Refinement and Proof Based Development of Systems Characterized by Continuous Functions

Guillaume Babin, Yamine Aït-Ameur, Shin Nakajima, Marc Pantel

Refinement and Proof Based Development of Systems Characterized by Continuous Functions

Guillaume Babin[1(✉)], Yamine Aït-Ameur[1], Shin Nakajima[2], and Marc Pantel[1]

[1] Université de Toulouse; IRIT / INPT-ENSEEIHT,
2 Rue Charles Camichel, Toulouse, France
guillaume.babin@irit.fr, {yamine,marc.pantel}@enseeiht.fr
[2] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
nkjm@nii.ac.jp

Abstract. The specification of cyber-physical systems usually relies on continuous functions over dense real numbers whereas their implementation is discrete. Proving the correctness of the discrete implementation with respect to the continuous specification remains a challenge in the presence of dense real numbers. In this paper, we propose a refinement-based formal method, relying on Event-B, for such developments. We illustrate our proposal with the development of a simple stability controller for a generic plant model. The continuous function that models the system behavior is refined as a discrete model of the same kind preserving stability expressed as a safety invariants of the continuous model. The obtained discrete model uses discrete time (instants modeled on $\mathbb{N}$), whereas the continuous model is based on dense time (on $\mathbb{R}$). The Rodin Platform, together with the Theory plug-in handling the `Real` datatype and its properties supported the whole developments and proofs.

Keywords: Continuous and discrete behaviors · Dense real numbers · Correct-by-construction · Formal methods · Proved refinements · Event-B

1 Introduction

According to Lee [20], cyber-physical systems (CPS) are defined as *integrations of computation, networking, and physical processes. Embedded computers and networks monitor and control the physical processes, with feedback loops where physical processes affect computations and vice versa.* Most of the time, a software part (the controller) drives the physical part (the plant) through a loop involving sensors and actuators. The CPS plant behavior is given by dense time continuous functions solution of differential equations. The CPS controller behavior is specified by continuous functions over dense time. The CPS software implements a discretization of these functions in order to control the CPS plant. This discretization proof is a key challenge in the CPS correctness proof.

DOI: 10.1007/978-3-319-25942-0_4

In the past years, several approaches relying on formal methods, like Hybrid automata [17] and model checking [5], have been set up to describe the behavior of the software controllers. Our proposal focuses on the verification of correct controllers obtained after discretization.

This paper show how proof and refinement based approaches handle the development of a correct-by-construction discrete controller starting from a dense time continuous function specification of the continuous controller. A complete incremental development relying on a theory of reals is conducted to synthesize a correct discretization of a continuous function. The approach exploits an axiomatization of mathematical reals. It maintains a safety invariant characterizing the physical plant of the studied system. Such invariant defines a safety envelope (which we called *safety corridor*) modeling a *stability property* in which the system must evolve i.e. for a continuous function f, we write $\forall t \in \mathbb{R}^+, f(t) \in [m, M]$ where t is a dense time parameter and the reals m and M define minimum and maximum values in $\mathbb{R}^+$ ensuring a correct behavior of the physical plant. In general, these values come from the physics of the studied system. The Event-B method is used to handle such formal developments. We illustrate our proposal with the development of a simple stability controller for a generic plant model.

This paper is structured as follows. Section 2 overviews the addressed problem of discretization. Section 3 summarizes the Event-B method. Sections 4 and 5 are the core of our proposal: the refinement strategy for any continuous function together with the corresponding requirements are given in section 4 while the complete Event-B development handling these requirements is provided in section 5. Related works and possible applications are sketched in section 6. The conclusion and some perspectives are given in the end.

2 Discretization of Continuous Functions

The behavior of many systems can be characterized by three states: the initial boot, the nominal behavior, and the final halt. Several CPS involving physical plants and software controllers follow this pattern such as energy production systems, smart systems, medical systems, etc. These systems are usually modeled by differential equations specifying dense time continuous functions. In order to control their behavior, one has first to discretize these continuous functions. The main safety property concerns stability where the function values shall be maintained inside a safety envelope i.e. an interval of correct values (called *corridor*).

The correct implementation of such continuous functions is a key point in ensuring the CPS safety. These ones shall be discretized in a correct manner that guarantees that the discrete behavior simulates the continuous one. In other words, the continuous states existing between two observed consecutive states of the discretization are also in the safety corridor. To achieve this goal, we follow a correct-by-construction approach based on a formal development of *any* continuous function discretization, making our development reusable and scalable. The approach relies on refinement and on the preservation of invariants. Discretization information are incrementally added while moving from the

continuous level to the discrete one. Event-B [1] and the Rodin Platform [2] have been set up to handle the developments.

3 The Event-B Method

An Event-B model [1] (see Table 1) is defined in a *MACHINE*. It encodes a state transition system which consists of: variables declared in the *VARIABLES* clause to represent the state; and events declared in the *EVENTS* clause to represent the transitions (defined by a Before-After predicate BA) from one state to another.

Table 1. Structure of Event-B machines

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**
$A(s,c)$	$I(s,cv)$
THEOREMS	**THEOREMS**
$T_c(s,c)$	$T_m(s,c,v)$
END	**VARIANT**
	$V(s,c,v)$
	EVENTS
	Event $evt \triangleq$
	any x
	where $G(s,c,v,x)$
	then
	$v :\mid BA(s,c,v,x,v')$
	end
	END

Table 2. Generated proof obligations for an Event-B model

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

A model also holds *INVARIANTS* and *THEOREMS* to represent its relevant properties. A decreasing *VARIANT* may introduce convergence properties when needed. An Event-B machine is related, through the *SEES* clause to a *CONTEXT* which contains the relevant sets, constants axioms, and theorems. The refinement capability [4], introduced by the *REFINES* clause, decomposes a model (thus a transition system) into another transition system containing more design decisions thus moving from an abstract level to a less abstract one. New variables and new events may be introduced at the refinement level. In a refinement, the invariant shall link the variables of the refined machine with the ones of the refining machine. A gluing invariant is introduced for this purpose. It preserves the proved properties and supports the definition of new ones.

Once an Event-B machine is defined, a set of proof obligations is generated. They are submitted to the prover embedded in the RODIN platform. Proof obligations associated to an Event-B model are listed in Table 2, here the prime notation is used to denote the value of a variable after an event is triggered. More details on proof obligations can be found in [1].

Use of Reals in Event-B. A recent evolution of the Event-B method allows to extend it with theories [13] similar to algebraic specifications. In the Rodin Platform, this evolution is provided by the *Theory plugin for Rodin* [3]. We need to model and reason on dense *reals*. We rely on the theory for *reals* and continuous functions, written by Abrial and Butler[1]. It provides a dense mathematical `REAL` datatype with arithmetic operators, axioms and proof rules.

Remark. From a tool point of view, the use of reals with the *Theory plugin for Rodin* introduces constants like `zero` and operators defined on the `REAL` datatype like `smr` for $<$, `gtr` for $>$ or `leq` for $\leq$. Casting operators need to be defined in order to work with other data types. These ones are used when discretizing continuous representations by refinement (see section 5.3).

4 Refinement Strategy

The mathematical model and the specification of the system behavior are sketched below. Following the approach defined in [23], the adopted refinement strategy consists in three steps: first, as shown in figure 1, we use three states to define a simple abstract controller that models the system; then, in a first refinement, we introduce a continuous controller characterizing its behaviors with a continuous function; finally, a second refinement builds a discrete controller.

4.1 The Illustrating System

The considered system goes through three phases. Figure 1 depicts its general behavior. First, it is booted (transition *boot* from state 1 to 2). After a while, once in state 2, it becomes operational in a nominal mode (*run* transition). Then, it stays a given amount of time in the nominal or running mode. When in nominal mode, it may be halted (*stop* transition from state 2 to state 3) for example in case a failure occurs or for maintenance purposes. This behavior is the one of a simple *abstract* system controller. When booting, the system cannot be stopped until it reaches the nominal mode. Other complex behavior scenarios can be defined with more complex transition systems.

In order to guarantee a correct behavior of the system, the previously defined controller shall fulfill the requirements from table 3. These ones ensure that the system is correctly controlled. For example, an energy production system requires that the power produced by a given system belongs to a specific interval or a pacemaker must be pacing when a sensed signal belongs to another specific interval.

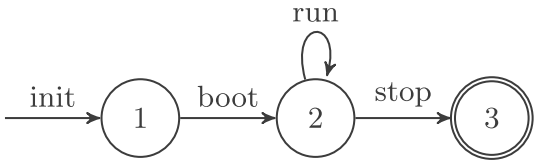


Fig. 1. Controller Automaton

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

Table 3. Requirements at the top level

At any time, the output value of the controlled system shall be less or equal to M in any mode.	Req.1
At any time, the output value of the controlled system shall belong to an interval $[m, M]$ in running mode.	Req.2
At any time, if any future output value of the controlled system does not belong to an interval $[m, M]$, then the system is stopped.	Req. 3

4.2 Continuous Controller

After modeling the system at an abstract level using three states, we introduce the continuous controller through the definition of a continuous function of the dense time $f : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ corresponding to the behavior of the system.

The requirements identified in the previous section, are rewritten (refined) to handle the introduced continuous function behavior (see table 4).

Table 4. Requirements at the first refinement

$m < M$	Req.0
$\forall t \in \mathbb{R}^+, f(t) \leq M$	Req.1
$\forall t \in \mathbb{R}^+, state(t) = 2 \Rightarrow f(t) \in [m, M]$	Req.2.1
$\forall t_1, t_2 \in \mathbb{R}^+, t_1 < t_2, state(t_1) = 2 \wedge f(t_2) \in [m, M] \Longrightarrow state(t_2) = 2$	Req.2.2
$\forall t_1, t_2 \in \mathbb{R}^+, t_1 < t_2, state(t_1) = 2 \wedge f(t_2) \notin [m, M] \Longrightarrow state(t_2) = 3$	Req. 3

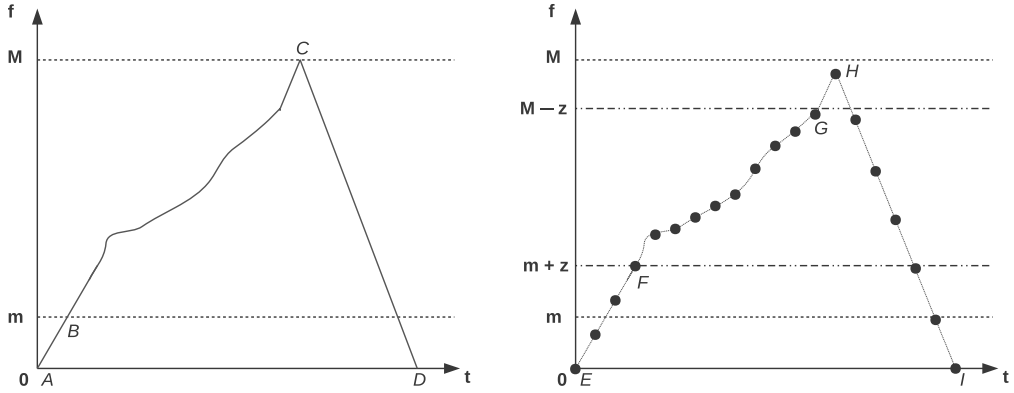
The control action over this system is a simple one. It consists in shutting down the system if the value of f goes out of range. The obtained continuous controller corresponds to a refinement of the abstract one from the previous section, it is described by a hybrid automaton [17]. We are aware that the control actions of the defined system are very simple. Our objective is to show how a controller (characterized by a simple state transition system) and a physical plant (characterized by a continuous function) can be formally integrated into a single Event-B formal development encoding incrementally a hybrid automaton.

The previously described behavior is depicted by the graph in figure 2(a). The system is initialized (at point A corresponding to the transition `init` to enter state 1). It reaches the running mode state at point B (corresponding to the event `boot` and entering state 2). The system stays in the safety corridor (between m and M in state 2). When point C is reached, the controller switches its state from state 2 to state 3 by the transition `stop` in order to prevent f from going over the threshold M. The system is then halted to reach point D (corresponding to state 3).

4.3 Discrete Controller

In order to implement the previous controller, we need to discretize the observation of the system behavior. In practice, when using computers to implement such controllers, time is observed according to specific clocks and frequencies. In other words, observations are discrete and depend on the available clocks.

Therefore, it is mandatory to define a correct discretization of time that preserves the continuous behavior introduced previously. This preservation entails the introduction of other requirements on the defined continuous function. Note that, in practice, these requirements correspond to requirements issued from the physical plant.



(a) Continuous controller (b) Discrete controller

Fig. 2. Examples of the evolution of the function f

It is mandatory to introduce a margin allowing the controller to anticipate the next observable behavior before incorrect behavior occurs. Let z be this margin. z is defined such that the derivation of the function f between two observed consecutive instants t_i and t_{i+1} shall not be greater than z. Formally, this is written as $z \geq \max_{i \in \mathbb{N}} |f(t_i) - f(t_{i+1})|$. We assume that a value for z exists (even if it is not the optimal one), it is obtained from the physical properties of the system. This means, that we need to identify the duration δt defining the amount of time between two consecutive states observed by the discrete controller. As a consequence, we write $z \geq \max_{t \in \mathbb{R}^+} |f(t) - f(t + \delta t)|$. In order for the problem to be well-defined, δt must be small enough so that the property $m + z < M - z$ holds. The set $\mathbb{D}$ of observation instants can be defined as $\mathbb{D} = \{t_i \mid t_i \in \mathbb{R} \wedge i \in \mathbb{N} \wedge t_0 = 0 \wedge t_{i+1} = t_i + \delta t\}$ and rewritten as $\mathbb{D} = \{t_i \mid t_i \in \mathbb{R} \wedge i \in \mathbb{N} \wedge t_0 = 0 \wedge t_i = i \times \delta t\}$.

As a consequence of this definition, the safety corridor becomes the interval $[m + z, M - z]$. Moreover, it becomes possible to observe, in the *running mode*,

Table 5. Requirements at the second refinement

$z > 0 \wedge m + z < M - z$	Req.0
$\forall t_i \in \mathbb{D}, f(t_i) \leq M$	Req.1
$\forall t_i \in \mathbb{D}, state(t_i) = 2 \Rightarrow f(t_i) \in [m + z, M - z]$	Req.2.1
$\forall t_i \in \mathbb{D}, state(t_i) = 2 \wedge f(t_i + \delta t) \in [m, M] \Longrightarrow state(t_i + \delta t) = 2$ $\Leftrightarrow \forall t_i \in \mathbb{D}, state(t_i) = 2 \wedge f(t_{i+1}) \in [m, M] \Longrightarrow state(t_{i+1}) = 2$ $\Leftrightarrow \forall n \in \mathbb{N}, state(n\,\delta t) = 2 \wedge f((n+1)\,\delta t) \in [m, M] \Longrightarrow state((n+1)\,\delta t) = 2$	Req.2.2
$\forall t_i \in \mathbb{D}, state(t_i) = 2 \wedge f(t_i + \delta t) \notin [m + z, M - z] \Longrightarrow state(t_i + \delta t) = 3$ $\Leftrightarrow \forall t_i \in \mathbb{D}, state(t_i) = 2 \wedge f(t_{i+1}) \notin [m + z, M - z] \Longrightarrow state(t_{i+1}) = 3$ $\Leftrightarrow \forall n \in \mathbb{N}, state(n\,\delta t) = 2 \wedge f((n+1)\,\delta t) \notin [m + z, M - z]$ $\Longrightarrow state((n+1)\,\delta t) = 3$	Req. 3

two consecutive instants t_i and t_{i+1} such that $f(t_i) \in [m+z, M-z]$ and $f(t_{i+1}) \notin [m+z, M-z] \wedge f(t_{i+1}) \in [m, M]$. This condition characterizes an exit from the safety corridor and thus the condition to stop the system and move to a stopping mode. Again, the previous requirements are refined to consider the discretization of time, using the two new parameters z and δt, and $\mathbb{D}$ (Table 5).

The safety margin z is defined such that if $f(n\,\delta t)$ is in $[m+z, M-z]$ then the value of f observed by the controller, $f((n+1)\,\delta t)$, is in $[m, M]$. The definition of this discretization guarantees that *Req.2.1* is fulfilled until the next value due to $\forall n \in \mathbb{N}, \quad \forall t \in [n \cdot \delta t, (n+1) \cdot \delta t], \quad |f(t) - f(n\,\delta t)| \leq z$. If the monitor observes a value in $[m, m+z[$ or in $]M-z, M]$, it shuts the system down because in the next step, the value might be out of range (*Req. 3*).

4.4 Top-Down Refinement

According to the previous definitions, the refinement starts from a generic definition of the system with the three identified events. The first refinement introduces the continuous function and the corresponding requirements of table 4. We start with a continuous model M_c of the system, describing the complete relevant physical behavior of the system. Then a second refinement defines the discrete model M_d of the behavior correctly glued with the continuous one. Here, the refined requirements of table 5 are taken into account. Gluing invariants, formalizing the refined requirements, are introduced in order to preserve the proofs and the behavior of the abstraction. When proving the refinement, we demonstrate that our discrete model is a correct implementation of the desired continuous behavior (the specification).

To summarize, in M_c, the continuous function $f_c : \mathbb{R} \longrightarrow \mathbb{R}$ is considered. In M_d, we introduce a discrete function $f_d : \mathbb{N} \longrightarrow \mathbb{R}$, where $i \in \mathbb{N}$ is an instant and δt is the time discretization interval duration. The functions f_d and f_c are glued by the following property: $\forall n \in 0..i, \; f_c(n \times \delta t) = f_d(n)$.

4.5 About Modeling of Time

In order to reduce the complexity of the proof of the discretization refinement corresponding to the introduction of f_d, we have split the behavior of f_c during an i^{th} discrete *macro step* $[t_i, \; (t_i + \delta t)]$ into three kinds of smaller discrete *micro steps* (see figure 3). For example, at the running state (or nominal phase), we define the following micro steps.

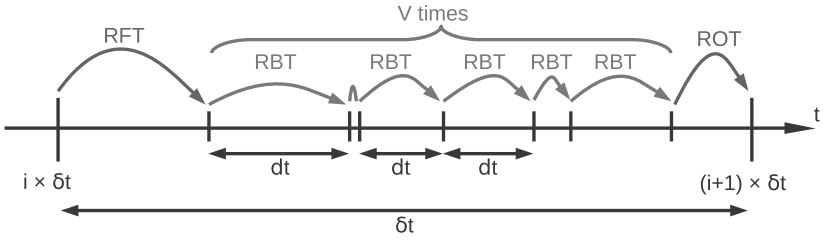


Fig. 3. Collapsing continuous time micro steps into a discrete time macro step

1. `RFT`: run from tick is the first micro step inside a macro step starting at a *tick* (a discrete time $t_i = i \times \delta t$). Its duration is strictly smaller than δt.

2. `RBT`: run between ticks is a micro step strictly in the macro step (not the first nor the last micro step in a macro step). Its duration is denoted $dt > 0$. A macro step contains V occurrences of such micro steps.

3. `ROT`: run on ticks is the last micro step in the macro step.

The Zeno problem is avoided by guaranteeing that the number of micro steps of type `RBT` is finite, and that $dt > 0$. From a modeling point of view, it will be formalized as a decreasing variant (natural number V in $\mathbb{N}$). The trace of micro steps between t_i and $t_{i+1} = t_i + \delta t$ is defined as `RFT (RBT)`^{V} `ROT`.

Our Event-B models introduce events aligned with these macro and micro steps either in the continuous case of in the discrete one.

5 A Formal Development of a Discrete Controller with Event-B

Our developments expressed within Event-B follow exactly the refinement strategy defined in section 4. According to [23], three development steps have been used. Contexts and machines are defined according to figure 4.

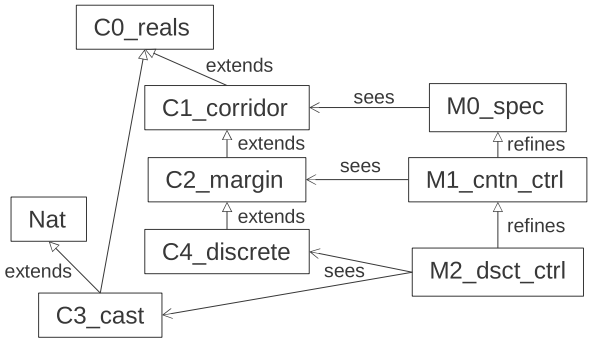


Fig. 4. Project structure

5.1 Abstract Machine: The Top-Level Specification

The top-level specification introduces the abstract controller with three events according to figure 1.

Needed Theories. To be able to handle real numbers and the corresponding theory, we have defined the context `C0_reals` which uses the theory defining reals. Listing 1.1 gives an extract of this context with axioms and theorems.

Several other axioms and theorems have been defined and proved. We show an extract of this theory. As mentioned in section 3 specific operators for manipulating reals are used.

```
CONTEXT C0_reals
CONSTANTS REAL_POS, REAL_STR_POS
AXIOMS
def01:REAL_POS={x | x ∈ REAL∧ leq(zero,x)}
def02:REAL_STR_POS={x| x∈ REAL
                        ∧ smr(zero,x)}
  ...
THEOREMS
  thm01: ∀a,b · ( a ∈ REAL ∧b ∈ REAL )
           ⇒( smr(zero,b) ⇒smr(a sub b , a) )
  thm02: ∀a,b · smr(a,b) ⇔¬leq(b,a)
  ...
END
```

Listing 1.1. Part of context C0_reals

```
CONTEXT C1_corridor
EXTENDS C0_reals
CONSTANTS m, M
AXIOMS
  axm01: m ∈ REAL_STR_POS
  axm02: M ∈ REAL_STR_POS
  axm03: smr(m,M)
END
```

Listing 1.2. Part of context C1_corridor

A second context defines the safety corridor with the values of m and M. Listing 1.2 defines this context `C1_corridor` extending the context `C0_reals`.

The Top-Level Event-B Machine. It defines the global continuous values issued from the controlled system. The machine introduces the invariant `inv03`, guaranteeing *Req*.1 and *Req*.2.1 stating that in running mode (identified by `active=true`), the continuous value (defining the values of a continuous function introduced in the first refinement) `fv` shall be correct. This machine also models the abstract controller with three events `boot`, `run` and `stop` corresponding to the transition system of figure 1. These events manipulate `fv` the real positive value corresponding to the current continuous value.

Listing 1.3 gives an extract of the top specification machine `M0_spec`. To keep this paper in a reasonable length, only details for the event `run` are given[2]. Therefore, the *Req*. 3 will not explicitly be handled in this paper, it mainly concerns the `stop` event.

```
MACHINE M0_spec SEES C1_corridor
VARIABLES fv, active
INVARIANTS
  inv01: fv ∈ REAL_POS
  inv02: active ∈ BOOL
  inv03: active = TRUE ⇒leq(m,fv) ∧leq(fv,M)
  inv04: active = FALSE ⇒fv = zero
EVENTS
  INITIALISATION ≙
  THEN
    act01: active := FALSE
    act02: fv := zero
  END
  boot ≙ ...
  run ≙
  ANY new_fv WHERE
    grd01: active = TRUE
    grd02: new_fv ∈ REAL_POS
    grd03: leq(m,new_fv) ∧leq(new_fv,M)
           // new_fv ∈ [m,M]
  THEN
    act01: fv := new_fv
  END
  stop ≙ ...
END
```

Listing 1.3. Extract of machine M0_spec

[2] The complete Event-B developments can be downloaded from http://babin.perso.enseeiht.fr/r/SETTA2015EventBModels.pdf

5.2 The First Refinement: Introducing Continuous Functions

Needed Theories. As shown on figure 4, the context `C2_margin` introducing the margin `z` is defined. Note that `axm02` corresponds to the requirement *Req*.0.

```
CONTEXT C2_margin EXTENDS C1_corridor
CONSTANTS z
AXIOMS
  axm01: z ∈ REAL_POS // z ∈ R+
  axm02: gtr(M sub m , (one plus one) mult z) // M−m > 2∗z
END
```

Listing 1.4. Extract of context C2_margin

The Event-B First Refinement with Continuous Functions. The first refinement `M1_cntn_ctrl` of the controller explicitly introduces

- the continuous function `fc` producing the values `fv` of the abstract machine and the corresponding invariant `prop01`
- dense time with the current instant noted `now`
- an important invariant `glue01` gluing the continuous values of the abstraction with the continuous function defined on dense time `fv = fc(now)`
- the variable `active_t` recoding the dense time where the system enters a running mode and the corresponding invariants `glue02`, `glue03` and `glue04` gluing the behavior of `active_t` with the `active` boolean variable.

The events of the `M1_cntn_ctrl` machine refine the ones of the top level specification. The `boot` event fixes the value of `active_t` and the `run` event builds the continuous function `fc` with steps of duration `dt`. `fc` becomes the function `nfc`, acting until `now+dt` instant.

```
MACHINE M1_cntn_ctrl REFINES M0_spec SEES C2_margin
VARIABLES
  fv, active, fc, now, active_t
INVARIANTS
  type01: now ∈ REAL_POS
  type02: fc ∈ REAL_POS →REAL_POS
  type03: active_t ∈ REAL_POS
  prop01: cnt_int(fc, zero, now) // fc is continous on [0,now]
  glue01: fv = fc(now)
  glue02: active = TRUE ⇒( ∀t ·t ∈ REAL ∧leq(active_t,t) ∧leq(t,now) ⇒
                                  ( leq(m plus z , fc(t)) ∧ leq(fc(t) , M sub z) ))
  glue03: ∀t · t ∈ REAL ∧leq(zero,t) ∧leq(t,now) ⇒leq(fc(t),M)
  glue04: active = TRUE ⇒leq(active_t,now)
EVENTS
  boot ≙ REFINES boot ...
   THEN
    ...
    act04: now := now plus dt
    act05: active_t := now plus dt

  run ≙ REFINES run
   ANY dt, nfc, new_fv WHERE
    ...
    grd04: dt ∈ REAL_STR_POS // dt > 0
    grd05: nfc ∈ REAL_POS ⇸REAL_POS
    grd06: dom(nfc) = {t | t ∈ REAL ∧leq(now,t) ∧leq(t , now plus dt)} // dom(nf) = [now,now+dt]
    grd07: nfc(now) = fc(now)
```

```
    grd08: nfc(now plus dt) = new_fv
    grd09: leq(fv,new_fv) ⇒(∀ t1,t2 · t1 ∈ dom(nfc) ∧t2 ∈ dom(nfc) ∧leq(t1,t2) ⇒
                                leq(nfc(t1) , nfc(t2)))
    grd10: cnt_int(nfc , now , now plus dt) //  nfc is continuous on [now,now+dt]
    grd11: leq(new_fv,fv) ⇒(∀ t1,t2 · t1 ∈ dom(nfc) ∧t2 ∈ dom(nfc) ∧leq(t1,t2) ⇒
                                leq(nfc(t2) , nfc(t1)))
    grd12: ∀t · t ∈ dom(nfc) ⇒leq(m plus z , nfc(t) ∧ leq(nfc(t) , M sub z)
   THEN
     ...
    act02: now := now plus dt
    act03: fc := fc ⩤nfc
   END
  stop ≙ REFINES stop...
END
```

Listing 1.5. Extract of machine M1_cntn_ctrl

The current instant `now` is incremented by the step duration `dt` as well. The guards of the event `run` introduce the relevant conditions to trigger this event.

Note that during the time interval of the step, the function `fc` shall be continuous and monotonic so as its value is never outside the safety corridor (`grd09` to `grd11`). This condition is fundamental when the function is discretized. Thus, `grd09` through `grd12` guarantee the requirement Req2.2 and are of particular importance when discretizing.

5.3 The Second Refinement: Introducing Discrete Representation

This refinement introduces the discretization function `fd` corresponding to the continuous function `fc` on each discrete observed instants. This fundamental property corresponds to requirement Req2.2 of table 5. It is expressed by the gluing invariants between the continuous controller and the discrete controller. It links the continuous f_c and and discrete f_d functions by the property $\forall n \in 0\,..\,i, f_c(n \times \delta t) = f_d(n)$ and is represented by invariant `glue01`.

```
CONTEXT C3_cast EXTENDS C0_reals, Nat
CONSTANTS cast
AXIOMS
  axm01: cast ∈ ℕ→REAL_POS      //  type
  axm02: cast(0) = zero     //   initial  case
  axm03: ∀a · a ∈ ℕ ⇒         // induction case
          (cast(a+1) = cast(a) plus one)
THEOREMS
  ...
  thm11: ∀a,b · (a ∈ ℕ ∧ b ∈ ℕ) // equiv. over '<'
          ⇒(a < b ⇔smr(cast(a),cast(b)))
  thm12: ∀a,b · (a ∈ ℕ ∧ b ∈ ℕ) // equiv. over '='
          ⇒(a = b ⇔cast(a) = cast(b))
  thm13: cast ∈ ℕ ⤖ cast[ℕ] // cast is a bijection
  ...
END
```

Listing 1.6. Definition and properties of the cast function

```
CONTEXT C4_discrete EXTENDS C2_margin
SETS VT
CONSTANTS
  tstep     //  discrete time step duration (δt)
  max_df    //  maximum delta for f during tstep
  RBT, RV
AXIOMS
  axm01: tstep ∈ REAL_STR_POS
  axm02: max_df ∈ REAL_POS
           //  max diff of f during tstep
  axm03: leq(max_df,z)
  axm04: partition(VT, {RBT}, {RV})
END
```

Listing 1.7. Extract of context C4_discrete

```
MACHINE M2_dsct_ctrl REFINES M1_cntn_ctrl SEES C3_cast, C4_discrete
VARIABLES
  fv, active, fc, now, active_t,
  fd // discrete power function
  i // the current instant number
  et // time elapsed from previous discrete value sampling time
  rs // remaining continuous micro steps inside the discrete macro step
  nv // next variant−related event type
INVARIANTS
  type01: fd ∈ 0..i →REAL_POS
  type02: i ∈ ℕ
  type03: et ∈ REAL_POS
  type04: rs ∈ ℕ
  type05: nv ∈ VT
  glue01: ∀ n · n ∈ 0..i ⇒fc(cast(n) mult tstep) = fd(n) // n ∈ 0..i ⇒fc(n∗tstep) = fd(n)
  glue02: now = (cast(i) mult tstep) plus et // now = i∗tstep + et
  inv01: ∀ n · n ∈ 0..i−1 ⇒(
              ∀ t · (leq(cast(n) mult tstep , t) ∧ leq(t , cast(n+1) mult tstep)) ⇒(
                     leq(fd(n) sub max_df , fc(t)) ∧ leq(fc(t) , fd(n) plus max_df)))
  inv02: ∀ t · (leq(cast(i) mult tstep , t) ∧ leq(t , now)) ⇒(
              leq(fd(i) sub max_df , fc(t)) ∧ leq(fc(t) , fd(i) plus max_df))
  inv03: smr(et,tstep)
VARIANT
  rs
EVENTS
  run_from_tick ≙ REFINES run
  WHERE
    ...
    grd13: et = zero
    grd14: smr(dt , tstep)
    grd15: ∀t · t ∈ dom(nfc) ⇒
             leq(fd(i) sub max_df , nfc(t))
             ∧ leq(nfc(t) , fd(i) plus max_df)
            // physical assumption
  THEN
    ...
    act04: et := et plus dt
    act05: rs :∈ ℕ
    act06: nv := RBT
  END

  run_between_ticks ≙ REFINES run
  WHERE
    ...
    grd13: smr(zero, et)
    grd14: smr(et plus dt , tstep)
    grd15: ∀t · t ∈ dom(nfc) ⇒
             leq(fd(i) sub max_df , nfc(t))
             ∧ leq(nfc(t) , fd(i) plus max_df)
    grd16: nv = RBT
    grd17: rs > 0
  THEN
    ...
    act04: et := et plus dt
    act05: nv := RV
  END

  run_variant ≙
  WHERE
    grd01: nv = RV
    grd02: rs > 0
  THEN
    act01: rs :| rs' ∈ ℕ∧ rs' < rs
    act02: nv := RBT
  END

  run_on_tick ≙ REFINES run
  WHERE
    ...
    grd13: et plus dt = tstep
    grd14: smr(zero,et)
    grd15: ∀t · t ∈ dom(nfc) ⇒
             leq(fd(i) sub max_df , nfc(t))
             ∧ leq(nfc(t) , fd(i) plus max_df)
   grd16: rs = 0
  THEOREMS
    thm03: cast(i+1) mult tstep = now plus dt
  THEN
    ...
    act04: i := i + 1
    act05: fd(i+1) := new_f
    act06: et := zero
  END

END
```

Listing 1.8. Extract of machine M2_dsct_ctrl

Needed Theories. Two contexts are introduced. As mentioned in section 3 the first context `C3_cast` is a technical context related to casting reals and integers. For example, the invariant $\forall n \in 0..i, f_c(n \times \delta t) = f_d(n)$ corresponding to `glue01` is written as `∀n · n∈0..i ⇒ fc(cast(n) mult tstep) = fd(n)`.

Note that the context `C3_cast` extends the `Nat` context[3] by Thai Son Hoang needed for handling inductive proofs on sets [4]. The last context `C4_discrete` introduces the discrete time macro steps duration `tstep` corresponding to δt on figure 3 and the values `RBT` and RV to identify the different events corresponding to the `run` event. It also defines the `max_df` constant corresponding to the maximum evolution of the function in a macro step is never more that the margin `z`. This assumption usually comes from the conditions on the physical plant.

The Event-B Refinement with Discretization. The defined machine `M2_dsct_ctrl` produces the discrete behavior of the continuous function `fc` with the discrete function `fd` glued by the invariant `glue01`. The other invariants `inv01` and `inv02` preserve Req2.2 and `inv03` states that the elapsed time `et` is less that the discrete time `tstep`. According to figure 3, three events for `ROT`, `RBT` and `RFT` are defined refine the `run` event. The `run_from_tick` (`RFT`) event starts the computation between two consecutive discrete values of function `fd` and fixes an arbitrary value of the variant `rs`.

The most interesting part in this machine relates to the `run_between_tick` (`RBT`) event which shall avoid the Zeno problem. For this purpose, each time this event is active, it triggers the event `run_variant` which decreases the variant. Once, this variant reaches the value 0, the `run_on_tick` (`ROT`) event is triggered to compute the final value corresponding to next discrete value of the function `fd`. Note that the guard `grd15` is fundamental to guarantee that the values do not exit the safety corridor. This assumption relates to the physical plant.

5.4 Proofs Statistics

All these models have been encoded within the Rodin Platform [2]. As shown on table 6, the main machine and the refinement led to 265 proof obligations.

Table 6. Rodin proofs statistics

Event-B model	Automatic proofs	Interactive proofs	Total
C0_reals	1	29	30
C1_corridor	0	6	6
C2_margin	0	10	10
C3_cast	11	26	37
C4_discrete	0	1	1
M0_spec (top-level)	11	6	17
M1_cntn_ctrl (1st ref.)	22	51	73
M2_dsct_ctrl (2nd ref.)	22	67	89
Total	67	198	265

[3] http://sourceforge.net/p/rodin-b-sharp/mailman/message/30378566/

[4] `induction: ∀S · S ⊆ ℕ ∧ 0 ∈ S ∧ (∀x · x ∈ S ⇒ x + 1 ∈ S) ⇒ ℕ ⊆ S`

67 were proved automatically and 198 needed numerous interactive proof steps. The interactive proofs mainly relate to the use of the Theory plugin for handling the reals. The lack of dedicated heuristics due to the representation of reals as an abstract data type, and not as a native type led to more interactive proofs.

6 Related Works and Applications

Two kinds of approaches for modeling hybrid systems can be distinguished. The first one relies on hybrid automata. They are mainly analyzed and verified by model checking [5]. Tools like HyTech [18] have succeeded in analyzing complex hybrid systems. While this approach enables automatic verification, it requires elaborate optimization techniques in order to handle the state space explosion as well as symbolic parameters and non-linear equations. To address these problems, logical analysis of hybrid automata brought interesting results [19]. They address classes of automata. The second kind of approaches relates to analysis of hybrid programs. One of the most successful tool is KeYmaera by Platzer et al. [22]. This tool is dedicated to hybrid system modeling and verification. It is equipped with an interactive theorem prover. Compared to Event-B, it does not provide a built-in refinement development operator. In the meantime, other approaches use Event-B to model hybrid systems. The work initiated in [23], and pursued in [12] proposes to model first the discrete events of a hybrid systems and then refine each event by introducing the continuous elements. It includes the use of a "now" variable, a "click" event that jumps in time to the next instant where an event can be triggered and simulated real numbers. In our proposal, we use this notion of "now" variable on dense time. Time jumps are encoded by the events. We use mathematical reals thanks to the latest developments of the Rodin Platform. Moreover, compared to [23], we have another refinement that introduces discretization of continuous elements. However, [23] incorporate analytical results from the study of differential equations into the Event-B models through the complementary use of Matlab/Simulink. The second proposed approach based on Event-B, initiated by Banach, is Hybrid Event-B [8]. This is an extension of Event-B which includes pliant events [7] as a way to model continuous behavior, allowing the direct use of differential equations in the modeling. However, there is no tool currently supporting this extension whereas our approach enabled us to develop and prove the models using available tools. Banach also worked on similar topics with ASM [9,10]. In our development we use reals defined by a minimal set of axioms. We do not use floating-point numbers, they may be introduced in a further refinement which is out of the scope of this paper. So, we are not exploiting the results from automated verification tools on floating-point numbers [21]. Static analysis [16] or abstract interpretation [14] (with tools such as Astrée [15]) have proved very powerful to analyze such programs. Our approach is at a modeling level. Moreover, the set of axioms for reals in the Theory plug-in we have used does not define reals in a constructive manner. So, we were not able to use the results obtained by the Coq [11] advanced proof tactics on reals. Indeed, our proofs have been discharged using the interactive prover of Rodin, leading to a large proof effort.

7 Conclusion

The development of cyber-physical systems needs to handle the behavior of the physical plant (environment). This behavior is usually described by continuous functions producing feedback information to the controller, which in turns produces orders to the actuators. In this paper, we have shown that it is possible to compose the development of both a controller and the corresponding behavior of the physical plant. The controller corresponds to a hybrid automaton. A simple one has been considered in this paper. It consists in booting, running and then stopping a physical plant (see figure 1). The main contribution of this paper concerns the synthesis of a discrete controller. We have shown that the synthesis of a correct-by-construction discretization of a continuous function associated to the behavior of a physical plant can be obtained by refinement. The proof of the preservation of the invariants gluing the continuous and discrete levels guarantees this correctness. We have introduced at the discrete level a variant guaranteeing that the model is Zeno-free. The Theory plug-in for the Rodin Platform and a theory of real numbers have been used to model continuous functions. To the best of our knowledge, this is the first attempt to model continuous controller discretization with the Event-B method and mathematical reals.

As future work, we plan to address more complex hybrid automata by generalizing the approach presented in this paper. A particular case we expect to study relates to the system substitution in case of failure for example, already addressed in the discrete case in [6]. Another research path concerns the refinement by floating point numbers as another discretization step. This refinement will use the intermediate value theorem as gluing invariant between the obtained discretization level and the floating point level. Finally, an effort should be devoted to handle more efficiently the complex proof process set up in this paper.

References

1. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering, 1st edn. Cambridge University Press, New York, NY, USA (2010)
2. Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. Int. J. Softw. Tools Technol. Transfer **12**(6), 447–466 (2010)
3. Abrial, J.-R., Butler, M., Hallerstede, S., Leuschel, M., Schmalz, M., Voisin, L.: Proposals for mathematical extensions for Event-B. Technical report (2009)
4. Abrial, J.-R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models: Application to Event-B. Fundamenta Informat. **77**(1), 1–28 (2007)
5. Alur, R.: Formal verification of hybrid systems. In: Chakraborty, S., Jerraya, A., Baruah, S. K., Fischmeister, S. (eds.) Proceedings of the 11th International Conference on Embedded Software, EMSOFT - ESWeek, Taipei, Taiwan, October 9–14, 2011, pp. 273–278. ACM (2011)
6. Babin, G., At-Ameur, Y., Pantel, M.: Formal verification of runtime compensation of web service compositions: A refinement and proof based proposal with Event-B. In: International Conference on SCC 2015 IEEE, pp. 98–105, June

7. Banach, R.: Pliant modalities in Hybrid Event-B. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods. LNCS, vol. 8051, pp. 37–53. Springer, Heidelberg (2013)
8. Banach, R., Butler, M., Qin, S., Verma, N., Zhu, H.: Core Hybrid Event-B I: Single Hybrid Event-B machines. Science of Computer Programming (2015)
9. Banach, R., Zhu, H., Su, W., Huang, R.: Formalising the continuous/discrete modeling step. In: Derrick, J., Boiten, E.A., Reeves, S. (eds.) Proceedings 15th International Refinement Workshop, Refine 2011, Limerick, Ireland, 20th June 2011, volume 55 of EPTCS, pp. 121–138 (2011)
10. Banach, R., Zhu, H., Su, W., Wu, X.: ASM and controller synthesis. In: Derrick, J., Fitzgerald, J., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., Riccobene, E. (eds.) ABZ 2012. LNCS, vol. 7316, pp. 51–64. Springer, Heidelberg (2012)
11. Boldo, S., Lelay, C., Melquiond, G.: Coquelicot: A user-friendly library of real analysis for Coq. Math. Comput. Sci. **9**(1), 41–62 (2015)
12. Butler, M., Abrial, J.-R., Banach, R.: From Action Systems to Distributed Systems: The Refinement Approach, chapter Modelling and Refining Hybrid Systems in Event-B and Rodin, p. 300. Taylor & Francis, February 2016
13. Butler, M., Maamria, I.: Practical theory extension in Event-B. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods. LNCS, vol. 8051, pp. 67–81. Springer, Heidelberg (2013)
14. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM POPL 1977, pp. 238–252, New York, NY, USA. ACM (1977)
15. Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: The ASTRÉE analyzer. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 21–30. Springer, Heidelberg (2005)
16. Goubault, É.: Static analyses of the precision of floating-point operations. In: Cousot, P. (ed.) SAS 2001. LNCS, vol. 2126, p. 234. Springer, Heidelberg (2001)
17. Henzinger. T. A.: The theory of hybrid automata. In: Inan, M.K., Kurshan, R.P. (eds.) Verification of Digital and Hybrid Systems, volume 170 of NATO ASI Series, pp. 265–292. Springer-Verlag (2000)
18. Henzinger, T.A., Ho, P.-H., Wong-Toi, H.: Hytech: A model checker for hybrid systems. International Journal on STTT **1**(1–2), 110–122 (1997)
19. Ishii, D., Melquiond, G., Nakajima, S.: Inductive verification of hybrid automata with strongest postcondition calculus. In: Johnsen, E.B., Petre, L. (eds.) IFM 2013. LNCS, vol. 7940, pp. 139–153. Springer, Heidelberg (2013)
20. Lee, E.A., Seshia, S.A.: Introduction to Embedded Systems - A Cyber-Physical Systems Approach. LeeSeshia.org, edition 1.5 edition (2014)
21. Muller, J.-M., Brisebarre, N., de Dinechin, F., Jeannerod, C.-P., Lefévre, V., Melquiond, G., Revol, N., Stehlé, D., Torres, S.: Handbook of Floating-Point Arithmetic. Birkhäuser (2010)
22. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer-Verlag, Heidelberg (2010)
23. Su, W., Abrial, J.-R., Zhu, H.: Formalizing hybrid systems with Event-B and the Rodin platform. Science of Computer Programming, 94, Part 2:164–202 (2014)