Formally Verified Architecture Patterns of Hybrid Systems Using Proof and Refinement with Event-B

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. Cyber-Physical Systems (CPS) play a central role in modern days technology. From simple thermostat controllers to more advanced autonomous cars, their versatility makes them perfect candidates for many applications, in particular for safety critical ones. Thus, their certification is a key issue and formal methods are good candidates to assess safety and produce associated certificates. Hybrid systems show continuous-time dynamics depending on mode that is required in several stages of the architecture of Cyber-Physical Systems. Our work addresses the problem of formally verifying hybrid systems using refinement and proof with Event-B. Our previous work [14] presented formally verified generic architecture patterns for designing centralised hybrid systems, based on our generic approach [15]. We extend this work and give a formally verified architecture pattern aimed at modelling distributed hybrid systems, featuring multiple plants and multiple controllers. We validate the approach and illustrate the use of the defined pattern on an extension of a very common case study, borrowed from literature.

Keywords: Hybrid systems · Cyber-physical systems · Architecture design patterns · Event-B · Refinement · Proof

1 Introduction

Cyber-Physical Systems (CPS) can be described as complex systems that integrate both discrete and continuous features [19]. Such system generally consists of a discrete algorithm or *controller* that interacts with a continuous process or *plant* in order to control its behaviour. The controller can retrieve information from the plant through sensors and may alter its behaviour with actuators.

Because of this hybridation, CPS are often regarded as quite hard to trust. However, their versatility, adaptability and price made them unavoidable in our everyday life, from Internet of Things (IoT) to smart systems (e.g. home automation, smart factories and so on), including, of course, critical systems such as transportation and medical devices. Being able to formally model and certify CPS is thus a major challenge nowadays.

This work was supported by grant ANR-17-CE25-0005 (The DISCONT Project https://discont.loria.fr) from the Agence Nationale de la Recherche (ANR).

A. Raschke et al. (Eds.): ABZ 2020, LNCS 12071, pp. 169–185, 2020.
https://doi.org/10.1007/978-3-030-48077-6_12

The design of formal modelling approaches for CPS development and/or certification has been addressed in various ways. In [4], Alur defines the hybrid automata formalism to model hybrid systems. Hybrid model-checkers such as HyTech, d/dt, PHaVer or SpaceEx can then be used to establish properties such as reachability.

In terms of modelling techniques, [18] have proposed HybridCSP as a hybrid extension of CSP [17]. [7] proposes a continuous extension of Action System. In the same manner, [8] proposes an hybrid extension to the Event-B method.

Proof-based approaches have also been used to try and formally prove CPS. [9] use Coq to that extent, starting from an annotated C program. [21] uses a special formalism (hybrid programs) to model and to prove hybrid systems using KeYmaera. Event-B has been used for modelling similar systems in [23] and [10].

However, all these approaches still require formal modelling expertise, where the developer needs to establish correctness using complex proof systems involving discrete and continuous mathematics features and proof rules. As a consequence, the use of these methods on a large scale is hindered, in particular in formal system engineering. So, easing CPS formal modelling and verification activities in presence of both discrete and continuous behaviours is still a challenge.

To address this challenge, we propose a systematic correct-by-construction approach to design hybrid systems based on the definition of architecture patterns. Indeed, one commonly used method in formal system engineering is to provide formalised generic patterns where relevant generic properties are established. Furthermore, those patterns can be instantiated for specific systems. In such a setting, the system developer selects the most adapted pattern and instantiates it. Proof obligations, in particular regarding well-definedness, may need to be discharged in order to inherit all the properties of the generic pattern.

In our previous work [14], we used Event-B to design and formalise commonly used architecture patterns (AP) for centralised hybrid systems. We based those patterns on our generic approach [15], allowing to model both discrete and continuous behaviours. In this paper, we extend these architecture patterns to model distributed hybrid systems i.e. systems that manage multiple autonomous subsystems, linked together by a communication network. A case study is given as a possible instantiation of this pattern, involving independent liquid tanks enforcing a global invariant that expresses safety properties.

This paper is organised as follows: Sect. 2 gives an overview of Event-B and Sect. 3 presents hybrid modelling features needed for hybrid system development. Section 4 introduces the architecture patterns identified when modelling hybrid systems. Section 5 recalls our generic method for designing hybrid systems in Event-B. Section 6 introduces a case study to support our work, which is solved in Sect. 7. Finally, Sect. 8 provides an assessment of the approach, and Sect. 9 concludes the paper and discusses possible future research directions.

2 Modelling Hybrid Systems with Event-B

Event-B method [2] supports the development of *correct-by-construction* complex systems. First order logic and set theory are used as core modeling language.

The design process consists of a series of refinements of an abstract model leading to the final concrete model. Refinement progressively contributes to add design decisions to the system.

Event-B *machines* formalize models described as state-transitions systems and a set of proof obligations are automatically generated for each model.

Notation. We use the superscripts ^{A} and ^{C} to 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
	$x^A :\mid BAP^A(\alpha^A, x^A, x^{A\prime})$	$x^{A\prime}, \alpha^A : W(\alpha^A, \alpha^C, x^A, x^{A\prime}, x^C, x^{C\prime})$
	END	THEN
	...	$x^C :\mid BAP^C(\alpha^C, x^C, x^{C\prime})$
		END
		...
(a)	(b)	(c)

- *Event-B Contexts (Table 1.a).* *Contexts* are the static part of a model. They set up all the definitions (carrier sets s and constants c), axioms (A) and theorems (T_{ctx}) needed to describe the required concepts.
- *Event-B Machines (Table 1.b).* A *machine* describes the dynamic part of a model as a transition system. A set of guarded events modifying a set of variables (state) represents the core concepts of a machine. *Variables* x, *invariants* $I(x)$, *theorems* $T_{mch}(x)$, *variants* $V(x)$ and *events* evt (possibly guarded by G and/or parameterized by α) are defined in a machine. *Invariants* and *theorems* formalize safety system properties while *variants* define convergence properties (reachability).
- *Event-B Refinements (Table 1.c).* A system is gradually designed by introducing properties (functionality, safety, reachability) at various abstraction levels. *Refinement* decomposes a *machine*, a state-transitions system, into a more concrete one, with more design decisions (refined states and events) while moving from an abstract level to a less abstract one. Abstract and

concrete variables are related by gluing invariants ensuring properties preservation between abstract and concrete models.
- *Proof Obligations (PO) and Property Verification.* To establish the correctness of an Event-B model, a set of POs are automatically generated from the calculus of substitutions. They need to be proved.
- *Extensions with mathematical theories.* In order to handle theories beyond set theory and first order logic, an Event-B extension to support externally defined mathematical theories has been proposed. It offers the capability to introduce new datatypes through the definition of new types, sets operators, theorems and associated rewrite and inference rules, in so-called *theories*.
- *Event-B and its IDE Rodin.* It offers resources for model edition, automatic PO generation, project management, refinement and proof, model checking, model animation and code generation. Several provers, like SMT solvers, are plugged to Rodin. In particular, a plug-in allows to define theories [11].

3 Hybrid Systems Modelling Features

Modelling hybrid systems requires handling of continuous behaviours. We thus need to access specific mathematical objects and properties, 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 [13–15].

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 those concepts as they are used in this paper are recalled below.

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

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

- **Discrete state** $x_s \in STATES$, variable that represents the controller's internal state. It evolves in a pointwise 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.

Hybrid Modeling Features. Modeling hybrid systems requires the introduction of multiple specific features which are defined below.

- **DE**(S) type for differential equations which solutions evolve over set S
- **ode**(f, η_0, t_0) represents the ODE[1] $\dot{\eta}(t) = f(\eta(t), t)$ with initial condition $\eta(t_0) = \eta_0$

[1] Ordinary Differential Equation.

```
THEORY
  TYPE PARAMETERS E, F
  DATA TYPES
    DE(F)
    CONSTRUCTORS
      ode(fun : ℙ(ℝ × F × F), initial : F, initialArg : ℝ)
  OPERATORS
    solutionOf <predicate> (D_R : ℙ(ℝ), η : ℝ⁺ ⇸ F, eq : DE(F))
    CauchyLipschitzCondition <predicate> (D_R : ℙ(ℝ), D_F : ℙ(F), eq : DE(F))
    Solvable <predicate> (D_R : ℙ(ℝ), eq : DE(F))
      direct definition
        ∃x · x ∈ (ℝ⁺ ⇸ F) ∧ solutionOf(D_R, x, eq)
    ...
END
```

Fig. 1. Differential equation theory snippet

- **solutionOf**$(D, \eta, \mathcal{E})$ is the predicate stating that function η is a solution of equation $\mathcal{E}$ on subset D
- **Solvable**$(D, \mathcal{E}, \mathcal{I})$ is the predicate stating that equation $\mathcal{E}$ has a solution defined on subset D so that the solution satisfies the constraint $\mathcal{I}$

These features have been encoded in a theory from which we show a snippet on Fig. 1 (the theory accumulates more than 150 operators and 350 properties).

Other, more specialised expressions and predicates are defined (*FlowEquation*, *FlowODE*) in additional theories. Note that all these definitions use *algebraic datatypes* together with axioms, theorems and proof rules.

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) append to the already existing function a new piece corresponding to its extended behavior (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:

$$x_p :|_{t \to t'} \mathcal{P}(x_s, x_p, x_p') \,\&\, \mathcal{I} \equiv \; [0, t] \lhd x' = [0, t] \lhd x \tag{PP}$$

$$\wedge \mathcal{P}(x_s, [t, t'] \lhd x_p, [t, t'] \lhd x_p') \tag{PR}$$

$$\wedge \forall t^* \in [t, t'], x_p'(t^*) \in \mathcal{I} \tag{LI}$$

We note $CBAP(x_s, x_p, x_p') \equiv PP(x_p, x_p') \wedge PR(x_s, x_p, x_p') \wedge LI(x_p, x_p')$. The operator consists of 3 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.

Note that this operator is well-defined if and only if $t' > t$, as otherwise the interval $[t, t']$ would not be well-defined.

From the above definition, shortcuts can be 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 $\mathcal{E} \in \mathbf{DE}(S)$: $x :\sim_{t \to t'} \mathcal{E} \,\&\, \mathcal{I} \;\equiv\; x :|_{t \to t'} \mathbf{solutionOf}([t, t'], x', \mathcal{E}) \,\&\, \mathcal{I}$.

4 Architecture Patterns for Modelling Hybrid Systems

One of the most common architectures found in CPS (see Fig. 2) is a discrete software controller, which interacts by some means (e.g. actuators) with a plant and its physical environment (continuous physical phenomenon) in a closed-loop schema. Input from sensors is processed and output is generated and communicated to actuators [12]. Commands from a user or another controller may also be addressed to the controller.

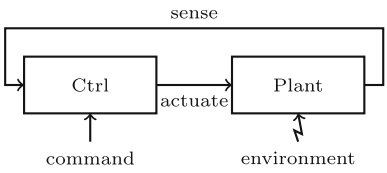


Fig. 2. Generic hybrid system representation

Controllers are characterised by discrete state variables and transitions corresponding to control decisions; as for plants, they are defined by continuous state variables whose evolution is generally described using differential equations. Sensors, user commands control decision and actuators modify these variables.

In this paper, we focus on the verification of the correctness of such discrete controllers, which require correct composition of discrete and continuous models. We claim that correctness should arise from a design process based on sound abstractions and models of the relevant laws of physics.

Hybrid systems may combine the behaviours of multiple separated components (plants or controllers), which can lead to very different control strategies, following the number of controllers and plants to be controlled. Therefore, the generic architecture given in Fig. 2 should be refined into three types of architecture patterns:

- ***Single-to-Single*** AP corresponds to hybrid systems with one controller and one plant. Examples of hybrid systems corresponding to this pattern addressed in the literature include the automatic car braking [15], the *signalised left-turn assist* [13], heating systems [16], etc.
- ***Single-to-Many*** AP describes hybrid systems with one controller and many plants (more than one). This pattern corresponds to centralised control. An example of hybrid system corresponding to this pattern is the control of a global volume distributed over several tanks formalised with hybrid automata and with Event-B in [14].
- ***Many-to-Many*** AP characterises hybrid systems with many controllers and many plants. This pattern refers to the case where several hybrid systems are integrated together to implement a given function. Examples of such systems are UAV or rover fleet control modelled in Event-B [22].

All the patterns defined above refine the one of Fig. 2. The controller and the plant components may be refined to one or many components. These refinements introduce specific properties and behaviours associated to each pattern. The *single-to-single* AP defines one discrete state for the controller and one continuous state for the plant. The *single-to-many* AP defines a controller with one discrete state able to build a global continuous state aggregating the many states of each and every plants.

Finally, the *many-to-many* AP allows to define distributed hybrid systems where each component has a partial view of all the other systems. Here, it is difficult to build a global state of the whole system. Therefore, an approximation of this global state is used by each system controller to take control decisions. Then, the correctness of this approximation shall be ensured to establish global invariants. In other words, local invariants associated to each hybrid system contribute to ensure the global invariant of the whole system composed of all the hybrid systems.

Note: it is worth noticing that the case of *many-to-many* AP may be defined as a set of hybrid systems corresponding to either *single-to-single* AP or *single-to-many* AP. In the last case, *single-to-many* is abstracted by a *single-to-single* system, providing modular verification.

5 Methodology for Hybrid System Design

The pattern presented in Sect. 4 served as a basis for setting up a methodology for hybrid system design. This methodology has been first presented in [15]. It consists of a generic Event-B model that abstracts hybrid systems following the pattern of Fig. 2. This model is then instantiated via refinement. Discrete models can be derived in the same manner [6].

Note that the generic model introduces typically continuous concepts such as differential equations and dense time. It therefore heavily relies on the theory extension of Event-B implemented as a plug-in (see Sect. 3).

5.1 A Generic Event-B Model for Hybrid Systems

VARIABLES t, x_s, x_p
INVARIANTS
inv1: $t \in \mathbb{R}$
inv2: $x_s \in$ STATES
inv3: $x_p \in \mathbb{R}^{\top} \nrightarrow S$
inv4: $[0, t] \subseteq \mathrm{dom}(x_p)$

Model State. The generic model deals with three variables. x_s represents the controller's discrete state that belongs to *STATES* set consisting of the states of the system's mode automaton.

x_p is the system's continuous state. It is a function of time (`inv3`) valued in the (continuous) *state space* S, usually $\mathbb{R}^n$. It represents the physical quantities that are sensed and/or controlled. Last, we recall that variable t models the physical, dense time.

Model Behaviour. The defined model follows the control-command principle depicted on Fig. 2. Two categories of events are defined. Discrete events are instantaneous. They are associated with changes in the state of the mode automaton either internal (*Transition* event) or induced by the sensing of the

plant's state (*Sense* event). Continuous events, on the contrary, are not instantaneous. They describe the Plant's behaviour, either following environmental changes (*behave* event) or caused by actuation (*actuate* event). Note that all these generic events will be refined later for developing particular hybrid systems.

Transition. Transition events (corresponding to *command* arrow and the *Ctrl* box of Fig. 2) model internal changes in the controller. They represent user commands, internal timers or non-deterministic choices that occur in the discrete part of the system (mode automata). It updates the state of the automaton (`act1`).

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 : $s \in \mathbb{P}1(\text{STATES})$
grd2 : $p \in \mathbb{P}(\text{STATES} \times \mathbb{R} \times S)$
grd3 : $(x_s \mapsto t \mapsto x_p(t)) \in p$
THEN
act1 : $x_s :\in s$
END

Sense. Sensing events (corresponding to *sense* arrow of Fig. 2) model changes in the controller induced by the reading of the plant's state, generally obtained from sensors. As they are fired according to the plant's state and to the mode automaton's state, they are guarded by a predicate over $x_p(t)$ and x_s (`grd3`). The purpose is to change state x_s in action `act1` of the mode automaton.

Behave. Behave events (corresponding to the *environment* arrow of Fig. 2) represent changes in the plant due to the environment: rain, wind, etc. These events enforce, in action `act1`, the dynamics of the plant to comply with a differential equation under solvability condition (`gdr2`) but without any condition on the state of the mode automaton.

Behave
ANY eq , t'
WHERE
grd1 : $eq \in DE(S)$
grd2 : **Solvable**$([t, t'], eq, \top)$
THEN
act1 : $t, x_p :\sim_{t \to t'} eq \ \& \ \top$
END

Actuate
ANY eq , s , H , t'
WHERE
grd1 : $eq \in DE(S)$
grd2 : **Solvable**$([t, t'], eq, H)$
grd3 : $s \subseteq \text{STATES}$
grd4 : $x_s \in s$
grd5 : $H \subseteq S$
grd6 : $x_p(t) \in H$
THEN
act1 : $t, x_p :\sim_{t \to t'} eq \ \& \ H$
END

Actuate. Actuation events (corresponding to the *actuate* arrow of Fig. 2) model changes in the plant induced by the controller (generally performed by actuators). These events enforce, in action `act1`, the dynamics of the plant to comply with a differential equation under solvability condition (`gdr2`) and a constraint H on the plant evolution domain (`gdr5` and `gdr6`). Moreover, unlike for *Behave*, since *Actuate* results from a change in the controller, it is guarded by a predicate on the mode automaton (`gdr4`).

As mentioned above, both *Behave* and *Actuate* are continuous events. They rely on the continuous evolution operators defined in Sect. 3. Both events enforce plant behaviour by setting up a corresponding differential equation.

5.2 Semantics

The semantics of hybrid models we use is close to the one of Hybrid Event-B [8], hybrid programs in [21] or continuous action systems [7,20].

In classical Event-B semantics, each model is associated with a discrete state-transition system, in which transitions are the fired machine events and states consist of the machine's variables. A system is hence characterised by a set of licit traces i.e. a set of fired events that abide by the system's invariants.

In our approach, discrete events are timeless, while continuous ones have a duration. In order to properly handle the modelling of continuous behaviours, the semantics of Event-B is enhanced to handle modelling of continuous phenomena which are, in nature, different from discrete behaviours. We have identified two categories of events: discrete (instantaneous) events, which use discrete assignments operators such as :| and := and continuous (not instantaneous) events that span over some duration and use continuous assignment operators, namely $:|_{t \to t'}$ and $:=_{t \to t'}$. Note that, if several (continuous or discrete) events guards are enabled, these enabled events are fired non deterministically.

A model is then defined as follows. After initialisation, continuous events (*Behave* and *Actuate* events) run continuously unless a discrete, instantaneous event is enabled (either a *Sense* or a *Transition* event). In this case, discrete events are preemptive. This protocol ensures that when the conditions (events' guards) are met, the controller is able to trigger control actions (*Sense* or *Transition*) that may or may not change the continuous behaviour of the plant (through triggering an *Actuate* event). Unlike *Actuate*, the *Behave* event neither requires control action to be triggered nor any plant evolution constraint H. Sensing actions using the *Sense* event will re-establish the correct plant behaviour via the control loop in order to further trigger an *Actuate* event.

5.3 The Generic Model in Rodin

The generic model is the entry point for the method. Specific hybrid system models are obtained by refining it, providing the various witnesses issued from event parameters and substituted variables. In itself, this model generates 13 proof obligations that are easily discharged. Among them there is an important obligation stating that if equation e is solvable then $x :\sim_{t \to t'} e$ is feasible.

This approach has been successfully applied to various case studies. [13,15] show a class of systems with one controller and one plant while [14] demonstrates the possible use of the method for a system with one controller and several plants.

Models for the generic approach, including the above-mentioned case studies can be found at https://www.irit.fr/~Guillaume.Dupont/models.php.

6 Case Study: The Water Tank Problem

We now illustrate our approach for formally verifying hybrid systems patterns with a well-known control theory problem: keeping the volume of liquid inside a tank between specific bounds proposed by [5].

6.1 Abstract System

The problem is depicted on Fig. 3 and can be described as follows: one or more tanks are filled with a liquid and connected to an input and an output pump.

A controller can access the global volume V of all tanks and may control their pumps to start filling or emptying them. The goal of the controller is to keep the whole volume between V_{low} and V_{high}.

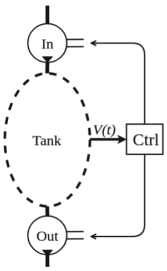


Fig. 3. Abstract tank

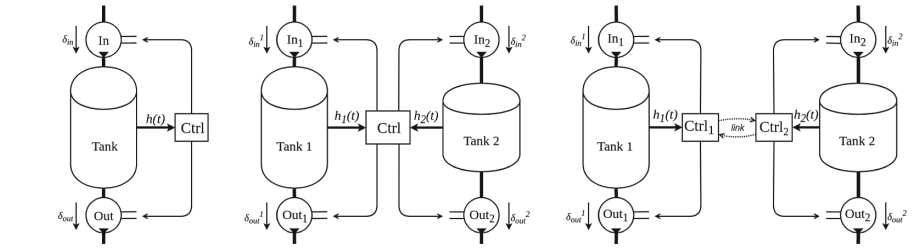
The following safety requirements are defined. Let V, be the volume of the tanks (continuous state being controlled). V is physically bounded by 0 and V_{max}, such that $V_{high} \leq V_{max}$, and it shall satisfy the following properties:

SAF1 The volume never overflows nor underflows: $V(t) \in [V_{low}, V_{high}]$
SAF2 The variation of the volume is bounded (to avoid excessive turmoil in the tank): $|\dot{V}(t)| < \Delta V_{max}$

At this level, it is not needed to know the specific characteristics of the tanks (i.e. their shapes, their number, the behaviour of the pumps, the way the controller accesses V and so on). They are simply abstracted away so as to keep this description as generic as possible. The system is later refined for specific tanks and using specific architecture patterns.

6.2 Architecture Patterns as Abstract System Refinements

The system formerly introduced can be refined to illustrate the three architecture patterns identified in Sect. 4 and depicted on Fig. 4.



(a) Single-to-Single (b) Single-to-Many (c) Many-to-Many

Fig. 4. Three refinement patterns for the case study

Single-to-Single Architecture Pattern. Within a refinement, the abstract model of Fig. 3 is instantiated by a concrete system composed of one controller managing one cylinder-shaped tank (see Fig. 4a). The abstract plant's volume is refined using the gluing invariant $V = B \cdot h$, where B is the surface of the cylinder's base and h is the height of liquid in the tank (easier to measure than the direct volume). Constraints on h are strengthened by the well-definedness condition $V_{max} \leq B \cdot H_{max}$, ensuring that the cylinder can contain (at least) volume V_{max}.

As a matter of simplification, the pumps are associated with a fixed flow rate and are either open (full flow) or closed (no flow), with no intermediate state. Therefore, a differential equation for the system is $\dot{h} = in \cdot \delta_{in} + out \cdot \delta_{out}$, where *in*, *out* are the states of the pumps and δ_{in}, δ_{out} are their respective flows.

This pattern has been previously instantiated in [13,15].

Single-to-Many Architecture Pattern. The same case study can be used to illustrate the second architecture pattern, which involves a single controller and many plants. In this case, we assume that the controller has a global view of the system. In other words, it *knows* all the plants' continuous state variables.

Figure 4b depicts a simplified case for two cylinder tanks, but it scales to any number of tanks of various shapes provided the differential equations that governs these plants are known. For two cylindrical tanks, the gluing invariant is $V = B_1 \cdot h_1 + B_2 \cdot h_2$ where B_1 and B_2 are the surface of the cylinders bases and h_1 and h_2 are the height of liquid in the tanks. The associated differential equations given as witnesses for instantiation are defined by a linear combination.

However, the interesting property in this instantiation relates to the feasibility of the refinement. Indeed, an additional well-definedness condition, expressed as an invariant, states that $V_{max} \leq B_1 \cdot H_{1,max} + B_2 \cdot H_{2,max}$ as to guarantee that the maximum abstract volume can be contained by the two cylinders representing the concrete plant.

This pattern has been thoroughly instantiated and studied in [14].

Note: All the Event-B models corresponding to the two architecture patterns discussed above are available at https://www.irit.fr/~Guillaume.Dupont/models.php. We did not discuss them in this paper due to space limitations. More details can be found in [14,15].

Section 7 below, focuses only on the Event-B models corresponding to the most complex architecture pattern: *many-to-many*.

7 Application of the Many-to-Many Architecture Pattern

In this section we present the last refinement chain corresponding to the *many-to-many* architecture pattern of Sect. 4. The details of the Event-B models are given below for the case study of the water tank following the instantiation of this specific architecture pattern from Fig. 4c.

Refinement Strategy. The refinement strategy is similar to the one used with the *single-to-single* and *single-to-many* patterns. It consists in instantiating the generic model of Sect. 5. Depending on the number of components (controllers/-plants), the generic parts for controller and plant are refined.

Note that the instantiation of the generic model is achieved by providing witnesses to the parameters of the generic events of the Event-B models, i.e. providing a witness for an existential proof obligation.

Two refinements leading to the final Event-B model are defined. First, an abstract tank model corresponding to the system presented in Sect. 6.1 is built

as an instance of the generic model of Sect. 5. Then, the final instantiated architecture pattern of Fig. 4c is modelled as a refinement of this model, providing witnesses for generic parameters. The two refinements are summarised below in Sects. 7.1 and 7.2.

7.1 Abstract Tank Model

MACHINE WaterTank_base **REFINES** Generic
VARIABLES t, V, x_s
INVARIANTS
- inv1: $V \in \mathbb{R}^+ \nrightarrow S$
- inv2: $[0, t] \subseteq \mathrm{dom}(V)$
- inv2: $V = x_p$
- inv3: $V \in \mathcal{D}^1([0, t], \mathbb{R}) \wedge \forall \tau \cdot \tau \in [0, t] \Rightarrow \left|\dot{V}(\tau)\right| \leq \Delta V_{max}$
- inv4: $\forall \tau \cdot \tau \in [0, t] \Rightarrow V_{low} \leq V(\tau) \leq V_{high}$

Machine State. The controlled variable is the volume V. As mentioned in Sect. 6.1 (SAF1 and SAF2), this quantity shall remain between V_{low} and V_{high} and its derivative ($\dot{V}$) shall be bounded by the ΔV_{max} constant.

The system operates in 4 modes: *Emptying* (volume decreases), *Filling* (volume increases), *Normal* (volume varies in an arbitrary way between V_{low} and V_{high}) and *Stable* (volume does not vary) defining the set *STATES*.

Transition and Sense. When the volume reaches V_{low} (resp. V_{high}) the system moves to *Filling* (resp. *Emptying*) mode. Outside of these restrictions, the system may evolve arbitrarily from one mode to another, via *transition* events. Transition events are guarded by a stricter version of the safety invariant as to prevent the system from deliberately going into an unsafe mode.

ctrl_sense_too_high **REFINES** *Sense*
WHERE
- grd1: $V_{high} \leq V(t)$

WITH s: $s = \{Emptying\}$
- p: $p = STATES \times \mathbb{R}^+ \times \{V^* \mid V_{high} \leq V^*\}$

THEN
- act1: $x_s := Emptying$

END

ctrl_transition_normal **REFINES** *Transition*
WHERE
- grd1: $V_{low} < V(t)$
- grd2: $V(t) < V_{high}$

WITH s: $s = \{Normal\}$
THEN
- act1: $x_s := Normal$

END

ctrl_actuate_pumps **REFINES** *Actuate*
ANY e, ss, t'
WHERE
- grd1: $e \in \mathbf{DE}(S)$
- grd2: $\mathbf{Solvable}([t, t'], e)$
- grd3: $FlowEq(ss, [t, t'], e)$
- grd4: $ss \in STATES$
- grd5: $x_s = ss$
- grd6: $V_{low} < V(t) < V_{high}$

WITH x'_p: $x'_p = V'$
- s: $s = \{ss\}$
- H: $H = \{V^* \mid V_{low} < V^* < V_{high}\}$

THEN
- act1: $V :\sim_{t \to t'} e \;\&\; \{V^* \mid V_{low} < V^* < V_{high}\}$

END

Behave and Actuate. The system performs actuation on the pumps. At this level, the shape of the tank(s) and the behaviour of the pumps are not known yet. The only constraint the actuation shall enforce is that whenever the system is in a specific state, the provided differential equation for actuation is such that its solutions have the expected behaviour (e.g. decreasing solutions when in *Emptying* mode, increasing solutions when in *Filling* mode, etc.).

This constraint is captured by the $FlowEq(x_s, D, e)$ predicate of guard `grd3` and is defined in a theory, where x_s is the controller's state, D is the domain on which the predicated behaviour is expected to be true and e is the given equation.

7.2 Many-to-many Model

The model presented below corresponds to the system depicted on Fig. 4c.

MACHINE WaterTank_2Ctrl_2Tanks **REFINES** WaterTank_base
VARIABLES t, V_1, V_2, V_1^{sim}, V_2^{sim}, x_s^1, x_s^2, Δ_1^{sim}, Δ_2^{sim}
INVARIANTS
inv11: $V_1 \in \mathbb{R}^+ \to S \wedge [0,t] \subseteq \mathrm{dom}(V_1)$
inv12: $V_1^{sim} \in \mathbb{R}^+ \to S \wedge [0,t] \subseteq \mathrm{dom}(V_1^{sim})$
inv13: $x_s^1 \in STATES$
inv14: $\forall \tau \cdot \tau \in \mathbb{R}^+ \Rightarrow V_1(\tau) + V_2^{sim}(\tau) \leq V_{high} - \Delta_2^{sim}$
$\wedge V_{low} + \Delta_2^{sim} \leq V_1(\tau) + V_2^{sim}(\tau)$
inv15: $\Delta_1^{sim} \in \mathbb{R}^+$
inv16: $\forall \tau \cdot \tau \in \mathbb{R}^+ \Rightarrow |V_2(\tau) - V_2^{sim}(\tau)| \leq \Delta_2^{sim}$
inv21−26: *— similar to inv11−16 with* V_2
inv01: $V = V_1 + V_2$
inv02: $x_s = guess\_gs(x_s^1 \mapsto x_s^2)$

Machine State. In this refinement, we want to control two tanks (although this could be extended to any number of tanks). Each tank has its own volume, V_1 and V_2 (see Fig. 4c) behaving as a global invariant. The abstract volume V is hence refined using the gluing invariant $V = V_1 + V_2$, and the safety invariant becomes $V_{low} \leq V_1 + V_2 \leq V_{high}$ (corresponding to $V_{low} \leq B_1 \cdot h_1 + B_2 \cdot h_2 \leq V_{high}$). Each volume V_i is bounded by $V_{i,max}$, and in order to have a coherent refinement we need to have $V_{max} \leq V_{1,max} + V_{2,max}$. The controller discrete abstract state x_s is glued (`inv02`) with the two discrete controllers states using the *guess_gs* operator.

Each tank is controlled by an independent controller. In a many-to-many pattern, a controller does not know exactly the state of the other controllers (i.e. what the other controllers are doing). However, the physics asserts that an estimation of this other state, and as such of the global state, can be built. To model this situation, two additional continuous variables, V_1^{sim} (resp. V_2^{sim}) are introduced. They allow the controller 2 (resp. 1) to simulate (i.e. estimate) V_1 (resp. V_2).

Because it is an estimation, V_i^{sim} is associated to a bound, Δ_i^{sim}, that represents the maximum error allowed for the controller to ensure a correct behaviour. We then need to have, at any time and for all i: $|V_i - V_i^{sim}| \leq \Delta_i^{sim}$, i.e.: V_i^{sim} is a *precise enough* approximation of V_i. Again, these properties are borrowed from the physics.

Transition and Sense. Controller 1 needs to enforce the (local) invariant $V_{low} + \Delta_2^{sim} \leq V_1 + V_2^{sim} \leq V_{high} - \Delta_2^{sim}$, and similarly for controller 2. This enforcement is used to prove the initially defined global invariant.

ctrl_sense_too_low_1 REFINES *ctrl_sense_too_low*
WHERE
grd1: $V_1(t) + V_2^{sim}(t) \leq V_{low} + \Delta_2^{sim}$
THEN
act1: $x_s^1 := Filling$
END

Behave and Actuate. The system's actuation is established using continuous refinement as presented in [14]: the witness for e (abstract differential equation) is a predicate that links the solutions of e_1 and e_2 (concrete differential equations) such that the sum $V_1^* + V_2^*$ of any pair of solutions (V_1^*, V_2^*) of (e_1, e_2) is a solution of e, in addition to having the relevant general constraints (namely, a correct behaviour as per the system's current state). The witness for V' is given to establish the invariant after actuation.

ctrl_actuate_pumps REFINES *ctrl_actuate_pumps*
ANY ss, e_1, e_2, ss_1, ss_2, V_1^{sim*}, V_2^{sim*}, t'
WHERE
grd01−02: $ss \in STATES \land ss = guess\_gs(ss_1 \mapsto ss_2)$
grd11−13: $e_1 \in \mathbf{DE}(S) \land \mathbf{Solvable}([t, t'], e_1) \land FlowEq(ss_1, [t, t'], e_1)$
grd14−15: $ss_1 \in STATES \land x_s^1 = ss_1$
grd16: $V_1^{sim*} \in \mathbb{R}^+ \nrightarrow S \land [t, t'] \subseteq \mathrm{dom}(V_1^{sim*})$
grd17: $\forall V_1^* \cdot V_1^* \in \mathbb{R}^+ \nrightarrow S \land [t, t'] \subseteq \mathrm{dom}(V_1^*) \land$
$\mathbf{solutionOf}([t, t'], V_1^*, e_1) \Rightarrow (\forall t^* \cdot t^* \in [t, t'] \Rightarrow |V_1^*(t) - V_1^{sim*}(t)| \leq \Delta_1^{sim})$
grd21−27: *−− similar to grd11−17 with* V_2
grd30: $V_{low} < V_1(t) + V_2^{sim}(t) < V_{high}$
grd31: $V_{low} < V_1^{sim}(t) + V_2(t) < V_{high}$
WITH
V': $V' = V_1' + V_2'$
e : $e \in \mathbf{DE}(S) \land \mathbf{Solvable}([t, t'], e) \land FlowEq(guess\_gs(ss_1 \mapsto ss_2), [t, t'], e) \land$
$(\forall V_1^*, V_2^* \cdot V_1^* \in \mathbb{R}^+ \nrightarrow S \land [t, t'] \subseteq \mathrm{dom}(V_1^*) \land V_2^* \in \mathbb{R}^+ \nrightarrow S \land [t, t'] \subseteq \mathrm{dom}(V_2^*) \land$
$\mathbf{solutionOf}([t, t'], V_1^*, e_1) \land \mathbf{solutionOf}([t, t'], V_2^*, e_2)$
$\Rightarrow \mathbf{solutionOf}([t, t'], V_1^* + V_2^*, e))$
THEN
act1: $V_1, V_2, V_1^{sim}, V_2^{sim} :|_{t \to t'}$
$\mathbf{solutionOf}([t, t'], V_1', e_1) \land \mathbf{solutionOf}([t, t'], V_2', e_2) \land$
$V_1^{sim\prime} = V_1^{sim*} \land V_2^{sim\prime} = V_2^{sim*}$
$\&\{V_{low} < V_1(t) + V_2^{sim}(t) < V_{high} \land V_{low} < V_1^{sim}(t) + V_2(t) < V_{high}\}$
END

8 Assessment

The work presented in this paper showed that the generic model proposed in [15] applies to different architecture patterns of hybrid systems. Below, we provide an assessment of the approach with respect to the proof effort and set up methodology. The models presented in this paper have been developed on the Rodin platform and all the generated proof obligations were discharged.

Complete models can be found at https://irit.fr/~Guillaume.Dupont/models.php.

Proof Effort. The abstract tank model generated 107 proof obligations, most of which are invariant (about 40%) or well-definedness (about 21%) related. Well-definedness also appears often in proofs subgoals. These POs are usually easy to prove, at least on paper. Feasibility POs, related to solution existence, are those difficult to prove.

As for the *many-to-many* model, it yields 156 proof obligations, among which a good proportion (53%) consists of invariant POs alone. Again, most of them are not hard to discharge. The model also yields quite a few guard strengthening POs (around 15%) that ensure that the controllers behave properly despite the estimation it makes of the system. But the hardest POs to discharge are the one regarding refinement (witness well-definedness and feasibility, and simulation).

A great interest of the proposed methodology is there: the only complex proofs to carry on are related to *refinement*. Proofs for complicated invariants and so on have been realised at the *abstract* level and are done once and for all.

Tool Support. Because of our heavy use of the theory plug-in in Rodin, proof automation (including SMTs and external provers) is nearly nonexistent for discharging the generated POs. Proof is thus mostly interactive, and even simple steps such as basic well-definedness are to be done fully manually using the interactive prover. That being said, the possibility to define rewrite and inference rules greatly improves the prover's overall ergonomy.

Methodology. The use of patterns as methodological basis is not new in system engineering. The availability of architecture patterns offers a methodological guide to system designers, who simply need to identify which pattern matches the hybrid system under design and instantiate it with refinements and witnesses.

The generic model offers a framework that is formally proven once and for all. It corresponds to a *customisation* of Event-B to offer resources for modelling controllers, plants, sensing and actuation, integrating both discrete and continuous behaviours. Proofs are done once for all and the designer does not need to re-prove them. This generic model is used as a ground model for further designs.

Each defined architecture pattern is formalised as an instance of the generic model. The pattern to be chosen for instantiation depends on the number of controllers and plants required in the model. Instantiation is performed using Event-B refinement.

One of the interests of the Event-B method is the capability to check well-definedness and feasibility conditions, which is particularly useful during instantiation. In our developments, it has been extensively used to provide conditions about the soundness of the defined instantiations. For example, it has been used to state that the cylinders given as refinement are capable of storing an abstractly specified volume of liquid V_{max}.

9 Conclusion and Future Work

This paper presented a framework for modelling hybrid systems. It relies on a formal model of different hybrid systems architecture patterns formalised with the Event-B method using the Rodin platform. These patterns, commonly used when designing hybrid systems, are characterised by the number of controlled plants and by the kind of control strategy (centralised or distributed). Because this framework is formalised at a generic level, it offers a systematic methodology for hybrid systems development and verification.

The approach extensively uses the mathematical extensions capabilities offered by the theory plug-in of Event-B, allowing to enrich Event-B models with continuous behaviours. Data types for reals, continuous functions, differential equations and so on have been defined within a sound Event-B theory. The available axioms and theorems were used to prove the relevant safety properties of the developed systems expressed as machine invariants. The developed models are scalable (modulo proof efforts), as they can deal an arbitrary number of state variables. Witnesses for the sets $STATES$ and S are provided at instantiation using gluing invariants.

This work revealed several research perspectives. Below, we summarise the identified future research actions.

Need for Other Domain Theories. Although the definition of generic architecture patterns has reduced the number and the complexity of proof obligations and their proofs, the proof effort still needs to be reduced. Providing other sound domain theories contributes to such a reduction. One of the main extensions to our work consists in enriching the proposed framework with other theories. Two kinds of theories are expected: theories for other types of control and theories

where the physics of considered plants is formalised. A library of such theories would help for such hybrid systems developments by making explicit knowledge in physics and in other related domains [3].

Methodology. From the method formalisation point of view, the major improvement is to leverage the formalisation of architecture patterns at a higher abstraction level to handle controllers and plants as first class mathematical objects.

Other patterns where the number of hybrid systems evolves dynamically could be considered. In this case, each system would have a partial knowledge of its environment. This kind of patterns may help to model autonomous aspects. However, defining safety properties remains a major challenge for such patterns.

Integration of Simulation Tools. To handle the traditional hybrid systems development processes where simulation is extensively used, coupling the developed models with simulation tools, like in [1], would help in animating these models.

Acknowledgment. We thank T. S. Hoang for his help with Rodin's Theory plug-in and R. Banach for the helpful discussions related to Event-B hybridation.

References

1. Project INTO-CPS: Integrated Tool Chain for Model-based Design of Cyber-Physical Systems. http://into-cps.au.dk/about-into-cps
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
3. Aït-Ameur, Y., Méry, D.: Making explicit domain knowledge in formal system development. Sci. Comput. Program. **121**(C), 100–127 (2016)
4. 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
5. Alur, R., Henzinger, T.A.: Modularity for timed and hybrid systems. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 74–88. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63141-0_6
6. Babin, G., Aït-Ameur, Y., Singh, N.K., Pantel, M.: A system substitution mechanism for hybrid systems in Event-B. In: Ogata, K., Lawford, M., Liu, S. (eds.) ICFEM 2016. LNCS, vol. 10009, pp. 106–121. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47846-3_8
7. Back, R.J., Petre, L., Porres, I.: Continuous action systems as a model for hybrid systems. Nord. J. Comput. **8**(1), 2–21 (2001)
8. 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)
9. Boldo, S., Clément, F., Filliâtre, J.C., Mayero, M., Melquiond, G., Weis, P.: Trusting computations: a mechanized proof from partial differential equations to actual program. Comput. Math. Appl. **68**(3), 325–352 (2014)
10. Butler, M., Abrial, J.R., Banach, R.: Modelling and refining hybrid systems in Event-B and Rodin. In: From Action Systems to Distributed Systems: The Refinement Approach. Computer and Information Science Series, pp. 29–42. Chapman and Hall/CRC (2016)

11. Butler, M., Maamria, I.: Practical theory extension in Event-B. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods. LNCS, vol. 8051, pp. 67–81. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39698-4_5
12. Cardenas, A.A., Amin, S., Sastry, S.: Secure control: towards survivable cyber-physical systems. System **1**(a2), a3 (2008)
13. 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., 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
14. 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)
15. 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
16. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings of 11th Annual IEEE Symposium on Logic in Computer Science, LICS, pp. 278–292. IEEE Computer Society (1996)
17. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Upper Saddle River (1985)
18. Jifeng, H.: From CSP to hybrid systems. In: Roscoe, A.W. (ed.) A Classical Mind, pp. 171–189. Prentice Hall International (UK) Ltd., Upper Saddle River (1994)
19. Lee, E.A., Seshia, S.A.: Introduction to Embedded Systems - A Cyber-Physical Systems Approach, 1.5 edn. LeeSeshia.org (2014). http://leeseshia.org/
20. Meinicke, L., Hayes, I.J.: Continuous action system refinement. In: Uustalu, T. (ed.) MPC 2006. LNCS, vol. 4014, pp. 316–337. Springer, Heidelberg (2006). https://doi.org/10.1007/11783596_19
21. Logical Foundations of Cyber-Physical Systems. Lecture Notes in Computer Science. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-63588-0_21
22. Singh, N.K., Aït-Ameur, Y., Pantel, M., Dieumegard, A., Jenn, E.: Stepwise formal modeling and verification of self-adaptive systems with Event-B. The automatic rover protection case study. In: 21st International Conference on Engineering of Complex Computer Systems, ICECCS 2016, pp. 43–52 (2016)
23. Su, W., Abrial, J.R., Zhu, H.: Formalizing hybrid systems with Event-B and the Rodin platform. Sci. Comput. Program. **94**(Part 2), 164–202 (2014). abstract State Machines, Alloy, B, VDM, and Z