# Correct Instantiation of a System Reconfiguration Pattern: A Proof and Refinement-Based Approach

Guillaume Babin, Yamine Aït-Ameur, Marc Pantel

# Correct instantiation of a system reconfiguration pattern: a proof and refinement-based approach

Guillaume BABIN*‡, Yamine AIT AMEUR*‡ and Marc PANTEL*‡
*Université de Toulouse ; INP, *IRIT* ;
2 rue Camichel, BP 7122, 31071 Toulouse Cedex 7, France
‡*CNRS* ; Institut de Recherche en Informatique de Toulouse ; Toulouse, France
guillaume.babin@irit.fr, yamine@enseeiht.fr, marc.pantel@enseeiht.fr

***Abstract*—System substitution can be defined as the capability to replace a system by another one that preserves the specification of the original one. It may occur in different reconfiguration situations like failure management or maintenance. When substituting a system at runtime, a key requirement is to correctly restore the state of the substituted one. This paper proposes a correct by construction generic model for system reconfiguration defined using formal methods, based on a system substitution operator. Systems are seen as state transition systems. This proposal relies on refinement and proofs. The formal development is conducted with the Event-B method. It consists in defining system substitution as a system composition operator associated to proof obligations. A generic formal model is developed using Event-B. Specific systems instantiate this generic model using a particular use of refinement-based on the definition of witnesses. This proposal is illustrated with an electronic commerce service.**

***Keywords*—system substitution, system reconfiguration, proof and refinement-based methods, Event-B**

## I. Introduction

Several formal system development approaches have proved the efficiency and the scalability of formal methods for realistic systems using deductive verification, model checking and abstract interpretation. These approaches have been integrated into system engineering life cycles and are currently set up in many engineering domains like aeronautic and space, transportation systems, medical systems or energy production. Checking that systems behave correctly is a key requirement in system engineering. Moreover, the capability to assert that families of systems behave correctly is often used for certification purposes to avoid system specific activities. Formal methods were shown to be good candidates to handle such verification processes and to supply well-founded argumentation for certification. One of the key properties studied in system engineering is the capability of a system to react to changes (e.g. failures, quality of service change, context evolution, maintenance, etc.). In this context, the objective of this proposal is twofold: first, to address the design of adaptive, resilient, dependable, self-$\star$ systems etc. which require the capability to reconfigure running systems (more precisely, reconfiguration can be seen as the substitution of a system by another one); second, to put formal methods (more precisely proof and refinement-based methods) into practice to model a system substitution operation for a family of systems whose behavior is characterized by transition systems. The Event-B method is used to perform the formal developments. Two different substitution relations are studied. The first one is a static substitution (corresponding to a *cold start*) that relies on refinement to characterize the set of systems that conform to the same specification. A class of potential implementation systems are thus characterized by refinement. The second one addresses the dynamic substitution (substitution at runtime or *warm start*). It relies on a composition operator that combines two systems that refine the same specification. This composition operator is parameterized by the *substitution* or *reparation property* ensuring that the current state (the state where the source system is halted) is correctly restored in the substitute system. Moreover, we identify three substitution modes for the composition operator: equivalent, degraded or upgraded substitute systems.

This paper proposes a generic system reconfiguration formal model developed using correct-by-construction stepwise refinement and proof-based formal methods. Event-B supports the whole formal development of the system substitution operator. The developed generic model can be instantiated to any number of systems to be substituted. The proposed approach is generic and an instantiation mechanism, based on a specific refinement with witnesses, is proposed to overcome the state space explosion problem usually encountered when model checking-based verification techniques are set up.

We have structured this paper as follows. First, the next section gives an overview of the Event-B method. Then, section III describes the proposed substitution operator through the definition of a parameterized composition operator for which proof obligations are synthesized, and section IV describes an application of this generalized approach. The mathematical setting describing the generalization of this approach is presented in section V. Then, the corresponding Event-B models handling this generalized model are described in section VI and the associated instantiation mechanism is described in section VII. The same case study is used to instantiate this generic model in section VIII. Then, an assessment of the proposed approach is shown in section IX, and section X gives some related work. Finally, a conclusion summarizes our contribution and some future research paths are discussed in the last section.

## II. Event-B: a Correct-by-Construction Method

An Event-B[1] model [1] (see Listing 1) is defined in a *MACHINE*. It encodes a state transition system which contains: variables, declared in the *VARIABLES* clause, that represent the states; and events, declared in the *EVENTS* clause, that

[1] http://www.event-b.org/

represent the transitions (defined by a Before-After predicate $BA$) from one state to another ($:|$ for the *becomes* operator).

**Context** $ctxt\_id\_2$
**Extends** $ctxt\_id\_1$
**Sets** $s$
**Constants** $c$
**Axioms** $A(s,c)$
**Theorems** $T_c(s,c)$
**End**

**Machine** $machine\_id\_2$
**Refines** $machine\_id\_1$
**Sees** $ctxt\_id\_2$
**Variables** $v$
**Invariant** $I(s,c,v)$
**Theorems** $T_m(s,c,v)$
**Variant** $V(s,c,v)$
**Events**
  **Event** Initialisation $\triangleq$
    **Any** x **Where** $G(s,c,x)$
    **Then** $v :| D(s,c,x,v')$
  **Event** evt $\triangleq$
    **Any** $x$ **Where** $G(s,c,v,x)$
    **Then** $v :| BA(s,c,v,x,v')$
**End**

Listing 1. Structures of Event-B contexts and machines

A model also contains *INVARIANTS* and *THEOREMS* that represent its relevant properties. A decreasing *VARIANT* introduces mandatory convergence properties. An Event-B machine is related through the *SEES* clause to a *CONTEXT* that contains the relevant sets, constants, axioms and theorems required to build an Event-B model. The refinement capability, introduced by the *REFINES* clause, builds a new model (thus a new transition system) that contains more design decisions representing the changes from an abstract level to a less abstract one. In a refinement, new variables and new events may be introduced. Gluing invariants are defined to link the variables of the refined machine with the ones of the refining machine. This refinement process ensures the preservation of proved properties and supports the definition of new refined models.

TABLE I. GENERATED PROOF OBLIGATIONS FOR AN EVENT-B MODEL

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

Once an Event-B machine is defined, a set of proof obligations is generated. They are passed to the prover embedded in the Rodin platform [2]. Proof obligations associated to an Event-B model are listed in Table I. The prime notation is used to distinguish between pre ($x$) and post ($x'$) variables. More details on proof obligations can be found in [1].

## III. OUR APPROACH FOR SYSTEM SUBSTITUTION

Studied systems are formalized as the state-transition systems. According to Figure 1, a system is initialized, then it evolves (`progress`) relying on state changes. A failure can occur during state change. The system may then be repaired, or isolated (`complete failure`).

Two main requirements are identified to allow the substitution of $Sys_S$ by $Sys_T$ (e.g. in case of failure) :

**[Req1.]** *Static substitution.* $Sys_S$ and $Sys_T$ are two systems implementing the same specification *Spec*.

**[Req2.]** *Dynamic substitution.* In case of failure of $Sys_S$, the system $Sys_T$ is activated at runtime. The state of $Sys_T$ will be initialized according to the current state of $Sys_S$.

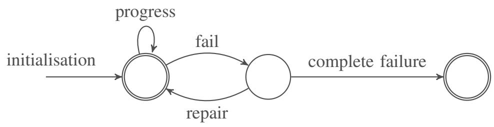


Fig. 1. Studied system behavior pattern

### A. The global system specification

Systems are formalized, within Event-B, as state-transition systems. Listing 2 shows a generic model representing the *Spec* machine for the global system specification. States are defined as a set of variables. Their correct values are constrained by invariants. States are initialized, and transitions are modeled as events (e.g. `evt`) that may affect state variables. These transitions model progress. An inductive invariant ($I_A$) ensures the correct behavior (safety) of the defined state-transition system, and an optional variant ($V_A$) ensures reachability.

**Context** C0
**Sets** $s$
**Constants** $c$
**Axioms** $A(s,c)$
**End**

**Machine** *Spec* **Sees** C0
**Variables** $v_A$
**Invariant** $I_A(s,c,v_A)$
**Variant** $V_A(s,c,v_A)$
**Events**
  **Event** Initialisation $\triangleq$
    **Then** $v_A :| D_A(s,c,v_A')$
  **Event** evt $\triangleq$
    **Any** $x_A$
    **Where** $G_A(s,c,v_A,x_A)$
    **Then** $v_A :| BA_A(s,c,v_A,x_A,v_A')$
**End**

Listing 2. An Event-B model for describing a system specification: a context `C0` and a machine *Spec*

### B. Static substitution

Every system refining the global specification is a candidate for system substitution. Listing 3 depicts two Event-B refinements of the specification *Spec*. They correspond to two systems $Sys_S$ and $Sys_T$ that implement the *same* specification *Spec*. A variable $m$ (for mode) has been added to express which system is used. Invariants $I_S$ and $I_T$ define relevant properties and ensure the preservation of the specification properties. This refinement fulfills requirement *Req1*.

**Machine** $Sys_S$
**Refines** *Spec* **Sees** C0
**Variables** $v_S, m$
**Invariant** $m = S \Rightarrow I_S(s,c,v_S)$
**Variant** $V_S$
**Events**
  **Event** Initialisation $\triangleq$
    **Then** $m := S$
      $\wedge\ v_S :| D_S(s,c,v_S')$
  **Event** evt **Refines** evt $\triangleq$
    **Any** $y_S$
    **Where** $m = S$
      $\wedge\ G_S(s,c,v_S,y_S)$
    **With** $y_S : x_S = y_S$
    **Then**
      $v_S :| BA_S(s,c,v_S,y_S,v_S')$
**End**

**Machine** $Sys_T$
**Refines** *Spec* **Sees** C0
**Variables** $v_T, m$
**Invariant** $m = T \Rightarrow I_T(s,c,v_T)$
**Variant** $V_T$
**Events**
  **Event** Initialisation $\triangleq$
    **Then** $m := T$
      $\wedge\ v_T :| D_T(s,c,v_T')$
  **Event** evt **Refines** evt $\triangleq$
    **Any** $y_T$
    **Where** $m = T$
      $\wedge\ G_T(s,c,v_T,y_T)$
    **With** $y_T : x_T = y_T$
    **Then**
      $v_T :| BA_T(s,c,v_T,y_T,v_T')$
**End**

Listing 3. Event-B models for *Sys_S* and *Sys_T* system substitutes for *Spec*

### C. Dynamic substitution

Two events are introduced (see Listing 4): the first one records the occurrence of a failure (`fail`) and halts the currently running system by setting the variable `m` to $F$; the second one (`repair`) transfers the control at runtime from $Sys_S$ to $Sys_T$ by 1) setting the variables of $Sys_T$ with values derived from the ones of the interrupted state of $Sys_S$. This assignment is possible and safe thanks to the definition of an *horizontal invariant* `P1` gluing the state variables of systems $Sys_S$ and $Sys_T$ and 2) initializing the variant for $Sys_T$ using a property `P2`. The variable `m` is then set to $T$ to transfer the control to $Sys_T$.

```
Machine Sys_G
Refines Spec
Sees C0
Variables v_S, v_T, m
Invariant  m = S ⇒ I_S(s, c, v_S)
           ∧ m = T ⇒ I_T(s, c, v_T)
           ∧ m = F ⇒ I_S(s, c, v_S)
Variant V_S + V_T
Events
  Event Initialisation ≙
    Then
      m := S
      v_S :| D_S(s, c, v'_S)
      v_T :| ⊤
  Event s_evt Refines evt ≙
   Any x_S
   Where m = S ∧ G_S(s,c,v_S,x_S)
   Then
      v_S :| BA_S(s,c,v_S,x_S,v'_S)
  Event t_evt Refines evt ≙
   Any x_T
   Where m = T ∧ G_T(s,c,v_T,x_T)
    Then
      v_T :| BA_T(s,c,v_T,x_T,v'_T)
  Event fail ≙
    Where m = S
    Then
      m := F
  Event repair ≙
    Where m = F
    Then
      v_S, v_T :| P1(v_S, v_T, v'_S, v'_T)
      V_T :| P2(V_S, V'_T)
      m := T
End
```

Listing 4. The resulting global system $Sys_G$

### D. Resulting global system

The resulting system composes systems $Sys_S$ and $Sys_T$ and the events `fail` and `repair` into one single Event-B machine as depicted on Listing 4. The obtained Event-B model encodes the substitution pattern of Figure 1. The mode `m` is set to the initial system (here $S$). The invariants $I_S$ and $I_T$ of each system are preserved, and when a failure occurs, the failing state preserves invariant $I_S$. *Req2* is thus satisfied.

### E. System substitution as a composition operator

This proposal can be seen as a parameterized (with the $P1$ and $P2$ parameters) system composition written as $Sys_S \underset{P1,P2}{\circ} Sys_T$. It defines the substitution of a system $Sys_S$ by another system $Sys_T$. Let us study the properties of this operator, i.e. the associated proof obligations. First, the events corresponding to $Sys_S$ and $Sys_T$ preserve the invariant because they preserved their respective invariants $I_S$ and $I_T$ in the static substitution. Second, the event `fail` also satisfies the invariant, since no state variable is modified by this event. The `repair` event is the only event concerned by the modification of the variables. It shall maintain the invariant. According to Table I, the associated proof obligation is defined as follows.

$$\begin{array}{l} A(s,c),\ m = S \Rightarrow I_S(s,c,v_S) \wedge m = T \Rightarrow I_T(s,c,v_T) \\ \quad \wedge\, m = F \Rightarrow I_S(s,c,v_S),\ m = F,\ P1(v_S,v_T,v'_S,v'_T) \wedge m' = T \\ \vdash m' = S \Rightarrow I_S(s,c,v'_S) \wedge m' = T \Rightarrow I_T(s,c,v'_T) \wedge m' = F \Rightarrow I_S(s,c,v'_S) \end{array}$$

The proof obligation for the preservation of the invariant in the `repair` event are obtained after simplifications.

$$\boxed{A(s,c) \vdash I_S(s,c,v_S) \wedge P1(v_S,v_T,v'_S,v'_T) \Rightarrow I_T(s,c,v'_T)} \qquad (1)$$

To conclude, the proof obligation corresponding to equation (1) is associated to the composition $Sys_S \underset{P1,P2}{\circ} Sys_T$. This equation defines the proof obligation associated to the substitution pattern we studied.

## IV. A CASE STUDY

This method has been applied to a case study of a basic electronic commerce system for web services compensation. The provided Event-B models are borrowed from [3].

### A. The system

The system enables the purchase of a set of products from a single supplier. A user selects some products in a cart, pays the corresponding fees, receives an invoice and then the products are delivered by the logistics part of the system (see Figure 2).

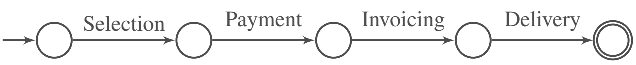


Fig. 2. High level view of the case study system

We suppose that during the *selection* step, a failure occurs due to an error on the supplier side. A failure signal is triggered. It enables the substitution of the supplier website by a new system composed of two other suppliers on two different websites. The two corresponding carts are filled such that the user does not lose or gain any products in the carts (corresponding to an equivalent system substitution case).

### B. The specification

The model encodes the state-transitions system of Figure 2.

*1) The context:* The context `C_1` introduces `STOCKS` a relation associating the available products for each site.

```
Context C_1
Sets
  PRODUCTS // all the products in the world
  SITES   // all the sites in the world
Constants STOCKS
Axioms
  Axm1 : finite(PRODUCTS)
  Axm2 : finite(SITES)
  Axm3 : card(SITES) ≥ 2
  Axm4 : STOCKS = SITES × PRODUCTS
End
```

Listing 5. The context C_1

*2) The top level specification:* corresponds to an Event-B machine (Listing 6) with the events of the state-transitions system of Figure 2. Only the details of the `selection` event are given. It fills a variable `carts` with an arbitrary cart which contains the desired products ($Grd3$), each one of these products being selected on a single site ($Grd4$).

```
Events
  Event initialisation ≙
   Then
    seq := 4
    P :∈ ℙ(PRODUCTS)
```

```
    carts := ∅
  Event selection ≙
    Any someCarts Where
     Grd1 : seq = 4
     Grd2 : someCarts ⊆ SITES × P
     Grd3 : ran(someCarts) = P
     Grd4 : ∀p. p ∈ ran(someCarts)⇒ card(someCarts⁻¹[{p}]) = 1
    Then
       seq := 3
       carts := someCarts
  Event payment ≙ Where Grd1 : seq = 3 Then seq := 2 . . .
  Event billing ≙ Where Grd1 : seq = 2 Then seq := 1 . . .
  Event delivery ≙ Where Grd1 : seq = 1 Then seq := 0 . . .
End
```

Listing 6. The events encoding the activities of the case study of Figure 2

### C. Possible implementations

We define two systems, *WS1* and *WS2*, implementing (refining) this specification. The two refinements of the `selection` event are given in Listings 7 and 8. The first system, *WS1*, uses one single cart and one website. It offers the possibility to add items individually to the cart.

```
Event addItem_WS1 ≙
 Any item
 Where item ∈ P \ ran(cart_WS1)
 Then cart_WS1 := cart_WS1 ∪ {site_1 ↦ item}

Event selection_WS1 ≙
Refines selection
 Where ran(cart_WS1) = P
 Then cart := cart_WS1
```

Listing 7. Refinement of the `selection` event for one site (*WS1*).

The second system, *WS2*, uses two websites with one cart on each website. A user may add items, one by one, to each of the carts, by choosing products on both websites. The cart of the specification is the union of the carts of both websites.

```
Event addItemA_WS2 ≙
 Any item
 Where item ∈ P \ ran(cart^A_WS2 ∪ cart^B_WS2)
 Then cart^A_WS2 := cart^A_WS2 ∪ {site_2A ↦ item}

Event addItemB_WS2 ≙
 Any item
 Where item ∈ P \ ran(cart^A_WS2 ∪ cart^B_WS2)
 Then cart^B_WS2 := cart^B_WS2 ∪ {site_2B ↦ item}

Event selection_WS2 ≙
Refines selection
 Where ran(cart^A_WS2 ∪ cart^B_WS2) = P
 Then cart := cart^A_WS2 ∪ cart^B_WS2
```

Listing 8. Refinement of the `selection` event for two sites (*WS2*).

### D. Failure and substitution

According to the presented methodology, the next step consists in introducing the failure and the substitution events. The basic definitions for failure modes are defined in the context `C_11`.

```
Context C_11 Extends C_1
Sets    FAILURE_MODES
Constants OK, NOK
Axioms
 axm1:partition(FAILURE_MODES, {OK}, {NOK})
End
```

Listing 9. Introduction of a context for failure modes

*1) Introduction of failures:* the `failure_WS1` event introduces failures on the first site *WS1*. It halts the system.

```
Event failure_WS1 ≙
 Where
  Grd1 : sys = WS1
  Grd2 : failureStatus = OK
 Then
  Act1 : failureStatus := NOK
```

Listing 10. Failure event for *WS1*

*2) System substitution:* the correctness of the substitution between the two systems relies on the correct restoration of the carts. Correctness is preserved by the *horizontal invariant* defined as $cart_{WS1} = cart^A_{WS2} \cup cart^B_{WS2}$. It guarantees that the products contained in the cart $cart_{WS1}$ already purchased on *WS1* (one website) are split in the carts $aCart^A_{WS2}$ and $aCart^B_{WS2}$ of *WS2* (two websites).

```
Event Repair_WS1_WS2 ≙
 Any
  aCart^A_WS2,aCart^B_WS2
 Where
  Grd1 : sys = WS1
  Grd2 : failureStatus = NOK
  Grd3 : aCart^A_WS2 ∪ aCart^B_WS2 = cart_WS1
  Grd4 : aCart^A_WS2 ∩ aCart^B_WS2 = ∅
 Then
  Act1 : sys := WS2
  Act2 : failureStatus := OK
  Act3 : cart^A_WS2 := aCart^A_WS2
  Act4 : cart^B_WS2 := aCart^B_WS2
```

Listing 11. The substitution event exploiting the horizontal invariant

The Event-B models we presented are borrowed from [3]. This simple case study will be used to illustrate the generalization of system substitution.

## V. Mathematical Setting for Substitution

The formal development sketched in the previous section shall be conducted every time a substitution case needs to be considered. In this sense, the previous approach provides a correct substitution mechanism, but it is not generic. Neither the development nor the verification processes can be reused. We advocate the use of a generic correct-by-construction approach. The proposed generalization consists in manipulating the described systems where systems become first-order objects manipulated by the Event-B models. States, transitions, invariants, variants, etc. become objects of the proposed model, and the described system behavior conforms to Figure 1.

This proposal first expresses the system substitution strategy at a higher level, and then reuses this development for each specific system substitution. The specific system is obtained by instantiation of the generic model. Instantiation is defined by a particular use of refinement. Specific systems, defining instances, are witnesses of the generic development.

### A. Variables and states

Variables, that represent states, belong to a set $Variables$. Their values are taken in the set $ValueElements$. Variables are associated to their values by the $Valuations \subseteq Variables \rightarrow\!\!\!\!\!\rightarrow \mathbb{P}(ValueElements)$ function.

## B. Initialization and progress

The initialization of the global system selects the first system to run. The `progress` event models the assignment of a new valuation for the system state variables.

## C. Systems

Systems belong to the set $Systems$ of all the systems. A system is a tuple that is defined as $system = \langle variables, variant, invariant, init, progress \rangle$, where:

- $variables$ is a set of variables representing the state of the system: $variables \subseteq Variables$
- $variant$ is a function producing the natural value of the variant from a valuation of the variables: $variant \in Valuations \to \mathbb{N}$
- $invariant$ is a predicate defined on the variables values: $invariant \in Valuations \to BOOL$
- $init$ and $progress$ are two before-after predicates recording the state changes.

## D. Systems substitution relation

System substitution requires the definition of a relation associating the source system states with the target system ones. As defined in equation 2, this relation is given by the definition of an invariant, named *horizontal invariant.*

$$\boxed{\begin{array}{l} \forall S_S, S_T \in Systems. \\ \forall Inv_H(S_S, S_T) \in \\ \qquad\qquad states(S_S) \times states(S_T) \to \text{BOOL}. \\ substitute\_states(S_S, S_T) = \\ \qquad\qquad \{(s_S, s_T) \in states(S_S) \times states(S_T) \mid \\ \qquad\qquad\qquad\qquad Inv_H(S_S, S_T)(s_S, s_T)\} \end{array}} \tag{2}$$

where[2] $states$ is a function returning the possible valuations of a given system: $states \in System \to Valuations$, and $Inv_H$ is a predicate defining the horizontal invariant involving the values of the variables of the source and target systems: $Inv_H \in System^2 \to Valuations^2 \to BOOL$

The invariant $Inv_H$ links the source and target states. It fulfills the role of $P1$ in the proof obligation defined in equation (1). In the generic model, its definition is given by an equivalence relation. Its definition entails the definition of the reparation relation $repair \in Systems^2 \times (Valuations \to BOOL)^2$. It is parameterized by two predicates $\psi$ and $\varphi$.

$$\boxed{\begin{array}{l} \forall S_S, S_T \in Systems. \quad \forall \psi \in states(S_S) \to \text{BOOL}. \\ \forall \varphi \in states(S_T) \to \text{BOOL}. \\ repair(S_S, S_T, \psi, \varphi) = \\ \qquad\qquad \{(s_S, s_T) \in substitute\_states(S_S, S_T) \mid \\ \qquad\qquad\qquad Inv_S(S_S)(s_S) \wedge \psi \Leftrightarrow Inv_S(S_T)(s_T) \wedge \varphi\} \end{array}} \tag{3}$$

where $Inv_S(S_X)(s_X)$ is the value (satisfied or not) of the system invariant of the system $S_X$ in the state $s_X$.

The predicates $\psi$ and $\varphi$ ($\neq false$) define different reparation or substitution modes.

- $\psi = True \wedge \varphi = True$ in the case $S_T$ is an equivalent system substitute,
- $\psi \neq True \wedge \varphi = True$ in the case $S_T$ upgrades $S_S$
- $\psi = True \wedge \varphi \neq True$ in the case $S_T$ degrades $S_S$

[2]If $E$ is a set, then $E^2$ denotes the Cartesian product $E \times E$

## E. Substitution property

The condition to substitute a system $S_S$ by a system $S_T$ is given by the $repairable\_equiv$ predicate characterizing the set of substitute systems.

$$\boxed{\begin{array}{l} repairable\_equiv(S_S) = \\ \qquad\qquad \exists S_T \in Systems \cdot repair(S_S, S_T, True, True) \neq \varnothing \end{array}} \tag{4}$$

According to equation (3), here the predicates $\psi$ and $\varphi$ are set to $True$ in equation (4) to obtain equivalence.

Finally, the generic system of systems setting is given by a graph characterized by the pair $SoS = (Systems, repair)$ where $Systems$ is the set of available systems (nodes) and $repair$ is the relation among the available systems (edges). The obtained graph of systems may be constrained by additional properties. For example, a property could be that each system has at least two substitute systems. This is out of the scope of this contribution.

# VI. An Event-B model for system substitution

The mathematical setting described above has been completely formalized[3] within the Event-B method. This formalization led to the definition of a context `C0` and of two machines `M0` and `M1`, the latter being the refinement of the former.

## A. Required definitions

The context `C0` (Listing 12) implements the theory associated to the system substitution relation. It defines `Systems`, `Variables` and their possible `Valuations`. `Systems` are sets characterizing the potentially available systems involved in a substitution. `States` and `Variables` are manipulated by the defined recovery mechanism. Note the introduction of the `system_of` function returning the system a state belongs to. Moreover, it also defines in `type10` the type of the *horizontal invariant* which associates corresponding repair states in systems. Property `prop8` ensures that this invariant is well-defined on the states to be recovered. The variant expression is accessed by the `fvar_of` function in `fun4` which returns, for a given state, the function which computes the value of the variant, while the `varval_of` function `fun5` returns, for a given state, the value of this variant.

```
CONTEXT C0
SETS Variables, ValueElements
CONSTANTS Valuations, VariablesSets, Systems, Systems_states, system_of,
    HorizontalInvs, varval_of
AXIOMS
    set1: finite (Variables)
    set2: finite (ValueElements)
    type1: Valuations ⊆ Variables ⇸ ℙ (ValueElements)
    type2: VariablesSets ⊆ ℙ (Variables)
    type3: Systems ⊆ VariablesSets × (Valuations ⇸ ℕ)
    type4: Systems_states ⊆ Systems × Valuations
     ...
    type10: HorizontalInvs ∈ (Systems × Systems) ⇸
                         ((Systems_states × Systems_states) → BOOL)
     ...
    prop1: VariablesSets ≠ ∅
    prop2: ∀ v1,v2 · (v1 ∈ VariablesSets ∧ v2 ∈ VariablesSets ∧ v1 ≠ v2)
                          ⇒ v1 ∩ v2 = ∅
    prop3: finite (Systems) ∧ Systems ≠ ∅
    prop4: ∀ vars,f_var · (vars ↦ f_var) ∈ Systems ⇒
             (∀ val · val ∈ Valuations ⇒
```

[3]The complete Event-B development is available on http://babin.perso.enseeiht.fr/r/HASE_2016_Models.pdf

```
                    (val ∈ dom(f_var) ⇔dom(val) = vars))
    prop5: Systems_states ≠ ∅
    prop6: dom(Systems_states) = Systems
    prop7: ∀ sys_st · sys_st ∈ Systems_states ⇒
                  dom(prj2(sys_st))= prj1(prj1(sys_st))
    prop8: ∀ s1,s2,sst1,sst2,b · ((s1↦ s2) ↦ {(sst1↦ sst2) ↦ b}
          ∈ HorizontalInvs )
                   ⇒( s1 = system_of(sst1) ∧s2 = system_of(sst2) )
    ...
    fun1: system_of = (λ syst_st ∈ System_states | prj1(sys_st)))
    ...
    fun4: fvar_of = (λ syst_st ∈ System_states | prj2(prj1(sys_st))))
    fun5: varval_of = (λ syst_st ∈ System_states |
                              fvar_of(sys_st)(prj2(sys_st))))
    ...
END
```

Listing 12. Context C0 containing basic definitions and properties

## B. Systems recovery behavior

The definition of the final obtained model conforms to the system behavior pattern depicted by the transition system of Figure 1.

*1) The top level specification:* The first abstract machine, M0 manipulates systems without considering system states yet. The available_systems and current_system variables define respectively all the available healthy systems for substitution and the current running system.

```
MACHINE M0 SEES C0
VARIABLES
  current_system, current_system_state
INVARIANTS
  type1: available_systems ⊆ Systems
  type2: current_system ∈ Systems
EVENTS
  Event INITIALISATION ≙ ...
  Event Fail ≙ ...
  Event Repair ≙ ...
  Event Complete_failure ≙ ...
END
```

Listing 13. Squeleton of machine M0

This machine only defines system modes and the failure together with the associated reparation. It models the fact that a system fails, is possibly repaired or isolated (complete_failure) in the treatment of the failure.

*2) First refinement:* Machine M1 below refines M0 to define the final complete generic substitution model. It introduces the variables and the states of the manipulated systems through two new variables available_systems_states and current_system_state.

```
MACHINE M1 REFINES M0 SEES C0
VARIABLES
  available_systems, available_system_states
INVARIANTS
  type1: available_systems ⊆ Systems_states
  type2: current_system_state ∈ System_states
  glue1: available_systems = dom(available_system_states)
  glue2: current_system = system_of(current_system_state)
VARIANT
  varval(current_system_state)
EVENTS
  Event INITIALISATION ≙ ...
  Event Fail Refines Fail ≙ ...
  Event Repair Refines Repair ≙
   Any ...
   Where
    ...
    grd9: HorizontalInvs(current_system ↦ next_system)
               (current_system_state ↦ next_system_state)
    ...
   Then
    current_system := ...
    current_system_state := ...
  Event complete_failure Refines complete_failure ≙ ...
  Event progress ≙
   Any new_valuation
   Where
       grd1: current_system ∈ available_systems
       grd2: new_valuation ∈ Valuations
       grd3: dom(new_valuation) =
               dom(valuation_of(current_system_state))
       grd4: fvar_of(current_system_state)(new_valuation)
               < varval_of(current_system_state)
   Then
       act1: current_system_state :=
               system_of(current_system_state) ↦ new_valuation
   End
END
```

Listing 14. Extract of the machine M1

Two important gluing invariants are defined. The first one glue1 denotes that the states of the available systems are effectively states of the available systems and the second one glue2 asserts that the current state is a state of the current running system. Here, the events are refined to handle the notion of system states. This refinement, 1) defines varval(current_system_state) as a variant to record the progress of the running system, 2) introduces the important event progress to record the behavior of the current running system. It defines the next state of the running system and ensures that the variant decreases (grd4), and 3) refines the repair event by choosing the next system and its state. Note that the guard grd9 defined in this event guarantees that the substitute system fulfills the *horizontal invariant* corresponding to the substitution property. Depending on this *horizontal invariant*, the system is substituted in equivalent, degraded or upgraded mode.

# VII. Instantiation with Event-B: by refinement

## A. The instantiation context: the principle

A context C0_instance extending C0 (Listing 12) is defined with concrete values for sets (Variables, ValueElements) and constants (Valuations, VariablesSets, Systems and System_states).

## B. Refinement and witnesses for instantiation: the principle

M1 of the generic model is instantiated by C0_instance context values. It is defined by M2 refining M1. The event progress is itself refined by the events progress_sysXY corresponding to the transitions in the specific system defined in C0_instance. Each transition of the instantiating system refines the progress event of the machine M1. Concrete event variables of M2 and abstract variables of M1 are glued with a witness On Listing 15, the variable new_state is instantiated by the witness new_state_sysA representing a concrete system variable for sysA.

```
Event eventA ≙
Any new_state
Where current_system_ok()
Then state := new_state
```

```
Event eventC Refines eventA ≙
Where current_system = sysA ∧ sysA_ok()
With new_state = new_state_sysA
Then state := new_state_sysA
```

Listing 15. Instantiation through refinement with witness

## VIII. Application to the Case study

The previous approach applies on the defined case study.

### A. The instantiation context: application to the case study

The instantiation context provides concrete values for the deferred sets of the context `C0`. Variables, systems, states, etc. are valued accordingly. Note the presence of the fundamental axiom `axm9` to ensure the correct system substitution. It corresponds to the reparation property introduced in section IV-D2.

```
CONTEXT C0_instance EXTENDS C0
CONSTANTS
  C1, C2a, C2b
  Prod1, Prod2, Prod3, Prod4, Prod5
  Sys1, Sys2
AXIOMS
  axm1: partition(Variables, {C1}, {C2a}, {C2b})
  axm2: partition(ValueElements, {Prod1}, {Prod2}, ... , {Prod5})
  axm3: Valuations = ({C1} → ℙ(ValueElements))
                    ∪({C2a,C2b} → ℙ(ValueElements))
  axm4: VariablesSets = {{C1},{C2a,C2b}}
  axm5: Sys1 = {C1} ↦ (λ val ·val ∈ {C1} → ℙ(ValueElements)
                | card(ValueElements) − card(val(C1)))
  axm6: Sys2 = {C2a,C2b} ↦ (λ val ·val ∈ {C2a,C2b} → ℙ(ValueElements)
                | card(ValueElements) − card(val(C2a) ∪ val(C2b)))
  axm7: Systems = {Sys1,Sys2}
  axm8: Systems_states = Systems × Valuations
  axm9: HorizontalInvs = { (Sys1 ↦ Sys2) ↦
          (λ (sst1 ↦ sst2) ·
                sst1 ∈ {Sys1} × ({C1} → ℙ(ValueElements))
                ∧ sst2 ∈ {Sys2} × ({C2a,C2b} → ℙ(ValueElements)) |
           bool(valuation_of(sst1)(C1)
                = valuation_of(sst2)(C2a) ∪ valuation_of(sst2)(C2b)))}
     ...
END
```

Listing 16. The instantiation context

### B. Use of refinement and witnesses for instantiation : application to the case study

The events of the `M1` machine are refined (by machine `M2`) for instantiation according to the principle of section VII-B.

```
MACHINE M2 REFINES M1 SEES C0_instance
EVENTS
  Event INITIALISATION ≙ ...
  Event failure_sys1 Refines failure ≙ ...
  Event failure_sys2 Refines failure ≙ ...
  Event repair_sys1_to_sys2 Refines repair ≙
    ...
    Where
      sys1_cart = new_sys2_cart1 ∪ new_sys2_cart2
    ...
  Event complete_failure Refines complete_failure ≙ .
  Event progress_sys1 Refines progress ≙ ...
  Event progress_sys2 Refines progress ≙ ...
END
```

Listing 17. The instantiation machine obtained by refinement

For instance, the `progress` event is refined by the `progress_sys1` event describing the progress for system *WS1*. It corresponds to the event `addItem_WS1` of Listing 7. The witness (`with` clause) of the event `progress_sys1` consists in adding a product in the cart `C1` of the website $site_1$ as done in event `addItem_WS1`.

```
  Event progress_sys1 ≙
  REFINES progress
  ANY new_prod
  WHERE
      grd1: current_system = Sys1
      grd2: Sys1 ∈ available_systems
      grd3: new_prod ∈ ValueElements
      grd4: new_prod ∉ sys1_cart
  WITH
      new_valuation: new_valuation = {C1 ↦ (sys1_cart ∪{new_prod})}
  THEN
      act1: sys1_cart := sys1_cart ∪{new_prod}
      act2: current_system_state :=
                Sys1 ↦ {C1 ↦ (sys1_cart ∪{new_prod})}
  END
```

Listing 18. The generic `progress` event of machine `M2`

## IX. Assessment

### A. Proof statistics

TABLE II. RODIN PROOFS STATISTICS

| Event-B Model | Generated proof Obligations | Automatic proofs | Interactive proofs |
|---|---|---|---|
| Context `C0` | 7 | 5 | 2 |
| Machine `M0` | 5 | 5 | 0 |
| Machine `M1` | 28 | 22 | 6 |
| Instantiation context `C0_context` | 3 | 2 | 1 |
| Instantiation machine `M2` | 54 | 39 | 15 |
| Total | 97 | 73 | 24 |

### B. Model checking or proof-based verification

Note that model checking techniques can be applied to automatically check the correctness of the instantiation. State exploration is possible since the sets have finite number of values in the context `C0_instance`. The instantiation mechanism defined above is based on refinement. The approach is scalable and does not face the state explosion problem. The key point for scalability concerns the instantiation of specific systems. Indeed, the development presented above is a generic one, defined at a meta-level, where the proof obligation associated to the correctness of the system substitution obtained in section III-E act as meta-theorem.

| Event of the model | Instantiation by a witness |
|---|---|
| **Event** $evt$ ≙<br>**Any** x<br>**Where**<br>grd1: $G(s,c,v,x)$<br>**Then**<br>act1: $v :\mid BA(s,c,v,x,v')$<br>**End** | **Event** ref_of_evt ≙<br>**Refines** $evt$<br>**Where** grd1: $G(s,c,v,y)$<br>**With** x: $x = y$ // witness<br>**Then**<br>act1: $v :\mid BA(s,c,v,y,v')$<br>**End** |

Listing 19. Proof based instantiation

The use of the $ANY$ generalized substitutions shows that the development considers any transition system described by a template corresponding to Figure 1 together with the associated invariants expressed in the corresponding Event-B models. According to the proof obligation associated to the $ANY$ substitution described in Table I, one proof strategy is to exhibit a witness for the parameter $x$. Two proof techniques, experimented in this paper have been developed. 1) The first one uses model checking with the ProB [4] model checker. We did not give the details of this approach in this paper. 2) The second technique relies on a proof-based approach (can be used if model checking fails) to check instance correctness. Such an approach consists in defining another model which refines the one presented in section V. Like in section VII, each $ANY$ event is refined by an event with a witness for each parameter.

The event refinement strategy is shown in Listing 19. The witnesses can be any transition system matching the pattern of Figure 1 whatever its size is. This second verification technique requires interactive proof efforts and ensures scalability.

### *C. Correct-by-construction formal methods*

The proposed approach is a generic one. The context `C0` describes the manipulated systems concepts (`systems`, `variables`, `HorizontalInvs`, etc. ). The concepts are manipulated as first-order objects in the machines `M0` and `M1` in order to encode the behavior pattern described with the events `Initialization`, `progress`, `fail`, `repair` and `complete_failure`. Let's note that the concept of transition is not manipulated as first order objects and thus not defined within the context `C0`. One may wonder why the transitions between states are not defined in this context `C0`. There are two main reasons for that: first, transitions are not explicitly manipulated by the substitution mechanism introduced in this paper. Second, the Event-B method provides a powerful built-in inductive proof technique based on invariant preservation by the events (see table I). The only proof effort relates to the correct event refinement. Note that in traditional correct-by-construction techniques like Coq [5] or Isabelle [6], classical inductive proof schemes are offered. One has first to describe the inductive structure associated to the formalized systems, then to give a specific inductive proof scheme for this defined inductive structure and finally to prove the correct instantiation. In the kernel definition of these techniques, the inductive process associated to transition systems corresponding to the pattern of Figure 1 and the refinement capability are not available as a built-in inductive proof process. transition together with corresponding inductive proof principles and the instantiation of transitions because event refinement is not available. Unlike Event-B, another meta-level is needed.

## X. Related work

Formal modeling of system reconfiguration has been studied by several authors. Regarding proof and refinement-based methods, the Event-B method has been applied to model fault-tolerance mechanisms in the field of critical multi-agent system by [7]. The authors used refinement at the heart of the approach. State-based formalisms illustrated by Event-B were also exploited by [8] in order to develop fault-tolerant system by modeling fault tolerance requirements, fault assumptions and modes of the system behavior through an additional viewpoints. In [9] Abstract State Machines (ASMs) model a combination of state-transition models with architectural descriptions. Other approaches studied system re-configuration as well. Model checking of timed automata has been used by [10] to model and study the robustness of self-adaptive decentralized systems. Some approaches used process algebras. Analysis of unplanned reconfiguration in dependable systems has also been modeled with process algebra [11]. Behavioral matching between substitute systems was defined by a bisimulation relation. Finally, there exists other approaches, offering an evolving number of systems, to define reconfigurable Byzantine-fault-tolerant distributed system [12].

## XI. Conclusion

This paper addresses the problem of *correct* system reconfiguration, where systems are described as state-transition systems. It provides a stepwise correct-by-construction approach that generalizes ad hoc system reconfigurations. This one relies on 1) the definition of a system that implements (i.e. refine) the same specification and 2) a system reconfiguration operator parameterized by a reparation property, namely a *horizontal invariant*. This one ensures that, when a failure occurs, the state of the source system is correctly restored to the state of the target system. Moreover, the approach is generic and it can be instantiated to any number of systems, thus it ensures scalability. An instantiation mechanism based on the definition of witnesses has been defined. Note that, since instantiation is performed by refinement, solely the last refinement step shall be proved at each instantiation. It corresponds to checking that the witnesses belong to the set of systems. From a methodological point of view, when instantiation by model checking does not scale up, one may use the defined instantiation mechanism based on witnesses. The whole proposed approach has been modeled within the Event-B method. Finally, the approach developed in this paper has been defined for system reconfiguration, but it can be deployed for other cases like redundancy with dissimilar systems or system monitoring.

## References


[1] J.-R. Abrial, *Modeling in Event-B: System and Software Engineering*, 1st ed. New York, NY, USA: Cambridge University Press, 2010.

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

[3] G. Babin, Y. Aït-Ameur, and M. Pantel, "Formal verification of runtime compensation of web service compositions: A refinement and proof based proposal with Event-B," in *Services Computing (SCC), 2015 IEEE International Conference on*, June 2015, pp. 98–105.

[4] J. Bendisposto, J. Clark, I. Dobrikov, P. Karner, S. Krings, L. Ladenberger, M. Leuschel, and D. Plagge, "Prob 2.0 tutorial," in *Proceedings of the 4th Rodin User and Developer Workshop*, ser. TUCS Lecture Notes, 2013.

[5] Y. Bertot and P. Castéran, *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*, ser. Texts in Theoretical Computer Science. Springer Verlag, 2004.

[6] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, ser. Lecture Notes in Computer Science. Springer, 2002, vol. 2283.

[7] I. Pereverzeva, E. Troubitsyna, and L. Laibinis, "A refinement-based approach to developing critical multi-agent systems," *International Journal of Critical Computer-Based Systems*, vol. 4, no. 1, pp. 69–91, 2013.

[8] I. Lopatkin and A. Romanovsky, "Rigorous development of fault-tolerant systems through co-refinement," School of Computing Science, University of Newcastle upon Tyne, Tech. Rep., January 2014.

[9] R. Mirandola, P. Potena, and P. Scandurra, "Adaptation space exploration for service-oriented applications," *Science of Computer Programming*, vol. 80, Part B, pp. 356 – 384, 2014.

[10] M. U. Iftikhar and D. Weyns, "A case study on formal verification of self-adaptive behaviors in a decentralized system," vol. 91, 2012, pp. 45–62.

[11] A. Bhattacharyya, "Formal modelling and analysis of dynamic reconfiguration of dependable systems," Ph.D. dissertation, Newcastle University School of Computing Science, January 2013.

[12] R. Rodrigues, B. Liskov, K. Chen, M. Liskov, and D. Schultz, "Automatic reconfiguration for large-scale reliable storage systems," *Dependable and Secure Computing, IEEE Transactions on*, vol. 9, no. 2, pp. 145–158, March 2012.