An Event-B Based Generic Framework for Hybrid Systems Formal Modelling

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

An Event-B Based Generic Framework for Hybrid Systems Formal Modelling

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

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

Abstract. Designing hybrid systems requires the handling of discrete and continuous behaviours. The formal verification of such systems revolves around the use of heavy mathematical features, and related proofs. This paper presents a generic and reusable framework with different patterns, aimed at easing the design and verification of hybrid systems. It relies on refinement and proofs using Event-B, and defines an easily extensible set of generic patterns in the form of theories and models that are proved once and for all. The model of any specific hybrid system is then produced by instantiating the corresponding patterns. The paper illustrates the use of this framework by proposing to realise a well-known case study of the inverted pendulum, which design uses the approximation pattern formally defined and verified in Event-B.

1 Introduction

Formal modelling of hybrid systems requires means to describe both continuous and discrete behaviours in a single setting. Several approaches have been proposed to address this specificity, in general via the integration of theories of continuous functions and differential equations on the one hand, and logic-based reasoning on state-transitions systems on the other hand. The most common methods use hybrid automata [3] to model such systems and hybrid model checking [4,14,15,18] to verify their properties. In addition, some other approaches such as hybrid CSP [9,19], hybrid programs [20,21], continuous action systems [5], refinement and proof based methods with Event-B [11–13,22] and Hybrid Event-B [6], have been developed as well.

In previous work, we extended Event-B modelling language via the development of various theories to design hybrid systems using a correct-by-construction approach [11–13,22]. Theories for continuous mathematics, an approximate refinement relation for approximation following the retrenchment principle [7] and different hybrid systems architectures have been formally modelled.

The *objective of this paper* is two-fold. First, it presents a generic and reusable framework, relying on Event-B, to support and ease the design of hybrid systems. It is built from the generalisation of the models we defined in our previous work and on their instantiation to model specific hybrid systems. This framework defines a set of formalised and reusable patterns, verified once and for all.

https://doi.org/10.1007/978-3-030-63461-2_5

Second, it demonstrates the application of this framework, and in particular the approximation pattern, with the development of the inverted pendulum case study, where approximate refinement is used to linearise non-linear dynamics.

The *organisation of this paper* is as follows. Next section presents the designed generic framework. Section 3 describes the case study of the inverted pendulum and Sect. 4 gives an overview of Event-B. Section 5 presents the generic models and theories composing the framework and Sect. 6 is dedicated to the development of the case study. Finally, Sect. 7 concludes the paper.

2 The Designed Framework

The generic framework for formal modelling and verification of hybrid systems relies on the various developments we have conducted to model and verify different types of hybrid systems [10–13]. These developments revealed several reusable building blocks seen as formal development patterns formalised in Event-B. Figure 1 depicts the framework and its different components, split in two categories: *reusable* and *specific*.

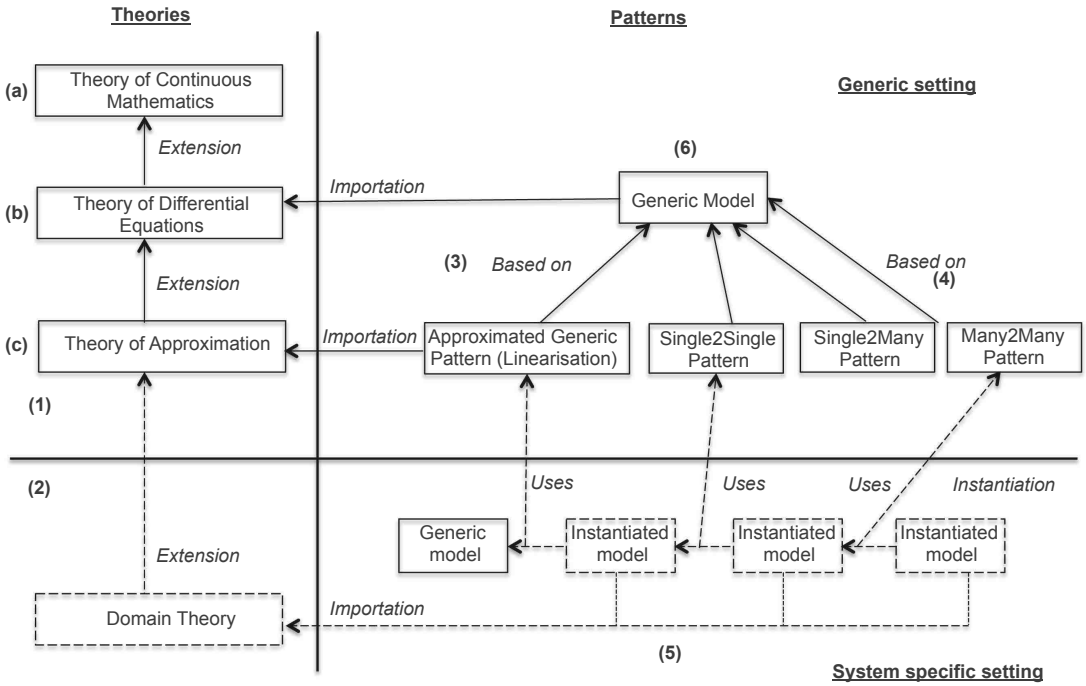

Fig. 1. Our framework: the big picture

2.1 Reusable Components

These components are the theories and the Event-B generic model and patterns to be instantiated for specific hybrid systems.

Relevant Theories (1 on Fig. 1). Event-B is based on set theory and first order logic; this low mathematical level is very expressive, but makes it difficult to

handle continuous features, essential in hybrid system modelling. These required mathematical concepts, not available in core Event-B, are defined within mathematical theories, referenced by the models. They make available reals, continuous functions, differential equations and associated properties. In addition, they also formalise approximation and define an approximate refinement operator, which is not available in native Event-B. These theories are defined incrementally, as denoted by the *Extends* operator.

Generic Model and Patterns for Hybrid Systems (3 and 4 on Fig. 1*).* They are parameterised Event-B models *proved once and for all.*

Generic Model (6 on Fig. 1*).* It formalises the generic pattern of Fig. 6. It is the root model from which all the other models are derived, using Event-B refinement. Plant and controller behaviours, together with sensing and actuation actions are meddled at a higher abstract level.

Architecture Patterns (4 on Fig. 1*).* These specific patterns introduce either centralised or distributed control and one or many controlled plants. Three Event-B models refining the generic model define three architecture patterns as *SingleToSingle* [12], *SingleToMany* [11] and *ManyToMany* [13].

Approximation Pattern (3 on Fig. 1*).* It consists of another Event-B model, refining the generic model and formalising a commonly used approximation operation realised by designers. In Event-B, this pattern encodes an approximate refinement operation following the principle of retrenchment. Linearisation is an example of such an approximation: a non-linear differential equation is approximately refined by a linear one.

The above introduced components represent a library of patterns deployed to model specific hybrid systems. They are proved once and for all.

2.2 Specific Components

These components are both theories and models developed for particular hybrid systems. They are obtained either by theories extensions or pattern instantiation.

Domain Theories (2 on Fig. 1*).* These specific theories describe the characteristics of the plant involved in the developed hybrid system, e.g.: kinematics of a car, robot motion, inverted pendulum, etc. In many cases, more than one theory may be needed, in particular when it involves different domains (signal processing, kinematics, etc.).

Instantiation Models (5 on Fig. 1*).* They are formal models for specific hybrid systems. They are obtained by applying the different patterns sequentially, starting with the generic model. Event-B refinement is used to instantiate those patterns, and witnesses are provided for the parameters of the generic model and patterns. These models refer to the domain theories to access the relevant characteristics of the considered system.

In the remainder of this paper, we show how the defined approximation pattern is deployed. It encodes an Event-B approximate refinement relationship. The case of the inverted pendulum is considered.

3 Case Study: The Inverted Pendulum

We consider the well-known case study of the inverted pendulum. This problem is particularly relevant as it imposes the use of *linearisation* in order to be correctly implemented. The case study is then realised in the Event-B based defined framework using Rodin.

3.1 Description

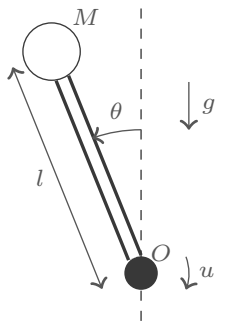

Fig. 2. Inverted pendulum

An object M is attached to a rigid rod of length l, that is itself attached to a step motor at point O. This point is also the origin of the coordinate system. The angle between the rod and the vertical axis is denoted θ, and the motor is capable of providing a torque, denoted u. The system is subject to standard G-force, of intensity g. The goal of the controller is to stabilise the rod in its vertical position by instrumenting the motor (and its torque u). From physics laws, we obtain the system's equation in θ (Fig. 2):

$$\ddot{\theta} - \frac{g}{l}\sin(\theta) = u\cos(\theta) \tag{1}$$

Equation 2 is derived from Eq. 1, as an ODE $\dot{\eta} = f_{NonLin}(\eta, u)$ where $\eta = [\,\theta\;\dot{\theta}\,]^\top$ and u is some control command:

$$f_{NonLin}((x_1, x_2), u) = (x_2, u\cos(x_1) + \frac{g}{l}) \tag{2}$$

The factor $\omega_0^2 = \frac{g}{l}$ is generally constant and ω_0 is the angular frequency (pulsatance) of the system, linked to the period of the pendulum's oscillations. This system is controllable when $\theta < \theta_{max}$, where θ_{max} is fixed by ω_0.

Due to the terms $\sin(\theta)$ and $\cos(\theta)$, the system's ODE is non-linear, meaning that it does not have an explicit solution, and the reachability is undecidable. However, when θ is small enough, it is possible to approximate $\sin(\theta)$ and $\cos(\theta)$; more precisely, given θ_{bound}, there exists δ such that, for any θ with $|\theta| < \theta_{bound}$, then $|\sin(\theta) - \theta| < \delta$ and $|1 - \cos(\theta)| < \delta$.

Assuming this condition holds, it is possible to approximate Eq. 1 to a simpler form, so-called *linearised*:

$$\ddot{\theta} - \frac{g}{l}\theta = v, \tag{3}$$

with v an adequate linear control command linked to u after linearisation. It can be expressed as the ODE $\dot{\eta} = f_{Lin}(\eta, v)$ where:

$$f_{Lin}((x_1, x_2), v) = (x_2, v + \frac{g}{l}x_1). \tag{4}$$

This ODE is linear, making it much easier to handle.

3.2 Requirements

The requirements of the system can be summarised as follows:

FUN1 The controller senses the angle (θ) of the pendulum (`:sense_angle`)
FUN2 If the value of the sensed angle is not 0, the controller sends a command to stabilise the pendulum at $\theta = 0$ (`:calculate_control`)
SAF1 For $|\theta| < \theta_{max}$, the system is always controllable
ENV1 The system is subject to perturbations that may cause its angle to vary
ENV2 There exists θ_{bound} such that $|\theta| < \theta_{bound}$; therefore, the non-linear system and the linearised system are always close up to $\delta > 0$

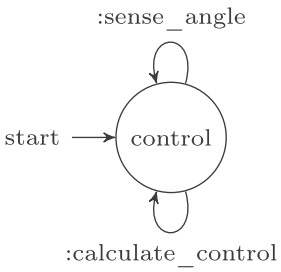

Fig. 3. System mode automaton

4 Event-B

Event-B [1] is a *correct-by-construction* method based on set theory and first order logic. It relies on a powerful state-based modelling language where a set of events allows for state changes[1] (see Table 1). A set of proof obligations (see Tables 2a and 2b) is automatically generated for each model. Event-B is associated with a proof system which contains a set of proof rules for formal reasoning. The design process of the system model consists of an abstract model leading to the final concrete model. Each refinement gradually introduces additional system design decisions.

Context (Table 1.a). A `Context` component describes the static properties. It introduces all the definitions, axioms and theorems needed to describe the required concepts using elementary components such as *Carrier sets* s, *constants* c, *axioms* A and *theorems* T_{ctx}.

Machines (Table 1.b). `Machine` describes the model behaviour as a transition system. A set of guarded events is used to modify a set of states using Before-After Predicates (BAP) to record variable changes. They use *variables* x, *invariants* $I(x)$, *theorems* $T_{mch}(x)$, *variants* $V(x)$ and *events* evt (possibly guarded by G and/or parameterized by α) as core elementary components.

Refinements (Table 1c). Refinement introduces different characteristics such as functionality, safety, reachability at different abstraction levels. It decomposes a *machine*, a state-transition system, into a less abstract one, with more design decisions (refined states and events) moving from an abstract level to a less abstract one (simulation relationship). Gluing invariants relating to abstract and concrete variables ensures property preservation.

[1] **Notation.** The superscripts ^{A} and ^{C} denote abstract and concrete features.

Table 1. Model structure

Context	**Machine**	**Refinement**
CONTEXT Ctx	MACHINE M^A	MACHINE M^C
SETS s	SEES Ctx	REFINES M^A
CONSTANTS c	VARIABLES x^A	VARIABLES x^C
AXIOMS A	INVARIANTS $I^A(x^A)$	INVARIANTS $J(x^A, x^C) \wedge I^C(x^C)$
THEOREMS T_{ctx}	THEOREMS $T_{mch}(x^A)$	...
END	VARIANT $V(x^A)$	EVENTS
	EVENTS	EVENT evt^C
	EVENT evt^A	REFINES evt^A
	ANY α^A	ANY α^C
	WHERE $G^A(x^A, \alpha^A)$	WHERE $G^C(x^C, \alpha^C)$
	THEN	WITH
	$x^A :\mid BAP^A(\alpha^A, x^A, x^A)$	$x^{A\prime}, \alpha^A{:}W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^C)$
	END	THEN
	...	$x^C :\mid BAP^C(\alpha^C, x^C, x^C)$
		END
		...
(a)	(b)	(c)

Table 2. Proof Obligations

(1) Theorems	$A \Rightarrow T_{ctx}$ $A \wedge I^A(x^A) \Rightarrow T_{mac}(x^A)$
(2) Invariant preservation (INV)	$A \wedge I_A(x^A) \wedge G_A(x^A, \alpha^A)$ $\wedge BAP^A(x^A, \alpha^A, x^{A'})$ $\Rightarrow I^A(x^{A\prime})$
(3) Event feasibility (FIS)	$A \wedge I_A(x^A) \wedge G^A(x^A, \alpha^A)$ $\Rightarrow \exists \alpha^A \cdot BAP^A(x^A, \alpha^A, x^{A\prime})$
(4) Variant progress	$A \wedge I^A(x^A) \wedge G^A(x^A, \alpha^A)$ $\wedge BAP^A(x^A, \alpha^A, x^{A\prime})$ $\Rightarrow \ V(x^{A\prime}) < V(x^A)$

(a) Machine Proof obligations

(5) Event Simulation (SIM)	$A \wedge I^A(x^A) \wedge J(x^A, x^C)$ $\wedge G^C(x^C, \alpha^C)$ $\wedge W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^{C\prime})$ $\wedge \boldsymbol{BAP^C\,(x^C, \alpha^C, x^{C\prime})}$ $\Rightarrow \boldsymbol{BAP^A\,(x_A, \alpha^A, x^{A\prime})}$
(6) Guard Strengthening (GS)	$A \wedge I^A(x^A) \wedge J(x^A, x^C)$ $\wedge W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^{C\prime})$ $\wedge \boldsymbol{G^C\,(x^C, \alpha^C) \Rightarrow G_A\,(x^A, \alpha^A)}$
(7) Invariant preservation (INV)	$A \wedge I^A(x^A)$ $\wedge G^C(x^C, \alpha^C)$ $\wedge W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^{C\prime})$ $\wedge BAP^C(x^C, \alpha^C, x^{C\prime})$ $\wedge \boldsymbol{J(x^A, x^C) \Rightarrow J(x^{A\prime}, x^{C\prime})}$

(b) Refinement Proof obligations

Proof Obligations (PO) and Property Verification. Tables 2a and 2b provide a set of proof obligations to guarantee Event-B model consistency, including refinements. These PO are automatically generated. They must be proven in order to establish the correctness of the defined model.

Extensions with Mathematical Theories. An extension of Event-B is defined [2] to support externally defined mathematical theories. It offers the introduction of new data types by defining new types, sets operators, theorems and associated rewrite and inference rules, all bundled in so-called *theories*.

Rodin. It is an Eclipse based IDE for Event-B project management, model edition, refinement and proof, automatic PO generation, model checking, model animation and code generation. It is equipped with standard provers, including support for external provers such as SMT solvers. A plug-in [8] is also available to support the development of mathematical theories.

5 Modelling the Generic Model and Patterns in Event-B

Modelling hybrid systems requires to handle continuous behaviours. We thus need to access specific mathematical objects and properties, which are not natively available in Event-B. These concepts such as differential equations and their associated properties have been modelled through an intensive use of Event-B theories and have been used to model various case studies found in [10–12].

This section describes the generic resources used by the defined framework. They correspond to the upper parts (1), (3), (4) and (6) of Fig. 1.

5.1 Theories for Continuous Mathematics and Differential Equations (1a and 1b on Fig. 1)

In order to deal with continuous objects, theories have been defined for continuous functions, (ordinary) differential equations as well as for their properties. They are used throughout the defined models. Their complete definitions are available at https://irit.fr/~Guillaume.Dupont/models.php. Some of these concepts as they are used in this paper are recalled below.

Hybrid Modelling Features. Modelling hybrid systems requires to introduce multiple basic operators and primitives defined below.

```
THEORY
  TYPE PARAMETERS E, S
  DATA TYPES
    DE(S)
    CONSTRUCTORS
      ode(f : ℙ(ℝ × S × S), η0 : S, t0 : ℝ)
  OPERATORS
    solutionOf <predicate> (DR : ℙ(ℝ),
        η : ℝ+ ⇸ S, eq : DE(S))
    Feasible <predicate> (xs : STATES,
        η : ℝ+ ⇸ S, DR : ℙ(ℝ), P :
        (ℝ+ ⇸ S) × (ℝ+ ⇸ S), I : ℙ(S))
    Solvable <predicate> (DR : ℙ(ℝ), eq :
        DE(S), I : ℙ(S))
    ...
END
```

Fig. 4. Differential equation theory snippet

- **DE**(S) type for differential equations which solutions evolve over set S
- **ode**(f, η_0, t_0) is the ODE (Ordinary Differential Equation) $\dot{\eta}(t) = f(\eta(t), t)$ with initial condition $\eta(t_0) = \eta_0$
- **solutionOf**$(D, \eta, \mathcal{E})$ is the predicate stating that function η is a solution of equation $\mathcal{E}$ on subset D
- **Solvable**$(D, \mathcal{E}, \mathcal{I})$ predicate states that equation $\mathcal{E}$ has a solution defined on subset D that satisfies the constraint $\mathcal{I}$
- **Feasible**$(x_s, x_p, D, \mathcal{P}, \mathcal{I})$, the feasible predicate states that, given x_s and x_p, there exists $x'_p \in D \to S$ such that $\mathcal{P}(x_s, x_p, x'_p)$ holds and $\forall t^* \in D, x'_p(t^*) \in \mathcal{I}$. In state x_s, the predicate

$\mathcal{P}$ holds for x_p and its next value x'_p on time interval D fulfils the constraint $\mathcal{I}$. It defines the feasibility condition of a continuous variable (e.g. a state in a model) change. This operator is used to define the continuous before-after predicate (CBAP).

These features are encoded in a theory from which we show a snippet on Fig. 4 (the theory accumulates more than 150 operators and 350 properties).

5.2 A Theory of Approximation (1c on Fig. 1)

In addition to the continuous mathematical objects of Sect. 5.1, a theory of approximation is required to implement approximate refinement in Event-B. In the following, we introduce the necessary concepts and operators related to approximation and used throughout this paper. Let us assume (E, d) to be a metric space with distance d.

Approximation ($\approx^\delta$). Let $x, y \in E$ and $\delta \in \mathbb{R}^+$. We say that x approximately equals to y by δ (or x is a δ-approximation of y) iff $x \approx^\delta y \equiv d(x, y) \leq \delta$.

δ-expansion. Let $S \subseteq E$ and $\delta \in \mathbb{R}^+$. The δ-expansion of S, noted $\mathcal{E}_\delta(S)$, is defined as $\mathcal{E}_\delta(S) = \{y \in E \mid \exists x \in S, x \approx^\delta y\} = \{y \in E \mid \exists x \in S, d(x, y) \leq \delta\}$.

δ-membership ($\in^\delta$). Let $\delta \in \mathbb{R}^+$, $S \subseteq E$ and $x \in E$. x belongs to S up to δ, denoted $x \in^\delta S$, iff x belongs to the δ-expansion of S. We write $x \in^\delta S \equiv x \in \mathcal{E}_\delta(S) \equiv \exists y \in S, d(x, y) \leq \delta$.

Extended δ-membership Operators. δ-membership is extended as follows.

- Let $f \in F \to E$ and $X \subseteq F$, then $f \in^\delta_X S \equiv \forall x \in X, f(x) \in^\delta S$
- Let $\Sigma \in F \to \mathbb{P}(E)$ (multivalued function), then $f \in^\delta_X \Sigma \equiv \forall x \in X, f(x) \in^\delta \Sigma(x)$

When X is omitted, the operator is applied on the function's domain of definition (i.e., $X = \mathrm{dom}(f)$).

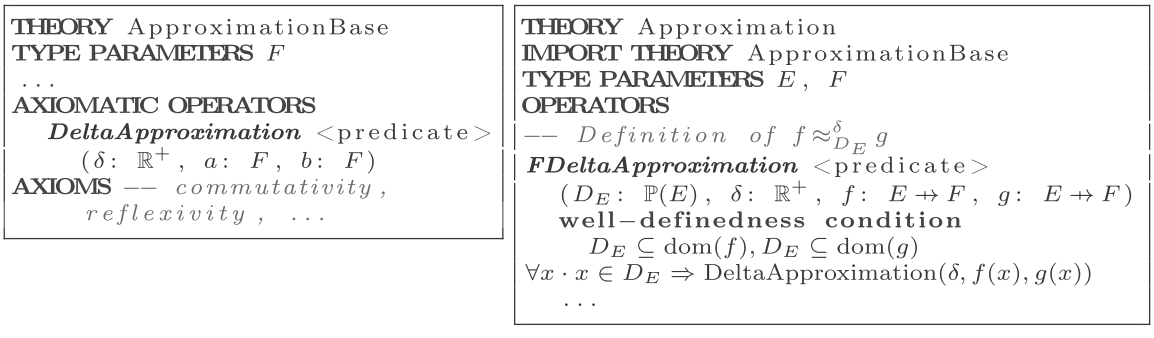

Fig. 5. Approximation theory excerpt

Note: δ-approximation ($\approx^\delta$) (resp. δ-membership ($\in^\delta$)) is a weak version of equality (resp. set membership). Indeed, when $\delta = 0$, by the separation property

of distance d, we obtain $x \approx^0 y \ \equiv \ d(x,y) \leq 0 \ \equiv \ x = y$. It follows that for any $S \subseteq E$, $\mathcal{E}_0(S) = S$ and thus $x \in^0 S \ \equiv \ x \in S$.

Implementation Using Theories. The above defined operators and concepts have been implemented in two Event-B theories (*ApproximationBase* and *Approximation*) from which an excerpt is given in Fig. 5. Typically, approximation ($\approx^\delta$) is expressed algebraically through the *DeltaApproximation* operator, while its extension to functions is implemented as the *FDeltaApproximation* operator.

5.3 The Generic Model (6 on Fig. 1)

As mentioned previously, the core Event-B does not support continuous behaviours. To handle such behaviours, we have introduced a generic model, acting as a meta-model encoding a hybrid automaton corresponding to the generic hybrid system structure depicted in Fig. 6. The notions of time, continuous states, continuous gluing invariants, continuous assignment and continuous events are introduced. The obtained model interleaves continuous events (with duration) and discrete events (instantaneous) as defined in [10–13].

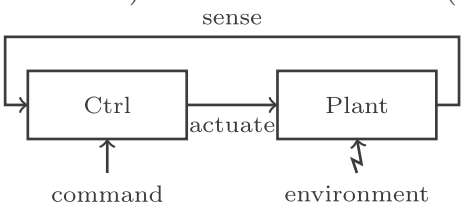

Fig. 6. Generic hybrid system pattern

The generic model is the entry point of the framework on which every pattern is based. It takes the form of an Event-B model that summarises and abstracts any hybrid system conforming to Fig. 6. Refinement is then used to derive any specific hybrid system from it.

Time. A notion of time is needed to define continuous behaviours. We thus introduce dense time $t \in \mathbb{R}^+$, modelled as a continuously evolving variable.

System State. According to the architecture of hybrid systems, we have identified two types of states:

- **Discrete state** $x_s \in \mathit{STATES}$ is a variable that represents the controller's internal state. It evolves in a point-wise manner with instantaneous changes.
- **Continuous state** $x_p \in \mathbb{R}^+ \rightarrow S$ represents the plant's state and evolves continuously. It is modelled as a function of time with values in space S.
  In the following, we use x to denote the union of discrete and continuous state variables.

Continuous Assignment. Continuous variables are essentially functions of time and are at least defined on $[0,t]$ (where t is the current time). Updating such variables, thus, requires to (1) make the time progress from t to $t' > t$, and (2) to append to the already existing function a new piece corresponding to its extended behaviour (on $[t,t']$) while ensuring its"past" (i.e. whatever happened on $[0,t]$) remains unchanged.

Similarly to the classic Event-B's before-after predicate (BAP), we define a *continuous before-after predicate* ($CBAP$) operator, denoted $:|_{t\to t'}$, as follows[2]:

$$x_p :|_{t\to t'} \mathcal{P}(x_s, x_p, x_p') \,\&\, \mathcal{I} \equiv \; [0,t] \lhd x' = [0,t] \lhd x \qquad (PP)$$
$$\wedge\, \mathcal{P}(x_s, [t,t'] \lhd x_p, [t,t'] \lhd x_p') \qquad (PR)$$
$$\wedge\, \forall t^* \in [t,t'], x_p'(t^*) \in \mathcal{I} \qquad (LI)$$

The operator consists of three parts: past preservation and coherence at assignment point (PP), before-after predicate on the added section (PR), and local invariant preservation (LI). The discrete state variables x_s do not change in the interval $[t,t']$ but the predicate $\mathcal{P}$ may use it for control purposes. We note $CBAP(x_s, x_p, x_p') \equiv PP(x_p, x_p') \wedge PR(x_s, x_p, x_p') \wedge LI(x_p, x_p')$.
From the above definition, shortcuts are introduced for readability purposes:

- Continuous assignment: $x :=_{t\to t'} f \,\&\, \mathcal{I} \;\equiv\; x :|_{t\to t'} x' = f \,\&\, \mathcal{I}$
- Continuous evolution along a solvable differential equation $\mathcal{E} \in \mathbf{DE}(S)$: $x :\sim_{t\to t'} \mathcal{E} \,\&\, \mathcal{I} \;\equiv\; x :|_{t\to t'} \mathbf{solutionOf}([t,t'], x', \mathcal{E}) \,\&\, \mathcal{I}$

The Generic Model in Event-B. Once all the features have been defined, we can describe the Event-B model.

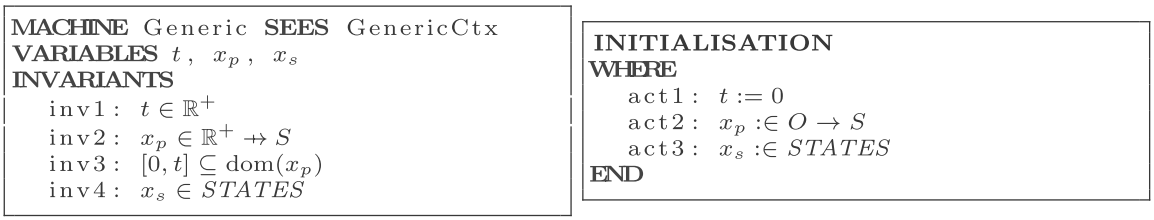

Fig. 7. Generic model Event-B machine header

The model handles three variables, time t, the continuous state x_p and the discrete state x_s constrained using invariants (`inv1-4`). They are initialised with 0 for t and using non-deterministic assignment for x_p and x_s. Further refinements provide more detailed value(s) (Fig. 7).

The events of the generic model follow the arrows of Fig. 6. Figure 8 shows the `Transition` and the `Sense` events modelling discrete state changes. Such change can arise following the detection of a change in the plant (sensing) or can be induced by the controller itself (`Transition`) after a calculation, at the end of a timer, and so on. This difference is captured by guards 2 and 3 of `Sense`, referencing the continuous state. `Transition` and `Sense` are so-called *discrete* events: they are timeless and instantaneous.

Figure 9 shows the other two types of events `Behave` and `Actuate` to model a change in the plant, induced either by a change in the controller (actuation) or by

[2] The $\lhd$ operator denotes the domain restriction operator.

Transition
ANY s
WHERE
grd1 : $s \in \mathbb{P}1(\text{STATES})$
THEN
act1 : $x_s :\in s$
END

Sense
ANY s , p
WHERE
grd1−2 : $s \in \mathbb{P}1(\text{STATES})$, $p \in \mathbb{P}(\text{STATES} \times \mathbb{R} \times S)$
grd3 : $(x_s \mapsto t \mapsto x_p(t)) \in p$
THEN
act1 : $x_s :\in s$
END

Fig. 8. Transition and sense events

Behave
ANY eq , t'
WHERE
grd0 : $t' \in \mathbb{R}^+ \wedge t' > t$
grd1 : $eq \in DE(S)$
grd2 : $\mathbf{Solvable}([t, t'], eq, \top)$
THEN
act1 : $t, x_p :\sim_{t \to t'} eq \,\&\, \top$
END

Actuate
ANY eq , s , H , t'
WHERE
grd0 : $t' \in \mathbb{R}^+ \wedge t' > t$
grd1−2 : $eq \in DE(S)$,$\mathbf{Solvable}([t, t'], eq, H)$
grd3−4 : $s \subseteq$ STATES, $x_s \in s$
grd5−6 : $H \subseteq S$,$x_p(t) \in H$
THEN
act1 : $t, x_p :\sim_{t \to t'} eq \,\&\, H$
END

Fig. 9. Transition and sense events

the environment (behave). Both events rely on a continuous assignment operator described above. The link between the controller and actuation is modelled by `grd3-4` in `Actuate` (absent from `Behave`). Also, the behaviour set in actuation is constrained by an evolution domain (`grd5-6`).

`Behave` and `Actuate` are *continuous* events: unlike discrete events, they have a duration. *Discrete* events are instantaneous and they preempt *continuous* ones.

Continuous Gluing Invariant. It is defined with the generic form $x_p^A \in \mathcal{O} \circ x_p^C$ where $\mathcal{O} \in S^C \leftrightarrow S^A$ is a relation linking abstract and continuous state-spaces. This invariant glues the abstract x_p^A and concrete x_p^C continuous variables. It is qualified as *exact* since it maps concrete values in S^C to abstract values in S^A using the $\in$ operator. Definition of an approximate gluing invariant, extending exact one, using the $\in^\delta$ operator is presented in next section.

5.4 The Approximation Pattern (3 on Fig. 1)

As mentioned in Sect. 2, we have chosen to illustrate the application of the generic framework using the approximation pattern. The choice of this pattern is motivated by the fact that 1) it uses an externally defined theory (see Sect. 5.2) not available in native Event-B and 2) it requires a specific refinement relationship, weakening classical refinement following the principle of retrenchment [7], and formalising the approximation of a continuous behaviour by another one. We particularly study the case of linearisation, when moving from a behaviour characterised by a non-linear differential equation to a behaviour characterised by a linear differential equation. The definition of this approximate refinement operation follows the approach of [16,17] where approximation is embedded in a simulation relationship. In addition, our definition offers an inductive process.

In this section, we present the approximation pattern as a refinement between an abstract machine (which elements are super-scripted with A) and a concrete machine (with superscript C). Figure 10 shows the respective headers of the machines. Approximation deals with continuous variables ($x_p^{A/C}$).

MACHINE M_A
VARIABLES t, x_s^A, x_p^A
INVARIANTS
inv1: $t \in \mathbb{R}^+$
inv2: $x_s^A \in STATES$
inv3: $x_p^A \in \mathbb{R} \nrightarrow S^A$
inv4: $[0,t] \subseteq \mathrm{dom}(x_p^A)$
inv5: $\forall t^* \cdot t^* \in [0,t] \Rightarrow x_p^A(t^*) \in \mathcal{I}^A$

MACHINE M_C **REFINES** M_A
VARIABLES t, x_s^C, x_p^C
INVARIANTS
inv2: $x_s^C \in STATES^C$
inv3: $x_p^C \in \mathbb{R} \nrightarrow S^C$
inv4: $[0,t] \subseteq \mathrm{dom}(x_p^C)$
inv5: $\forall t^* \cdot t^* \in [0,t] \Rightarrow x_p^C(t^*) \in \mathcal{I}^C$
inv6: $\boldsymbol{x_p^A \in^\delta \mathcal{O} \circ x_p^C}$
inv7: $J_s(x_s^C, x_s^A)$

Fig. 10. Machine header

The approximation pattern is applied at the refinement level using *approximated* relations instead, and built using the operators defined in Sect. 5.2, e.g. $\approx^\delta$ or $\in^\delta$ (see Fig. 10). It is formalised by `inv6` whee the $\in$ operator is replaced by its approximated version ($\in^\delta$).

EVENT $\mathbf{Sense}_A$
WHEN grd1: $x_p^A(t) \in \mathcal{G}^A$
THEN act1: $x_s^A :\in STATES$
END

EVENT $\mathbf{Sense}_C$ **REFINES** $\mathbf{Sense}_A$
WHEN grd1: $x_p^C(t) \in \mathcal{G}^C$
THEN act1: $x_s^C :\in STATES$
END

Fig. 11. Sense event

Sensing events (Fig. 11) remain relatively unchanged compared to normal refinement. Guard $\mathcal{G}^C$ must be defined carefully: $\mathcal{G}^C$ shall be stronger than $\mathcal{G}^A$, *taking into account the error* allowed by approximate refinement (guard strengthening PO).

Actuation (Fig. 12) is almost unchanged. The provided *witness* (`WITH` clause) shall ensure preservation of approximation after occurrence of the `Actuate` event. This witness leads to a feasibility proof obligation to guarantee that the property $x_p^{A\prime} \in^\delta \mathcal{O} \circ x_p^{C\prime}$ holds (i.e. approximation holds).

EVENT $\mathbf{Actuate}_A$
ANY t_p
WHEN
grd1: $t_p \in \mathbb{R} \wedge t_p > t$
grd2: $x_s = State$
grd3: $\mathbf{Feasible}(x_p^A, [t, t_p], \mathcal{P}^A, \mathcal{I}^A)$
grd4: $x_p^A(t) \in \mathcal{I}^A$
THEN
act1: $x_p^A :|_{t \to t'} \mathcal{P}^A(x_s^A, x_p^A, x_p^{A\prime}) \,\&\, \mathcal{I}^A$
END

EVENT $\mathbf{Actuate}_C$ **REFINES** $\mathbf{Actuate}_A$
ANY t_p
WHEN
grd1: $t_p \in \mathbb{R} \wedge t_p > t$
grd2: $x_s = State$
grd3: $\mathbf{Feasible}(x_p^C, [t, t_p], \mathcal{P}^C, \mathcal{I}^C)$
grd4: $x_p^C(t) \in \mathcal{I}^C$
WITH $\boldsymbol{x_p^{A\prime} \in^\delta \mathcal{O} \circ x_p^{C\prime}}$
THEN
act1: $x_p^C :|_{t \to t'} \mathcal{P}^C(x_s^C, x_p^C, x_p^{C\prime}) \,\&\, \mathcal{I}^C$
END

Fig. 12. Actuate event

Table 3. Refinement POs for the generic model: case of approximate refinement

(5) Event Simulation (SIM)	$A \wedge x_p^A \in \mathbb{R} \nrightarrow S^A \wedge [0,t] \subseteq \mathrm{dom}(x_p^A) \wedge x_p^C \in \mathbb{R} \nrightarrow S^C \wedge [0,t] \subseteq \mathrm{dom}(x_p^C)$ $\wedge x_p^A \in \mathcal{I}^A \wedge x_p^C \in \mathcal{I}^C \wedge x_p^C(t) \in \mathcal{G}^C$ $\wedge \boldsymbol{x_p^A \in^\delta \mathcal{O} \circ x_p^C} \wedge x_p^{A\prime} \in^\delta \mathcal{O} \circ x_p^{C\prime}$ $\wedge PP(x^C, x^{C'}) \wedge PR(x^C, x^{C'}) \wedge LI(x^C, x^{C'})$ $\Rightarrow PR(x^A, x^{A'}) \wedge LI(x^A, x^{A'})$
(6) Guard Strengthening (GS)	$A \wedge x_p^A \in \mathbb{R} \nrightarrow S^A \wedge [0,t] \subseteq \mathrm{dom}(x_p^A) \wedge x_p^C \in \mathbb{R} \nrightarrow S^C \wedge [0,t] \subseteq \mathrm{dom}(x_p^C)$ $\wedge x_p^A \in \mathcal{I}^A \wedge x_p^C \in \mathcal{I}^C$ $\wedge \boldsymbol{x_p^A \in^\delta \mathcal{O} \circ x_p^C} \wedge x_p^{A\prime} \in^\delta \mathcal{O} \circ x_p^{C\prime}$ $\wedge \boldsymbol{x_p^C(t) \in \mathcal{G}^C \Rightarrow x_p^A(t) \in \mathcal{G}^A}$
(7) Invariant Preservation (INV)	$A \wedge x_p^A \in \mathbb{R} \nrightarrow S^A \wedge [0,t] \subseteq \mathrm{dom}(x_p^A) \wedge x_p^C \in \mathbb{R} \nrightarrow S^C \wedge [0,t] \subseteq \mathrm{dom}(x_p^C)$ $\wedge x_p^A \in \mathcal{I}^A \wedge x_p^C \in \mathcal{I}^C \wedge x_p^C(t) \in \mathcal{G}^C \wedge \boldsymbol{x_p^A \in^\delta \mathcal{O} \circ x_p^C}$ $\wedge PP(x^C, x^{C'}) \wedge PR(x^C, x^{C'}) \wedge LI(x^C, x^{C'})$ $\boldsymbol{\Rightarrow x_p^{C\prime} \in \mathbb{R} \nrightarrow S^C \wedge [0,t'] \subseteq \mathrm{dom}(x_p^{C\prime}) \wedge x_p^{C\prime} \in \mathcal{I}^C \wedge x_p^{A\prime} \in^\delta \mathcal{O} \circ x_p^{C\prime}}$

Revisited Proof Obligations. Approximation, similar to the *concedes* relation of retrenchment, extends the standard refinement operation which proof obligations are given in Table 2b. The use of well-definedness and witnesses in approximate refinement leads to an updated set of proof obligations (highlighted in bold) in Table 3.

Exact Refinement as a Particular Case of Approximate Refinement. We note that, when $\delta = 0$ in the operators defined in Sect. 5.1, we actually find back standard exact operators: $\approx^0 \equiv =$, $\in^0 \equiv \in$, etc. By restriction/strengthening, this means that, for $\delta = 0$, defined approximate refinement looks like exact refinement.

5.5 The Architecture Patterns (4 on Fig. 1)

Architecture patterns have been introduced in order to model the different types of structures hybrid systems may have: one controller controlling one plant (simple control, *Single2Single*), one controller and several plants (centralised control, *Single2Many*) and several controllers with several plants (distributed control, *Many2Many*). These patterns have been thoroughly studied, formalised and implemented in [11–13] respectively.

6 Modelling Hybrid Systems

Modelling specific hybrid systems follows the bottom part of Fig. 1. Two steps are identified: the first step introduces definitions relevant to the system (Fig. 1(2)), completing the generic theories of Fig. 1(1) with the relevant types, axioms and theorems for modelling the specific features of the system to design. The second step (Fig. 1(5)) is performed by refining and instantiating patterns to obtain the desired system (used patterns of Fig. 1). This process is exemplified below with the inverted pendulum case study.

6.1 Application to the Case Study

Our framework is used to address the case study introduced in Sect. 3. The exact use of the framework is depicted on Fig. 13, and follows the two steps discussed before: first, a theory for the physics of the inverted pendulum is defined (Fig. 13 (2)); second, the *Single2Single* pattern is applied to the generic model in order to derive a non-linear pendulum model. Finally, the approximation pattern is used to derive a linearised pendulum model from the non-linear one.

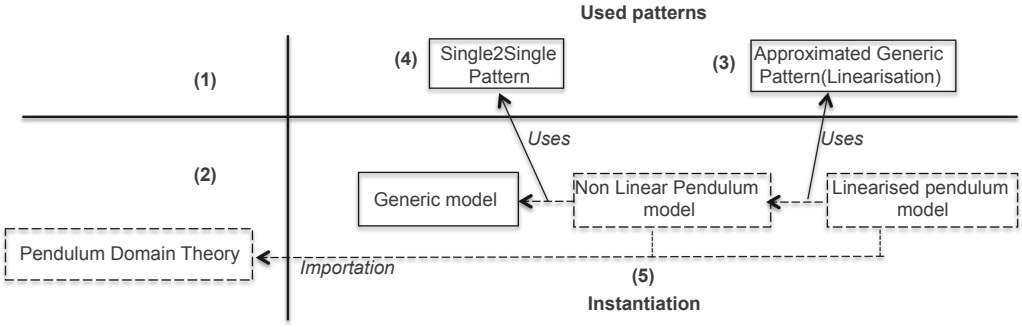

Fig. 13. Framework application to the case study

Step 1: A Theory for Simple Inverted Pendulums (2 in Fig. 13). Before modelling the actual system, we need to develop a domain theory of pendulums, that holds every important concepts needed to model this kind of system: differential equations (both non-linear and linearised) and adequate controls for the systems, as well as various physical and mathematical properties that will help in establishing the system's correctness. The definitions of such a theory correspond to (2) in Fig. 13.

THEORY InvertedPendulum IMPORTS Approximation
OPERATORS
  $\mathbf{PendulumFun_{NonLin}}$ <expression> $(\omega_0 : \mathbb{R})$
    direct definition
      $(\lambda t \mapsto (x_1 \mapsto x_2) \mapsto u \cdot t \in \mathbb{R}^+ \wedge (x_1 \mapsto x_2) \in \mathbb{R}^2 \wedge u \in \mathbb{R} \mid x_2 \mapsto (u \cos(x_1) + \omega_0^2 \sin(x_1)))$
  $\mathbf{Pendulum_{NonLin}}$ <expression> $(\omega_0 : \mathbb{R},\ x_0 : \mathbb{R}^2,\ t_0 : \mathbb{R})$
    direct definition code$(\mathbf{PendulumFun_{NonLin}}(\omega_0), x_0, t_0)$
  $\mathbf{PendulumFun_{Lin}}$ <expression> $(\omega_0 : \mathbb{R})$
    direct definition
      $(\lambda t \mapsto (x_1 \mapsto x_2) \mapsto u \cdot t \in \mathbb{R}^+ \wedge (x_1 \mapsto x_2) \in \mathbb{R}^2 \wedge u \in \mathbb{R} \mid x_2 \mapsto (u + \omega_0^2 x_1))$
  $\mathbf{Pendulum_{Lin}}$ <expression> $(\omega_0 : \mathbb{R},\ x_0 : \mathbb{R}^2,\ t_0 : \mathbb{R})$
    direct definition code$(\mathbf{PendulumFun_{Lin}}(\omega_0), x_0, t_0)$
AXIOMATIC DEFINITIONS
  OPERATORS
    theta_max <expression> $(\omega_0 : \mathbb{R}) : \mathbb{R}$
    $\mathbf{PendulumControl_{NonLin}}$ <expression> $(\omega_0 : \mathbb{R},\ (\theta_0, \dot{\theta}_0) : \mathbb{R}^2,\ t_0 : \mathbb{R}^+) : \mathbb{R} \nrightarrow \mathbb{R}$
    $\mathbf{PendulumControl_{Lin}}$ <expression> $(\omega_0 : \mathbb{R},\ (\theta_0, \dot{\theta}_0) : \mathbb{R}^2,\ t_0 : \mathbb{R}^+) : \mathbb{R} \nrightarrow \mathbb{R}$
    ...

Fig. 14. Pendulum theory excerpt

Listing of Fig. 14 gives an extract of the pendulum defined domain theory. It mainly defines the differential equations associated with both the non-linear ($Pendulum_{NonLin}$) and linearised ($Pendulum_{Lin}$) pendulum systems. It also proposes control functions for both systems ($PendulumControl_{NonLin}$ and $PendulumControl_{Lin}$ resp.) which are algebraically defined together with useful properties used in proofs.

Step 2: Non-Linear Inverted Pendulum Model (5 in Fig. 13). The *Single2Single* architecture pattern is used to derive, by refinement of the generic model, a first model of the inverted pendulum, which features the non-linear differential equation. This step correspond to (4) in Fig. 13.

CONTEXT PendulumCtx **EXTENDS** GenericCtx
CONSTANTS ω_0 , θ_{max} , θ_0 , control
AXIOMS
axm1−2: $\omega_0 \in \mathbb{R}$, $\omega_0 \neq 0$
axm3−5: $\theta_{max} = theta\_max(\omega_0)$, $\theta_0 \in \mathbb{R}$, $|\theta_0| < \theta_{max}$
axm6: partition($STATES$, {control})
END

The context for this model defines the system's pulsatance (ω_0 in `axm1-2`) and its associated maximum controllable angle (θ_{max} in `axm3-5`). Last, the only state of the system's mode automaton is declared in *control* (`axm6`).

MACHINE Pendulum **REFINES** Generic
SEES PendulumCtx
VARIABLES
t , θ , $\dot{\theta}$, t^{sense} , θ^{sense} , $\dot{\theta}^{sense}$, $control\_fun$
INVARIANTS
inv1−4: $\theta \in \mathbb{R} \nrightarrow \mathbb{R}$, $\dot{\theta} \in \mathbb{R} \nrightarrow \mathbb{R}$, $[0,t] \subseteq \text{dom}(\theta)$, $[0,t] \subseteq \text{dom}(\dot{\theta})$
inv5: $x_p = [\theta\ \dot{\theta}]^\top$
inv6: $\forall t^* \cdot t^* \in [0,t] \Rightarrow |\theta(t^*)| < \theta_{max}$
inv7: $x_s = \text{control}$
inv8−10: $t^{sense} \in \mathbb{R}^+$, $\theta^{sense} \in \mathbb{R}$, $\dot{\theta}^{sense} \in \mathbb{R}$
inv11−12: $control\_fun \in \mathbb{R} \nrightarrow \mathbb{R}$, $[t^{sense}, +\infty[\subseteq \text{dom}(control\_fun)$
inv13: $|\theta^{sense}| \leq \theta_{max}$

INITIALISATION
WITH
x_p': $x_p' = \{0 \mapsto (\theta_0 \mapsto 0)\}$
x_s': $x_s' = control$
THEN
act1: $t := 0$
act2: $\theta := \{0 \mapsto \theta_0\}$
act3: $\dot{\theta} := \{0 \mapsto 0\}$
act4: $t^{sense} := 0$
act5: $\theta^{sense}, \dot{\theta}^{sense} := \theta_0, 0$
act6: $control\_fun := PendulumControl_{NonLin}(\omega_0, (\theta_0 \mapsto 0), 0)$
END

Fig. 15. Machine header and initialisation

Listings of Fig. 15 give the machine header and the initialisation of the system. The continuous state is the vector $[\dot{\theta}\,\theta]^\top$ defined in `inv1-4`. `inv5` glues this continuous state to the generic one (x_p). It is constrained by `inv6`. The mode automaton of the system defines the *control* (`inv7`) state and the discrete state of the machine comprises variables to store the observation of the system when sensing (*sense* variables super-scripted of `inv8-10`). At initialisation θ is set to an arbitrary value θ_0 and the control function *control_fun* (`inv11-12`) is assigned to the non-linear differential equation modelling the behaviour of the pendulum borrowed from the `InvertedPendulum` theory.

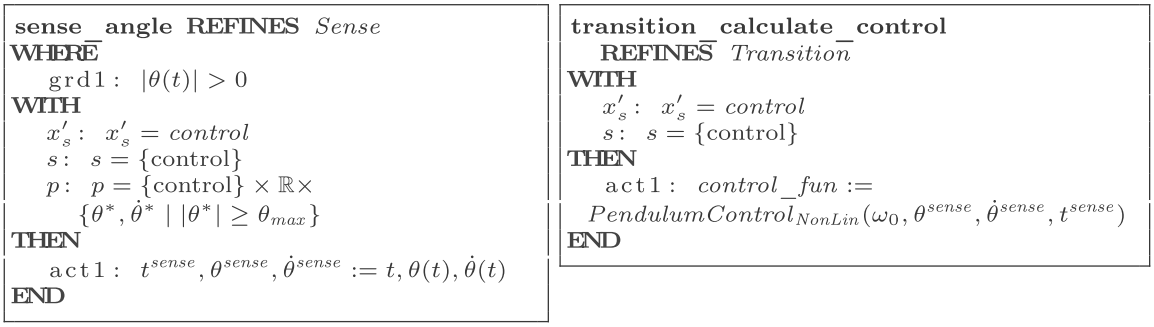

Fig. 16. Sensing and transition

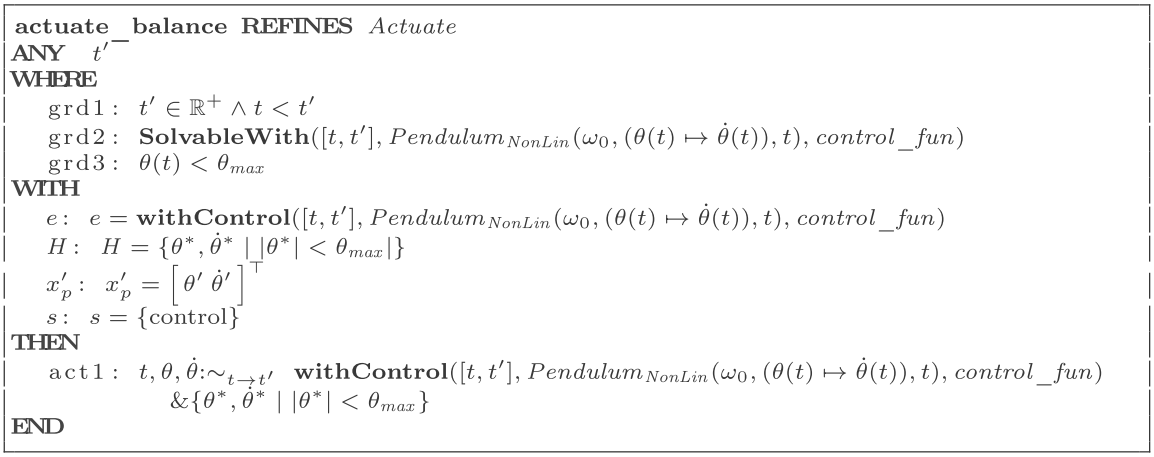

Fig. 17. System actuation

Following the hybrid automaton of Fig. 3, the system defines two discrete events: the sensing event `sense_angle` reads and stores the continuous state in the *sense* variables, and the transition event `transition_calculate_control` uses the stored continuous state to set up an adequate control function, stored in *control_fun*. An actuation event updates the plant's behaviour with the $Pendulum_{NonLin}$ differential equation, associated with *control_fun*'s new value (Fig. 17).

Step 3: Linearised Inverted Pendulum Model (5 in Fig. 13). The approximation pattern ((3) in Fig. 13) is used to refine the non-linear pendulum model into a linearised one. The theory of approximation of Sect. 5.2 as well as the domain theory of pendulums given in Sect. 6.1 allow us to set up an approximate refinement relationship between the two linear and non-linear models.

CONTEXT PendulumLinCtx **EXTENDS** PendulumCtx
CONSTANTS δ , δ_{ctrl} , θ_{bound}
AXIOMS
axm1−2: $\delta \in \mathbb{R}, 0 < \delta$
axm3: $\delta_{ctrl} = PendulumControlDelta(\omega_0, \delta)$
axm4−7: $\theta_{bound} \in \mathbb{R}, 0 < \theta_{bound}$, $\theta_{bound} < \theta_{max}, \delta < \theta_{bound}$
END

The context of this system extends the one for the non-linear pendulum. It introduces a fixed δ (`axm1-2`), to model the maximum difference between the state of both system models as well as a stricter bound for θ (θ_{bound} in `axm4-7`). Using the pendulum theory, it is possible to synthesise δ_{ctrl}, the maximum difference between the controls of each system model (`axm3`) (Fig. 18).

MACHINE PendulumLin **REFINES** Pendulum
SEES PendulumLinCtx
VARIABLES t , t^{sense} , $control\_fun\_lin$, θ_{Lin} , $\dot{\theta}_{Lin}$, θ^{sense}_{Lin} , $\dot{\theta}^{sense}_{Lin}$
INVARIANTS
inv1−4: $\theta_{Lin} \in \mathbb{R} \nrightarrow \mathbb{R}, \dot{\theta}_{Lin} \in \mathbb{R} \nrightarrow \mathbb{R}$, $[0,t] \subseteq \mathrm{dom}(\theta_{Lin}), [0,t] \subseteq \mathrm{dom}(\dot{\theta}_{Lin})$
inv5: $[\theta_{Lin}\ \dot{\theta}_{Lin}]^\top \approx^{\delta}_{[0,t]} [\theta\ \dot{\theta}]^\top$
inv6: $\forall t^* \cdot t^* \in [0,t] \Rightarrow |\theta(t^*)| < \theta_{bound} \wedge |\dot{\theta}_{Lin}(t^*)| < \theta_{bound} - \delta$
inv7: $control\_fun\_lin \in \mathbb{R} \nrightarrow \mathbb{R}$
inv8: $control\_fun\_lin \approx^{\delta_{ctrl}}_{[t^{sense},+\infty[} control\_fun$
inv9−10: $\theta^{sense}_{Lin} \in \mathbb{R}$, $\dot{\theta}^{sense}_{Lin} \in \mathbb{R}$
inv11: $|\theta^{sense}_{Lin}| \leq \theta_{bound} - \delta$
inv12: $[\theta^{sense}\ \dot{\theta}^{sense}]^\top \approx^{\delta} [\theta^{sense}_{Lin}\ \dot{\theta}^{sense}_{Lin}]^\top$

INITIALISATION
WITH
θ'; $\theta' = \theta_0$, $\dot{\theta}'$: $\dot{\theta}' = 0$
$\theta^{sense\prime}$: $\theta^{sense\prime} = \theta_0$, $\dot{\theta}^{sense\prime}$: $\dot{\theta}^{sense\prime} = 0$
$control\_fun'$: $control\_fun' = PendulumControl_{NonLin}(\omega_0, (\theta_0 \mapsto 0), 0)$
THEN
act1: $t := 0$
act2: $t^{sense} := 0$
act3: $\theta_{Lin} := \{0 \mapsto \theta_0\}$
act4: $\dot{\theta}_{Lin} := \{0 \mapsto 0\}$
act5: $control\_fun\_lin := PendulumControl_{Lin}(\omega_0, (\theta_0 \mapsto 0), 0)$
act6: $\theta^{sense}_{Lin}, \dot{\theta}^{sense}_{Lin} := \theta_0, 0$
END

Fig. 18. Machine header and initialisation

The machine header, presented in Fig. 18 is close to the abstract non-linear model with a new state $[\theta_{Lin}\ \dot{\theta}_{Lin}]$ (`inv1-4`). It is glued with the non-linear abstract state via the **approximate gluing invariant**, `inv5`. Both abstract and concrete states have strengthened constraints (`inv6`) to ensure the existence of the approximation relationship. The control function *control_fun* is refined by *control_lin_fun* (`inv7`). It is linked to the abstract control using the approximated refinement gluing invariant of `inv8`. Refined versions of the sensing variables are introduced. θ^{sense}_{Lin} and $\dot{\theta}^{sense}_{Lin}$ are defined in `inv9-10` and constrained in `inv11`. They are linked to the abstract sensing variables using `inv12` gluing invariant. Last, `Initialisation` on the right-hand side of Fig. 18 updates the state variables. Simple witnesses (`WITH` clause) are provided for the refined (disappearing) variables.

sense_angle REFINES *sense_angle*
WHERE
grd1 : $|\theta_{Lin}(t)| > 0$
WITH
$\theta^{sense\prime}, \dot{\theta}^{sense\prime}$:
$[\theta^{sense\prime}\ \dot{\theta}^{sense\prime}]^\top \approx^{\delta} [\theta^{sense\prime}_{Lin}\ \dot{\theta}^{sense\prime}_{Lin}]^\top$
THEN
act1 : $t^{sense}, \theta^{sense}_{Lin}, \dot{\theta}^{sense}_{Lin} :=$
$t, \theta_{Lin}(t), \dot{\theta}_{Lin}(t)$
END

transition_calculate_control REFINES
transition_calculate_control
WITH
control_fun′ :
$control\_fun' \approx^{\delta}_{[t^{sense}, +\infty[}\ control\_fun\_lin'$
THEN
act1 : $control\_fun\_lin :=$
$PendulumControl_{Lin}(\omega_0, \theta^{sense}_{Lin}, \dot{\theta}^{sense}_{Lin}, t^{sense})$
END

Fig. 19. Linear refined sense and transition with approximation

The `sense` and `transition` events (Fig. 19) update system variables. Witnesses are provided to link the state variables of the abstract and refined models.

actuate_balance REFINES *actuate_balance*
ANY t'
WHERE
grd1 : $t' \in \mathbb{R}^+ \wedge t < t'$
grd2 : **SolvableWith**$([t, t'], Pendulum_{Lin}(\omega_0, (\theta_{Lin}(t) \mapsto \dot{\theta}_{Lin}(t)), t), control\_fun\_lin)$
grd3 : $|\theta_{Lin}(t)| < \theta_{bound} - \delta$
WITH
$\theta, \dot{\theta}$: $[\theta'\ \dot{\theta}'] \approx^{\delta}_{[0,t']} [\theta'_{Lin}\ \dot{\theta}'_{Lin}]$
THEN
act1 : $t, \theta_{Lin}, \dot{\theta}_{Lin} :\sim_{t \to t'}$
withControl$([t, t'], Pendulum_{Lin}(\omega_0, (\theta_{Lin}(t) \mapsto \dot{\theta}_{Lin}(t)), t), control\_fun\_lin)$
$\&\{\theta^*, \dot{\theta}^* \mid |\theta^*| < \theta_{bound} - \delta\}$
END

Fig. 20. Linear refined actuation with approximation

Last, the `actuation` event of Fig. 20 updates the state variables by providing a witness using the `WITH` clause for the abstract continuous state using the defined approximation. It is central to maintain the approximated gluing invariant.

6.2 Assessment

The main advantage of the defined framework is proof reuse. Indeed, proofs are realised at the generic level and do not need to be discharged again. The only remaining proofs relate to the instantiation of the pattern (under the form of refinement POs) and the specific features of the model, namely invariants.

The first refinement generated 100 POs. 34% of them relate to refinement, while the vast majority of the others are about well-definedness (33%) of the operators and invariants (37%), most of which are typing invariants. The second one generated 63 POs. 19% of them come from refinement, and more specifically when using the approximation pattern. Again, a significant number of POs relate to well-definedness (33%) and invariants (44%) are mainly typing invariants. The interactive proofs have been carried out using rewriting rules, deductive rules application, and external automatic provers calls, combined in tactics.

The theory plug-in is still in the early stage of development, it hinders proof automation. For this reason and because our models extensively rely on it, proofs had to be done interactively. All the models shown in this paper can be accessed at https://irit.fr/~Guillaume.Dupont/models.php.

7 Conclusion

The definition of the proposed framework results from the different experiments and models that we defined in previous work. Some of the patterns are identified from our Event-B developments for a simple controlled system [10,12], centralised control of many plants [11] and distributed control of many controllers [13].

In this paper, we have shown how the defined framework of Fig. 1 is put into practice to model the inverted pendulum case study. First, we applied the *Single2Single* architecture pattern and then the *Approximation* pattern as depicted in Fig. 13 to obtain a verified linearised model of the inverted pendulum.

The Event-B method together with its IDE Rodin proved powerful to support the formalisation of such generic patterns as parameterised Event-B models. These patterns and the necessary theories are proved to be correct once and for all. Specific hybrid systems models are obtained by instantiation i.e. by providing witnesses for the parameters of the generic models satisfying the properties (invariants) expressed at the generic models level. Only this instantiation step requires to be checked, the other proofs are reused, they are not re-proved again.

The defined framework is open and can be enriched, at the generic level, with new patterns and other theories. The added patterns may be connected through refinement to existing ones or may use new other theories. Each time a pattern is added, it needs to be formally verified. Examples of patterns that can be added are: discretisation pattern, PID[3] controller pattern, introduction of theories for partial differential equations or delayed differential equations, etc. In addition, other theories axiomatising different domains from physics should be defined in order to broaden the use of the defined framework.

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., Leuschel, M., Schmalz, M., Voisin, L.: Proposals for mathematical extensions for Event-B. Technical report (2009). http://deploy-eprints.ecs.soton.ac.uk/216/
3. 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

[3] Proportional, Integral, and Derivative.

4. Asarin, E., Dang, T., Maler, O.: The d/dt tool for verification of hybrid systems. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 365–370. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45657-0_30
5. Back, R.-J., Petre, L., Porres, I.: Generalizing action systems to hybrid systems. In: Joseph, M. (ed.) FTRTFT 2000. LNCS, vol. 1926, pp. 202–213. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45352-0_17
6. 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)
7. Banach, R., Poppleton, M., Jeske, C., Stepney, S.: Engineering and theoretical underpinnings of retrenchment. Sci. Comput. Program. **67**(2–3), 301–329 (2007)
8. Butler, M., Maamria, I.: Practical theory extension in Event-B. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods. Lecture Notes in Computer Science, vol. 8051, pp. 67–81. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39698-4_5. Essays Dedicated to Jifeng He on the Occasion of his 70th Birthday
9. Chaochen, Z., Ji, W., Ravn, A.P.: A formal description of hybrid systems. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) HS 1995. LNCS, vol. 1066, pp. 511–530. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0020972
10. Dupont, G., Aït-Ameur, Y., Pantel, M., Singh, N.K.: Hybrid systems and Event-B: a formal approach to signalised left-turn assist. In: Abdelwahed, E.H., et al. (eds.) MEDI 2018. CCIS, vol. 929, pp. 153–158. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02852-7_14
11. Dupont, G., Aït-Ameur, Y., Pantel, M., Singh, N.K.: Handling refinement of continuous behaviors: a refinement and proof based approach with Event-B. In: 13th International Symposium TASE, pp. 9–16. IEEE Computer Society Press (2019)
12. 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
13. Dupont, G., Aït-Ameur, Y., Pantel, M., Singh, N.K.: Formally verified architecture patterns of hybrid systems using proof and refinement with Event-B. In: Raschke, A., Méry, D., Houdek, F. (eds.) ABZ 2020. LNCS, vol. 12071, pp. 169–185. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-48077-6_12
14. Frehse, G.: PHAVer: algorithmic verification of hybrid systems past HyTech. Int. J. Softw. Tools Technol. Transf. **10**(3), 263–279 (2008)
15. Frehse, G., et al.: SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_30
16. Girard, A., Pappas, G.J.: Approximate bisimulation relations for constrained linear systems. Automatica **43**(8), 1307–1317 (2007)
17. Girard, A., Pappas, G.J.: Approximation metrics for discrete and continuous systems. IEEE Trans. Automat. Contr. **52**(5), 782–798 (2007)
18. Henzinger, T.A.: The theory of hybrid automata. In: Inan, M.K., Kurshan, R.P. (eds.) Verification of Digital and Hybrid Systems. NATO ASI Series, vol. 170, pp. 265–292. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-642-59615-5_13
19. Jifeng, H.: From CSP to hybrid systems. In: Roscoe, A.W. (ed.) A Classical Mind, pp. 171–189. Prentice Hall International (UK) Ltd. (1994)
20. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. **41**(2), 143–189 (2008)

21. 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
22. Su, W., Abrial, J.R., Zhu, H.: Formalizing hybrid systems with Event-B and the Rodin platform. Sci. Comput. Program. **94**, 164–202 (2014)