A System Substitution Mechanism for Hybrid Systems in Event-B

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

▶ **To cite this version:**

Guillaume Babin, Yamine Aït-Ameur, Neeraj Kumar Singh, Marc Pantel. A System Substitution Mechanism for Hybrid Systems in Event-B. International Conference on Formal Engineering Methods, Nov 2016, Tokyo, Japan. pp.106–121, 10.1007/978-3-319-47846-3_8. hal-03172256

HAL Id: hal-03172256

https://hal.science/hal-03172256

Submitted on 18 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A System Substitution Mechanism for Hybrid Systems in Event-B

Guillaume Babin, Yamine Aït-Ameur(✉), Neeraj Kumar Singh, and Marc Pantel

Université de Toulouse, IRIT/INPT-ENSEEIHT,
2 Rue Charles Camichel, Toulouse, France
guillaume.babin@irit.fr, {yamine,nsingh,marc.pantel}@enseeiht.fr

Abstract. Changes like failure or loss of QoS are key aspects of hybrid systems that must be handled during their design. Preserving the system state is a common requirement that can be ensured by reconfiguration relying on system substitution. The specification and design of these systems usually rely on continuous functions whereas their implementation is discrete. Moreover, the associated safety properties are characterized by a safety envelope defining safe system states. This paper presents a novel approach for formalizing the system substitution mechanism for hybrid systems, in which the system substitution maintains a safety envelope of the given hybrid system during system failure or switching from one supporting system to another. Proving the correctness of the discrete implementation of the defined reconfiguration mechanism for hybrid systems is a challenging problem. In this purpose, we propose to combine system substitution and incremental system modeling to ensure correct discretization. We rely on the Event-B method and the Rodin Platform with the *Theory* plug-in to develop the system models and carry out the proofs on dense real numbers.

Keywords: System reconfiguration and substitution · Continuous and discrete behaviors · Formal methods · Refinement and proof · Event-B

1 Introduction

Context. Cyber Physical Systems refer to the tight integration and coordination between computational and physical resources [18]. In these systems, a software component, the controller, manages the physical parts of the system. The early models for such systems usually rely on continuous functions. The controller is then implemented in a discrete manner thus combining continuous environment models with discrete controller models, building an hybrid system.

Proving the correctness of discrete implementations of continuous controllers is a challenging problem. Formal methods allow checking the correctness of such system functional requirements, including the required safety properties. Due to these core benefits, they have been adopted for designing and developing

DOI: 10.1007/978-3-319-47846-3_8

the new age of discrete controllers that must satisfy their original continuous specification [19] for building safe and reliable hybrid systems.

To prevent a system failure, controllers must react according to environment changes to keep a desired state or to meet minimum requirements that maintain a safety envelope for the system. A safety envelope is a safe over-approximation of system states. It can be modeled as invariants that define a set containing all possible system states under its nominal conditions. One key property studied in system engineering is the ability to take actions according to an evolving behavior. It may occur in different situations (e.g. failures, quality of service change, context evolution, maintenance, etc.). Most safety critical systems, such as avionics, nuclear, automotive and medical devices, whose failure could result in loss of life, including reputation and economical damage, use reconfiguration or substitution mechanisms to prevent losing the quality of system services required for system stability when a random failure occurs.

In our earlier work, we proposed both a correct by construction system substitution mechanism [8,9] and a strategy to derive discrete controllers from continuous specifications [6]. In [8,9], we defined the reconfiguration mechanism to maintain a safety property for a system (defined as a state-transitions system) during failure or to switch from one supporting system to another. The defined approach has been successfully applied, for the discrete case, on web services [7]. But it is not applicable straightforwardly for hybrid systems which need to handle continuous features. In [6], we presented the formal development of a continuous controller that is refined by a discrete one preserving the continuous functional behavior and the required safety properties. This work helped us formulating more general strategies, that we aim to develop in this paper, for the development of system substitution for hybrid systems using formal techniques.

Objective of this Paper. We target modeling hybrid systems, and providing modeling patterns for reconfiguration, using a correct by construction approach. We provide a generic system substitution mechanism for hybrid systems that allows maintaining a safety envelope during the system failure or switching from one supporting system to another using stepwise refinement in Event-B [3]. Moreover, we show how the defined substitution or reconfiguration mechanism applies to handle hybrid systems characterized by continuous functions using discrete functions. More precisely, we investigate the modeling of continuous systems in discrete form by preserving the continuous behavior. For hybrid systems, the system substitution is usually not instantaneous as it takes time to restore the state of the substituted system. We propose a special treatment to handle it. The primary use of the models is to assist in the construction, clarification, and validation of the continuous controller requirements to build a digital controller in case of system reconfiguration or system substitution. In this development, we use the Rodin Platform [4,16] to manage model development, refinement, proofs checking, verification and validation.

Paper Organization. The remainder of this paper is organized as follows. Section 2 presents preliminary details for system substitution mechanisms and the required modeling framework. Section 3 summarizes the studied systems and associated problems, including the informal requirements of the selected system. Section 4 explores an incremental proof-based formal development of system substitution for hybrid systems. Section 5 discusses our approach, and Sect. 6 presents related work and compares the results of this work with existing work. Finally, Sect. 7 concludes the paper with some future research directions.

2 Preliminaries

This section provides a comprehensive overview on system substitution mechanisms, for both continuous and discrete functions, that illustrates our proposal, and a basic overview on the Event-B modeling framework.

2.1 System Substitution Mechanism

System substitution allows to replace a system by another system that provides the same service. It can be used to ensure high availability in case of failure as required for safety critical systems such as avionics, nuclear, automotive and medical devices, where failure could result in loss of life, including reputation and economical damage. In general, system substitution can occur in any state of the system. We focus on *warm start* tagged as *Dynamic substitution*, where the substitute system will recover as much data and state variable values as possible from the halting state of the original system. *Dynamic substitution* allows replacing a failed system Sys_S with a new one Sys_T starting from the last running state of Sys_S. Thus, Sys_T must be initialized according to the last running state of Sys_S. In order to ensure that both systems provide the same services, they must implement the same specification $Spec$ according to the recovery states.

2.2 The Modeling Framework

Event-B [3] is a formal modeling notation, in which the event-driven approach extends the B-method [2]. The Event-B language has two main components, *context* and *machine*, to characterize the systems. A *context* describes the static structure of a system using *carrier sets*, *constants*, *axioms* and *theorems*, and a *machine* describes the dynamic structure of a system using *variables, invariants, theorems, variants* and *events*. Table 1 shows a formal organization of a model, in which various clauses (i.e. VARIABLES, EVENTS) are used to introduce the required modeling components for specifying the given system requirements. For instance, the clause VARIABLES represents the state and the clause EVENTS represents the transitions (defined by a Before-After predicate (BA)) of a system. A list of events can be used to model possible system behaviors that modify the state variables by providing appropriate *guards* in a *machine*. A model also

Table 1. Model structure

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 $evt \triangleq$
	any x
	where $G(s,c,v,x)$
	then
	$v :\mid BA(s,c,v,x,v')$
	end
	END

Table 2. Proof obligations

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	$A(s,c) \wedge I(s,c,v)$ $\wedge G(s,c,v,x)$ $\wedge BA(s,c,v,x,v')$ $\Rightarrow V(s,c,v') < V(s,c,v)$

contains INVARIANTS and THEOREMS clauses to represent its relevant properties to check the correctness of the formalized behavior. A VARIANT clause can be used to introduce convergence properties in a machine. Moreover, the terms like *refines, extends*, and *sees* are mainly used to describe the relation between components of Event-B models.

The Event-B modeling language supports a *correct by construction* approach to design an abstract model and a series of refined models for developing any large and complex system. The refinement, introduced by the REFINES clause, decomposes a model (thus a transition system) into another transition system containing more design decisions when moving from an abstract level to a less abstract one. Refinement supports modeling a system gradually by introducing safety properties at various refinement levels. New variables and new events may be introduced in a new refinement level. These refinements preserve the relation between the abstract model and its corresponding refined concrete model, while introducing new events and variables to specify more concrete behaviors of the system. The defined abstract and concrete state variables are linked by introducing *gluing invariants.*

The Rodin Platform provides rich tool support for model development using the Event-B language. It includes project management, model development, proof assistance, model checking, animation and automatic code generation. Once an Event-B model is modeled and syntactically checked in the Rodin Platform, then a set of proof obligations is generated with the help of the Rodin tools. Theses generated proof obligations are further passed to the inbuilt Rodin prover. The main proof obligations associated to an Event-B model are listed in Table 2, in which 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 [3].

The Theory Plug-In. A recent extension of the Event-B language allows extending it with theories [5] similar to algebraic specifications. In the Rodin Platform, this is provided by the *Theory* plug-in [13]. We formalize and analyze a system substitution mechanism applied to hybrid systems, that use the `REAL` datatype for state variables. Thus, we rely on the *Real* theory, written by Abrial and Butler[1] that provides a dense mathematical `REAL` datatype with arithmetic operators, an axiomatic semantics and proof rules.

3 Studied Systems

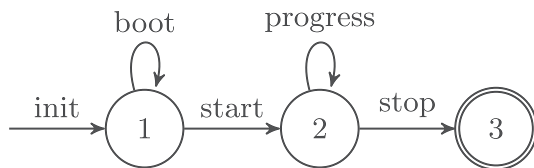


Fig. 1. Behavior of studied systems

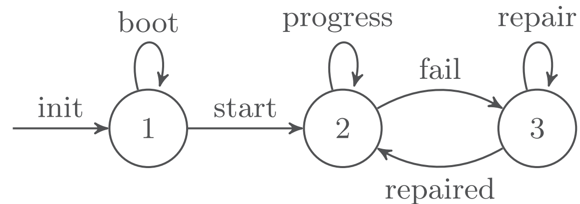


Fig. 2. System substitution

In this section, we describe the studied family of simple systems as patterns including the mechanism for system substitution. These ones are depicted in Fig. 1 for the system and Fig. 2 for the substitution mechanism. They are formalized as state-transition systems. Their behaviors are characterized by three states: *boot* (1), *progress* (2) and *stopped* (3). The *boot* state is the initial state, and the *progress* state is the nominal running state. According to Fig. 1, after initialization, a system enters the *booting* state, denoted as *state 1*, which may take a certain amount of time. If a system does not require the booting phase, then the system initialization is followed by a *start* transition without any delay. After this one, the system moves into the *progress* state, denoted as *state 2*. If the system stops, it switches into the *stopped* state, denoted as *state 3*.

3.1 Problem Statement

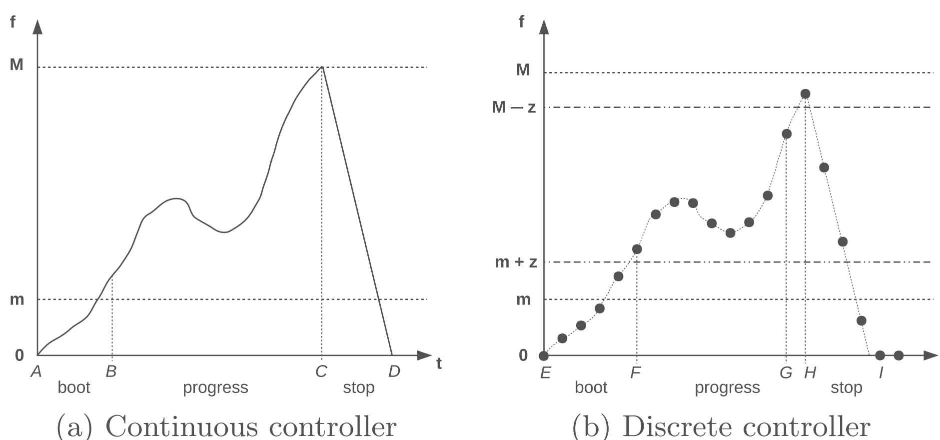
The substitution mechanism allows maintaining the running state of a given system in case of failure or decreasing QoS by replacing it with another one that provides the required behavior. A basic substitution pattern is defined by the state-transitions system of Fig. 2. When a failure occurs, the running system is halted (*fail* transition), then repaired in *state 3* where the state of the substitute system is restored from the halted system. Finally, the control is given to the substitute system (transition *repaired* from *state 3* to *state 2*). The substitution correctness has been studied in different cases (equivalent, degraded or upgraded cases). This mechanism (Fig. 2) shall satisfy the following requirements: (1) Preserving the required system behavior of the original system; (2) Restoring the halted system correctly.

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

Refinement is used to fulfill the first requirement. Several refinements may implement the same specification thus providing a class of systems that are candidate for substitution. The second requirement is expressed as a relation restoring the state variables of the substituted and substitute systems that must preserve the invariant and properties of the original specification. Details can be found in [8,9]. Substitutions can be instantaneous when it consists in restoring state variables that fulfill the specification invariant as shown in the case of web services compensation from [7]. But, for hybrid systems, it may require some time. The *repair* transition on state 3 of Fig. 2 must handle the repair process duration. This case is adressed in this contribution and the system behavior must be preserved during that duration.

3.2 Informal System Requirements

The hybrid systems behaviors models usually rely on continuous functions over time. Figure 3a depicts such a function f whose nominal value (after initialization) must stay in the safety enveloppe $[m, M]$. The time intervals $[A, B]$, $]B, C]$ and $]C, D]$ correspond respectively to state 1, 2 and 3 of Fig. 1. Any system controller, including a reconfiguration one, must observe the behavior of the system (here the function f) and act (preserve or change the system mode) to keep the observation in the safety enveloppe. Such observations and actions are usually implemented by a software that requires the discretization of the continuous functions. Figure 3b depicts such a discrete form for f. The time intervals $[E, F]$ $]F, H]$ and $]H, I]$ correspond respectively to state 1, 2 and 3 of Fig. 1. In the software that implements such controllers, time is observed according to specific clocks and periods. Therefore, it is mandatory to define a correct discretization of time that preserves the observed continuous behavior introduced previously. This preservation entails the introduction of other requirements on the defined continuous function. With respect to a time interval δt, the margin z is defined as respecting: $z \geq \max_{t,\delta t \in \mathbb{R}^+} |f(t) - f(t + \delta t)|$ (the evolution of f is assumed to be bounded) and $m + z < M - z$ (for consistency). Note that, in practice, these



(a) Continuous controller (b) Discrete controller

Fig. 3. Examples of the evolution of the function f

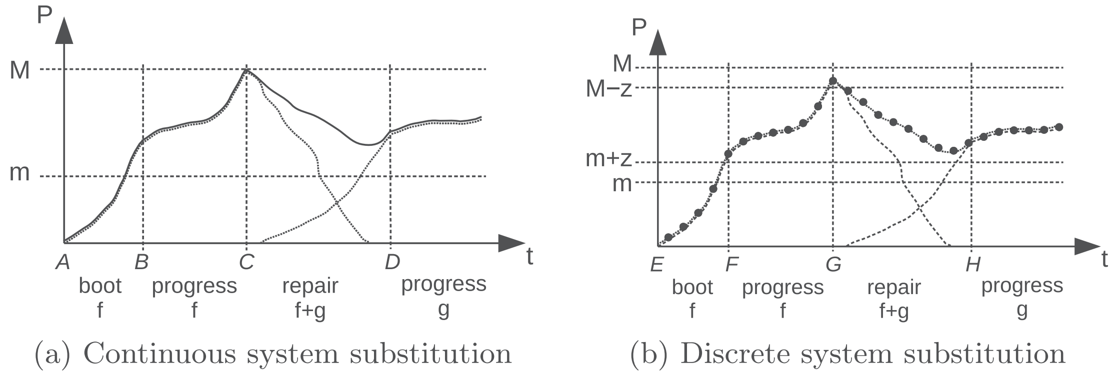
requirements are usually satisfied by the physical plant (f is usually a smooth continuous function).

Two continuous functions f and g characterize the behavior of two hybrid systems Sys_f and Sys_g. We assume that these systems maintain their observed output within the safety envelope $[m, M]$. Thus, they can substitute each other since they fulfill the same safety requirement. In this paper, we study the substitution of Sys_f by Sys_g after a failure occurrence (see requirements of Table 3).

Figure 4a and b show the substitution scenario in both continuous and discrete cases. The X axis describes time change and the vertical dashed lines model state transitions according to the behavior depicted in Fig. 2. Observe that during the repairing process (state 3 of Fig. 2) function f (associated with Sys_f) decreases due to its failure while function g (associated with Sys_g) is booting. The invariant states that $f + g$ belongs to the safety envelope $[m, M]$ during the repair (between C and D in the continuous case of Fig. 4a or G and H in the discrete case on Fig. 4b). Finally, the progress state 2 is reached a second time with Sys_g as the running system.

Table 3. Requirements in the abstract specification.

Requirement	
At any time, the feedback information value of the controlled system shall be less or equal to M in any mode	Req. 1
At any time, the feedback information value of the controlled system shall belong to the safety envelope $[m, M]$ in *progress* mode	Req. 2
The system feedback information value can be produced either by f, g or $f + g$ (f and g being associated to Sys_f and Sys_g)	Req. 3
The system Sys_f may have feedback information values outside $[m, M]$	Req. 4
At any time, in the *progress* mode, when using Sys_f, if the feedback information value of the controlled system equals to m or to M, Sys_f must is stopped	Req. 5



(a) Continuous system substitution

(b) Discrete system substitution

Fig. 4. Examples of the evolution of the function f

4 Formal Development

This section describes the stepwise formal development of studied systems in an abstract model and a sequence of refined models. The abstract model formalizes only the system initial behavior, while the refined models are used to define the concrete and more complex behaviors in a progressive manner that preserves the required safety properties at every refinement level.

Due to the limitation of the paper length, we only include a brief description of the model development and refinements. We invite readers to rely on the complete formal model available at [1] to understand the basic steps of the formal development, refinements and associated safety properties.

4.1 The Required Contexts

Contexts define the relevant concepts needed for our developments. The context *C_reals* (see Listing 1.1) defines the positive real numbers and theorems helpful for discharging the proofs. This context uses the `REAL` type for real numbers defined in the *Theory Real* by Abrial and Butler. Listing 1.2 introduces the constants *MODE_X* defining the different system modes (F, G and R for Sys_f, Sys_g and *Repair* modes) belonging to the *MODES* set.

```
CONTEXT C_reals −− Continuous functions
CONSTANTS
    REAL_POS, REAL_STR_POS
AXIOMS −− Axioms and theorems
        −− for continuous functions
    def01: REAL_POS = {x | x ∈ REAL ∧ 0≤ x)}
 ....
END
```

Listing 1.1. Context C_reals

```
CONTEXT C_modes
SETS
    MODES
CONSTANTS
    MODE_F, MODE_R, MODE_G
AXIOMS
    axm1: partition(MODES, {MODE_F},
                  {MODE_R}, {MODE_G})
END
```

Listing 1.2. Modes definition

The previous two contexts (*C_envelope* and *C_margin*) deal with the definition of a safety envelope. As mentioned in the requirements defined in Table 3, we define the interval of safe values as $[m, M]$ in the continuous case and $[m + z, M - z]$ with margin z in the discrete case.

```
CONTEXT C_envelope −− Safety envelope
EXTENDS C_reals
CONSTANTS
    m, M
AXIOMS
  axm01: m ∈ REAL_STR_POS
  axm02: M ∈ REAL_STR_POS
  axm03: smr(m,M)
THEOREMS
  thm01: m ≤ M
  thm02: 0 ≤ m
  thm06: 0 ≤ M
  thm03: ∀x · m ≤ x ⇒ x ∈ REAL_POS
  thm05: ∀a · m ≤ a ⇒ 0 ≤ a
END
```

Listing 1.3. Context C_envelope

```
CONTEXT C_margin −− Safety envelope margin
EXTENDS C_envelope
CONSTANTS
  z
AXIOMS
  axm01: z ∈ REAL_POS −− z ∈ R+
  axm02: M−m > 2∗z
THEOREMS
  thm03: 0 ≤ M−z
  thm06: z ≤ M−m
  thm07: m ≤ M−z
  thm08: m+z ≤ M
  thm10: m+z ≤ M−z
 ...
END
```

Listing 1.4. Context C_margin

4.2 Abstract Model: Definition of a Mode Controller

As shown in Fig. 2, we use three states to define a simple abstract controller (a mode automata) that models the system substitution through mode changes. Machine *M0* (see Listing 1.5) describes the abstract specification of the reconfiguration state-transitions system depicted in Fig. 2. The modes are used in the events guards to switch from one state to another. At initialization, Sys_f is started (*MODE_F*), it becomes active when the *active* variable is true (Sys_f ended the booting phase). When a failure occurs, progress of Sys_f is stopped. The controller enters in the repairing mode *MODE_R*. Once the system is repaired, the mode is switched to *MODE_G* and Sys_g enters the progress state.

```
MACHINE M0
SEES C_modes
VARIABLES
    active      -- true when the system is started
    md          -- running mode of the system
INVARIANTS
    type01: active ∈ BOOL
    type03: md ∈ MODES
    tech01: active = FALSE ⇒ md = MODE_F
EVENTS
    INITIALISATION=
    THEN
        act1: active := FALSE
        act2: md := MODE_F
    END
    boot = WHERE
        grd1: active = FALSE
        grd2: md = MODE_F
    END
    start= WHERE
        grd1: active = FALSE
        grd2: md = MODE_F
    THEN
        act1: active := TRUE
    END
    progress = WHERE
        grd2: active = TRUE
        grd1: md = MODE_F ∨ md = MODE_G
    END
    fail = WHERE
        grd2: active = TRUE
        grd1: md = MODE_F
    THEN
        act1: md := MODE_R
    END
    repair= WHERE
        grd2: active = TRUE
        grd1: md = MODE_R
    END
    repaired = WHERE
        grd2: active = TRUE
        grd1: md = MODE_R
    THEN
        act1: md := MODE_G
    END
END
```

Listing 1.5. The mode automata

4.3 First Refinement: Introduction of the Safety Envelope

The first refinement introduces the safety envelope $[m, M]$: the main invariant satisfied by all functions: f initially, $f + g$ during substitution and g after substitution. Machine *M1*, defined in Listing 1.6, refines *M0*. It preserves the behavior defined in *M0* and introduces two kinds of events: environment events (event name prefixed with *ENV*) and controller events (event name prefixed with *CTRL*) [23]. The *ENV* events produce the system feedback observed by the controller.

In this refinement, three new real variables f, g and p are introduced. f and g record the feedback information of Sys_f and Sys_g individually, while p records the feedback information of both systems before, during and after substitution. The variable p corresponds to f of Sys_f in *MODE_F*, g of Sys_g in *MODE_G* and $f + g$ of combined Sys_f and Sys_g in *MODE_R* corresponding to the system reparation (invariants *mode01* to *mode05*). In all cases, p shall belong to the safety envelope (invariants *envelope01* and *envelope02*). The *CTRL* events correspond to refinements of the abstract events of *M0*. They modify the control variable *active* and *md*. The *ENV* events observe real values corresponding to

the different situations where Sys_f and Sys_g are running or when Sys_f fails and Sys_g boots. This last situation corresponds to the reparation case.

```
MACHINE M1 REFINES M0
SEES C_envelope, C_modes
VARIABLES
    active, md, p, f, g
INVARIANTS
        ...
    envelope01: p ≤ M
    envelope02: active = TRUE ⇒ m ≤ p

    mode01: md = MODE_F ⇒ p = f
    mode04: md = MODE_F ⇒ g = 0
    mode02: md = MODE_R ⇒ p = f + g
    mode03: md = MODE_G ⇒ p = g
    mode05: md = MODE_G ⇒ f = 0
THEOREMS
    ....
EVENTS
    INITIALISATION=
     ....
    CTRL_started REFINES start =
    WHERE
        grd3: m ≤ p ∧ p ≤ M
    END
    ENV_evolution_f REFINES progress =
    ANY new_f
    WHERE
        grd2: active = TRUE ∧ md = MODE_F
        grd5: f ≠ m ∧ f ≠ M
        grd3: m ≤ new_f
        grd4: new_f ≤ M
    THEN
        act1: f := new_f
        act2: p := new_f
    END
```

```
    CTRL_limit_detected_f REFINES fail =
    WHERE
        grd5: f = m ∨ f = M
    END

    ENV_evolution_fg REFINES repair =
    ANY new_f, new_g
    WHERE
        grd3: m ≤ new_f + new_g
        grd4: new_f + new_g ≤ M
        grd5: 0 ≤ new_f
        grd6: new_f ≤ f
        grd7: g ≤ new_g
        grd8: new_g ≤ M
    THEN
        act1: f := new_f
        act2: g := new_g
        act3: p := new_f + new_g
    END

    CTRL_repaired_g REFINES repaired =
    WHERE
        grd3: m ≤ g
        grd4: g ≤ M
        grd5: f = 0    -- f+g to g is continuous
    END

    ENV_evolution_g REFINES progress =
     ...
END
```

Listing 1.6. Refinement with ENV and CTRL events

4.4 Second Refinement: Continuous Behavior and Dense Time

The behaviors of continuous controllers defined on dense time are modelled by continuous functions introduced by this refinement. This behavior is modelled in Machine *M2* (See Listing 1.7). It corresponds to Fig. 4a. Once the modes and the observed values are correctly set, the next refinements are straightforward. They correspond to a direct reuse of the development of a correct discretization of a continuous function proposed in [6].

Continuous functions f_c, g_c, p_c corresponding to variables f, g, p from *M1* are introduced. A real positive variable now represents the current time. The gluing invariants ($glue01$ for example $p = p_c(now)$) connect the variables of machine *M1* with the continuous functions values at time now. In the same way, each event of *M1* is refined. Time steps dt are introduced and the continuous functions are updated by the environment ENV events. The continuous functions are updated on the interval $[now, now + dt]$ and now is updated to $now := now + dt$. The control $CTRL$ events observe the value $p_c(now)$ to decide whether specific actions on the mode md_c variable are performed or not. Listing 1.7 shows an extract of this machine and a detailed description of this refinement is given in [1,6].

```
MACHINE M2 REFINES M1
SEES C_corridor, C_thms
VARIABLES
    now, p_c, f_c , g_c
    ...
INVARIANTS
    type01: now ∈ REAL_POS
    glue01: p = p_c(now)
    glue02: f = f_c(now)
    glue03: g = g_c(now)
    corridor01: ∀t · t ∈ [0,now] ⇒ p_c(t) ≤ M
    ...
EVENTS
...
 ENV_evolution_f
    REFINES ENV_evolution_f =
    ANY dt, new_f_c
    WHERE
        ...
      grd5: f_c(now) = new_f_c(now)
      grd6: ∀ t · t ∈ [now,now+dt] ⇒
                        new_f_c(t) ∈ [m,M]
  WITH
        new_f: new_f = new_f_c(now + dt)
  THEN
     act1: now := now + dt
     act2: p_c := p_c ⩤ new_f_c
     act3: f_c := f_c ⩤ new_f_c
      ...
  END
...
END
```

Listing 1.7. Machine M2

```
MACHINE M3 REFINES M2
SEES C_discrete, ...
VARIABLES
    p_d, f_d , g_d
    i   -- the current instant number
    et  -- time elapsed from previous discrete
        --                   value sampling time
    ...
INVARIANTS
    type01: f_d ∈ 0..i → REAL_POS
                -- similar for p_d and g_d
    type04: i ∈ ℕ
    glue01: ∀ n· n ∈ 0..i ⇒ f_c(n∗tstep)=f_d(n)
                -- similar for p_d and g_d
    glue02: now = i∗tstep + et
...
EVENTS
...
    ENV_evolution_f_on_tick
       REFINES ENV_evolution_f =
    ANY dt, new_f_c
    WHERE
        new_f_c ∈ [now,now+dt] → REAL_POS
         ...
    THEN
        act01: f := new_f
        act02: now := now + dt
        act03: f_c := f_c ⩤ new_f_c
        act04: i := i + 1
        act05: f_d(i+1) := new_f_c(now+dt)
        act06: et := 0
         ...
    END
...
END
```

Listing 1.8. Machine M3

4.5 Third Refinement: Discretization of the Continuous Behavior

This last refinement models a discrete controller. A discrete function is associated to values of the continuous function at each discrete time steps. The discrete behavior is given in Machine *M3* (See Listing 1.8). It models the behavior from Fig. 4b following the work in [6]. Again, we follow the same approach as for the refinement of the continuous behavior. As mentioned in the context *C_margin*, the margin z is defined, such that $0 < z \wedge m+z < M-z \ \wedge \ M-m > 2 \times z$. This margin defines, at the discrete level, the new safety envelope $[m+z, M-z] \subset [m, M]$. The new discrete variables f_d, g_d, p_d of *M3* are glued to f_c, g_c, p_c of *M2*. They correspond to discrete observations of f_c, g_c, p_c. The discretization step is defined as δt. Each environment event corresponding to a continuous event is refined into three events: the first one corresponds to discrete time now, the second one to discrete time $now + \delta t$ and the third one to any time in $]now, now+\delta t[$. In this discrete modeling, the last event ensures the correctness of refinement. Moreover, it must be *Zeno* free, so we introduce a decreasing variant in this refinement. The discrete controller observes only the events on time jumps from now to $now + \delta t$. Note that due to the discretization and the introduction of the z margin, a possible failure can be detected when $p_d(now) \in [m, m+z[\ \vee \ p_d(now) \in]M-z, M]$. The predicted behavior is enforced by the discrete controller that detects a limit before the value of m or M is reached. This situation is depicted in Fig. 4b at instant G.

4.6 Model Analysis

This section gives the proof statistics through detailed data about generated proof obligations. Event-B supports *consistency checking* which shows that a list of events preserves the given invariants, and *refinement checking* which ensures that a concrete machine is a valid refinement of an abstract machine. The whole formal development is presented through one abstract model and a sequence of three refinement models to cover the possible operations of system substitution of hybrid systems.

Table 4. Proof Statistics

Model	Total number of POs	Automatic proof	Interactive proof
Abstract model (M0)	5	5 (100 %)	0 (0 %)
First refinement (M1)	93	48 (52 %)	45 (48 %)
Second refinement (M2)	209	71 (34 %)	138 (66 %)
Third refinement (M3)	425	78 (18 %)	347 (82 %)
Total	**732**	**202 (28 %)**	**530 (72 %)**

Table 4 gives the proof statistics for the development using the Rodin tool. To guarantee the correctness, we established various invariants in the incremental refinements. This development resulted in 732 (100 %) proof obligations, of which 202 (28 %) were proved automatically, and the remaining 530 (72 %) were proved interactively using the Rodin prover (see Table 4). These interactive proof obligations are mainly related to the complex mathematical expressions and the use of *Theory* plug-in for `REAL` datatype, which are simplified through interaction, providing additional information to assist the Rodin prover.

5 Discussion

System substitution is a mechanism that allows to maintain the running state of a given system in case of any failure by preserving the required behavior. Specially, for developing critical systems, it is highly required to mitigate any risk of failure. On the other hand, stepwise refinement always plays an important role in designing a complex and large system systematically through progressive development. For developing the system substitution mechanism for hybrid systems, the stepwise refinement played an important role to preserve the required behavior and safety properties. As mentioned earlier, refinement is a core concept in Event-B development, and applying the refinement steps in a systematic order is always useful for designers to know what decisions must be taken for introducing system behaviors in each new refinement level. We identified the following development steps to integrate our system substitution mechanism for hybrid systems: (1) Define a set of modes for the controller; (2) Define a safety

envelope to preserve the desired behavior; (3) Handle the continuous behavior and dense time; (4) Model the discretization of the continuous function.

The proposed work is an extension of our previous work [6,8]. In [8], we have developed a generic formal model for system substitution and in [6], we have proposed the stepwise formal development for modeling continuous function using concrete functions. In this paper, we have used our existing approaches for addressing the challenges related to formal modeling and verification for the system substitution for hybrid systems. As far as we know, there are no similar published work. This work is a preliminary step for applying a system substitution mechanism for hybrid systems. We use the *Theory* plug-in for describing the hybrid systems and the required properties. In this experiment, we found that proof are quite complex and the existing Rodin tool support is not powerful enough to prove the generated proof obligation automatically. In fact, we need to assist the Rodin provers to find the required assumptions and predicates to discharge the generated proof obligations. On the other hand, we also found that the *Theory* plug-in is not yet complete. We have defined several assumptions and theorems in our model to help the proving process with the *Real* theory.

6 Related Work

Cyber-physical systems are strongly connected to their operating environment. Thus, the systems can adapt to environment changes to ensure the functional correctness. System reconfiguration is a key element to implement such kinds of systems that is proposed by several researchers. In [11], π-calculus and process algebra are used for system modeling, including reconfiguration, by exploiting behavioral matching based on bi-simulation. An Event-B approach was also proposed in [9]. The B-method is used for validating dynamic re-configuration of the component-based distributed systems using proofs techniques for consistency checking and temporal requirements [17]. Dynamic reconfiguration allows to stay in a system in a stable state using self-configuration and self-healing techniques. Rodrigues et al. [22] presented the dynamic membership mechanism as a key element of a reliable distributed storage system. Event-B is demonstrated in the specification of cooperative error recovery and dynamic reconfiguration for enabling the design of a fault-tolerant multi-agent system, and to develop dynamically reconfigurable systems to avoid redundancy [20]. Model checking of timed automata has been used by [15] to model and study the robustness of self-adaptive decentralized systems.

Cyber-physical systems belong to the class of hybrid systems, thus hybrid automata can be used to model the system requirements. The developed model can be verified through model checking tools, such as HyTech [14]. This approach enables automatic verification by exploring state space and required properties. Usually, model checking tools suffer from state explosion problem that impairs the use of any large model during verification process. Alternatively, theorem provers can be used to analyze and verify hybrid programs. The KeYmaera [21] tool, including an interactive theorem prover, is dedicated to hybrid system

modeling and verification. In [12,23], the development of an hybrid system is proposed using the correct by construction approach, where first, it specifies the discrete model and then refines 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 work [6], we use this notion of "now" variable on dense time, and time progression is defined by events. We use the *Theory* plug-in to model the continuous functions, and another layer of refinement that introduces discretization of continuous elements. Banach et al. [10] proposed Hybrid Event-B that is an extension of Event-B, which contains pliant events to model continuous behavior by using differential equations during system modeling. However, there is currently no tool support for this extension, whereas our approach [6] enabled us to develop and to prove the models using available tools. In our work, we use real numbers defined by a minimal set of axioms without addressing floating-point numbers, which is out of the scope of this paper.

7 Conclusion

Hybrid systems are dynamic systems that combine continuous and discrete behaviors to model complex critical systems, such as avionics, medical, and automotive, where an error or a failure can lead to grave consequences. For critical systems, recovering from any software failure state and correcting the system behavior at runtime is mandatory. The substitution mechanism is an approach that can be used to recover from failure by replacing the failed system. Its use for hybrid systems is a challenging problem as it requires to maintain a safety envelope through discrete implementation of continuous functions. To address this problem, we have presented a refinement based formal modeling and verification of system reconfiguration or substitution for hybrid systems by proving the preservation of the required safety envelope during the process of system substitution. In this paper, we have extended our work on system substitution to handle systems characterized by continuous models. First, we formalized the system substitution at continuous level, then we developed a discrete model through refinement by preserving the original continuous behavior. The whole approach is supported by proofs and refinements based on the Event-B method. Refinements proved useful to build a stepwise development which allowed us to gradually handle the requirements. Moreover, the availability of a theory of real numbers allowed us to introduce continuous behaviors which usually raise from the description of the physics of the controlled plants. All the models have been encoded within the Rodin Platform [4]. These developments required many interactive proofs in particular after the introduction of real numbers. The interactive proofs mainly relate to the use of the *Theory* plug-in for handling real numbers. Up to our understanding, the lack of dedicated heuristics due to the representation of real numbers as an axiomatically-defined abstract data type, and not as a native Event-B type together with our limited experience in defining tactics led to this number of interactive proofs.

This work opened several research directions. First, the models defined in this work handled a single parameter for information feedback with a simple safety envelope (interval that the value must belong to). We plan to investigate the reformulation of this problem when several parameters will be considered. In this case, the safety envelope becomes a more complex expression (a constraint solving problem). The second possible extension of this work is related to parametrization of the safety envelope with time. In other words, instead of having constant interval bounds, we may define bound functions $m(t)$ and $M(t)$. Other properties like elasticity could be expressed. However, this extension requires a powerful prover on real numbers and constraint solving problems techniques. Another possible extension of this work is the development of simulation. The integration of simulation or co-simulation to validate the formal model hypotheses will undoubtedly strengthen the approach. Finally, studying particular systems through realistic case studies is another objective of our work.

References

1. Models. http://babin.perso.enseeiht.fr/r/ICFEM_2016_Models/
2. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996). http://ebooks.cambridge.org/ebook.jsf?bid=CBO9780511624162
3. Abrial, J.R.: Modeling in Event-B: System and Software Engineering, 1st edn. Cambridge University Press, New York (2010)
4. Abrial, J.R., Butler, M., Hallerstede, S., Hong, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. Int. J. Softw. Tools Technol. Transf. **12**(6), 447–466 (2010)
5. Abrial, J.R., Butler, M., Hallerstede, S., Leuschel, M., Schmalz, M., Voisin, L.: Proposals for mathematical extensions for Event-B. Technical report (2009)
6. Babin, G., Aït-Ameur, Y., Nakajima, S., Pantel, M.: Refinement and proof based development of systems characterized by continuous functions. In: Li, X., et al. (eds.) SETTA 2015. LNCS, vol. 9409, pp. 55–70. Springer, Heidelberg (2015). doi:10.1007/978-3-319-25942-0_4
7. Babin, G., Aït-Ameur, Y., Pantel, M.: Formal verification of runtime compensation of web service compositions: a refinement and proof based proposal with Event-B. In: IEEE International Conference on Services Computing, pp. 98–105 (2015)
8. Babin, G., Aït-Ameur, Y., Pantel, M.: Correct instantiation of a system reconfiguration pattern: a proof and refinement-based approach. In: IEEE International Symposium on High Assurance Systems Engineering (HASE), pp. 31–38 (2016)
9. Babin, G., Aït-Ameur, Y., Pantel, M.: Trustworthy cyber-physical systems engineering. In: Romanovsky, A., Ishikawa, F. (eds.) A Generic Model for System Substitution. Chapman and Hall/CRC, Boca Raton (2016)
10. Banach, R., Butler, M., Qin, S., Verma, N., Zhu, H.: Core hybrid Event-B I: single hybrid Event-B machines. Sci. Comput. Program. **105**, 92–123 (2015)
11. Bhattacharyya, A.: Formal modelling and analysis of dynamic reconfiguration of dependable systems. Ph.D. thesis, Newcastle University, January 2013
12. Butler, M., Abrial, J.R., Banach, R.: From Action Systems to Distributed Systems: The Refinement Approach, chap. Modelling and Refining Hybrid Systems in Event-B and Rodin, pp. 29–42. Chapman and Hall/CRC., April 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. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HyTech: a model checker for hybrid systems. Int. J. Softw. Tools Technol. Transf. **1**(1–2), 110–122 (1997). http://dx.doi.org/10.1007/s100090050008
15. Iftikhar, M.U., Weyns, D.: A case study on formal verification of self-adaptive behaviors in a decentralized system. In: Kokash, N., Ravara, A. (eds.) 11th International Workshop on Foundations of Coordination Languages and Self Adaptation (FOCLASA 2012), EPTCS, vol. 91, pp. 45–62 (2012)
16. Jastram, M., Butler, M.: Rodin User's Handbook: Covers Rodin V.2.8. CreateSpace Independent Publishing Platform, USA (2014). ISBN 10: 1495438147, ISBN 13: 9781495438141, http://handbook.event-b.org
17. Lanoix, A., Dormoy, J., Kouchnarenko, O.: Combining proof and model-checking to validate reconfigurable architectures. Electron. Notes Theor. Comput. Sci. **279**(2), 43–57 (2011)
18. Lee, E.A., Seshia, S.A.: Introduction to Embedded Systems - A Cyber-Physical Systems Approach. LeeSeshia.org, 1.5 edn. (2014). http://leeseshia.org/
19. Lin, H.: Mission accomplished: an introduction to formal methods in mobile robot motion planning and control. Unmanned Syst. **02**(02), 201–216 (2014)
20. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: A refinement-based approach to developing critical multi-agent systems. Int. J. Crit. Comput.-Based Syst. **4**(1), 69–91 (2013)
21. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010). http://symbolaris.com/lahs/
22. Rodrigues, R., Liskov, B., Chen, K., Liskov, M., Schultz, D.: Automatic reconfiguration for large-scale reliable storage systems. IEEE Trans. Dependable Secure Comput. **9**(2), 145–158 (2012)
23. Su, W., Abrial, J.R., Zhu, H.: Formalizing hybrid systems with Event-B and the Rodin platform. Sci. Comput. Program. **94**, 164–202 (2014)