Correct Instantiation of a System Reconfiguration Pattern: A Proof and Refinement-Based Approach
Guillaume Babin, Yamine Aït-Ameur, Marc Pantel

To cite this version:
Guillaume Babin, Yamine Aït-Ameur, Marc Pantel. Correct Instantiation of a System Reconfiguration Pattern: A Proof and Refinement-Based Approach. 17th IEEE International Symposium on High Assurance Systems Engineering (HASE 2016), Jan 2016, Orlando, FL, United States. pp.31–38, 10.1109/HASE.2016.47. hal-03155038

HAL Id: hal-03155038
https://hal.archives-ouvertes.fr/hal-03155038
Submitted on 3 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.
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-* 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-B1 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

1http://www.event-b.org/
represent the transitions (defined by a Before-After predicate $BA$) from one state to another (\(\rightarrow\)) for the `$becomes$` operator).

\[
\begin{align*}
\text{Theorems} & : A(s, c) \Rightarrow T_s(s, c, v) \\
\text{Invariant preservation} & : A(s, c) \land T_s(s, c, v) \land T_m(s, c, v) \Rightarrow T_m(s, c, v') \\
\text{Event feasibility} & : A(s, c) \land T(s, c, v) \land T_m(s, c, v) \Rightarrow \exists v' : BA(s, c, v, v') \\
\text{Variant progress} & : A(s, c) \land T(s, c, v) \land BA(s, c, v, v') \Rightarrow V_s(s, c, v') < V(s, c, v)
\end{align*}
\]

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 machine are 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 | $A(s, c) \Rightarrow T_s(s, c)$ |
| Invariant preservation | $A(s, c) \land T(s, c, v) \land T_m(s, c, v) \Rightarrow T_m(s, c, v')$ |
| Event feasibility | $A(s, c) \land T(s, c, v) \land T_m(s, c, v) \Rightarrow \exists v' : BA(s, c, v, v')$ |
| Variant progress | $A(s, c) \land T(s, c, v) \land BA(s, c, v, v') \Rightarrow V_s(s, c, v') < V(s, c, v)$ |

### 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 $Sys_S$.

### 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., $e \in \mathcal{V}$) 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.

### 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.

---

**Fig. 1. Studied system behavior pattern**

**Listing 1. Structures of Event-B contexts and machines**

A model also contains `INVIANTS` and `THEOREMS` that represent their 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.

**Listing 2. An Event-B model for describing a system specification: a context $C_0$ and a machine $Spec$**

**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 (call) and halts the currently running system by setting the variable \( m \) to \( F \); the second one (repair) transfers the control at runtime from \( S_{\text{sys}} \) to \( S_{\text{sys}} \) by 1) setting the variables of \( S_{\text{sys}} \) with values derived from the ones of the interrupted state of \( S_{\text{sys}} \). This assignment is possible and safe thanks to the definition of an horizontal invariant \( P1 \) gluing the state variables of systems \( S_{\text{sys}} \) and \( S_{\text{sys}} \) and 2) initializing the variable for \( S_{\text{sys}} \) using \( P2 \). The variable \( m \) is then set to \( T \) to transfer the control to \( S_{\text{sys}} \).

\[
A(s, c) = I_2(s, c, v) \land P1(v, v, v, v) \Rightarrow I_1(s, c, v)
\]  

To conclude, the proof obligation corresponding to equation (1) is associated to the composition \( S_{\text{sys}} \circ P1P2 S_{\text{sys}} \). 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](image)

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 \( \text{STOCKS} \) a relation associating the available products for each site.

<table>
<thead>
<tr>
<th>Event</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>Payment</td>
</tr>
<tr>
<td>Invoicing</td>
<td>Delivery</td>
</tr>
</tbody>
</table>

![Listing 5. The context C_1](image)

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 \( \text{carts} \) with an arbitrary cart which contains the desired products (\( \text{Grd3} \)), each one of these products being selected on a single site (\( \text{Grd4} \)).

<table>
<thead>
<tr>
<th>Event</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event initialization</td>
<td>End</td>
</tr>
<tr>
<td>seq := 4</td>
<td></td>
</tr>
<tr>
<td>( P \in \mathbb{P} \text{(PRODUCTS)} )</td>
<td></td>
</tr>
</tbody>
</table>
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
Then cartWS1 := cartWS1 ∪ {site1 → item}
End

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(cartWS2)
Then cartWS2 := cartWS2 ∪ {site2 → item}
End

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
End

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 \( cartWS1 = cartWS2 ∪ cartWS2 \). It guarantees that the products contained in the cart \( cartWS1 \) already purchased on WS1 (one website) are split in the carts \( aCartWS2 \) and \( aCartWS2 \) of WS2 (two websites).

```
Event Repair_WS1_WS2
Any
aCartWS2 := cartWS2
Where
Grd1 : sys = WS1
Grd2 : failureStatus = NOK
Grd3 : aCartWS2 ∪ aCartWS2 = cartWS1
Grd4 : aCartWS2 ∪ aCartWS2 = ∅
Then
Act1 : sys := WS2
Act2 : failureStatus := OK
Act3 : cartWS2 := aCartWS2
Act4 : cartWS2 := aCartWS2
End

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 \( Valuations \subseteq Variables × ValueElements \) 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 = (variables, variant, invariant, init, progress), 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 \(\rightarrow\) \(\mathbb{N}\)
- invariant is a predicate defined on the variables values: invariant \(\in\) Invariants \(\rightarrow\) 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.

\[
\text{Inv}(S_\text{S}, S_\text{T}) \subseteq \text{states}(S_\text{S}) \times \text{states}(S_\text{T}) \rightarrow \text{BOOL}
\]

\[
\text{Inv}(S_\text{S}, S_\text{T})(s_\text{S}, s_\text{T}) = \{ (s_\text{S}, s_\text{T}) \in \text{states}(S_\text{S}) \times \text{states}(S_\text{T}) | \text{Inv}(S_\text{S}, S_\text{T})(s_\text{S}, s_\text{T}) \}
\]

where\(^2\) states is a function returning the possible valuations of a given system: states \(\in\) System \(\rightarrow\) Valuations, and Inv\(\text{H}\) is a predicate defining the horizontal invariant involving the values of the variables of the source and target systems: Inv\(\text{H}\) \(\in\) System\(^2\) \(\rightarrow\) Valuations\(^2\) \(\rightarrow\) BOOL\(^2\).

The invariant Inv\(\text{H}\) 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 repair relation repair \(\in\) Systems\(^2\) \(\times\) [Valuations \(\rightarrow\) BOOL\(^2\)]. It is parameterized by two predicates \(\psi\) and \(\varphi\).

\[
\text{Inv}(S_\text{S}, S_\text{T})(s_\text{X}) \in \text{states}(S_\text{X}) \rightarrow \text{BOOL}
\]

\[
\text{Inv}(S_\text{S}, S_\text{T})(s_\text{X}) = \{ (s_\text{S}, s_\text{T}) \in \text{substitute_states}(S_\text{S}, S_\text{T}) | \text{Inv}(S_\text{S}, S_\text{T})(s_\text{S}, s_\text{T}) \}
\]

where Inv\(\text{S}\)(\(s_\text{X}\)) is the value (satisfied or not) of the system invariant of the system \(S_\text{X}\) in the state \(s_\text{X}\).

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

- \(\psi = \text{True} \land \varphi = \text{True}\) in the case \(S_\text{T}\) is an equivalent system substitute,
- \(\psi = \text{False} \land \varphi = \text{True}\) in the case \(S_\text{T}\) upgrades \(S_\text{S}\)
- \(\psi = \text{True} \land \varphi = \text{False}\) in the case \(S_\text{T}\) degrades \(S_\text{S}\)

E. Substitution property

The condition to substitute a system \(S_\text{S}\) by a system \(S_\text{T}\) is given by the repairable_equiv predicate characterizing the set of substitute systems.

\[
\text{repairable_equiv}(S_\text{S}) = \exists S_\text{T} \in \text{Systems} . \text{repair}(S_\text{S}, S_\text{T}, \text{True}, \text{True}) \neq \emptyset
\]

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

Finally, the generic system of systems setting is given by a graph characterized by the pair \(\text{SoS} = (\text{Systems}, \text{repair})\) 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 \(C_0\) and of two machines \(M_0\) and \(M_1\), the latter being the refinement of the former.

A. Required definitions

The context \(C_0\) (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 \(\text{type10}\) 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 valvar_of function fun5 returns, for a given state, the value of this variant.

\(^2\)If \(E\) is a set, then \(E^2\) denotes the Cartesian product \(E \times E\)

\(^3\)The complete Event-B development is available on http://babin.perso.enseeiht.fr/HASE_2016_Models.pdf
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.

Listing 12. Context C0 containing basic definitions and properties

```
MACHINE M0 SEES C0
VARIABLES
  available_systems, current_system
INVARIANTS
  available_systems ⊆ Systems
  current_system ⊆ Systems
EVENTS
  Event INITIALIZATION
  Event Repair
  Event Complete_failure
END
```

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.

Listing 13. Skeleton of machine M0

```
(\{val ∈ dom(f_var) ⇔ dom(val) = var\})
prop6: Systems_states ≠ ∅
prop7: dom(Systems_states) = Systems
prop8: ∀ s1, s, sst ∈ Systems_states ⇒
  dom(\{sys.at = s | sys ∈ Systems_states\}) = Systems_states
prop9: (s1 = system_of(sst1)) ∧ (s2 = system_of(sst2)) ⇒
  HorizontalInv \{ HorizontalInv \}
  ⇒ \{(s1 = system_of(sst1)) ∧ (s2 = system_of(sst2)) \}
fun1: system_of = (\lambda sys.at ∈ Systems_states \mapsto system_of(sys.at))
fun4: varval_of = (\lambda sys.at ∈ Systems_states \mapsto varval_of(sys.at))
fun5: varval_of = (\lambda sys.at ∈ Systems_states \mapsto varval_of(sys.at))
```

Listing 14. Extract of the machine M1

```
VII. INSTANCE 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, Valuations, Values, 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.

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 $C_0$. 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.

<table>
<thead>
<tr>
<th>CONTEXT $C_0$</th>
<th>EXTENDS $C_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Prod1$, $Prod2$, $Prod3$, $Prod4$, $Prod5$</td>
<td>$Sys1$, $Sys2$</td>
</tr>
</tbody>
</table>

**AXIOMS**

$axm1$: partition($\{C_1\}$, $\{C_{2a}$, $C_{2b}\}$) $axm2$: partition($\{\text{Values}\}$, $\{\text{Prod1}, \text{Prod2}\}$, $\ldots$, $\{\text{Prod45}\}$) $axm3$: $\text{Valuations} = \{C_1\} \rightarrow F(\text{Values})$ $axm4$: $\text{VariablesSets} = \{\text{C1}, \text{C2a}, \text{C2b}\}$ $axm5$: $\text{Sys1} = (\lambda \text{val} \in \{C_1\} \rightarrow F(\text{Values})) (\text{C2a}, \text{C2b})$ $axm6$: $\text{Sys2} = (\lambda \text{val} \in \{\text{C2a}, \text{C2b}\} \rightarrow F(\text{Values}))$ $\text{Card}(\text{VariablesSets}) = \text{Card}(\text{val}(\text{C1}))$ $\text{Card}(\text{val}(\text{C2a}))$ $\text{Card}(\text{val}(\text{C2b}))$

**Listing 16.** The instantiation context

| END |

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

The events of the $M_1$ machine are refined (by machine $M_2$) for instantiation according to the principle of section VII-B.

**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 $C_1$ of the website site1 as done in event `addItem_WS1`.

```
MACHINE M2 REFINES M1 SEEDS C0_instance
EVENTS
  Event INITIALISATION $\triangleleft$
  Event_failure_sys1 $\triangleleft$
  Event_failure_sys2 $\triangleleft$
  Event_failure_systo_sys2 $\triangleleft$
  Event_complete_failure $\triangleleft$
  Event_progress_sys1 $\triangleleft$
  Event_progress_sys2 $\triangleleft$

WHERE
  $sys1\_cart = new\_sys1\_cart1 \cup new\_sys2\_cart2$
  $current\_system = Sys1$
  $available\_systems$
END
```

| END |

IX. ASSESSMENT

A. Proof statistics

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.

**Listing 18.** The generic progress event of machine $M2$

```
BEGIN
  new_valuation = new_valuation $\cap (C1 \Rightarrow (sys1\_cart \cup \{new\_prod\}))$
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 \( C_0 \) describes the manipulated systems concepts (systems, variables, HorizontalInv, etc.). The concepts are manipulated as first-order objects in the machine \( M_0 \) and \( M_1 \) in order to encode the behavior pattern described with the events Initialization, progress, fall, 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 \( C_0 \). One may wonder why the transitions between states are not defined in this context \( C_0 \). 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 deductive proof technique based on inductive preservation by the events (see table 1). 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 algebra. 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 sys-