braking* is triggered when the car is very close to the stop signal SP. In this case, we must decrease the car velocity by assigning $-B$ to $ctrlV$. To model this system using our approach, we followed the schema depicted in Fig. 2. The whole models of this development can be downloaded from https://cloud.lacl.fr/index.php/s/aiKiPxkrfmWpakR.

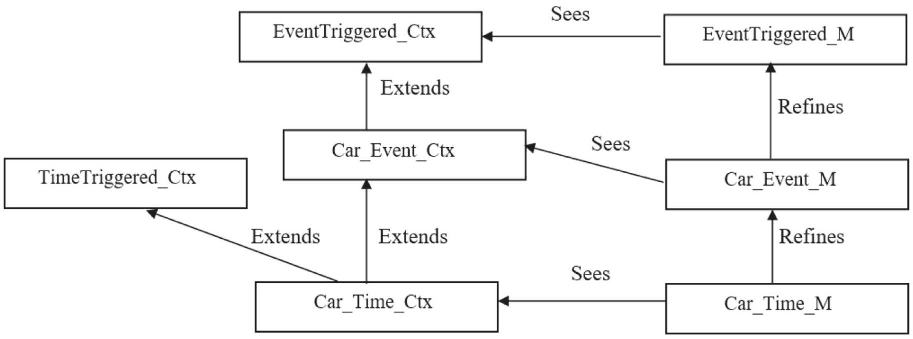

Fig. 2. *Stop sign* case study development schema.

Machine *Car_Event_M* refines Machine *EventTriggered_M* through replacing the generic state variable *plantV* by the physical state variables associated with the *Stop sign* case study, p and v. This replacement is done using the operator *bind* defined in the differential equations theory [6]. The physical part is modeled by the *Plant_event_car* event. This latter refines the *Plant* event by adding a witness that replaces the evolution of the generic state variable *plantV* by the evolution of the position p and the velocity v represented by the *f_evol_plantV* defined in the associated context. State *accelerate* is modeled using the *Ctrl_acceleration_car* event that refines the *Ctrl_normal* event.

State *braking* is modeled through replacing the value *evade_value* by $-B$ in the *Ctrl_evade* event.

Machine *Car_Time_M* refines Machine *Car_Event_M* in order to preserve the system safety property, $p \leq SP$ and $0 \leq v$. If we prove that $safeTime$ implies $safe$, and the *Car-Event-Triggered* model satisfies the property $p \leq SP$, so we can say that the *Car-Event-Triggered* model also satisfies this property. *Car_Time_M* is based on the Machine *TimeTriggered_M*, therefore we added the variable d and its evolution.

To prove that *Car_Time_M* refines *Car_Event_M* we must prove the three associated POs presented in Sect. 5.3. As we mentioned above, the choice of the parameters *nrml_value* and *evade_value* is the key to prove the safety of the system which can be done by using external mathematical tools for a parametric analysis since the differential equation of the *Stop Sign* case study is linear.

5.5 Comparing Event-B Refinement with Differential Refinement Logic

Type of Refinement. Event-B refinement is based on the execution traces starting from the initial state. Therefore, to prove that a concrete Event-B machine refines an abstract one, we have to establish that the set of execution traces of the concrete one is included in that corresponding to the abstract one. The refinement of dR$\mathcal{L}$ is based on reachable states. In hybrid automata and hybrid programs, a state is defined by a couple $(x\_s, plantV)$ composed of the current discrete state x_s and the current value of the continuous variable $plantV$. Therefore, to prove that a hybrid program α refines another hybrid program β ($\alpha \leq \beta$), we have to establish that the set of reachable states from a state s following the transitions of α is included in the set of reachable states from the same state s following some transitions of β.

Both Event-B refinement and dR$\mathcal{L}$ allow preserving the safety properties of the refined model. This is ensured in dR$\mathcal{L}$ through combining refinement relations and modalities. Despite the several features of dR$\mathcal{L}$'s refinement, computing reachable states for non linear system requires solving non-linear real arithmetic problems which is difficult in general [15]. Moreover, dR$\mathcal{L}$ refinement does not preserve the safety properties on the traces, but it is less constrained than the Event-B refinement.

Proofs Complexity. As we mentioned earlier, dR$\mathcal{L}$ has introduced a refinement strategy based on comparing reachable states for hybrid programs. Using this refinement strategy, one can start with an ideal system where the controller has continuous control over the system behavior (*Event-Triggered* system), then introduces a more realistic system where the controller interrupts the physical part at least every ϵ time (*Time-Triggered* system). The main advantage of dR$\mathcal{L}$ is that it uses differential equations to describe the continuous evolution of a given hybrid system by employing differential invariants, differential cuts, and differential refinement techniques. Moreover, the refinement relation between *Time* and *Event-Triggered* systems have been successfully proved using the d$\mathcal{RL}$'s refinement proof rules. Despite these advantages, dR$\mathcal{L}$ is not supported by any prover, which makes the proofs difficult to achieve in the case of complex systems especially for systems with more than two modes. Through using Event-B, we can overcome this limitation since its support tools aid in discharging proof obligations either automatically or with the interactive prover. Therefore using our approach, we can model an hybrid system with more than two modes.

The major limitation in using Event-B to model and verify hybrid systems is the absence of support for the continuous aspects of CPSs, such as continuous time and differential equations. As we mentioned, the approach proposed in [6] has tried to overcome this limitation by defining an Event-B theory that includes different kinds of differential equations. Using the abstract model of this approach, it becomes possible to represent the reasoning on hybrid programs in Event-B.

6 Conclusion and Future Work

In this paper, we have presented a proof-based approach that uses Event-B and its refinement technique to specify and verify *Event-Triggered* systems and *Time-Triggered* systems. We have defined two generic templates for these systems, directly inspired from the dR$\mathcal{L}$ specification, that represent hybrid systems as hybrid programs. dR$\mathcal{L}$ proof obligations have been defined to establish the refinement of the *Event-Triggered* template by the *Time-Triggered* template. Then we have compared the Event-B refinement with the dR$\mathcal{L}$ refinement and the generated POs. This led us to define new refinement POs in Event-B. One of the main advantages of Event-B is its support tools (provers, model-checkers, ...) to discharge POs, contrary to dR$\mathcal{L}$.

To demonstrate the usability of our approach, we have experimented it on a *Stop Sign* case study. In this case study, the differential equations that represent the evolution of the physical part are linear and can be easily solved. To handle more difficult differential equations we need to use an external tool like Mathematica [16], a symbolic mathematical computation system. Moreover, our approach is still in the abstract level where all transitions are instantaneous. It does not take into account the duration between the sending of continuous measurements by the sensors and their processing by the controller as well as the duration between the sending of actions by the controller and their execution. As future work, we plan to define a refinement of the *Time-Triggered* model to model these durations. We also plan to integrate Mathematica as a back-end tool in the Rodin platform to resolve differential equations.

References

1. ANR-17-CE25-0005: DISCONT ANR project (2017). https://discont.loria.fr

2. Lee, E.A.: Cyber physical systems: design challenges. In: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), pp. 363–369. IEEE (2008)
3. Kopetz, H.: Event-triggered versus time-triggered real-time systems. In: Karshmer, A., Nehmer, J. (eds.) Operating Systems of the 90s and Beyond. LNCS, vol. 563, pp. 86–101. Springer, Heidelberg (1991). https://doi.org/10.1007/BFb0024530
4. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reasoning **41**(2), 143–189 (2008). https://doi.org/10.1007/s10817-008-9103-8
5. Loos, S.M., Platzer, A.: Differential refinement logic. In: 2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), pp. 1–10. IEEE (2016)
6. Dupont, G., Aït-Ameur, Y., Pantel, M., Singh, N.K.: Proof-based approach to hybrid systems development: dynamic logic and Event-B. In: Butler, M., Raschke, A., Hoang, T.S., Reichl, K. (eds.) ABZ 2018. LNCS, vol. 10817, pp. 155–170. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91271-4_11
7. 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
8. Fulton, N., Mitsch, S., Quesel, J.-D., Völp, M., Platzer, A.: KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 527–538. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_36
9. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
10. 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. Transf. **12**(6), 447–466 (2010). https://doi.org/10.1007/s10009-010-0145-y
11. Butler, M., Maamria, I.: Mathematical extension in Event-B through the Rodin theory component (2010)
12. Banach, R., Butler, M., Qin, S., Verma, N., Zhu, H.: Core hybrid Event-B I: single hybrid Event-B machines. Sci. Comput. Programm. **105**, 92–123 (2015)
13. Butler, M., Abrial, J.-R., Banach, R.: Modelling and refining hybrid systems in Event-B and Rodin (2016)
14. 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. Transf. **18**(1), 67–91 (2015). https://doi.org/10.1007/s10009-015-0367-0
15. Chen, X.: Reachability analysis of non-linear hybrid systems using Taylor models. Ph.D. thesis, Fachgruppe Informatik, RWTH Aachen University (2015)
16. Wolfram, S.: The Mathematica Book, 5th edn. Wolfram Media, Champaign (2003)