A Formal Guidance Approach for Correct Process Configuration

Souha Boubaker[1(✉)], Amel Mammar[1], Mohamed Graiet[2], and Walid Gaaloul[1]

[1] SAMOVAR, Telecom SudParis, CNRS, Universite Paris-Saclay, Évry, France
{souha.boubaker,amel.mammar,walid.gaaloul}@telecom-sudparis.eu
[2] ISIMM, Monastir University, Monastir, Tunisia
mohamed.graiet@imag.fr

Abstract. Configurable process models are recently gaining momentum as a basis for process *design by reuse*. Such models are designed in a generic manner to group common and variable parts of similar processes. Since these processes are usually large and complex, their configuration becomes manifestly a difficult task. This is why, an increasing attention is being paid to help achieving the process models configuration in a correct and domain-compliant manner. In this work, we propose an Event-B based formal approach that guides the process analyst to easily derive *correct* process variants while considering business domain constraints provided by *configuration guidelines*. To show the effectiveness of our approach, we conduct experiments on a case study.

Keywords: Business process management · Configurable process model · Process variants · Formal verification · Event-B

1 Introduction

Configurable process models are increasingly adopted by companies due to their capability of grouping the common and variable parts of similar processes. According to a specific business need, process models are configured and customized by selecting one design option for each configurable element. The obtained processes are called *variants*. Several approaches have been proposed for the aim of process variants configuration [10,14,20]. A number of them have attempted to help achieving this in a correct manner [2,3]. One of the most important correctness criterion for Business Process Management (BPM) is the soundness property [2]. Hence, the configurable process model should respect a set of structural constraints (e.g. no isolated activities or dead flows). Therefore, a design-time verification should be applied, since the initial configurable processes should be correct. In addition, the resulting derived variants should fulfill a set of behavioral constraints to avoid issues such as deadlock and lack of synchronization.

Furthermore, *configuration guidelines* describing business domain constraints have been introduced [19,20] in order to further limit the configuration decisions.

Q.Z. Sheng et al. (Eds.): ICSOC 2016, LNCS 9936, pp. 483–498, 2016.
DOI: 10.1007/978-3-319-46295-0_30

These guidelines denote the best practices in a given domain. While several approaches have attempted to provide guidance to analysts in selecting configuration choices according to a specific domain requirements [8,17], these configuration choices are most often applied *manually* leaving the designer the full responsibility for applying correct ones. Thus, the correctness of resulting variants is most often difficult to preserve even if domain constraints are respected.

This work provides a systematic approach to guide the process analyst to easily configure process models while, not only preserving correctness, but also complying with domain requirements. The approach targets to answer two questions: (1) Is a configurable process model **correct**? (2) Which configuration choices analyst should take to obtain a **correct** variant **with respect to a specific domain constraints**? To do so, we define an Event-B based formal approach allowing first to analyze and check the correctness of a configurable process (*Objective 1*) and to produce correct variants (*Objective 2*). Configuration guidelines rules are also injected to our formal model to ensure that the obtained variants comply with their domain constraints (*Objective 3*). In essence, we formally define and verify constraints related to (i) structural and behavioral correctness properties; and (ii) domain-based configuration guidelines. Then, we use Event-B tools to perform an incremental verification by checking these constraints at each intermediate step of the configuration procedure.

The remainder of this paper is organized as follows. In Sect. 2, we motivate our approach using an example, used also to illustrate our contribution. We present the related work in Sect. 3. Then, we present basic concepts of Event-B method in Sect. 4. In Sect. 5, we give an overview of our approach. Section 6 illustrates our formalization of process configuration and its corresponding constraints using Event-B. The approach verification and validation using the RODIN tool are depicted in Sect. 7. In Sect. 8, our approach is evaluated using a case study. Finally, we conclude and provide insights for future work.

2 Motivation and Requirements

configurable business process integrates multiple process variants of a same business process in a given domain through variation points. These points are referred to as *configurable elements*. The configuration decision of a configurable element is made at design-time [20]. The non-configurable elements represent the commonalities in the configurable model.

Motivating Example. An example of a configurable process model for hotel reservation and car rental agency is captured by Fig. 1. This agency has many branches in different cities and countries. Each branch performs one variant of this process model which may differ in terms of its structure and behavior according to its specific needs. The customer first submits a request through a web form (*a1*). Next, five main functionalities are proposed: (1) the recommendation, i.e. the process fragment starting from *ops2* and ending with *a6*; (2) hotels and cars searching, i.e. the process fragment starting from *ops5* and

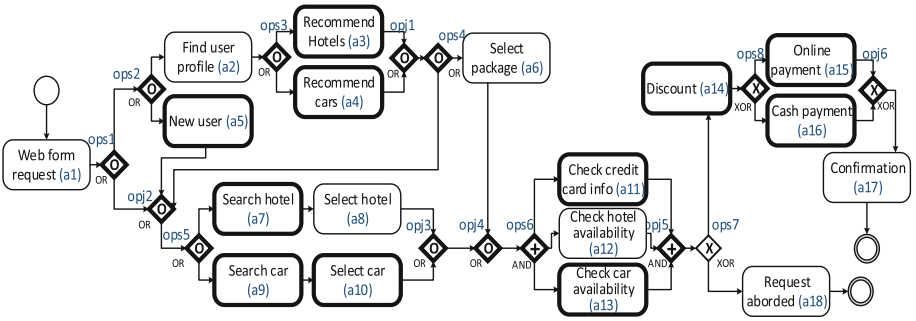


Fig. 1. A configurable hotel and car reservation process model

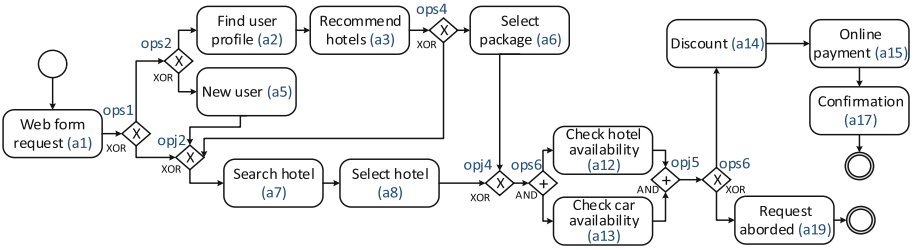


Fig. 2. A process variant derived from the configurable process in Fig. 1

ending with *opj3*; (3) checking phase, i.e. the process fragment starting from *ops6* and ending with *opj5*; (4) discount offer (*a14*); and, (5) payment, i.e. the process fragment starting from *ops8* and ending with *opj6*.

The process is modeled using the Configurable Business Process Model and Notation (C-BPMN) [7], a configurable extension to BPMN. We consider four main control flow elements: activity (represented with a rectangle), edge (control flow edges represented with arrows), event (represented with a circle) and connector (represented with a diamond). Three main connectors, OR (◯), exclusive OR (×) and AND (+) are used to model the splits (e.g. *ops1*) and joins (e.g. *opj1*). C-BPMN includes two configurable elements: activities and connectors. This example presents 24 configurable elements (13 connectors and 11 activities) which are highlighted with a thicker border. For instance, activities *a1* and *a18* are non-configurable, so they should be included in every configured variant. Whereas, the activity *a11* and the connector *ops1* may vary from one process to another, as they are configurable.

A connector may be configurable to restrict its behavior by (i) changing its type (e.g. from OR to AND), or/and (ii) restricting its incoming or outgoing branches. A connector may change its type according to a set of configuration constraints [20] (see Table 1). Each row corresponds to the initial type that can be

Table 1. Constraints for the configuration of connectors [20]

FROM-TO	OR	XOR	AND	seq
OR	√	√	√	√
XOR		√		√
AND			√	

mapped to one or more types in columns. For example, an *OR* type can be configured to any type while an *AND* remains unchangeable. The connector *AND* should never be configured to a sequence. Figure 2 shows an example of a process variant derived from the configurable process of Fig. 1. In this variant, the analyst does not need neither the recommendation functionality for cars (*a4*) nor the option to rent a car (*a9* and *a10*). This refers to configuring *ops3* to a sequence starting from *a3* (i.e. the outgoing branch of *ops3* starting from *a4* is removed) and configuring *ops5* to a sequence starting with *a7* (i.e. the outgoing branch of *ops5* starting from *a9* is removed).

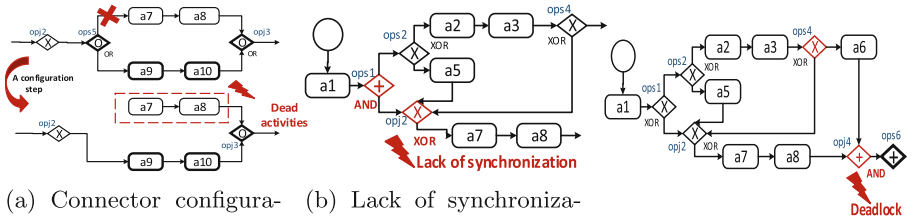
Also, a configurable activity may be needed in a process variant and not in another depending on specific requirements. Hence, these activities can be included (i.e. ON) or excluded (i.e. OFF) from the model [20]. In the process variant of Fig. 2, the analyst does not need the card checking functionality but does need the creation of a new user and the discount functionalities. This refers to configuring *a11* to OFF and *a5* and *a14* to ON: *a11* is removed in the resulting variant whereas *a5* and *a14* are kept.

Configuration Guidelines. To comply with specific domain business needs, the process analyst needs further guidelines to derive specific variants. *Configuration guidelines* provide recommendations and proposed best practices for a specific domain [8,20]. An example of such guidelines satisfied by the variant of Fig. 2 is: *"if the hotel recommendation functionality is included (i.e. a3) in the derived variant, then the hotel searching functionality (i.e. a7) should be also included."* These guidelines are expressed in the form of logical *If-Then*-rules where the if and then parts contain configurations of different configurable elements.

Correctness Checking. In Fig. 3a, *ops5* has been configured to a sequence starting from *a9* (the edge between *ops5* and *a7* disappears). Thus, the produced process is not sound, since activities *a7* and *a8* become unreachable from the initial event: they are dead as they can never be executed. In this paper, we aim at preventing these configurations by formally ensuring that every connector configuration involving outgoing or incoming branches restriction is implicitly followed by a transformation phase allowing to remove the isolated activities from the resulting process. Thereafter, an isolated node is either unreachable from the initial event, or are not on a sequence leading to a final event.

Besides checking structural correctness of the configurable process model, we aim to also ensure the behavioral correctness of the derived process variants [2]. Since processes may be complex with a large number of inter-dependencies between the different configuration alternatives, configuring a process model becomes a quite difficult task. Therefore, analysts may easily be mistaken in their choices which undermine the correctness of the resulting variant. In the following, we illustrate some soundness problems [22] that would happen during the process configuration resulting from mismatches between splits and joins.

– In Fig. 3b, the join operator *opj2* has been configured to an XOR while the connector *ops1* had been already configured to an AND-Split. The two outgoing branches from the AND-split will be activated, however, the XOR-join needs the completion of exactly one of its incoming branches. This leads to an improper termination of the process.
– In Fig. 3c, the connector *ops4* has been configured to an XOR-Split and the corresponding join *opj4* to an AND-join. This implies a deadlock, as only one branch is activated after the XOR-join, whereas the AND-join needs the completion of all its incoming branches.



(a) Connector configuration to a sequence (b) Lack of synchronization (c) Deadlock

Fig. 3. Examples of configuration mistakes from the configurable process in Fig. 1

In view of these situations, the process configuration options should be evaluated with respect to all configuration soundness constraints as well as configuration guidelines to derive correct variants. Hence, our contribution consists not only in proving the correctness of the process configuration steps but also in guiding analyst choices with respect to these constraints using the ProB model checker.

3 Related Work

Several approaches have been proposed to model variability in configurable process models by restricting its behavior through configurable nodes [10,14,20]. A number of them tried to ensure the process configuration correctness [19]. Mainly, two criteria related to process variants correctness were defined: structural and behavioral correctness.

Table 2 provides a comparative overview of some configuration approaches in light of our evaluation criteria (inspired from [19]). We further decompose the *Correctness Support* column into three sub-criteria: (i) *structural* correctness, (ii) *soundness* and (iii) compliance with *domain*-specific configuration guidelines.

In [1,2,20], Petri net was used to formalize and verify correctness and soundness properties of Configurable EPC (C-EPC) processes. They highlight the soundness property and they derive propositional logic constraints that guarantee the behavioral correctness of the configured model. The approach in [3] used partner synthesis. Authors in [13] discuss ensuring soundness of variant models

in the Provop framework which extends the process variants by *options*[14]. In [21], authors use CoSeNets (Configurable Service Nets) that allow to achieve correctness because of their no cycles syntactic restrictions. However, even if these proposals try to achieve configuration correctness, they nevertheless often lack the necessary guidance to become adaptable to a given domain and do not support the BPMN notation.

Existing approaches for assisting process analysts in selecting desirable configuration choices according to specific domain requirements are most often manual. The questionnaire-based approach in [17] offers guidance for configuring process models using a set of questions defined by domain experts and answered by designers. Authors in [16] introduce the use of configuration rules in order to configure a reference process template. In [6,11], authors propose a feature-oriented approach based on feature models to represent variability. Moreover, in [20], the notion of configuration guideline is introduced in order to meet specific domain requirements. Authors in [8] attempted to use configuration guidelines for assisting analysts in BPMN configuration. However, they do not consider any correctness criterion.

In our previous work [9], an Event-B based approach for deriving correct process variants was proposed. The current work strengthens this previous one by the integration of configuration guidelines, and conducting experiments with a group of users allowing to prove the approach usefulness. Thus, our approach intends to meet the defined criteria in Table 2. First, we consider three levels of verification: structural correctness, soundness and compliance with domain-specific configuration guidelines. These three issues were usually discussed individually. Then, we define a systematic formal approach for guiding analyst in deriving correct and domain-compliant BPMN variants using a step-based Event-B model animated by the ProB tool. Event-B is of special interest, since it supports incremental verification allowing to produce a correct specification by construction by proving the different constraints of the model at each step.

Table 2. Evaluation of related configuration approaches

Approaches	Criteria						
	Process modeling language	Correctness support			Guidance support	Incremental verification	Formal specification
		Structural	Soundness	Domain			
[1–3, 20]	C-EPC	+	+	+	−	−	+
[13]	Block-structured	+	+	−	−	−	±
[21]	CoSeNets	+	±	−	−	−	±
[16]	Block-structured	+	+	−	±	−	±
[6, 11]	Block-structured	+	+	−	±	−	+
[17]	C-EPC	±	−	+	+	−	+
[8]	C-BPMN	−	−	+	+	−	−
Our approach	C-BPMN	+	+	+	+	+	+

4 The Event-B Method

Event-B [4] is a state-based formal method for modeling and analyzing systems. It is based on classical logic and set theory. An Event-B model uses two types of components to describe a system: machines and contexts. A machine contains dynamic elements that describe the state of the system, which are variables v and events E to describe the behavior of a system. Variables are constrained by invariants *I(v)*, which are supposed to hold whenever the state of the system change. Whereas a context represents the static part of the model, consisting of sets, constants, and axioms that specifies their properties. Machines can be linked to each other by a *refinement* relation. To have access to its elements, a context is seen by a machine and its refinements. A context may be also extended by another to introduce more elements.

An event takes the form: evt $\triangleq$ **any** x **where** G **then** Act **end**; where x is the list of event parameters, G denotes a conjoined list of predicates defining the guard that are the necessary conditions for the event to occur. An action Act is a simple assignment to a state variable to describe the consequence of the event occurrence. In this paper, we restrict ourselves to the deterministic assignment, the becomes equal substitution, denoted by $(x := e)$.

To cope with the complexity of a system, Event-B defines several abstraction levels, using refinements, in order to gradually introduce the different elements of a system. A step wise refinement approach produces a *correct specification by construction* since we prove the different properties of the system at each step. Event-B is supported by the eclipse based RODIN platform [5] on which different external tools (provers, animators, model-checkers) can be plugged in order to animate/validate a formal development.

5 Approach Overview

This section gives an overview of our contribution detailed in the next sections. Figure 4 depicts the configuration procedure allowing to obtain correct process variants using Event-B as a formal method. Basically, we defined two abstraction levels: the first level introduces our model for process model configuration allowing to preserve correctness (machine M0, see Sect. 6.1). In this machine, configuration steps and their correctness are ensured by events and invariants (see Sect. 6.2). Next, configuration guidelines are formally integrated to our model in the second abstraction level as model refinement of the first level (machine M1, see Sect. 6.3). Event-B defines proof obligations to guarantee that the invariants are preserved by all events (see Sect. 7.1).

First of all, the correctness of the configurable process is verified with respect to the different invariants. The configuration can start only if all the invariants are verified. This allows to achieve our *objective 1* (defined in the Introduction). Then, the analyst uses ProB animator [18] to perform configuration steps involving each element configuration (see Sect. 7.2): firstly, the guards of each event are evaluated (step 1). These guards include both correctness and domain constraints. Then, only events whose guards are verified are enabled (step 2). Thus, the configuration step can be applied (step 3).

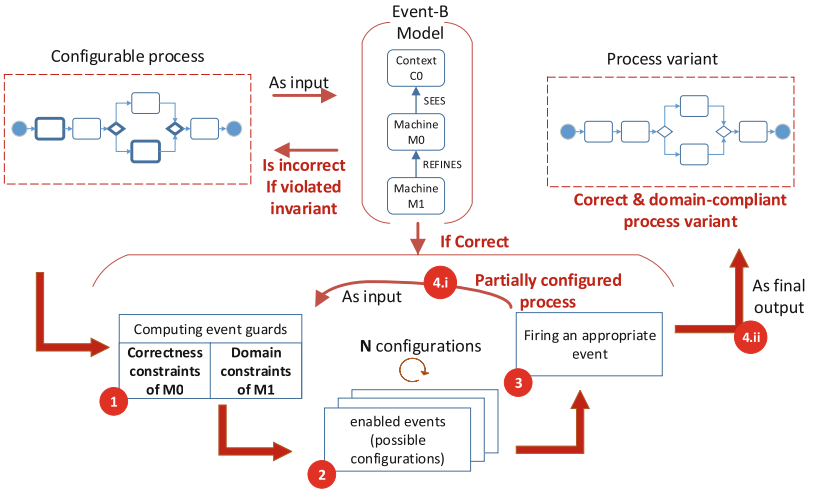


Fig. 4. Our approach overview

The set of potential configuration options is updated after each step. These steps are repeated (step 4.i) while there are configurable elements. As a result, the analyst derives a correct and domain-compliant process variant (step 4.ii), satisfying our *objectives 2* and *3*.

6 Event-B Formal Modeling of Process Configuration

In this section, we use the Event-B method to formally specify our configuration approach. Based on a correct Configurable process model (see Sect. 6.1), we define configuration steps (see Sect. 6.2) allowing to derive *correct* process variants with respect to a set of *configuration guidelines* (see Sect. 6.3). Thereafter, the analyst is able to generate correct variants thanks to this model. Due to lack of space, we outline in this paper the basic elements of our formalization[1].

6.1 Formalizing Configurable Process Models

We start by presenting the context *C0* which holds the following finite sets: (i) *BPS*, which defines the set of possible processes, (ii) *NODES*, which contains three values denoting types of nodes: activities (i.e. *ACTS*), split connectors (i.e. *CON_S*), and join connectors (i.e. *CON_J*), and (iii) *TYPES*, which defines three types of connectors: *OR*, *XOR* and *AND*.

[1] The complete Event-B model can be downloaded from our web page http://www-inf.it-sudparis.eu/SIMBAD/tools/GuideBPMEventB.

Then, we define the machine *M0* which sees the context *C0* described above. The variables of *M0* and their typing invariants are given in Listing 1. We define a variable *BP* to store the created processes. To map each process to its nodes, we introduce the relation *BP_Nodes* from *BP* to *NODES* (*Inv2*). We define the start and the end events as activities using respectively the total function *Initial* (*Inv3*) and the relation *Final* (*Inv4*); since we assume that a process has exactly one initial activity but may have several final ones. In BPMN, each connector, either a split or a join, has a type. This is modeled using the total function *CON_Type* (*Inv5*). The control flow perspective describes activities and their execution ordering through different constructors [15]. This execution order is modeled using the total function *SEQ* (*Inv6*). Finally, we define a total *Boolean* function *Configurable_Nodes* (*Inv7*) to state whether a given node is configurable or not in each process in which it appears.

Listing 1. M0's variables and typing invariants

```
MACHINE M0 SEES C0
VARIABLES BP BP_Nodes Initial Final SEQ CON_Type
INVARIANTS
Inv1 : BP ⊆ BPS
Inv2 : BP_Nodes ∈ BP ↔ NODES
Inv3 : Initial ∈ BP → ACTS
Inv4 : Final ∈ BP ↔ ACTS ∧ dom(Final) = BP
Inv5 : CON_Type ∈ P_Nodes ▷ (CON_S ∪ CON_J) → TYPES
Inv6 : SEQ ∈ BP → (NODES ↔ NODES)
Inv7 : Configurable_Nodes ∈ BP_Nodes → BOOL
```

Structural Constraints. To ensure consistent and structurally correct process control flow, we define a set of constraints to be respected. We illustrate some of them in Listing 2: (i) except the initial and the final nodes, each activity have exactly one outgoing (*Inv11*[2]); (ii) a split connector has at least two outgoings arcs (*Inv14*); and (iii) a join connector has exactly one outgoing arc (*Inv15*).

Listing 2. Structural constraints invariants

```
....
Inv11 : ∀bp.(bp ∈ BP ⇒ (ACTS ◁ SEQ(bp)) ∈ ACTS ∩ BP_Nodes[{bp}] \ Final[{bp}]
                    → BP_Nodes[{bp}] \ Initial[{bp}]
...
Inv14 : ∀bp, nd.(bp ∈ BP ∧ nd ∈ CON_S ∧ bp ↦ nd ∈ BP_Nodes ⇒ card(SEQ(bp)[{nd}]) ≥ 2)
Inv15 : ∀bp.(bp ∈ BP ⇒ CON_J ◁ SEQ(bp) ∈ CON_J ∩ BP_Nodes[{bp}] → NODES) ...
```

A process is considered to be *sound* if it fulfills the following two conditions: (1) all nodes of the process can be activated, i.e. every node can be reached from the initial activity, as depicted by *Inv20* in Listing 3 where *cls*[3] is the transitive closure of a relation; and (2) for each activity in the process, there is at least one possible path leading from this activity to a final activity, i.e. the termination is always possible. This condition is captured by *Inv21* of Listing 3.

Listing 3. Soundness constraints invariants

```
Inv20 : ∀bp, node.(bp ↦ node ∈ BP_Nodes ∧ node ≠ Initial(bp) ⇒ node ∈ (cls(SEQ(bp)))[{Initial(bp)}])
Inv21 : ∀bp, node.(bp ↦ node ∈ BP_Nodes ∧ node ∉ Final[{bp}]⇒
      (cls(SEQ(bp)))[{node}] ∩ Final[{bp}] ≠ ∅)
```

[2] $A \lhd f$ denotes a domain restriction: $A \lhd f = \{x \mapsto y | x \mapsto y \in f \wedge x \in A\}$.

[3] $cls(r)$ denotes the closure of the relation r defined, for each relation $(r \in S \leftrightarrow S)$, by: (1) $cls(r) = \bigcup_{i=1..\infty} r^i$; (2) $r^1 = r$; and (3) for each $n >= 2 r^n = (r; r^{n-1})$ The transitive closure formulations were expressed as machine theorems.

Behavioral Constraints. The configuration of a business process model may affect the soundness by two types of potential errors: *lack of synchronization* and *deadlocks* [22]. These situations result from a mismatch between splits and joins. To formally prevent these situations during configuration procedure, we defined six invariants: three for the splits and three for the joins. These invariants should be preserved by all the events defined to capture configuration operations. For instance, the lack of synchronization could be captured by joining AND-split flows with XOR-join flows (see Fig. 3b). Thanks to *Inv22* (Listing 4[4]), this situation is not allowed in our model. Specifically, having a AND-split operator *ops* (line 2), for each couple of outgoing nodes *n1* and *n2* (lines 3 and 4), the first common node[5] *opj* (lines 4 to 7) should be an AND or a not yet configured OR connector that should be eventually configured as an AND (lines 9 and 10). Similar invariants are defined to ensure a deadlock-free control flow.

Listing 4. Synchronization invariant

```
1 Inv22:∀bp, ops, n1, n2.(bp ↦ ops ∈ BP_Nodes ▷ CON_S
2  ∧ CON_Type(bp ↦ ops) = AND
3  ∧ n1 ∈ SEQ(bp)[{ops}]∧
4   n2 ∈ SEQ(bp)[{ops}] ∧ n1 ≠ n2
5⇒ (∀opj.opj ∈(∪t.t ∈ ((cls(SEQ(bp)))[{n1}] ∪ {n1})
6    ∩ ((cls(SEQ(bp)))[{n2}] ∪ {n2}) ∧
7   SEQ(bp) ~ [{t}] ∩ (((cls(SEQ(bp)))[{n1}] ∪ {n1})
8    ∩ ((cls(SEQ(bp)))[{n2}] ∪ {n2})) = ∅ | {t})
9  ⇒ (CON_Type(bp ↦ opj) = AND ∨
10  (CON_Type(bp ↦ opj) = OR ∧
11  Configurable_Nodes(bp ↦ opj) = TRUE)) ))
```

6.2 Formalizing Configuration Steps

In this section, we describe the formal modeling of the configurable elements: *activity*, and *connector*. In this formalization, each configuration step is performed by an appropriate event. In order to derive correct variants, we define a set of constraints using invariants and we prove that each event preserves them.

Activity Configuration. A configurable activity could be included or excluded in a process variant according to the analyst choice. To define this activity configuration, two invariants and two events are introduced.

With regard to events, activity configuration is performed through either: (i) *ConfigureACTON* event which keeps the activity; or (ii) *ConfigureACTOFF* event which excludes it. We present in Listing 5 the *ConfigureACTOFF* event. Based on a configurable process *bp1*, a configured process *bp2* is a result of excluding an activity *act*. As guard, *act* must be configurable (*grd3*). This event allows *bp2* to inherit from *bp1*: (i) its nodes whilst removing *act* (*act2*), (ii) its initial and final activities (*act3* and *act4*), (iii) all its nodes relations (i.e. SEQ(bp1)) while removing *act* dependencies and creating a new one connecting *act* successor and predecessor (*act5*), (iv) its configurable nodes (*act6*), and (iiv) types of its connectors. Finally, we define *bp2* as a configuration of *bp1* whilst excluding *act* (*act8*) [9]. Similarly, the event *ConfigureACTON* allows to maintain the same process by keeping the configurable activity[6].

[4] The inverse of a function f, (f^{-1}), is denoted in Event-B as ($f\sim$).

[5] Having two nodes n1 and n2, the *first common node* is the first node which belongs to the transitive closure of both nodes n1 and n2.

[6] More details can be found in http://www-inf.it-sudparis.eu/SIMBAD/tools/GuideBPMEventB.

Listing 5. Excluding activity event

```
ConfigureACTOFF ≙   ANY    bp1   bp2   act
WHERE
grd1: bp1 ∈ BP ∧ act ∈ ACTS ∧ bp1 ↦ act ∈ BP_Nodes ∧ bp2 ∈ BPS \ BP
grd2: Configurable_Nodes(bp1 ↦ act) = TRUE
  ...
THEN
act1: BP := BP ∪ {bp2}
act2: BP_Nodes := BP_Nodes ∪ ({bp2} × (BP_Nodes[{bp1}] \ {act}))
act3: Initial(bp2) := Initial(bp1)
act4: Final := Final ∪ ({bp2} × (Final[{bp1}]))
act5: SEQ(bp2) := (({act} ⩤ SEQ(bp1)) ⩥ {act}) ∪((SEQ(bp1)) ~ [{act}] × SEQ(bp1)[{act}])
act6: Configurable_Nodes := Configurable_Nodes ∪ (⊔ node.node ∈ BP_Nodes[{bp1}] \ {act}
      | {bp2 ↦ node ↦ Configurable_Nodes(bp1 ↦ node)})
act7: CON_Type := CON_Type ∪ (⊔ con.con ∈ BP_Nodes[{bp1}] ∩ (CON_S ∪ CON_J)
      | {bp2 ↦ con ↦ CON_Type(bp1 ↦ con)})
act8: Is_Configuration_OFFAct := Is_Configuration_OFFAct ∪ {bp2 ↦ bp1}
```

Connector Configuration. A connector configuration has to consider the following requirements: (1) the configuration constraints for each type of connector, (2) only configurable nodes can be removed, and (3) the connectors types matching checking in order to prevent erroneous situations.

Concretely, in order to obtain a well-structured configured process, an invariant for each configuration choice should be respected. These configuration choices are insured by two events (either split or join) for each connector type. For instance, the event *ConfigureORSplit* allows configuring a configurable split connector from OR type to any type (according to Table 1) while preserving the number of branches greater than two. Recall that each branch can be removed only if all its nodes are configurable. Furthermore, all events should not lead to deadlock or lack of synchronization: for example, for every pair of outgoing branches if the corresponding join is an AND, then the split should be configured to an AND as well. More details about this event and other similar ones (e.g. *ConfigureORJoin*) are given in [9].

Finally, a connector can be configured to a sequence *Seq(N)* by keeping a single branch starting by the node *N*. This is modeled using the event *ConfigureCONSToSeq* for a split connector (resp. *ConfigureCONJToSeq* for a join).

6.3 Injecting Configuration Guidelines in the Model

Process providers may define specific business domain constraints for their process configurations. Thus, *configuration guidelines* are introduced to depict relevant inter-dependencies between the configuration decisions in order to be inline with domain constraints and best practices. Such guidelines are expressed via logical expressions of the form *If-Then*-rules. Both the *if* and *then* parts contain statements about binding configurable nodes to concrete values [20]. An example of such rules is: "*if* a9=OFF and ops5=*Seq*(a7) *then* a14=OFF", i.e. if the car searching and selection functionalities are excluded in a given variant, then the discount activity is excluded too. Note that the *if* part may contain many conditions and the *then* part contains only one statement.

In order to integrate these domain constraints in our model, we define a second abstraction level M1 that refines the first one, M0. We define for each type

of *then* statement one invariant. For instance, the relation *ConfigurationG_ACT* (*inv1*, Listing 6) defines a guideline related to the configuration of an activity. Thus, the guideline may have five different conditions: an activity configuration (line 1), a split or join configuration to a type (line 2), and a split or join configuration to a sequence (line 3). Similarly, *ConfigurationG_CONS* defines the guideline for a split connector configuration.

Listing 6. Guidelines invariants

```
1  Inv1 : ConfigurationG_ACT ∈ P(ACTS × CONF) ×
2     P(CON_S × TYPES×P1(NODES)) × P(CON_J × TYPES×P1(NODES)) ×
3     P(CON_S × NODES) × P(CON_J × NODES) ↔ ACTS × CONF
4  Inv2 : ConfigurationG_CONS ∈ P(ACTS × CONF) ×
5     P(CON_S × TYPES×P1(NODES)) × P(CON_J × TYPES×P1(NODES)) ×
6     P(CON_S × NODES) × P(CON_J × NODES) ↔ CON_S × TYPES×P1(NODES)
    ...
```

As each configuration step must fulfill the configuration guidelines, we refined our abstract events by adding one guard for each guideline. For instance, considering the same example above, we have $\{a9 \mapsto OFF\} \mapsto \emptyset \mapsto \emptyset \mapsto \{ops5 \mapsto a7\} \mapsto \emptyset \mapsto (a14 \mapsto OFF) \in$ *ConfigurationG_ACT*. Thus, we have two conditions consisting of $\{a9 \mapsto OFF\}$ and $\{ops5 \mapsto a7\}$ that if satisfied, *a14* should be mapped to *OFF*. Hence, we added a guard in the event *ConfigureACTON* to ensure that in order to set an activity to *ON* at least one condition is not satisfied in a guideline leading to the configuration of this activity to *OFF*. In this particular case, *a14* can be set to *ON* if *a9* and *op5* have been both configured and *a9* has been set to *ON* or *ops5* has not been configured as a sequence of *a7*. Reciprocally, the configuration *a14* to *ON* is not allowed if at least one of *a9* and *op5* is not configured yet or both have been configured according to the guideline.

7 Verification and Validation

7.1 Verification Using Proof Obligations

In order to demonstrate that the formal specification of configurable process models is correct, a the number of generated proof obligations (POs) should be discharged. Using the Rodin tool [5], our model generated 358 proof obligations; most of them (272 POs $\simeq$ 76 %) were automatically discharged; more complex ones (86 POs $\simeq$ 24 %) required the interaction with the provers to help them find the right rules to apply but also to define additional rules that may lack in the rule base of the prover. These POs ensure that the invariants which model the different constraints on the configurable business processes and the derived variants, are always satisfied (i.e. they hold initially; and each event preserves them). For each event of the form (**WHEN** G **THEN** Act) with G and Act representing the guard and the action respectively, the following proof obligation is generated to verify that the execution of the action Act under the guard G permits to preserve the invariant [4]: $(Inv \wedge G) \Rightarrow [Act]Inv$.

An example of the proofs, we have established, concerns the event *ConfigureACTOFF* correctness with respect to the invariant $inv20$: we have to prove

that even if an activity *act* is removed (set to OFF), it remains possible to reach each node from the initial one. This holds since we have added a control from linking the predecessor of *act* to its successor. To discharge this proof that refers to the closure of a relation, we have added the rule defining the closure of the union of two relation s and r:

$$r \in t \leftrightarrow t \wedge s \in t \leftrightarrow t \Rightarrow cls(r \cup s) = cls(r)((id(t) \cup cls(r)); s)+; (id(t) \cup cls(r))$$

7.2 Validation by Animation

Now, based on a correct model, we validate our Event-B specification by animation and model checking using the ProB plugin [18]. Concretely, we play and observe different scenarios and check the behavior of our model by showing at each step the values of each variable, which events are enabled or not.

For instance, we process the animation of the scenario captured by Fig. 3b as follows. After initializing the model using the process in Fig. 1, all invariants should be respected to ensure the correctness of the configurable process model. Next, we process our scenario by triggering enabled events, and at each configuration step, we observe that invariants are always re-established: (1) we trigger the *ConfigureORSplit* event to configure the split operator *ops1* from OR to an AND ($to = AND$) while maintaining the same branches, (2) *ops3* and *opj1* are configured (using *ConfigureToSeq* event) to a sequence starting from *a3* (*a3* is set to *ON* as well), (3) the activity *a7* is set the to *ON* (using *ConfigureACTON*) since *a3* in included in the previous step (the mapping of a7 to OFF is not allowed in accordance with the guideline defined in Sect. 2), (4) *ops5* and *opj3* are also configured to a sequence starting from *a7* (only this branch could be preserved, since the second branch nodes are configurable), next, (5) when configuring the join operator *opj2*, the only allowed alternative is to fire the event *ConfigureORJoin* with the connector type parameter AND (see Fig. 5). By restricting configuration choices, we guaranteed that the resulted variant have not improper termination caused by the lack of synchronization.

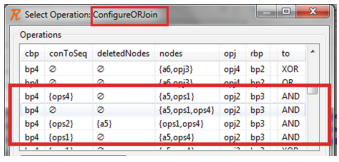


Fig. 5. The connector opj2 configuration restriction using ProB

8 Case Study

In order to evaluate the practical usefulness and identify the opportunities of using our approach, we conducted a case study with a group of business process experts and analysts. We examine its objectives, analyze and discuss its practical experience in conducting business process model configuration in the following.

Objective. The main goal of our work is to evaluate how our approach helps and guides analysts in generating correct and domain-compliant process configuration. Therefore, we define the following research question: How can our approach assist process analyst in applying correct configuration steps?

To answer this question, we formulate three hypotheses: our approach allows *(H1)* to save time and facilitate the identification of the configuration steps; *(H2)* to guarantee a correct process model at each configuration step; and *(H3)* to derive domain-compliant process variants based on the configuration guidelines.

Design, Data Collection and Execution. Our case study is a real configurable supervision process adopted by Orange, a French telecom industrial partner. Different variants of this process are used by Orange affiliates in different cities and countries according to their specific needs. Based on 28 variants, a set of configuration guidelines was generated by an automated approach and validated by a domain expert [8].

With a population of 9 participants that are familiar with process configuration, we targeted experiments to derive a set of different variants using the considered configurable process model. With this purpose, we divided the population into three groups of 3 people each. After a workshop organized to explain the basics needed in this study, the first group (G_1) is asked to manually derive a maximum of process variants without any guidance. Then, the second group (G_2) is also asked to manually derive process variants, but, while providing them with the generated configuration guidelines rules. Whereas, the third group (G_3) is provided with the complete Event-B model (installed under the RODIN tool) and asked to generate process variants with respect to the allowed configuration choices by the model checking. So, participants of latter group can apply only configuration steps that are allowed by our model. As mentioned in the previous sections, this model includes correctness and domain constraints. However, the first two groups take the burden of verifying the correctness of their choices.

The resulted process variants are then collected for comparison. In order to answer the identified research question and confirm its hypotheses, we evaluated the results according to two parameters: (1) the time needed to derive process variants for the different groups, (2) the number of errors for the identified correctness and domain constraints.

Results Analysis and Findings. Regarding the time needed to derive variants, the group G1 took in average 16 min and the group G2 took in average 14 min, whereas the group G3 took only 5 min. Table 3 shows the distribution of the time according to the correctness and the business criteria. Through this table, we notice that the more participants of G1 and G2 take time in deriving variants the less correctness errors are detected. This can explain that participants are making a special effort. Also, it is clear that the first two groups took much more time in deriving correct and domain-compliant variants than the group G3. It is worth noting that all derived variants by G3 contain neither structural nor behavioral correctness errors. No domain errors are detected as well. Moreover, the participants of group G3 affirmed that the ProB model checker is quite straightforward to use and it assisted them in defining appropriate configuration steps. They easily followed the enabled events to make their choices which helped them to be compliant not only to correctness constraints

but also to domain recommendations. As a result, it can be concluded that our approach allows (1) to save time and to assist users in defining their configuration choices, which supports the hypothesis *H1*; and (2) to respect correctness and domain constraints, supporting *H2* and *H3*.

Table 3. The average time in minutes unit spent to derive variants either correct (C) or not (¬C), and either business-complaint (B) or not (¬B)

Group	Variant			
	C & B	C & ¬B	¬C & B	¬C & ¬B
G_1	23	17	15	8
G_2	17	×	11	×
G_3	5	×	×	×

Threats to Validity. First, the small number of the collected process variants, used to generate our configuration guidelines, can be considered as a threat of validity. However, in this study we have chosen 28 variants that are relevant and depict various business needs. Secondly, one case study has been only conducted by 9 participants. We believe that a larger group of participants with varied backgrounds need to be used to highlight the validity and reliability of the experiments results. We leave this to future work.

9 Conclusion

In this paper, we propose a formal Event-B based approach to derive correct variants from well-defined configurable processes. To do so, we introduce a step-based configuration approach to guide the analyst by providing at each step the potential configuration choices. We have succeeded to verify structural constraints of configurable process model (e.g. each node reachable from the initial activity, has always the option to complete). We have also reached our goal in preserving the variants soundness (no deadlocks and lack of synchronization situations). Finally, our approach respects domain requirements provided by *configuration guidelines* as well. As future work, we plan to extend the proposed approach by considering the required configurable cloud resource allocation [12] and adding a formal verification phase.

References

1. van der Aalst, W.M.P., Dumas, M., Gottschalk, F., ter Hofstede, A.H.M., La Rosa, M., Mendling, J.: Correctness-preserving configuration of business process models. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 46–61. Springer, Heidelberg (2008)

2. Aalst, W.V.D., et al.: Preserving correctness during business process model configuration. Formal Aspects Comput. **22**(3–4), 459–482 (2008)
3. Aalst, W.V.D., Lohmann, N., Rosa, M.L.: Ensuring correctness during process configuration via partner synthesis. Inf. Syst. **37**(6), 574–592 (2012)
4. Abrial, J.R.: Modeling in Event-B: System and Software Engineering, 1st edn. Cambridge University Press, New York (2010)
5. Abrial, J.R., et al.: Rodin: an open toolset for modelling and reasoning in Event-B. STTT **12**(6), 447–466 (2010)
6. Asadi, M., Mohabbati, B., Grner, G., Gasevic, D.: Development and validation of customized process models. J. Syst. Softw. **96**, 73–92 (2014)
7. Assy, N.: Automated support of the variability in configurable process models (2015)
8. Assy, N., Gaaloul, W.: Extracting configuration guidance models from business process repositories. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 198–206. Springer, Heidelberg (2015)
9. Boubaker, S., Mammar, A., Graiet, M., Gaaloul, W.: An Event-B based approach for ensuring correct configurable business processes. In: The 23rd IEEE International Conference on Web Services, ICWS (2016)
10. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H., La Rosa, M.: Configurable workflow models. Int. J. Coop. Inf. Syst. (IJCIS) **17**(2) (2008)
11. Groner, G., Boskovic, M., Silva Parreiras, F., Gasevic, D.: Modeling and validation of business process families. Inf. Syst. **38**(5), 709–726 (2013)
12. Hachicha, E., Assy, N., Gaaloul, W., Mendling, J.: A configurable resource allocation for multi-tenant process development in the cloud. In: Nurcan, S., Soffer, P., Bajec, M., Eder, J. (eds.) CAiSE 2016. LNCS, vol. 9694, pp. 558–574. Springer, Heidelberg (2016). doi:10.1007/978-3-319-39696-5_34
13. Hallerbach, A., Bauer, T., Reichert, M.: Guaranteeing soundness of configurable process variants in provop. In: IEEE Conference on Commerce and Enterprise Computing, CEC, pp. 98–105 (2009)
14. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the provop approach. J. Softw. Maintenance Evol. **22**(6–7), 519–546 (2010)
15. Kiepuszewski, B., Hofstede, A.T., Aalst, W.V.D.: Fundamentals of control flow in workflows. Acta Informatica **39**(3), 143–209 (2002)
16. Kumar, A., Yao, W.: Design and management of flexible process variants using templates and rules. Comput. Ind. **63**(2), 112–130 (2012)
17. La Rosa, M., Van Der Aalst, W., Dumas, M., ter Hofstede, A.: Questionnaire-based variability modeling for system configuration. Softw. Syst. Model. **8**(2), 251–274 (2008)
18. Leuschel, M., Butler, M.: ProB: an automated analysis toolset for the B method. Int. J. Softw. Tools Technol. Transf. **10**(2), 185–203 (2008)
19. Rosa, M.L., Aalst, W.V.D., Dumas, M., Milani, F.: Business process variability modeling: a survey (2013)
20. Rosemann, M., Aalst, W.V.D.: A configurable reference modelling language. Inf. Syst. **32**(1), 1–23 (2007)
21. Schunselaar, D.M.M., Verbeek, E., van der Aalst, W.M.P., Raijers, H.A.: Creating sound and reversible configurable process models using CoSeNets. In: Abramowicz, W., Kriksciuniene, D., Sakalauskas, V. (eds.) BIS 2012. LNBIP, vol. 117, pp. 24–35. Springer, Heidelberg (2012)
22. Van Dongen, B., Mendling, J., Aalst, W.V.D.: Structural patterns for soundness of business process models. In: Enterprise Distributed Object Computing Conference, pp. 116–128 (2006)