Handling Refinement of Continuous Behaviors: A Refinement and Proof Based Approach with Event-B

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

2019 International Symposium on Theoretical Aspects of Software Engineering (TASE)

Handling Refinement of Continuous Behaviors: A Proof Based Approach with Event-B

G. Dupont*, Y. Aït-Ameur*, M. Pantel* and N. K. Singh*
*INPT-ENSEEIHT/IRIT, University of Toulouse, France
{guillaume.dupont,yamine,marc.pantel,nsingh}@enseeiht.fr

***Abstract*—Cyber-physical systems (CPS) are taking a crucial role in various areas of our society and industry. Yet, because of their hybrid nature (i.e. the integration of both continuous and discrete features), their design and verification are not easy to handle, in particular when they are part of a critical system. Their certification requires to exhibit a formal argumentation that formal methods should be able to provide.**

This paper addresses the formal development of CPS using correct-by-construction refinement and proof based approaches. It relies on the Event-B formal method. In addition to modeling both the discrete and continuous parts of a CPS, this paper presents a novel approach in two steps.

First it shows that the generic formal model we have defined, integrating both discrete and continuous behaviors, can be instantiated by various kinds of CPS. Fundamentally, continuous behaviors modeled by differential equations mingle with discrete transition systems (mode automaton), which model discrete behaviors. Here, refinement is used as a decomposition mechanism.

Second, it expands the refinement operation, well mastered in the discrete world, to cover continuous behaviors. We show that different levels of abstraction of continuous aspects can be glued in a refinement chain. The proposed approach has been completely formalized using Event-B on the Rodin platform and a case study based on water tanks is used to illustrate it.

***Index Terms*—hybrid systems, Event-B, proof, refinement**

I. Introduction

The increasing preponderance of computers and automation in our modern society has led to a new category of systems called *Cyber-Physical Systems* or CPS. According to E. Lee [1], CPS refer to the tight integration and coordination between computational and physical resources. Conceptually, a CPS consists of a software component, the controller, which interacts with a physical environment or *plant* [1]–[4]. The software periodically takes into account information on the plant state and its environment provided by *sensors*, and control the plant behavior *via* orders sent through *actuators*.

CPS are rapidly gaining a foothold in our everyday life: from Internet of Things (IoT) to smart energy grids, from autonomous transportation systems to medical assistants, it appears that CPS are becoming omnipresent.

The close entanglement of the discrete and continuous worlds poses important challenges in CPS design and certification, especially in safety critical systems. A rigorous and formal modeling framework, based on mathematics and logic, contribute to take on these challenges.

Context. If the discrete part of a CPS can be formalized using classical modeling techniques (transition systems, process algebra, etc.) well studied in the literature, its continuous part requires other kinds of modeling. In particular, continuous behaviors are described using *differential equations*. Thus, the formal definition of CPS relies substantially on the formal definition of these equations, their various natures (linear, ordinary, first order, etc.) and most of all their properties (existence of solution, smoothness, boundedness, stability, etc.). Like for the software part, formal models have been proposed for the continuous part. The literature in control theory is rich and provides solid mathematical solutions.

Defining formal methods supporting modeling and offering reasoning capabilities for both discrete and continuous worlds is the key issue for rigorous CPS design. It represents the research question our work is addressing. The literature is full of frameworks supporting the development of CPS. HyTech [5] and dReach [6] tackle the problem using model-checking while Ptolemy [7] uses simulation and modularity to design so-called *reactive systems*. Proof and refinement based approaches also constitute a promising path with KeYmaera [8], [9], Hybrid Event-B [10] or Coquelicot [11].

Our approach relies on proof and refinement based formal methods. We have shown in [12], [13] that Event-B [14] and its Integrated Development Environment (IDE) Rodin [15] support the design and proof of hybrid systems without requiring any extension. We devised a generic model that has been successfully deployed, using refinement, on various case studies borrowed from the literature [16], [17].

Objective of the paper. However, our approach [12], [13] mainly focuses on the design, by refinement, of the software part i.e. the *controller* assuming the plant is well and thoroughly defined. *Our objective is to pay the same attention to the design of the plant by providing a refinement operation in the continuous part, i.e the* plant. Therefore, in this paper, we extend our approach to the design of the *plant*. We show that a plant model, expressed in Event-B, can be produced by refinement in parallel with the development of the controller. A continuous refinement relation is defined, allowing one to enrich the continuous behavior of a CPS while preserving invariants and safety properties expressed at the abstract level.

Organization of the paper. This paper is organized as follows. Section II motivates our work and Section III presents the Event-B modeling framework and the required ingredients for modeling hybrid systems. The basic concepts of the hybrid system modeling in Event-B we have set up are summarized in Section IV. A refinement strategy for abstract system plants is

DOI 10.1109/TASE.2019.00009

proposed in Section V. Section VI describes the development of the selected case study and Section VIII provides an assessment and discusses generic modeling concepts and proof details. Section IX presents related work. Finally, Section X concludes the paper and discusses a future research agenda.

II. Motivation

Refinement of discrete models is a well-established concept. It has been extensively studied and the literature is full of seminal studies on refinement [18]–[22]. It addressed both centralized and distributed systems.

This notion is at the heart of the Event-B method. It allows developers to gradually design a system by adding either static (data refinement) or behavioral (events refinement by simulation) details while moving from an abstract to a concrete model. In addition to the preservation of the invariants defined at the abstract level by the refinement relation, Event-B supports the definition of new invariants in the refined model.

The notion of refinement has been extended to continuous behaviors. Simulation relations and temporal logics over reals have been defined in [23]–[25]. However, compared to discrete models, few modeling frameworks allow to reason on the refinement of continuous behaviors. One can mention [26]–[29].

Similarly to the refinement relation defined for discrete models, i.e. models with discrete state transitions (discrete time jumps), we claim that 1) it is possible to define a refinement relation for continuous models (models with continuous state transitions and continuous time jumps) for Event-B without requiring any extension, and 2) that it can be used for designing a complex system by a sequence of continuous refinements of an abstract plant leading to a concrete plant.

Moreover, we believe that integrating, **in a single framework**, the refinement relation inherent to discrete models and its continuous counterpart greatly enhances the use of refinement-based approaches for the development of hybrid systems and helps the formal verification of controllers for such systems.

In the remaining part of this paper, we show that the approach we developed in [12], makes it possible to define a refinement relation to refine an abstract continuous model by another one. Such a refinement is formalized in Event-B.

III. Event-B: A Refinement and proof based Approach

Event-B [14] is a *correct-by-construction* formal method to design complex systems. The design process consists of a series of refinement of an abstract model (specification) leading to the final concrete model. Refinement progressively adds design decisions to the system.

A. Overview

a) Modeling: Event-B Machines.: The Event-B language uses set theory and first order logic as basic notations. Two main components are defined to build system models: *context* and *machine* (See Table I). A *context* is the static part of a model. It sets up all the definitions, axioms and theorems needed to describe the manipulated concepts. *Carrier sets* s, *constants* c, *axioms* $A(s,c)$ and *theorems* $T_c(s,c)$ are introduced.

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* v, *invariants* $I(s,c,v)$, *theorems* $T_m(s,c,v)$, *variants* $V(s,c,v)$ and *events* evt (possibly guarded G) are defined in a machine.

Invariants and *theorems* formalize system properties to check the correctness of the formalized behavior while *variants* define convergence properties (reachability).

CONTEXT	**MACHINE**
ctxt_id_2	*machine_id_2*
EXTENDS	**REFINES**
ctxt_id_1	*machine_id_1*
SETS	**SEES**
s	*ctxt_id_2*
CONSTANTS	**VARIABLES**
c	v
AXIOMS	**INVARIANTS**
$A(s,c)$	$I(s,c,v)$
THEOREMS	**THEOREMS**
$T_c(s,c)$	$T_m(s,c,v)$
END	**VARIANT**
	$V(s,c,v)$
	EVENTS
	Event evt
	any x
	where $G(s,c,v,x)$
	then
	$v :\mid BA(s,c,v,x,v')$
	end
	END

Table I – Model structure

b) Refinement of Event-B models.: Refinement decomposes a model (thus a sate-transitions system) into another transition system with more design decisions while moving from an abstract level to a less abstract one. A system is gradually designed (modeled) by introducing properties (functionality, safety, reachability) at various refinement levels. Event guards may be strengthened and new variables and events may be introduced in order to model more concrete behavior of the system. Refinements preserve the relation between the refining model and the refined one. Abstract and concrete state variables are linked by introducing *gluing invariants*.

c) Proof Obligations (PO) and Property Verification : To establish the correctness of an Event-B model (machine or refinement) the POs (automatically generated from the calculus of substitutions) need to be proved.

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

Table II – Proof obligations

The main POs are listed in Table II. The prime notation, allowing to define Before-After Predicates (BAP), denotes the value of a variable after an event is triggered. POs of Table II require to demonstrate that theorems hold (1), each event preserves invariants (Induction (2)), each event can be triggered (feasibility (3)) and variant decreasing (convergence (4)).

Regarding refinement, two more relevant POs need to be discharged. First, the simulation PO to assert that the refined (concrete) event simulates the corresponding abstract event. Second, the refined events may strengthen the abstract guards

to specify concrete conditions. More details on proof obligations can be found in [14].

A proof system, with a set of proof rules on the Event-B concepts, is associated with Event-B.

The Rodin platform [15] is an open source Eclipse-based tool for engineering Event-B models. It offers resources for model edition, project management, refinement and proof, model checking, model animation and code generation. Several provers, like SMT solvers, are plugged to Rodin.

B. A Mathematical Extension for Event-B

Event-B is based on first order logic and basic set theory. In addition to that, it "natively" supports integer numbers and basic arithmetic. However, the handling of hybrid systems and thus continuous behaviors requires an access to various mathematical objects and properties, not natively available in Event-B.

An Event-B extension to support externally defined mathematical theories has been proposed in [30]. It offers the capability to introduce new datatypes through the definition of new types, sets operators, theorems and even new rewrite and inference rules all packed in so-called *theories*. This extension has been developed as a plug-in Rodin [31].

C. A domain theory for hybrid systems in Event-B

To model hybrid systems, the mathematical concepts related to the description of continuous behaviors that are not available in Event-B are formalized in a theory. We defined several operators and sets that allows us to make use of core concepts in control theory such as (ordinary) differential equations, solvability and so on. A theory for differential equations has hence been defined (see Figure 1).

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

Fig. 1 – Differential Equation Theory Snippet

In particular, we have defined:

- **DE**(S) the set (type) of **D**ifferential **E**quations with solutions in S (*space set* of the differential equation).
- **solutionOf**(D,η,eq) a predicate stating that the function $\eta \in D \rightarrow S$ is a solution of $eq \in \mathbf{DE}(S)$.
- **Solvable**(D,eq) a predicate stating that equation $eq \in \mathbf{DE}(S)$ has a solution on domain D.
- **ode**(F,η_0,t_0) defines the ODE[1] $\dot{\eta}(t) = F(t, \eta(t))$ with initial condition $\eta(t_0) = \eta_0$.

Other expressions and predicates are defined ($FlowEquation$, $FlowODE$) in this theory. It should be noted that these definitions are **algebraic** and based on datatypes.

[1]Ordinary Differential Equation

IV. Modeling Hybrid Systems with Event-B: Our Approach

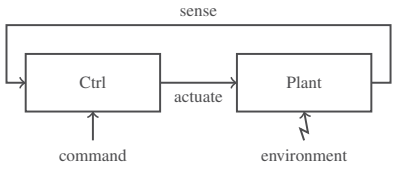

Fig. 2 – Generic Hybrid System Representation

One of the most common architectures found in CPSs (see Figure 2) is a discrete software controller which interacts 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 [32]. Commands from a user or another controller may also be addressed to the controller.

We are concerned with the verification of the correctness of such discrete controllers, which requires correct integration of discrete and continuous models. Correctness should arise from a design process based on sound abstractions and models of the relevant physical laws.

A. An Approach to Hybrid System Modeling in Refinement-based Methods

In [12], we presented a generic approach to hybrid system modeling. This approach is inspired by the work of R. Back on continuous action systems [26], Platzer in [9] and R. Banach in [10], where both continuous and discrete behaviors are integrated in a single model definition.

However, instead of defining a new domain specific modeling for hybrid systems requiring the development of a new specific framework, *our approach relies on the use of the core Event-B method and the theory plug-in* to model domain-specific features and manipulate concepts relative to continuous behaviors modeling. Thus, Event-B machines integrate both discrete and continuous events.

Our approach relies on the definition of a generic model, formalizing the architecture pattern of Figure 2. It is modeled as a generic machine that ought to be instantiated for specific hybrid systems thanks to refinement.

B. The generic Event-B model

A machine `System` with 4 types of events is defined.

```
MACHINE System
VARIABLES t, x_s, x_p
INVARIANTS
  inv1: t ∈ ℝ⁺
  inv2: x_s ∈ STATES
  inv3: x_p ∈ ℝ⁺ → S
EVENTS
INITIALISATION
THEN
  act1: t := 0
  act2: x_s :∈ STATES
  act3: x_p :∈ ℝ⁺ → S
END

Progress
THEN
  act1: t :| t' ∈ ℝ⁺ ∧ t < t'
END
```

a) Controller/plant state: Time (t initially 0) is a read-only, ever-progressing (*Progress* event) variable. The system defines 2 state variables: a continuous (plant) state x_p, function of time, and a discrete (controller) state, x_s valued in the abstract set `STATES`. They are initialized with a non-deterministic assignment operator $:\in$, refined later at instantiation.

b) Transition Events: They symbolize changes in the controller induced by its mode automaton. Namely, they model internal decisions or user commands.

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

c) Sense Events: They model changes in the controller induced by changes in the plant. They are triggered by the reading (sensing) of values from a sensor.

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

Both *Transition* and *Sense* events are *instantaneous*.

Behave
ANY eq
WHERE
grd1: $eq \in DE(S)$
grd2: $\text{Solvable}([t, +\infty[\,, eq)$
THEN
act1:
$x_p :| x'_p \in \mathbb{R}^+ \to S \wedge$
$[0, t[\lhd x'_p = [0, t[\lhd x_p \wedge$
$\text{solutionOf}([t, +\infty[\,, [t, +\infty[\lhd x'_p, eq)$
END

Actuate
ANY eq, s
WHERE
grd1: $eq \in DE(S)$
grd2: $\text{Solvable}([t, +\infty[\,, eq)$
grd3: $s \subseteq \text{STATES}$
grd4: $x_s \in s$
THEN
act1:
$x_p :| x'_p \in \mathbb{R}^+ \to S \wedge$
$[0, t[\lhd x'_p = [0, t[\lhd x_p \wedge$
$\text{solutionOf}([t, +\infty[\,, [t, +\infty[\lhd x'_p, eq)$
END

d) Behave Events: They express the possible spurious changes or *perturbations* in the plant itself. Those changes are a result of the effects of the environment on the plant: wind, rain, temperature, etc.

e) Actuate Events: They represent the action of the controller on the plant, generally via actuators.

These two types of event are not instantaneous: they last some time. They also enforce the fundamental property of *past preservation* for continuous behaviors: at time t, the past of the continuous state ($[0, t[\lhd x'_p$) remains unchanged, while its future ($[t, +\infty[\lhd x'_p$) is modified, becoming the solution of the given differential equation eq. This particular construct is a *specific assignment operator in the continuous world*.

f) Instantiation by Refinement: Concrete hybrid systems designs are synthesized via the instantiation of this generic model through refinement. The set STATES is valued to contain the various states of the controller (states of a mode automaton), S is set to the space where the system evolves ($\mathbb{R}^n, n \in \mathbb{N}$). Last, witnesses are given for x_p and eq (differential equation modeling the plant). For example, the plant may be modelled by an ordinary differential equation and x_p is valued in a real vector space (typically, $\mathbb{R}^n$).

C. About semantics

Discrete events are timeless while continuous ones have a duration. After initialization, continuous events run continuously unless a discrete *instantaneous (timeless)* event is enabled. In this case, that discrete event is preemptive. This protocol asserts that when the conditions are met, the controller is able to trigger control actions.

V. Refining Continuous Systems

The objective of defining a refinement relation on the continuous part of a CPS model is to ease the design of physical behavior by introducing design decisions progressively. The verification of the correctness of the development is disseminated along the refinement chain. To set up such a relation in Event-B, we need to address 1) data refinement for continuous state variables, 2) invariants and gluing invariants linking abstract and continuous state variables and 3) continuous event refinement.

a) Data refinement: The idea behind refinement of continuous state variables in Event-B is similar to data refinement for discrete state variables. The approach consists of redefining, in a refined machine, the abstract state variables of the abstract machine into concrete state variables. This refinement shall produce new functions over time for the continuous state. For example, an abstract tank volume $V(t)$ may be refined with a specific formula corresponding to a concrete tank shape (e.g. $\pi R^2 \cdot h(t)$ for a cylinder of radius R and height h).

b) Gluing, local and global invariants: Refinement shall preserve the behavior and correctness of the abstract model. Gluing invariants are defined for this purpose. They link abstract and concrete state variables. In our tank example, a safety envelope (*safety requirement*) for the volume $V(t)$ of the plant such as $V(t) \in [V_{low}..V_{high}]$ may be defined. Invariants may also be defined on derivatives (e.g. the variation of the volume shall be smooth and thus bounded $\left|\dot{V}(t)\right| < \Delta V_{max}$). Such invariants are called **global invariants**, at the plant level. When this abstract plant is refined by the cylinder (introduced above), then the invariant $V(t) = \pi R^2 \cdot h(t) \wedge h(t) \in [0..H_{max}]$, where H_{max} is the maximum height of the cylinder, defines a **gluing invariant** for the continuous part of the hybrid system.

Additional invariants may be introduced to assert feasibility properties on the continuous part; in other words, to ensure that the concrete plants are correct realizations of the abstract one. Again, for our example, the cylinder (concrete plant) shall satisfy $\pi R^2 \cdot H_{max} \geq V_{max}$. Other invariants, not expressible at the abstract level, may be added. These invariants are examples of the notion of **local invariants**.

c) Event refinement: The refinement of continuous events is close to the one of discrete events. It is based on a simulation relationship defined on states continuously observed (continuous simulation as defined in [23]–[25]). One may consider that an abstract plant is refined by several plants (e.g. a set of cylinders) whose behaviors are described by one or more continuous events. This situation is depicted by the case study defined in next section.

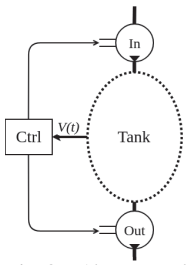

Fig. 3 – Abstract Tank

VI. Case Study

We study the control of a liquid tank of arbitrary size and shape connected to two pumps, one to fill the tank with liquid and the other one drain the tank (see Figure 3). The content of the tank is physically bounded by its capacity and is between 0 and V_{max}.

A controller actuates the input and output pumps and senses the liquid vol-

ume level of the tank through a sensor. Additional requirements are introduced. First, the volume of liquid in the tank shall fluctuate between V_{low} and V_{high}. Indeed, when

the volume goes below V_{low}, the controller shall activate the input pump to fill it up. Conversely, if the volume goes beyond V_{high} the controller shall activate the output pump to drain it down. Second, because of a possible fragility of the tank, the volume variation in time shall be bounded to avoid excessive turmoil.

Here, continuous refinement has multiple interests:

- the behavior of the pumps is not yet specified. They could be simple open-close systems or more subtle continuous systems;
- the shape of the tank (or even *tanks*) nor the way we get the volume (which is hardly ever given as is) are not yet specified as well.

Next section shows how continuous refinement is set up on the defined case study.

VII. Continuous Refinement: Application to the Case Study

To position the case study in the context of our generic approach [12], an overview of our development is given by describing the controller and the continuous parts.

A. Overview of the Development

The controller operates in 4 modes

- **Stable:** no pump is active, the volume inside the tank does not change.
- **Normal:** no particular control on the pumps, the volume inside the tank changes in an uncontrolled way.
- **Emptying:** pumps are controlled so that the volume in the tank decreases.
- **Filling:** pumps are controlled so that the volume in the tank increases.

Every state associated with each mode is reachable from any other state through a *Transition* event. Additionally, the controller automatically goes into *Emptying* mode when the tank's volume reaches V_{high} and into *Filling* mode when it reaches V_{low}. There are thus two *Sense* events able to detect when the volume is too high or too low.

Volume $V(t)$, is the continuous state being controlled. It is physically bounded between 0 and V_{max} (depending on the size of the tank) and shall behave such that, for every t,

- it remains within the required bounds:

$$V(t) \in [V_{low}, V_{high}] \qquad \text{(SAF1)}$$

- its variation is bounded by ΔV_{max}:

$$|\dot{V}(t)| < \Delta V_{max} \qquad \text{(SAF2)}$$

B. Refinement Strategy

A first model of the abstract tank instantiating the generic approach is produced as a first refinement. It mainly defines the controller events (transition and sensing) as well as the invariants on the volume V. A second refinement, a continuous one, precises which type of differential equation is used to model the volume's behavior. Finally, a third (continuous) refinement concretizes the abstract tank as two cylindrical tanks with *height* as controlled (continuous state) variable.

C. First Refinement: Abstract Tank Model

The abstract tank system instantiates the generic model. It refines sensing and transition events. It selects the right differential equation in the corresponding *Actuate* event (in *Emptying* or *Filling* modes).

CONTEXT WaterTank0Ctx **EXTENDS** SystemCx
CONSTANTS
Emptying, Filling, Stable, Normal,
V_{max}, V_{high}, ⋯
ΔV_{max}, ⋯
AXIOMS
axm1: partition(STATES,{Emptying},{Filling},{Stable},{Normal})
axm2...: $V_{max} \in \mathbb{R}$, $V_{max} > 0$, ...

MACHINE WaterTank0 **REFINES** System
SEES WaterTank0Ctx
VARIABLES t, x_s, V
INVARIANTS
inv1: $V \in \mathbb{R}^+ \to S$
inv2: $x_p = V$
inv3: $\forall t \cdot t \in \mathbb{R}^+ \Rightarrow V(t) \in [V_{low}, V_{high}]$
inv4: $\forall t \cdot t \in \mathbb{R}^+ \Rightarrow |\dot{V}(t)| < \Delta V_{max}$

The associated Event-B context defines the needed constants as well as a few axioms on them. It also defines the system's modes. The Event-B machine refines the generic `System` machine. It defines time (t) as well as the discrete and continuous states (x_s, V). `inv2` links the abstract and concrete states (gluing invariant). `inv3` and `inv4` represent global safety invariants corresponding (resp.) to (SAF1) and (SAF2).

Initialization enforces the system to be in *Stable* mode (i.e.: no volume variation) at the beginning. To that extent, we set V to be the solution of $NoFlow(V_0)$, a differential equation that yields a constant solution of value V_0.

INITIALISATION **REFINES** INITIALISATION
WITH $x'_p = V'$
THEN
act1: $t := 0$
act2:
$V :| V \in \mathbb{R}^+ \to S \wedge$
solutionOf($\mathbb{R}^+$, V, $NoFlow(V_0)$)
act3: $x_s :=$ stable

Transition and *Sense* events follow the pattern given in the adjacent listing. *Transitions*' guards enforce a safe behavior whenever changing mode (e.g. we can empty the tank if the volume is above V_{low}). Conversely, *Sense* events trigger whenever the system's state needs correction (e.g. when the volume exceeds V_{high}).

ctrl_sense_too_high
REFINES Sense
WHERE $V(t) \geq V_{high}$
WITH
$s =$ {Emptying}
$p =$ STATES $\times \mathbb{R}^+ \times$
$\{v* \mid v* \geq V_{high}\}$
THEN
act1: $x_s :=$ Emptying
END
ctrl_transition_emptying
REFINES Transition
WHERE $V(t) > V_{low}$
WITH $s =$ {Emptying}
THEN
act1: $x_s :=$ Emptying
END

Finally, the *Actuate* event is simply a rewrite of the abstract one replacing x_p by V. We also constraints equation eq to enforce a category of behavior; that is, to constraint the shape of the equation's solutions.

ctrl_actuate_emptying **REFINES** Actuate
ANY eq
WHERE
grd1: $eq \in DE(S)$
grd2: Solvable($[t, +\infty[$, eq)
grd3: $flowEquation$(Emptying, $[t, +\infty[$, eq)
grd4: $x_s =$ Emptying
WITH $x'_p = V'$, $s =$ {Emptying}
THEN
act1:
$V :|$
$V' \in \mathbb{R}^+ \to S \wedge$
$[0, t[\lhd V' = [0, t[\lhd V \wedge$
solutionOf($[t, +\infty[$, $[t, +\infty[\lhd V'$, eq)
END

The constraints are encapsulated in the *flowEquation* predicate, which states that given a solution V^* to eq: 1) the variation of V^* matches the current state (e.g. when in *Emptying* mode V^* should decrease) and 2) the value of V^* ranges between 0 and V_{max}, for physical reasons.

D. Second Refinement: ODE Refinement

The second refinement constraints further the equation modeling the system, forcing it to be an ODE. The whole controller part remains unchanged (as well as the system's variables). Only the *continuous part* of *Actuate* is refined (*continuous event refinement*).

The parameter eq disappears in favour of Φ, the function of the ODE of the system ($\dot{V}(t) = \Phi(t, V(t))$). Just like for eq, we impose constraints on this function so that the behavior it describes matches the requirement of the system. These constraints are encapsulated in the *flowODE* predicate defined in the continuous concepts theory.

```
MACHINE WaterTank1 REFINES WaterTank0
...
ctrl_actuate_emptying
REFINES ctrl_actuate_emptying
ANY Φ
WHERE
  grd1: Φ ∈ ℝ+ × S → S
  grd2: flowODE(Emptying, [t, +∞[, Φ)
  grd3: Solvable([t, +∞[, ode(Φ, V(t), t))
  grd4: x_s = Emptying
WITH eq = ode(Φ, V(t), t)
THEN
  act1:
    V :|
      V' ∈ ℝ+ → S∧
      [0, t[ ◁ V' = [0, t[ ◁ V∧
      solutionOf([t, +∞[, [t, +∞[ ◁ V',
        ode(Φ, V(t), t)
      )
END
```

E. Third Refinement: Dual-Tank Concrete Plant

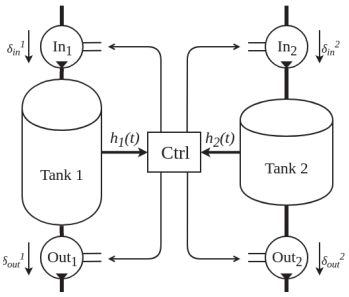

Fig. 4 – Diagram of the New System

1) Specification: In this refinement, the abstract tank is refined by two cylinder-shaped tanks. The tanks have a (fixed) circular section of area πR_1^2 (resp.) πR_2^2, and the controller does not have access to the volume directly, but it can sense the *height* of liquid in each tanks (h_1 and h_2). Each tank i is equipped with an input $[In_i]$ and output pump $[Out_i]$ that can be either activated $[= 1]$ or deactivated $[= 0]$ (discrete behavior). In active mode, the input pump for Tank 1 (resp. Tank 2) has a flow of δ_{in}^1 (resp. δ_{in}^2), the output pump for Tank 1 (resp. Tank 2) has a flow of δ_{out}^1 (resp. δ_{out}^2). The tanks are physically independent and controlled by their own differential equations:

$$\begin{aligned} \dot{h_1}(t) &= In_1 \cdot \delta_{in}^1 - Out_1 \cdot \delta_{out}^1 = \Delta_1(In_1, Out_1) \\ \text{and} \quad \dot{h_2}(t) &= In_2 \cdot \delta_{in}^2 - Out_2 \cdot \delta_{out}^2 = \Delta_2(In_2, Out_2) \end{aligned} \tag{1}$$

However, these two pumps are controlled by a single controller (*one* controller for *many* plants). As a matter of simplification, we impose that in *Emptying* mode every output pump is running and no input pump is and vice versa for *Filling* mode. In *Stable* mode no pump is active and in *Normal* mode there is no constraint on the pumps. For readability reasons, we denote by $\Delta_i(In_i, Out_i)$ the function for the ODE of tank i with pump state In_i and Out_i. Once the concrete plant is defined, the abstract volume V is refined (data refinement) by the *linear combination*

$$V = \pi R_1^2 \cdot h_1 + \pi R_2^2 \cdot h_2 \tag{2}$$

Moreover, as the tanks have different shapes, they have their own H_{1max} and H_{2max}, and we can write $V_{max} = \pi R_1^2 \cdot H_{1max} + \pi R_2^2 \cdot H_{2max}$.

```
MACHINE
      WaterTank_DualTank_Cylinder
REFINES WaterTank1
...
VARIABLES t, x_s, h_1, h_2
INVARIANTS
  inv1: h_1 ∈ ℝ+ → S
  inv2: h_2 ∈ ℝ+ → S
  inv3: V = πR_1^2 · h_1 + πR_2^2 · h_2
```

2) Modeling: This machine refines the previous one. It sees a specific context (not detailed here for space reasons) with the definitions and axioms of the various constants used throughout the model (R_1, H_{1max}, H_{10}, etc.).

The header also declares the new state variables of the system: V is replaced by h_1 and h_2 whose types are enforced in `inv1` and `inv2`. `inv3` gives the gluing invariant given in Equation 2.

Initialization sets the system in *Stable* mode, i.e. no pump is active and neither h_1 nor h_2 varies. The witness for V' is directly derived from the gluing invariant.

```
INITIALISATION REFINES INITIALISATION
WITH V' = πR_1^2 · h'_1 + πR_2^2 · h'_2
THEN
  act1: t := 0
  act2:
    h_1, h_2 :|
      h'_1 ∈ ℝ+ → S ∧ h'_2 ∈ ℝ+ → S∧
      solutionOf(ℝ+, h'_1, NoFlow(H_10))∧
      solutionOf(ℝ+, h'_2, NoFlow(H_20))
  act3: x_s := stable
```

```
ctrl_sense_too_high REFINES
      ctrl_sense_too_high
WHERE πR_1^2 · h_1(t) + πR_2^2 · h_2(t) ≥ V_high
THEN
  act1: x_s := Emptying
END
```

As V is no more appearing in this refinement, the guards of *Transition* and *Sense* events must be rewritten. We just need to inject the gluing invariant in the formula to express V using h_1 and h_2.

```
ctrl_actuate_emptying REFINES ctrl_actuate_emptying
WHERE x_s = Emptying
WITH V': V' = πR_1^2 · h'_1 + πR_2^2 · h'_2
      Φ: Φ ∈ ℝ+ × S → S∧
      flowODE(Emptying, [t, +∞[, Φ)∧
      Solvable([t, +∞[, ode(Φ, V(t), t))∧
      ∀h_1^*, h_2^*:
        h_1^* ∈ [t, +∞[→ S ∧ h_2^* ∈ [t, +∞[→ S∧
        solutionOf ([t, +∞[, h_1^*, ode(Δ_1(0, 1), h_1(t), t)) ∧
        solutionOf ([t, +∞[, h_2^*, ode(Δ_2(0, 1), h_2(t), t))
        ⇒ solutionOf ([t, +∞[, πR_1^2 · h_1^* + πR_2^2 · h_2^*, ode(Φ, V(t), t))
```

The *Actuate* event is refined using a witness for V' derived from the gluing invariant. But the most important part in this refined event is the witness provided for Φ. Indeed, a direct relation between Φ and the ODEs characterizing h_1 and h_2 is not available, so the witness for Φ cannot be of the form $\Phi = \ldots$. It has to be a more complex predicate that ensures that

1) Φ does not violate the invariants and guards of the abstract event;
2) given h_1^* solution of the ODE characterized by $\Delta_1(0, 1)$ and h_2^* solution of the ODE characterized by $\Delta_2(0, 1)$ (given in Equation 1), then

$$V^* = \pi R_1^2 \cdot h_1^* + \pi R_2^2 \cdot h_2^*$$

must be a solution of the ODE characterized by Φ

The second point is fundamental for continuous refinement. A continuous refinement rule is defined. *An equation $\mathcal{E}_C$ refines $\mathcal{E}_A$ if any solution of $\mathcal{E}_C$ is also a solution of $\mathcal{E}_A$.*

THEN
act1 :
$h_1, h_2 :|$
$h'_1 \in \mathbb{R}^+ \to S \wedge h'_2 \in \mathbb{R}^+ \to S \wedge$
$[0, t[\lhd h'_1 = [0, t[\lhd h_1 \wedge \text{solutionOf}\left([t, +\infty[, h'_1, \text{ode}(\Delta_1(0,1), h_1(t), t)\right) \wedge$
$[0, t[\lhd h'_2 = [0, t[\lhd h_2 \wedge \text{solutionOf}\left([t, +\infty[, h'_2, \text{ode}(\Delta_2(0,1), h_2(t), t)\right)$
END

The remaining part of the *Actuate* event is built following the abstract event. Past is preserved for h_1 and h_2 and their future is set as a solution of their respective ODE.

VIII. Assessment

The models for the case study presented in Section VI have been developed with Rodin. They are available at `https://www.irit.fr/~Guillaume.Dupont/models.php`.

Below, some technical details on the resulting proof effort for these models are given. We also discuss the possible use of the refinement chain we defined.

A. Proofs

The generic model yields 13 proof obligations that are easily discharged. The abstract tank (`WaterTank0`) generates 134 proof obligations, most of which are actually quite similar as the events are akin. Those proofs are not really complicated and often rely on the use of very generic theorems. The ODE-based tank (`WaterTank1`) produces 132 proof obligations, mainly refinement related (i.e. simulation, witness and guard strengthening). Those POs lead to longer proofs, often studded with well-definedness subgoals, not very complicated but very redundant. Last, the concrete model (`WaterTank_DualTank_Cylinder`) yields 94 proof obligations, again most of them being refinement related. They are substantially harder to discharge; especially because of the witnesses' form. Moreover, because not all the Rodin automatic provers are interfaced with the *theory plug-in*, our extensive use of it the heavily hinders proof automation. Thus, a lot of proofs have to be done completely manually.

B. Refinement Strategy

The refinement chain proposed is intentionally very generic. It can easily be extended both vertically (i.e. enriching the system's behavior) and horizontally (i.e. adding new types of behaviors). For example, it would be relevant to add a refinement level after `WaterTank1` that refines the machine with two abstract tanks, and then later refine this machine with cylinder-shaped containers associated with each abstract tank. Similarly, it is possible to refine `WaterTank1` or `WaterTank0` with a single tank, three tanks, n tanks, and so on without substantial increase of the proof effort.

Lastly, the situation we presented sets up a *single* controller and *many* plants. Other architecture patterns can be handled by our approach. Indeed, it is possible to refine the system differently so as to design a more complex system where *many* controllers control *many* plants. This kind of scheme, distributed and autonomous, is very recurring in the domain of cyber-physical systems. They do however raise other types of challenges, especially when it comes to ensure the conservation of *global* invariants.

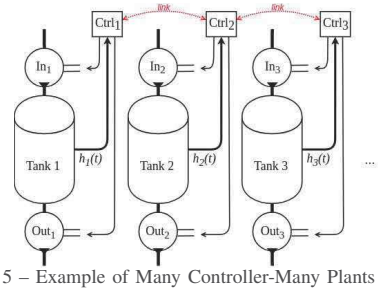

Fig. 5 – Example of Many Controller-Many Plants System

IX. Related work

Many approaches for the verification of CPS rely on

- model checking, mainly to study reachability problems, like *dReach*, *Phaver*, etc. studying hybrid systems whose plant behaviors are modelled using different types of differential equations (first/second order, linear/non linear etc.)
- static analysis of programs based on abstract interpretation. These approaches consist in building safety envelopes characterizing a safe behaviour of the modelled hybrid system.
- *KeYmaera* based at program level using a proof based approach grounded on dynamic logic for hybrid programs
- *Coq* and *Coquelicquot* based code analysis

All the proof-based approaches we have reviewed above use theories of reals. These theories support the definition of relevant properties like continuity and differentiation features or invariants to characterize real variables regions or to describe Taylor series. The approaches of Platzer [8], [16], Banach [10] and Boldo [11] support the explicit definition of differential equations. Time is implicitly considered in these approaches through these differential equations although *KeYmaera* supports explicit time definition using a differential equation of the form $t' = cst$. [11] deals with C programs using a suite of proof tools focusing on the implementation of ODE/PDE while *KeYmaera* [16] is deployed on hybrid programs that provide an abstract model of a hybrid system in a closed-loop modeling approach where ODEs are proved symbolically. Observe that there is no bibliographic reference between the approaches of [11] and of [16]. In [10], a similar approach to [8] is adopted. The added value of this approach is the use of refinement to define a step-wise formal development preserving the invariants in the different refinement levels. But, up to now, there is no tool supporting the approach.

The approaches of [33] and [34] use Event-B and the Rodin [15] platform to model hybrid systems in a closed-loop model. Time is explicitly modelled using a specific state variable. The authors consider continuous functions and they define discrete and continuous transitions preserving invariants characterizing the correct behaviour of the described hybrid

system. Refinement proved itself useful for the stepwise design of a hybrid system.

X. Conclusion

This paper addressed the problem of refining the continuous part of a CPS. We have proposed refinement relations and their proof obligations within Event-B. The concept of *refinement*, a well-known, powerful technique widely used in program development and certification, has been defined to handle continuous behaviors. We have shown that both discrete and continuous refinements can be deployed in a single integrated development. We then successfully used this principle to address a complete case study, in which an abstract specification *for the plant* has been continuously refined in two ways.

This case study demonstrates further the interest of the generic approach defined in [12] while showing a possible use of continuous refinement. It also lays out a path for other type of refinement that we plan to explore in the future. One of the promising research directions is to define a framework handling different formal generic CPS architecture patterns (single/many controllers and single/many plants). Discretization issues are also envisioned in order to offer discrete controllers synthesis capabilities.

Acknowledgement

The authors thanks R. Banach for the fruitful discussions on modelling hybrid systems.

References

[1] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*, 1st ed. LeeSeshia.org, 2014.

[2] E. A. Lee, "Constructive models of discrete and continuous physical phenomena," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-15, Feb 2014.

[3] ——, "The past, present and future of cyber-physical systems: A focus on models," *Sensors*, vol. 15, no. 3, pp. 4837–4869, 2015.

[4] I. Akkaya, P. Derler, S. Emoto, and E. A. Lee, "Systems engineering for industrial cyber–physical systems using aspects," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 997–1012, May 2016.

[5] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "Hytech: A model checker for hybrid systems," in *Computer Aided Verification*, O. Grumberg, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 460–463.

[6] S. Kong, S. Gao, W. Chen, and E. Clarke, "dreach: δ-reachability analysis for hybrid systems," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. Baier and C. Tinelli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 200–205.

[7] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogenous systems," *Int. Journal in Computer Simulation*, vol. 4, no. 2, 1994.

[8] A. Platzer, "Differential dynamic logic for hybrid systems." *J. Autom. Reas.*, vol. 41, no. 2, pp. 143–189, 2008.

[9] A. Platzer and J.-D. Quesel, "KeYmaera: A hybrid theorem prover for hybrid systems." in *IJCAR*, ser. LNCS, A. Armando, P. Baumgartner, and G. Dowek, Eds., vol. 5195. Springer, 2008, pp. 171–178.

[10] R. Banach, M. Butler, S. Qin, N. Verma, and H. Zhu, "Core Hybrid Event-B I: Single Hybrid Event-B machines," *Science of Computer Programming*, 2015.

[11] S. Boldo, F. Clément, J.-C. Filliâtre, M. Mayero, G. Melquiond, and P. Weis, "Trusting computations: A mechanized proof from partial differential equations to actual program," *Computers & Mathematics with Applications*, vol. 68, no. 3, pp. 325–352, 2014.

[12] G. Dupont, Y. Aït-Ameur, M. Pantel, and N. K. Singh, "Proof-based approach to hybrid systems development: Dynamic logic and event-b," in *Abstract State Machines, Alloy, B, TLA, VDM, and Z*, vol. LNCS 10817, 2018, pp. 155–170.

[13] G. Dupont, Y. A. Ameur, M. Pantel, and N. K. Singh, "Hybrid systems and event-b: A formal approach to signalised left-turn assist," in *New Trends in Model and Data Engineering - MEDI 2018 International Workshops, IWCFS, REMEDY,*, ser. Communications in Computer and Information Science, vol. 929. Springer, 2018, pp. 153–158.

[14] J.-R. Abrial, *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010.

[15] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin, "Rodin: an open toolset for modelling and reasoning in Event-B," *International Journal on Software Tools for Technology Transfer*, vol. 12, no. 6, pp. 447–466, 2010.

[16] J.-D. Quesel, S. Mitsch, S. Loos, N. Aréchiga, and A. Platzer, "How to model and prove hybrid systems with keymaera: a tutorial on safety," *International Journal on Software Tools for Technology Transfer*, vol. 18, no. 1, pp. 67–91, Feb 2016.

[17] N. Aréchiga, S. M. Loos, A. Platzer, and B. H. Krogh, "Using theorem provers to guarantee closed-loop system properties," in *2012 American Control Conference (ACC)*, June 2012, pp. 3573–3580.

[18] J. Abrial, S. A. Schuman, and B. Meyer, "Specification language," in *On the Construction of Programs*, 1980, pp. 343–410.

[19] R. J. R. Back and K. Sere, "Stepwise refinement of action systems," in *Mathematics of Program Construction*, J. L. A. van de Snepscheut, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 115–138.

[20] D. Gries, "Data refinement and the transform," in *Program Design Calculi, Proceedings of the NATO Advanced Study Institute on Program Design Calculi, Marktoberdorf, Germany, July 28 - August 9, 1992.*, 1992, pp. 93–119.

[21] C. Morgan, *Programming from specifications*. Prentice Hall, 1994.

[22] H. Jifeng and T. Hoare, "Equating bisimulation with refinement," *Technical Report 282, UNU-IIST*, 2003.

[23] M. Reynolds, "Continuous temporal models," in *AI 2001: Advances in Artificial Intelligence, 14th Australian Joint Conference on Artificial Intelligence*, ser. LNCS, M. Stumptner, D. Corbett, and M. J. Brooks, Eds., vol. 2256. Springer, 2001, pp. 414–425.

[24] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[25] H. Lin, "Mission accomplished: An introduction to formal methods in mobile robot motion planning and control," *Unmanned Systems*, vol. 2, no. 2, pp. 201–1 131 652, 2014.

[26] R.-J. Back, L. Petre, and I. Porres, "Continuous action systems as a model for hybrid systems," *Nord. J. Comput.*, vol. 8, no. 1, pp. 2–21, 2001.

[27] L. Meinicke and I. J. Hayes, "Continuous action system refinement," in *Mathematics of Program Construction, 8th International Conference, MPC 2006*, ser. Lecture Notes in Computer Science, T. Uustalu, Ed., vol. 4014. Springer, 2006, pp. 316–337.

[28] S. Mitsch, J.-D. Quesel, and A. Platzer, "Refactoring, refinement, and reasoning," in *FM 2014: Formal Methods*, ser. LNCS, C. Jones, P. Pihlajasaari, and J. Sun, Eds., vol. 8442. Springer, 2014.

[29] R. Banach, "Formal refinement and partitioning of a fuel pump system for small aircraft in hybrid event-b," in *2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE)*, July 2016, pp. 65–72.

[30] J.-R. Abrial, M. Butler, S. Hallerstede, M. Leuschel, M. Schmalz, and L. Voisin, "Proposals for mathematical extensions for Event-B," Tech. Rep., 2009.

[31] M. J. Butler and I. Maamria, "Practical theory extension in event-b," in *Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday*, 2013, pp. 67–81.

[32] A. A. Cardenas, S. Amin, and S. Sastry, "Secure control: Towards survivable cyber-physical systems," *System*, vol. 1, no. a2, p. a3, 2008.

[33] W. Su, J.-R. Abrial, and H. Zhu, "Formalizing hybrid systems with event-b and the rodin platform," *Science of Computer Programming*, vol. 94, no. Part 2, pp. 164 – 202, 2014, abstract State Machines, Alloy, B, VDM, and Z.

[34] M. Butler, J.-R. Abrial, and R. Banach, *From Action Systems to Distributed Systems: The Refinement Approach*, ser. Computer and Information Science Series. Chapman and Hall/CRC, Apr. 2016, ch. Modelling and Refining Hybrid Systems in Event-B and Rodin, pp. 29–42.