Handling Continuous Functions in Hybrid Systems Reconfigurations: A Formal Event-B Development

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

▶ **To cite this version:**

Guillaume Babin, Yamine Aït-Ameur, Neeraj Kumar Singh, Marc Pantel. Handling Continuous Functions in Hybrid Systems Reconfigurations: A Formal Event-B Development. ABZ 2016 - 5th International Conference Abstract State Machines, Alloy, B, TLA, VDM, and Z, May 2016, Linz, Austria. pp.290–296, 10.1007/978-3-319-33600-8_23 . hal-03155064

HAL Id: hal-03155064
https://hal.science/hal-03155064

Submitted on 2 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.

Handling Continuous Functions in Hybrid Systems Reconfigurations: A Formal Event-B Development

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

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

Abstract. This paper presents a substitution mechanism for systems having a continuous behavior. It shall preserve the safety property stating that the output of both systems remain in a safety envelope. The whole approach is formalized using Event-B, and relies on the Rodin tools and a theory of Reals provided by the Rodin Theory Plug-in to check the internal consistency with respect to safety properties, invariants and events.

1 Introduction

This paper relies on stepwise refinement in Event-B [2] to contribute to the formalization and verification of controllers in Cyber Physical Systems, relying on a generic substitution mechanism. In this work, we show how to apply the defined substitution as a reconfiguration mechanism to handle hybrid systems characterized by continuous functions which can be solutions of differential equations In this case as we model elements from the physical world, system substitutions are not instantaneous. We extensively use our previous work [4] on discrete controllers synthesis from continuous behavior descriptions. The primary use of the models is to assist in the construction, clarification, and validation of the continuous behavior controller requirements to build a digital controller in the presence of system reconfiguration or system substitution. In this development, we use the Rodin Platform [3,9] to manage model development, refinement, proofs checking, verification and validation.

2 The Event-B Method

An Event-B model [2] is defined by *contexts* and *machines*. It encodes a state-transitions system which consists of: *variables* to represent the state; and *events* to represent the transitions (defined by before-after predicates). A model also contains *invariants* to represent its relevant properties. A *variant* may ensure convergence properties when needed. An Event-B machine can *see* contexts that

DOI: 10.1007/978-3-319-33600-8_23

contain the relevant *sets*, *constants* and *axioms*. Refinement allows to add more behavior properties and system requirements by refining the abstract model to a more concrete one. New variables and new events may be introduced at the refinement level. Once a machine is defined, generated proof obligations need to be proven in order to demonstrate the preservation of the invariants.

Use of Reals in Event-B. In our work, we are interested in formalizing and analyzing system specifications while using reals. Therefore, we rely on the theory for *reals*, written by Abrial and Butler[1]. It provides a dense mathematical *REAL* datatype with arithmetic operators, axioms and proof rules.

3 Our Previous Work

This section recalls the seminal results we obtained for both system substitution and discretization of continuous functions. The work presented in this section is detailed in [4] and all the Event-B models related to the discretization of continuous functions are available in [1].

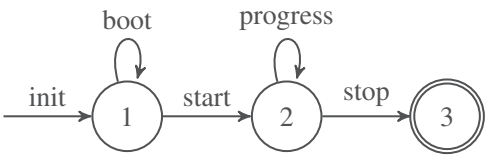


Fig. 1. Behavior of studied systems

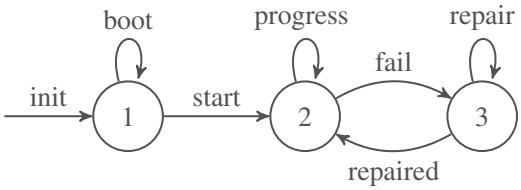


Fig. 2. System substitution

Studied Systems. They are formalized as state-transition systems. The behavior of such systems can be characterized by two states: the initial one (usually called *boot*) and the nominal one (*progress*). According to Fig. 1, after initialization, a system enters into the *booting* state (state 1). Then, after a *start* transition, the system progresses (state 2, known as *running* state). If the system stops, it switches to state 3.

System Substitution [5,6]**.** One of the key properties studied in system engineering is the capability of a system to react to changes. It may occur in different situations (e.g. failures, change of quality of service, context evolution, maintenance, etc.). System substitution can be defined as replacing a system by another one while preserving the required behavior.

In [5,6], we have developed a formal model (pattern) for system substitution described in the Event-B modeling language. This pattern is depicted in Fig. 2. When a failure occurs, the running system is halted (*fail* transition), it is repaired in state 3 where the state of the substitute system is restored from the one of the

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

halted system. Finally, the control is restored to the substitute system (transition *repaired*). The correctness of this substitution has been studied in different cases (equivalent, degraded or upgraded ones). The defined substitution mechanism (Fig. 2) preserves the behavior of the original specification and restores correctly the state of the halted system.

Discretization of Continuous Functions [4]**.** To control a system, in particular for system reconfiguration, it is required to observe (monitor the feedback behavior of the function) and to control (maintain or change system mode) the system. Such observation and control is performed by a software requiring the discretization of continuous functions. When using computers to implement such controllers, time is observed according to specific clocks and frequencies. 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. Note that, in practice, these requirements are usually satisfied by the physical plant.

In [4], we reported the stepwise formal development in Event-B of a correct discretization of a Lipschitz continuous function f characterizing a hybrid system Sys_f. The development consists in the following steps.

- **Specification: the mode controller.** It models systems whose behaviors are described in the state-transitions system of Fig. 1. The safety requirement ensures that the observed values remain in the safety envelope defined by the interval $[m, M]$.
- **Introduction of continuous behaviors.** This refinement consists in introducing a continuous function f defined on the domain of positive real numbers (for dense time). The observed values of f will belong to the defined safety envelope $[m, M]$.
- **The second refinement: correct discretization of continuous behaviors.** Discretization requires the introduction of a margin allowing the controller to anticipate (predictive control) the next observable behavior before incorrect behavior occurs. Let $z > 0$ be this margin. We use the Lipschitz continuous function property of f to define the amount of time between two consecutive states observed by the discrete controller. We introduced macro time steps of duration δt (discrete control sampling time interval). As a consequence, we define z such that $z \geq \max_{t \in \mathbb{R}^+} |f(t) - f(t + \delta t)|$. In order to make it well-defined, δt must be small enough so that the property $m + z < M - z$ holds. As a consequence, the set $\mathbb{D}$ of observation instants can be defined as, $\mathbb{D} = \{t_i \mid t_i \in \mathbb{R} \wedge i \in \mathbb{N} \wedge t_0 = 0 \wedge t_{i+1} = t_i + \delta t\}$
  It can be 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 result of the definitions of z and δt, if $f(n \times \delta t)$ is in $[m + z, M - z]$ then we can safely predict using the Lipschitz condition that the next value of f observed by the controller, $f((n + 1) \times \delta t)$, is in $[m, M]$.

4 Hybrid System Substitution in the Presence of Continuous Behaviors

We consider two continuous functions f and g characterizing the behavior of two hybrid systems Sys_f and Sys_g. We also assume that these two systems maintain their feedback information value in the safety envelope $[m, M]$. As a consequence, these two systems can substitute each other since they fulfill the same safety requirement. The studied scenario consists in substituting Sys_f after a failure by Sys_g (see requirements in Table 1).

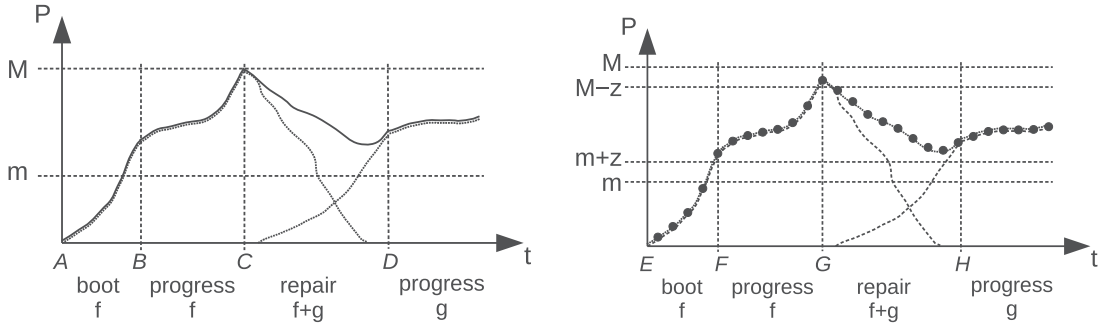
Figure 3a and b show the substitution scenario in both continuous and discrete cases. The X axis describes time passing and the vertical dashed lines model state transitions according to the behavior depicted in Fig. 2. Observe that during repair (state 3 of Fig. 2), the function f (associated with Sys_f) decreases while the function g (associated with Sys_g) is booting. The invariant states that the function $f + g$ belongs to the safety envelope $[m, M]$ during the repair. Finally, the system returns to *progress* (state 2) using Sys_g as the running system.

Table 1. Requirements in the abstract specification.

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 an interval $[m, M]$ 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

4.1 Refinement Strategy

The substitution process is defined for replacing Sys_f by Sys_g, both systems being described by the state-transitions system of Fig. 2. Following the approach defined in [10], the adopted refinement strategy consists of an abstract specification and three refinements: *Definition of a Mode controller (M0), Introduction of the safety envelope (M1), Handling continuous behavior and dense time (M2)* and *Discretization of the continuous behavior (M3).*



(a) Continuous system substitution (b) Discrete system substitution

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

4.2 The Event-B Models

We describe the stepwise formal development of the models corresponding to the refinement strategy sketched above. All the Event-B models are available in [1].

– **The required contexts.** Contexts define the relevant concepts needed for our developments. The context *C_reals* defines a set of theorems for positive reals, continuous and monotonic functions. This context uses the *REAL* type for real numbers defined in *RealTheory* (using the Theory plug-in for Rodin). The context *C_modes* defines the constants *MODE_X* representing the different system modes (F, G and R for Sys_f, Sys_g and *Repair* modes) belonging to the *MODES* set. The next two contexts (*C_envelope* and *C_margin*) deal with the definition of the safety envelope. They define the intervals of safe values: $[m, M]$ in the continuous case, and $[m + z, M - z]$ with margin z in the discrete case.

– **The root machine: Definition of a Mode controller.** Machine *M0* describes the reconfiguration state-transitions system depicted in Fig. 2. The modes are used in the events guards that allow the system 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 reparation mode *MODE_R*. Once the reparation is completed, the mode is switched to *MODE_G* and Sys_g enters into a progress state.

– **First refinement: Introduction of the safety envelope.** Machine *M1* 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*) [10]. The *ENV* events produce the system feedback observed by the controller. Three continuous variables f, g and p are introduced. f and g record the feedback information values of Sys_f and Sys_g individually, while p records the feedback value of both systems before, during and after substitution. 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 reparation after failure. Once the system has booted, p must belong to the safety envelope in all the cases. The *CTRL* events correspond to refinements

of the abstract events of *M0*. They modify the control variable *active* and *md* (the mode). The *ENV* events observe real values corresponding to the different situations where Sys_f and Sys_g are running or when Sys_f failed and Sys_g is booting. This last situation corresponds to the reparation case.

– **Second refinement: Continuous behavior and dense time.** Machine *M2* (refining *M1*) describes the continuous behavior (Fig. 3a). Once the modes and the observed values are correctly set, the next refining events are straightforward. They correspond to a direct reuse of the development of a correct discretization of a continuous function as done in [4]. Indeed, continuous variables f_c, g_c, p_c and md_c corresponding to the functions f, g, p and md to the modes in *M1* are introduced. A real positive variable now is defined to represent the current time. The gluing invariants (for example $p = p_c(now)$) connect the variables of *M1* with the continuous variables at time now. In the same way, each event of *M1* is refined. A non-deterministic time step dt is introduced and the continuous functions are updated by the *ENV* events on the interval $[now, now + dt]$ while now is updated to $now := now + dt$. The control *CTRL* events observe the value $p_c(now)$ to decide whether specific actions on mode md_c variable are performed or not. A detailed description of this refinement is given in [4].

– **Third refinement: Discretization of the continuous behavior.** Following [4], the discrete behavior is described in a Machine *M3* (Fig. 3b). As mentioned in the context *C_margin*, the margin z is defined, such as $0 < z \wedge m + z < M - z$ and $M - m > 2 \times z$. This margin defines, at the discrete level, the new safety envelope as $[m + z, M - z] \subset [m, M]$. The new discrete variables f_d, g_d, p_d and md_d of *M3* are glued to f_c, g_c, p_c and md_c of *M2*. They correspond to discrete observations feedback of f_c, g_c, p_c and md_c. The discretization step is defined as δt. Each environment event corresponding to a continuous event is refined by three events. One discrete event starting a time interval δt, one for time intervals included in δt and the next discrete event (third one) at end of δt. The second event may occur several times, a variant enforces it to be Zeno free. The discrete controller only observes the events at time $n \times \delta t$.

5 Conclusion

Modeling hybrid systems using Event-B was studied by several other authors [7,8,10]. In this paper, we have extended our work on system substitution to handle systems characterized by continuous models. First, we modeled system substitution at a continuous level, then we provided a discrete model for substitution which preserves the original continuous behavior. This work reused previous results we obtained on the correct system reconfiguration modeling and the correct discretization of continuous behaviors. By correctness, we mean the preservation of system information feedback in a safety envelope. The whole approach is supported by proof and refinement based on the Event-B method. Refinement proved useful to build a stepwise development which allowed us to gradually handle the requirements. Moreover, the availability of a theory of

reals allowed us to introduce continuous behaviors which usually raise from the description of the physics of the controlled plants. All these models have been developed within the Rodin platform [3] and the developed formal models can be downloaded from [1].

References

1. Models. http://babin.perso.enseeiht.fr/r/ABZ_2016_Models/
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering, 1st edn. Cambridge University Press, New York (2010)
3. 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. International Journal on Software Tools for Technology Transfer **12**(6), 447–466 (2010). http://dx.doi.org/10.1007/s10009-010-0145-y
4. 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
5. Babin, G., Ait-Ameur, Y., Pantel, M.: A generic model for system substitution. In: Romanovsky, A., Ishikawa, F. (eds.) Trustworthy Cyber Physical Systems Engineering. CRC Press Taylor & Francis Group (2016)
6. Babin, G., Ait-Ameur, Y., Pantel, M.: Correct instantiation of a system reconfiguration pattern: a proof and refinement-based approach. In: 2016 IEEE High Assurance Systems Engineering Symposium, HASE 2016, Orlando, FL, USA, January 7–9, 2016. IEEE Computer Society Press (2016)
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. 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, p. 300. Taylor & Francis, February 2016. http://www.taylorandfrancis.com/books/details/9781498701587/
9. Jastram, M.: Rodin User's Handbook (Oct 2013). http://handbook.event-b.org
10. Su, W., Abrial, J.R., Zhu, H.: Formalizing hybrid systems with Event-B and the Rodin platform. Sci. Comput. Program. **92**(2), 164–202 (2014). http://www.sciencedirect.com/science/article/pii/S0167642314002482