Inf Syst E-Bus Manage (2016) 14:443–467
DOI 10.1007/s10257-015-0290-7

ORIGINAL ARTICLE

Variability patterns for business processes in BPMN

Alaaeddine Yousfi[1,2] · **Rajaa Saidi**[1,3] · **Anind K. Dey**[2]

Received: 10 April 2015 / Revised: 5 July 2015 / Accepted: 1 August 2015 /
Published online: 25 August 2015

Abstract Many entities, both in academia and the business sector, urge an efficient improvement of business processes. However, when it comes to addressing this point, each slight disparity in the business rules and/or objectives translates into a separate model, which is neither practical nor acceptable as it burdens the host process-aware information system with repetitive and almost verbatim instances. To solve this issue, we propose considering variability. Variability will serve as a business process improvement technique to efficiently design and run a variable business process throughout different business situations that are similar to one another is some ways yet differ in others. First, we define variability within the context of business processes. Second, we present a set of variability patterns and explain how they are used. We validate our approach via the business process improvement patterns known and used by the community. The variability design patterns are a series of business process improvement patterns for building business process with variability and efficiently acting on the improved process performance metrics.

Keywords Business process improvement · Business process · BPMN · Variability · Variability pattern

✉ Alaaeddine Yousfi
aeyousfi@cmu.edu

Rajaa Saidi
r.saidi@insea.ac.ma

Anind K. Dey
anind@cs.cmu.edu

[1] LRIT, Research Unit Associated to the CNRST (URAC 29), FSR, Mohammed V University, Rabat, Morocco

[2] HCII, Carnegie Mellon University, Pittsburgh, PA, USA

[3] INSEA, BP 6217, Rabat, Morocco

1 Introduction

Business Processes (BPes) are steadily gaining more ground in controlling how workflows are managed and handled. These components contribute greatly to the optimization of important performance metrics such as timeliness, effectiveness and correctness. They are dynamic entities able to handle a diversity of business situations Predonzani et al. (1999). However, business processes are still intuitively modeled in a very basic and ad hoc fashion. They are typically designed and used based on what one feels to be true without conscious reasoning or an explicit approach.

This has meant that when an engineer develops several business processes, she/he represents and executes them separately from each other, even with a slight disparity in the business rules/objectives. As a result, the host Process-Aware Information System (PAIS) Mejia Bernal et al. (2010) ends up having an abundance of repetitive and almost verbatim business processes, which is neither practical nor acceptable due to the fact that this approach undermines the three major concerns of software engineers; cost, quality and time Sinnema and Deelstra (2007).

As such, let us consider the following examples:

Example 1 Production Line

Adam, Gina and Emily stop by McDonald's to order a Big Mac. Each of these people has the following business requirements with regard to her/his burger to be:

- Adam does not want pickles in his burger.
- Gina does not want sauce and lettuce in her burger.
- Emily wants a regular burger.

Example 2 Assembly Line

Boeing currently builds its commercial airplanes across the state of Washington. Each model is characterized by multiple variants. For instance, the 777 can be 772ER, 772LR, 773ER or 777F. Moreover, each variant has multiple options such as the engine type (e.g., Rolls Royce, General Electric). Say United and Delta place the following specific orders:

- United orders a 772LR with Rolls Royce engines.
- Delta orders a 773ER with General Electric engines.

Based on both examples, McDonald's does not make new production lines to fulfill each different requirement of the same Big Mac. Neither does Boeing build multiple assembly lines to fulfill each different requirement of the same 777. A production line costs money. So does an assembly line. For McDonald's, a clerk handles all the customer requirements through the same system that is offering multiple variants (e.g., pickle, sauce and lettuce). For Boeing, the airlines' requirements are fulfilled through the same assembly line for each model (e.g., one existing assembly line for the 777 in Everett, Washington). Note both examples use a variation throughout the same production/assembly line of reference to fulfill the specific requirements of each customer.

The production (e.g., McDonald's) and the assembly (e.g., Boeing) lines are two specific examples of what was generalized in early 1990s (when business processes were officially introduced) as workflow. A business process workflow is a sequence of connected steps that go from a mission objective to a business objective. Here, the goal is to propose a generic approach that fits within the generic nature of a business process and models it in an efficient way. Efficiency refers to the level of disparity between the business process outcome (product or service) and the business situation (a set of business requirements).

To model business processes in an efficient way, we apply the technique of Variability Hallerbach et al. (2010), Park and Yeom (2011), Khan et al. (2011), Galster and Avgeriou (2011). Variability will serve as a Business Process Improvement (BPI) Shtub and Karni (2010) technique to achieve more efficient results throughout several business situations. This concept has been defined in a variety of ways, including:

1. Variability is the propriety of an object to being changeable and the capacity of a system to be tailored van Eijndhoven et al. (2008).
2. Variability is the ability of a core asset to vary in the different product contexts, under conditions of environment, in a preplanned fashion, within the product line scope Santos et al. (2010).

Subsequently, our new definition of variability, which adapts to the context of business processes, is split among the highest and lowest levels of abstraction:

– ***highest or design-time***: Variability is a technique for designing business process(es), in which business rules and/or objectives are similar to one another in some ways but different in others, by factorizing the similarity(ies) and grouping the difference(s) into a business process of reference.
– ***lowest or run-time***: Variability is the ability of a business process to change its behavior, within the scope of the pre-modeled eventualities already anticipated in the business process of reference.

Why variability? Via variability we obtain a number of advantages that we enumerate below (by gain):

1. the wasted time and money spent on modeling separate business processes with commonalities is reduced. (*time* and *cost*)
2. the expenditures for maintenance are minimized because of the business processes' flexibility. (*cost*)
3. a business process life-cycle lasts longer since agility reduces maintenance frequency. (*time* and *cost*)
4. portions of business processes can be reused in other business processes due to the reason that variability supports reusability. (*time* and *cost*)

5. the process variants are dynamic and reflect the dynamics of real-world environments. (*quality*)

All in all, without variability, fulfilling the business requirements of each instance is usually accomplished by treating each variation as a distinct process, which leads to redundancy and inconsistency among the business processes Milani et al. (2012). As a matter fact, the goal of this paper is to propose a solution to coherently and efficiently design and run business processes with similarities yet have some differences. Building on that, the contributions of this paper are threefold. First, we establish the new concepts of variable business process, variable partition and variability objective. Second, we define two forms of variability patterns: product patterns and process patterns. The first are design elements while the second are two straightforward processes for generating business processes with variability at both design-time and run-time.

A variability modeling technique is a procedure for modeling variability, while a variability mechanism is a way to introduce or implement variability. In fact, this manuscript is structured as follows: Sect. 2 synthesizes previous work in both directions of variability (modeling technique and mechanism) and highlights their shortcomings. Section 3 describes our approach in managing variability within business processes in BPMN while Sect. 4 presents an illustrative example of our approach. Afterwards, Sect. 5 discusses the validation of the approach while Sect. 6 concludes this paper, discusses the results and considers future work.

2 Existing variability modeling techniques

Although BPMN[1] is a BPML[2] that has been shown to have great value in Business Process Management (BPM) Jeston and Nelis (2014), its variability modeling techniques are still relatively nascent. Authors of Santos et al. (2010) suggest analyzing existing BP models, to discover possible variations, and applying a Non-Functional Requirement (NFR) procedure so as to assure a process configuration. They unfortunately fail to address how their proposed NFR methodology can be used to produce the resulting diagrams. In addition, they consider all BPMN elements to be potential variation points which is somehow paradoxical since, for instance, saying a pool may be a decision point appears unsound. Similarly, authors of Weidmann et al. (2011) offer an incipient approach as it still does not explain how their methodology can be applied to real world use cases. They plan to explore this in their future work.

Alternatively, the variability modeling techniques proposed by La Rosa et al. (2008), Schnieders and Puhlmann (2006) are a promising stepping stone toward our suggested approach. The authors borrow the concept of stereotype from UML 2 to establish their mechanisms. Hereafter, we synthesize their work.

[1] Business Process Model and Notation.

[2] Business Process Modeling Language.

Based on the achievements of La Rosa et al. (2008), each BPMN task can have one of the seven stereotypes ($\ll$ *VarPoint* $\gg$, $\ll$ *Abstract* $\gg$, $\ll$ *Null* $\gg$, $\ll$ *Optional* $\gg$, $\ll$ *Alternative* $\gg$, $\ll$ *Variant* $\gg$ *and* $\ll$ *Default* $\gg$) attached to it. Judging by their specification, we can group, actually, each of which into two separate sets referring to, either a Rule of Choosing (RC), or, a Rule of Being Chosen (RBC).

$$RC \in \begin{cases} \ll Abstract \gg \\ \ll Null \gg \\ \ll VarPoint \gg \\ \qquad \cup \\ \ll Optional \gg \\ \ll Alternative \gg \end{cases} \quad \textbf{AND} \quad RBC \in \begin{cases} \ll Default \gg \\ \ll Variant \gg \end{cases}$$

Figure 1 shows an excerpt of a stereotyped business process model using the "Basic Mechanisms". Variability here sets the rules regarding the ordering of a credit card(s). On account of this variability, the customer may order a credit card from her/his bank depending on the variants offered (package card, gold and platinum). The task "Order credit card" is the *Generic Task*. The presence of the label $\ll$ *Stereotype* $\gg$ indicates that the rule for ordering the card(s) differs depending on the value of this label, which may be one of the RC set elements.

To get Table 1, we substitute the label $\ll$ *Stereotype* $\gg$ from the generic task with each value belonging to the first subset of RC apart; we identify the default variant and count the number of additional ones. Note, we disregard both $\ll$ *Optional* $\gg$ and $\ll$ *Alternative* $\gg$, in this paper, since they are exceptional cases of $\ll$ *Null* $\gg$ and $\ll$ *Abstract* $\gg$, respectively, with a unique variant.

BPMN basic stereotypes, even though pioneering, are considered elementary since they only accept atomic variants and offer simple variability. To address this, authors of Schnieders and Puhlmann (2006) propose a set of advanced variability

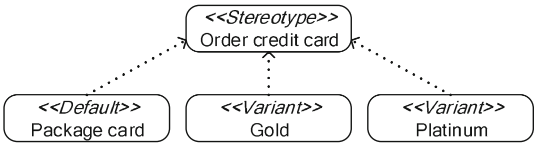


Fig. 1 Stereotyped *basic mechanism* variability

Table 1 Basic mechanisms' explanatory and synthesized table (for the case of three variants)

Basic mechanism	Default variant	Cardinality
$\ll$ *Abstract* $\gg$	Package card	1
$\ll$ *Null* $\gg$	None ($\ll$ *Default* $\gg$ is replaced by $\ll$ *Variant* $\gg$ on Fig. 1)	0..3
$\ll$ *VarPoint* $\gg$	Package card	1..3

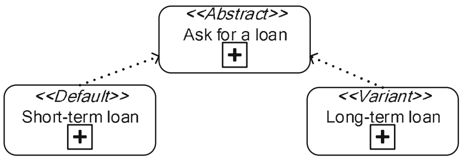


Fig. 2 Stereotyped *ESP mechanism* variability

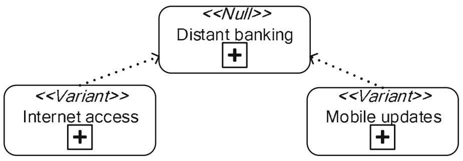


Fig. 3 Stereotyped *extension points mechanism* variability

mechanisms in which the variants are involved. Five complex mechanisms are introduced (Encapsulation of Sub-Processes (ESP), Parameterization, Inheritance, Extension Points and Design Patterns). The encapsulation of sub-processes along with the extension points are, ultimately, progressive forms of the two basic mechanisms ≪ *Abstract* ≫ and ≪ *Null* ≫, respectively. Unlike these two basic mechanisms, the two advanced mechanisms support non-atomic variants which are sub-processes. The design patterns are, however, excluded in some references limiting the number of advanced mechanisms to four since this component is simply a combination of the others. For this reason, we will deal, henceforward, with the first four evolved mechanisms and not consider the "Design Patterns".

The *Encapsulation of Sub-Processes (ESP)* imposes that a single and unique sub-process variant be chosen. Figure 2 elucidates how a bank customer should ask for either a short-term loan or a long-term loan with the former option being the default one. Choosing both types of loans is not authorized. This is analogous to the ≪ *Abstract* ≫ basic stereotype in the cardinality except for the variants' atomic nature.

The *Extension Points* mechanism specifies that all offered options (sub-processes) are possible, including no choice. Figure 3 illustrates that the bank customer may subscribe to Internet-based remote banking, the Mobile-based remote banking, to both services or to neither (no default variant is required). This is analogous, too, to the ≪ *Null* ≫ basic mechanism except for the variants' atomic nature.

The *Inheritance* mechanism, as mentioned by Schnieders and Puhlmann (2006), modifies an existing (default) sub-process by adding either activities or pools to match specific business rules. This supports having alternative variation points. Figure 4 shows how the default and parent sub-process (withdraw money) can either be instantiated alone or the specialized and child sub-process (withdraw money plus the backup task) will take over by reusing the sub-process (withdraw money) for instantiation, in case the ATM is depleted and the cash is no longer available, to suggest a nearby ATM with available cash.

The *Parameterization*, unlike all the previous mechanisms, offers the possibility to interact with both event and data. That is to say, this mechanism is both data-based and/or event-based, which means, it could either represent a branching

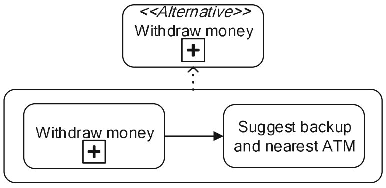


Fig. 4 Stereotyped *inheritance mechanism* variability

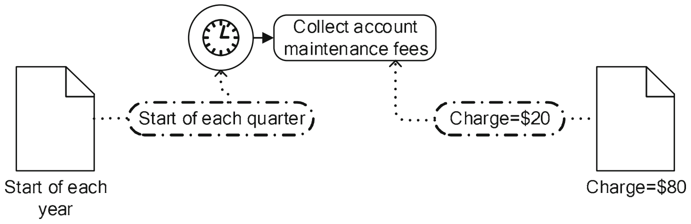


Fig. 5 Stereotyped *parameterization mechanism* variability

variability in the process where the alternatives are triggered upon an event that occurs at the variation point, or, upon the using of process data. Figure 5 presents an example with a double parameterization (data-based and event-based). As indicated, the first variability simulates the frequency of collecting bank account maintenance fees (event-based) that is either yearly or quarterly, while the second defines the charge (data-based) that needs to be collected.

Although these synthesized mechanisms model variability in BPMN, they suffer from the following significant issues:

1. By adding the stereotypes, the mechanisms extend BPMN and burden its diagrams with superfluous notations. The models appear non readable. Still, they are reserved to BPMN and stereotyped BPMN acquainted entities. More importantly, we believe variability can be stereotype-free in BPMN.
2. It is not clear how to take these mechanisms and apply them to real field cases. No existing approach discusses taking over the stereotyped diagrams and applying them to field work.
3. There is redundancy in some basic and advanced mechanisms. For example, the $\ll$ *Abstract* $\gg$ and the *ESP* mechanisms are similar in cardinality. So why two mechanisms?
4. These mechanisms are both less open and less flexible for event(s) handling. *Parameterization* offers very restrictive support for modeling events, while the other mechanisms do not offer any possibility for modeling events.

The following section presents our approach to circumvent these four weaknesses. In doing so, we reduce the complexity of process diagrams, time needed and modeling costs by skipping unnecessary phases while designing variability for business processes in BPMN. Moreover, we introduce variability patterns to motivate component reuse and standardize the variability mechanisms. We also outline the variability configuration and derivation patterns to guide the designer and ease the modeling task.

3 Proposed variability modeling technique

To identify and manage variability in a procedural, organized and standardized methodology, we build our solution on the concept of patterns. Patterns are problems and approaches to solving problems described in a generalized form Predonzani et al. (1999). They depict a recurrent issue and its core solution. The pattern solution makes the design more flexible, elegant and reusable Gamma et al. (1994). Here, we use ***two types of patterns***: *Product Patterns* that represent a model to apply and *Process Patterns* that set a technique to follow Conte et al. (2002). The use of *product patterns* helps overcome redundancy and inconsistency among the business process models by factorizing business processes that are similar to one another in some ways but are different in others, in a business process of reference. Still, for each new business rule the change is incremental (not radical). In other words, the pattern already exists as a template and the designer only adds new variants to fulfill each new business requirement. On the other hand, the *process patterns* Ambler (1998) clarify the straightforward steps to be taken to build a BP with variability both at design-time and run-time. In short, patterns result in easier uptakes by business process modelers and therefore information system designers.

3.1 Business process with variability

We designate by a Variable Business Process (VBP), a business process embedding variability. Variability embedment is characterized by the presence of *at least one Variable Partition* like indicated on the definition diagram of Fig. 6. The variable partition delimits the discrepancy zone, in the process, where multiple possibilities may occur. Each variable partition is distinguished by a *Variability Objective* which describes the discrepancy handled by the variable partition (e.g., select notification method, order credit card). Also, it is important to point out that variable partitions may overlap within the same process. When overlapping, smaller variability objectives are grouped into bigger ones and so forth (e.g., select package card into choose debit card into order product).

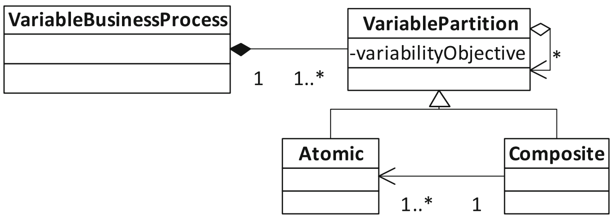


Fig. 6 Variable business process definition model

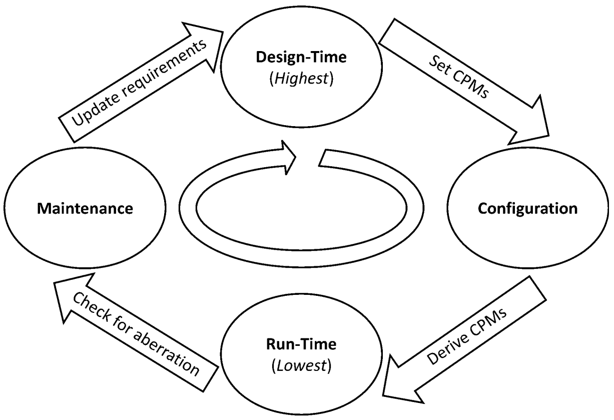


Fig. 7 Variability mutation cycle within business processes

Table 2 Variability facets at the highest and lowest levels of abstraction

Variability facet	Design-time (highest)	Run-time (lowest)
Business rule	Virtual	Concrete
Variability goal	Anticipation	Adaptation
Activity	Generic	Specific
Variability partition	Variable partition	Adapted partition
Business process model	Configurable process model	Derived process model

Variable partitions are, indeed, the cornerstone of our approach for building VBPes. Therefore, we focus on presenting our standardized approach for modeling these entities based on the concept of patterns. By embedding variable partitions into the high-level abstraction process models, we build Configurable Process Models (as known as CPMs) Recker et al. (2007), Derguech et al. (2012), Chun et al. (2009). CPMs are a step forward in the systematic reuse of process models. They offer the possibility for modeling BPes that are similar to one another in many ways yet differ in some other ways. The CPM is, then, considered a reference model to be employed in distinct use cases.

Our definition of variability (see Sect. 1) is split into design-time and run-time. Actually, variability mutates from the highest to the lowest level of abstraction and so forth throughout its cycle (Fig. 7). The way it is viewed and how it operates is different at each stage of the cycle. Table 2 groups the different factors in relation to the highest and lowest levels. In the modeling stage, variability is more about anticipation of changes and dynamics: at this point, the business rules are virtual, and the activities are generic. For the execution level, the goal is adaptation which means, the business rules are concrete regarding a certain use-case scenario and engender the activities to be specific.

3.2 Variability design patterns

3.2.1 Patterns specification

The variability design patterns are our product patterns for designing VBPes. They share the same ground rules as Workflow Patterns Aalst et al. (2003) since they innately deal with problems such as sequence, choice and synchronization. In addition, they handle recurrent variability design problems and their solutions by serving as a template for reducing the modeling and simplifying the maintenance of each variable partition to be.

Based on the meta-model of Fig. 8, this section describes the standardized representation we adopt for modeling variable partitions in an automatic and formal way. These partitions will be embedded in the highest level abstraction diagrams of the targeted processes.

Because BPMN v2.0 OMG (2011) does not offer an event-based OR gateway, we extend this version by adding the latter gateway as indicated with the **thick Generalization** in the meta-model. We believe by adding a new gateway into BPMN v2.0 we provide more flexibility for handling the event-triggered variable partitions. The semantic of the event-based OR gateway that we introduce is similar to the data-based one, except for the branching that is event-based. For the graphical representation, we add an upper case **E** in the center of the data-based OR shape.

Authors of Dumas et al. (2010) and Gschwind et al. (2008) discuss how to structure process models. They take already existing unstructured models and transform them into structured ones. As a continuation to their work, the patterns we introduce inherently help building both structured process diagrams from scratch and less error prone model designs. ***Moreover***, the patterns we propose meet, by definition, the requirements of control-flow posed by BPMN OMG (2011), which

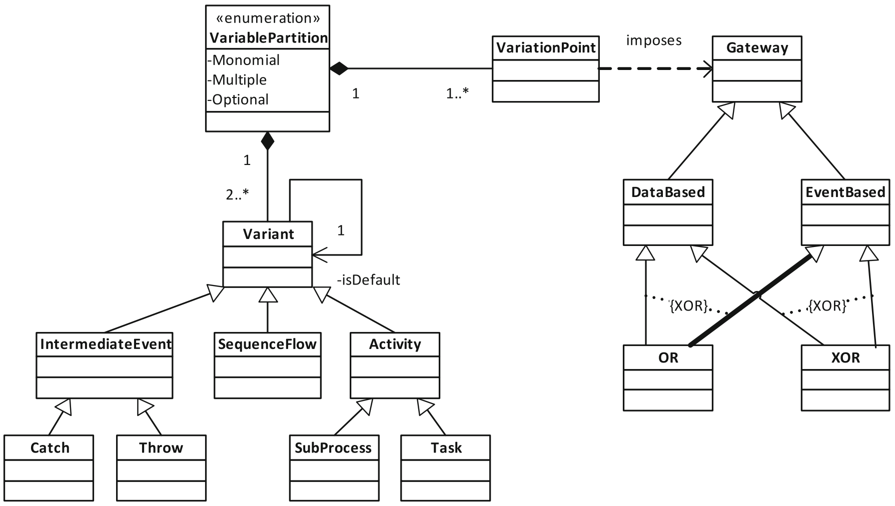


Fig. 8 Variability design patterns in BPMN meta-model

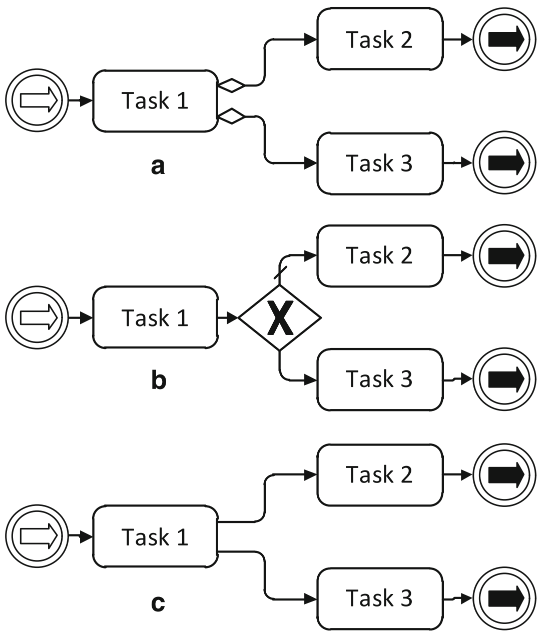


Fig. 9 Example of control-flow situations in BPMN

enables the process model(s) holding the partitions to be flawless. As an illustration, Fig. 9 draws a comparison between a process excerpt model both in uncontrolled and controlled design versions. In the bloc Fig. 9a "*task 1*" holds, for instance, two branching conditions. If the process designer in case (a) intends, for example, to impose a single path in the workflow, these branching conditions MUST be mutually exclusive. If she/he does not pay attention to such a critical condition, the process will get blocked. To avoid such a disastrous situation, it is better to use a gateway (Fig. 9b). In this case, it is of type XOR (data-based) with a catch-up branching-condition-free default path. Obviously, the argument stands for itself in case of OR or even non-conditional branching (Fig. 9c).

As indicated in the meta-model (Fig. 8), the variable partition definition (the area in the business process where multiple possibilities may occur) is linked to both its cardinality as well as its triggering method. For the cardinality, we define three types:

1. ***Monomial***: a **sole** variant among the ones offered is authorized. A **default** one **must** be specified.
2. ***Multiple***: **several** variants among the ones offered are authorized. A **default** one **must** be specified.
3. ***Optional***: a **no** choice as well as **several** variants among the ones offered is/are authorized. A **default** one is the pristine sequence flow path.

Table 3 Variability design patterns formal definition (TM: triggering method)

Pattern name		Problem		Solution		Consequences
TM	Cardinality	TM	Cardinality	Divergent gateway	Convergent gateway	
Data-based	Monomial	Data	1	XOR (data)	XOR (data)	
Data-based	Multiple	Data	1..*	OR (data)	OR (data)	
Data-based	Optional	Data	*	OR (data)	OR (data)	
Event-based	Monomial	Event	1	XOR (event)	XOR (data)	Improve business process
Event-based	Multiple	Event	1..*	OR (event)	OR (data)	
Event-based	Optional	Event	*	OR (event)	OR (data)	

1..* The value of the cardinality could be between 1 and n with n an integer

* either 0, 1, 2, … n

When we add the triggering method to the cardinality description, we arrive at our definition of variable partitions. For that purpose, we define two types of triggering methods:

1. ***Data-based variable partition***: the **variants** are linked to the **variation point** by a flow of **data**.
2. ***Event-based variable partition***: the **variants** are linked to the **variation point** by a flow of **events**.

Building on the previous classifications (cardinality and triggering method), our variability design patterns are defined in Table 3, following the format introduced by Gamma et al. (1994). The set holds six elements. Each of which is bearing the *name* "triggering method (e.g., Data-based) cardinality (e.g., Monomial)". The pattern *problem* is described by the triggering method along with the cardinality. The *solution* provides the gateways (divergent and convergent) to be adopted for solving the problem. For all patterns, the *consequences* are optimizing time, cost and quality in engineering VBPes.

3.2.2 Data-based variability design patterns

Meta-Models are by definition structural diagrams. To set the requirements about the order in which the BPMN core modeling elements appear in a data-based design pattern, we introduce the architecture of Fig. 10. *The ellipsis (...) indicate that the number of variants is variable and up to the modeler*. The patterns are delimited by two gateways; a divergent gateway placed at the beginning of the variable partition (a variation point) and a convergent one located at the end. The nature of the first gateway depends absolutely on the description of each single mechanism. The second gateway,

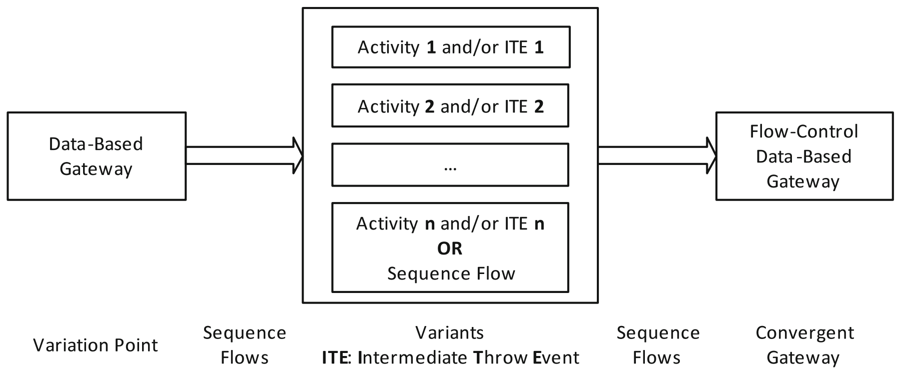


Fig. 10 Data-based variability design patterns architecture

though, is placed to control the flow and avoid invalid models. As the BPMN creators dictate OMG (2011) "*If the flow does not need to be controlled, then a Gateway is not needed*". Otherwise, it is. The variants may be activity(ies), intermediate throwing events or a pristine sequence flow in case of an optional variable partition.

As indicated in the meta-model, we introduce three cardinalities of variability: Monomial, Multiple and Optional. Table 4 exemplifies the data-based patterns. The variants [intermediate event(s) and/or the activity(ies)] may vary as long as they follow the architecture of Fig. 10. *For the sake of clarification, uniformity and space distribution we use three variants as illustration.*

3.2.3 Event-based variability design patterns

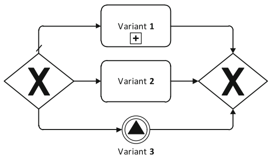
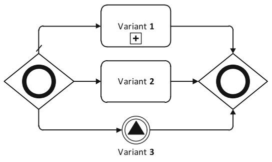
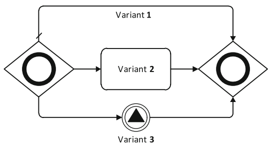
Similarly to the data-based patterns, the event-based patterns are three types: Monomial, Multiple and Optional. In this case, the variants are event(s)-triggered. Figure 11 illustrates our event-based patterns architecture. Unlike the data-based one, this current representation contains a *must-have* layer for catching the upcoming events. According to the BPMN v2.0 specification, the only intermediate catching events to be implemented are: Message, Signal, Timer, Conditional and Multiple. On the other hand, all the other characteristics of the data-based architecture stand the same.

Table 5 illustrates the event-based design patterns. The layer of intermediate catch events as well as the variants [intermediate events and/or activity(ies)] MAY vary as long as the architecture of Fig. 11 is respected. The Intermediate Timer Event is placed to catch up with the flow in case no event is triggered after a certain time. It is equivalent to the "slash" of the data-based architecture. Note, we use the event-based OR gateway that we introduced at the beginning of this section for the Multiple and Optional patterns.

3.3 Variability configuration pattern (*design-time*)

Variability configuration takes place at design-time following the definition we introduce in Sect. 1. Here, we discuss the process pattern for variability configuration. Its specification is discussed as follows:

Table 4 Examples of data-based variable partitions for the case of a three variant data-based design pattern

Data-based design pattern	Cardinality (gateway)	Example of variants	Default variant	Variable partition
Monomial	1 (XOR)	Variant **1** Subprocess (Activity) Variant **2** Task (Activity) Variant **3** Signal (ITE)	Variant 1	
Multiple	1..* (OR)	Variant **1** Subprocess (Activity) Variant **2** Task (Activity) Variant **3** Signal (ITE)	Variant 1	
Optional	* (OR)	Variant **1** Sequence flow Variant **2** Task (Activity) Variant **3** Signal (ITE)	Variant 1	

[1..*] The value of the cardinality could be between 1 and n with n an integer

* either 0, 1, 2, … n

- **Pattern Name**: Variability Configuration
- **Problem**: Business rules/objectives that are similar to one another in some ways yet differ in others
- **Solution**:

```
Step 1. Identify and group business rules/objectives with commonality(ies)
  ⟹ For each group of Step1.
  Step 2. Identify the discrepancy(ies) in the group
  Step 3. Identify the variability objective(s) and delimit the variable partition(s)
    ⟹ For each variable partition
    Step 4. Identify the variants
    Step 5. Identify the relationship among the variants (cardinality)
    Step 6. Identify the variants triggering method (event or data)
    Step 7. Apply the corresponding variability design pattern
    ⟸ End
  Step 8. Merge the variable partition within the process model
  ⟸ End
```

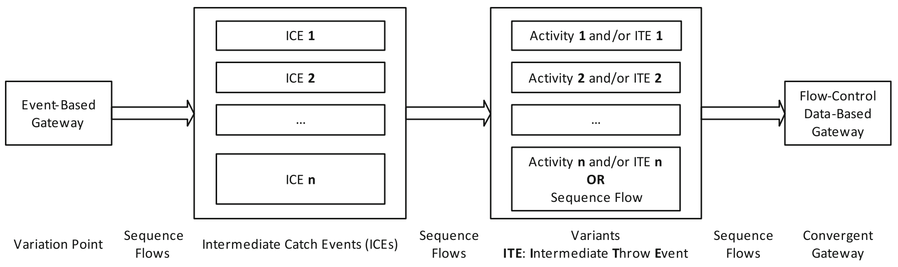


Fig. 11 Event-based variability design patterns architecture

– **Consequences**: Build variable business processes at design-time (aka configurable process models).

For indication, the right arrow in the pattern solution, means getting in the down-level process while the left one symbolizes getting out to the up-level process. Also, the first step can be accomplished following the work of Dijkman et al. (2011), van Dongen et al. (2008), which we take for granted. Figure 12 clarifies the pattern solution in BPMN. Referring to the process modeling elements, we impose the use of the Manual Task for identifying business rules/objectives with commonalities. Still, we use Ad-hoc type sub-process to denote that the order of processing is not fundamental. Otherwise, either business rules or partitions, the order of treating each, does not make the difference in generating the outcome BP with variability arrangements. To keep traceability over the process levels, we place the (Start and End) events on the boundaries of each single and expanded sub-process.

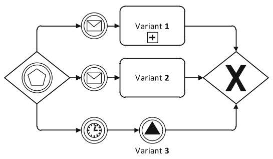
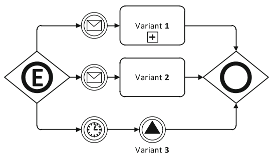
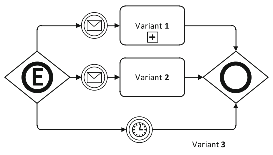
It is important to point out that the designer can approach the partitions either ascending or descending. It means that the modeler may start with smaller partitions and go higher to the bigger ones (ascending) until every condition is entirely covered or the opposite (descending). We recall that the partitions may overlap (see definition model in Fig. 6).

Although built in a three level architecture, the variability configuration pattern solution does not suffer from complexity since most instructions are purely accomplished through observation (steps 2, 3, 4, 5 and 6). Furthermore, its presence along with the set of design patterns can provide great assistance to the designer by guiding her/him through the steps for generating CPMs. As such, the likelihood of digressing from the design goal is reduced.

Example: Since the variability configuration pattern is repetitive for each group of business rules and variable partitions, we focus this illustrative example on producing one single variable partition. So, let us consider the following group of business rules:

– Once a credit card is inserted into an ATM, a security check procedure is launched before offering the transactions menu.

Table 5 Examples of event-based variable partitions for the case of a three variant event-based design pattern

Event-based design pattern	Cardinality (gateway)	Example of variants	Default variant	Variable partition
Monomial	1 (XOR)	Variant **1** Subprocess (Activity) Variant **2** Task (Activity) Variant **3** Signal (ITE)	Variant 3	
Multiple	1..* (OR)	Variant **1** Subprocess (Activity) Variant **2** Task (Activity) Variant **3** Signal (ITE)	Variant 3	
Optional	* (OR)	Variant **1** Subprocess (Activity) Variant **2** Task (Activity) Variant **3** Sequence flow	Variant 3	

– Depending on the bank's records, a credit card may either be expired, suspended (after entering an erroneous code three times), declared stolen or valid.
– If a credit card is reported as stolen, both the authorities and the network of local offices are informed. The card is then confiscated by the ATM.
– In case a credit card is suspended, a suspension message is displayed to the user and the card gets ejected from the ATM.
– If the credit card is expired, an expiration message is displayed to the user and the card is confiscated by the ATM.

By going back to the variability configuration pattern solution, the outcome of each step is discussed as follows (we skip steps 1 and 8 because the work is repetitive through variable partitions and their merger is discussed throughout the illustrative example section):

Step 2. Identify the discrepancy(ies) in the group: there is a discrepancy between the card statuses and their intended follow-ups.
Step 3. Identify the variability objective and delimit the variable partition: the variability objective is *check credit card validity*. Its corresponding partition contains the follow-ups to each credit card reported status plus the delimitation gateways.
Step 4. Identify the variants: according to the text of the business rules, four statuses imply four variants: stolen, expired, suspended and valid.

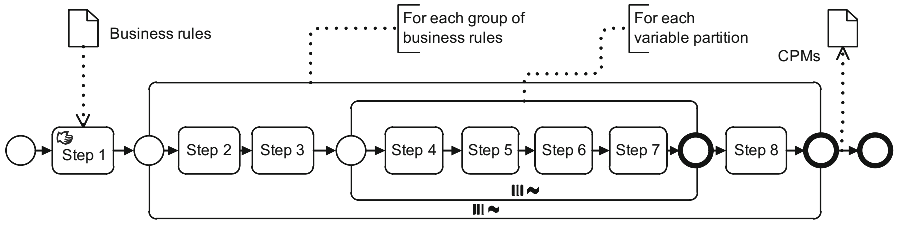


Fig. 12 Variability configuration pattern solution in BPMN

Step 5. Identify the relationship among the variants (cardinality): according to the text of the business rules "...a credit card may ***either*** be...", implies a cardinality of one.
Step 6. Identify the variants triggering method (event or data): according to the text of the business rules "Depending on the ***bank's records***...", implies a data-based variability.
Step 7. Apply the corresponding variability design pattern: XOR (data-based), implies a monomial data-based variability design pattern.

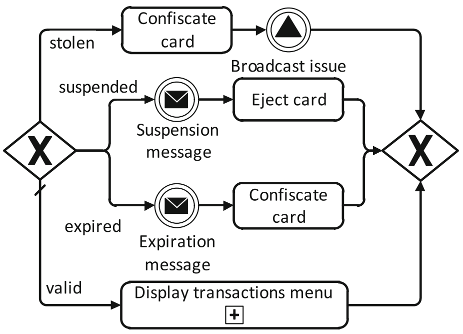
Figure 13 presents the graphical representation of the variable partition "*Check Credit Card Validity*" in BPMN. The default path is the one leading to the display of the transactions menu.

3.4 Variability derivation pattern (*run-time*)

Once the BP designer finishes applying the variability configuration pattern, she/he will end up with CPMs as an outcome. Each CPM is, considered a reference model that is susceptible to undergo a derivation procedure, with the goal of getting a Derived Process Model (DPM), with regard to each business situation. The variability derivation pattern specification is presented as follows:

- **Pattern Name**: Variability Derivation
- **Problem**: Configurable process model
- **Solution**: *Derivation procedure* Identify and omit the the superfluous variant(s) as a response to a business situation needs.
  - if the design pattern is of type monomial or optional with one variant selected, the exclusive gateways (variation point and flow-control) are omitted.
  - if the design pattern is of type multiple or optional with more than one variant selected, the inclusive gateways (variation point and flow-control) are replaced with parallel ones.
- **Consequences**: Derived process model.

Fig. 13 "Check credit card validity" variable partition



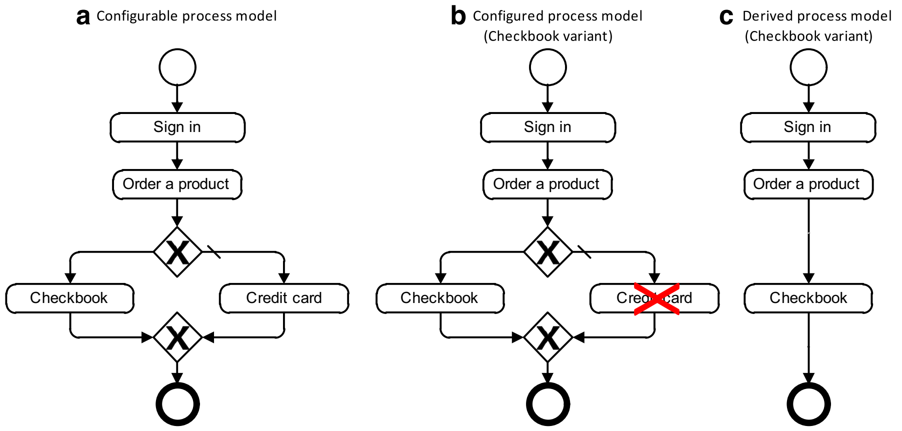

a Configurable process model
b Configured process model (Checkbook variant)
c Derived process model (Checkbook variant)
Sign in
Order a product
Checkbook
Credit card

Fig. 14 CPM derivation

Example:

The process of Fig. 14a holds a single variable partition of type monomial data-based, whose variability objective is "Order a product". Judging by this variability, the client may order a credit card or a checkbook but not both or neither. The CPM gathers the possibilities. Figure 14b illustrates how the desired variant is held while the unwanted one is removed. The derived process is the result placed in Fig. 14c that should be executed. This process is only presented for illustration purposes. Otherwise, it is possible to change the business rule and allow the customer to order both the credit card and the checkbook simultaneously.

Analogically to the four issues we highlighted about current variability modeling techniques at end of Sect. 2, we can affirm how our new approach addresses each of which as follows:

1. the design patterns are definite and do not hold any stereotype. Because they are modeled in basic BPMN, the patterns are straightforward and require no foreknowledge besides BPMN basics.

2. the configuration and derivation patterns take the design patterns to a concrete level and make them use-case ready through a clear set of steps.
3. the six patterns are posed with six concise specifications and no redundancy.
4. the meta-model and the two architectures are very flexible. They offer a plethora of possibilities for designing event-based variable partitions or data-based variable partitions with event(s) as variant(s).

In the same vein, our pattern-based approach falls under the BPI research area. Adopting patterns implicitly indicates opting for reuse and therefore saving time and money. Variability design patterns add the quality optimization feature since they model more alternatives to fulfill a wide range of business requirements. Additionally, variability configuration/derivation patterns specify the process of using the design patterns to attain more efficient variable business processes.

4 Illustrative example

Inspired from a real-life process, this illustrative example is a simplified version of the process whose business objective is “order a burger” at a typical Five Guys restaurant.[3] In this example, we show how we deal with variability configuration and derivation using our technique. We also indicate how variable partitions can overlap.

4.1 Variable business process scenario

For legibility and space reasons, we alter some of the characteristics related to the process “order a burger” at a typical Five Guys restaurant such as the number of toppings as well as some other steps in the process flow. We also assume that each customer orders exactly one burger. All in all, the updated scenario of the process is described as follows:

– In the order taken by the *order clerk*, the *customer* selects one of the four types of burgers offered (ham burger, cheese burger, bacon burger and veggie sandwich). She/He then chooses the toppings to have (lettuce, pickles, tomatoes, mayo, grilled onions). Ordering French fries is optional. If and when ordered, the customer chooses between Five Guys style and Cajun Style. The fries are offered in three sizes, i.e., little, medium and large. The drinks are also optional, they come either regular or large.
– Once the order is specified, the customer is given the price and proceeds with the payment.
– After the payment is received, a receipt is printed for the customer with the order number and an order ticket is sent to the clerk who cooks the patties (*patties clerk*) and hands them to the clerk who assembles the burger (*assembling clerk*),

[3] fiveguys.com.

unless the order is about a veggie sandwich, then the step of cooking the patties is skipped.
- When assembled, the clerk transfers the burger to another clerk who handles the fries (*fries clerk*), in case French fries were ordered. Otherwise, it goes straight to the delivery.
- The *delivery clerk* calls the customer's order number and hands the final product to her/him.

4.2 Variable business process at design-time

First of all, six participants are involved in the process (customer, order clerk, patties clerk, assembling clerk, fries clerk and delivery clerk). This condition implies that the process "order a burger" is indeed a collaboration process (a process with two or more participants). Figure 15 shows the configurable process model of "ordering a burger". The model encloses seven variable partitions. We use groups to circumscribe and highlight them. The groups delimit the variable partitions VP1 whose variability objective is "Choose burger type" and is generated from a data-based monomial pattern, VP2 whose variability objective is "Choose toppings" and is generated from a data-based multiple pattern, VP3 whose variability objective is "Order fries" and is generated from a data-based optional pattern, VP4 whose variability objective is "Choose style of fries" and is generated from a data-based monomial pattern, VP5 whose variability objective is "Choose size of fries" and is generated from a data-based monomial pattern, VP6 whose variability objective is "Order drink" and is generated from a data-based optional pattern and VP7 whose variability objective is "Choose size of drink" and is generated from a data-based monomial pattern. Note, the partitions might overlap as it turns out to be the case of (VP3, VP4 and VP5) on one side and (VP6 and VP7) on another side.

4.3 Variable business process at run-time

A variable business process at run-time can be also called a derived business process. Before proceeding with the derivation let us first consider that the business situation specifies the following business requirements:

- VP1 $\Rightarrow$ ham burger
- VP2 $\Rightarrow$ add tomatoes, grilled onions, lettuce
- VP3 $\Rightarrow$ want to order fries
- VP4 $\Rightarrow$ select Five Guys style
- VP5 $\Rightarrow$ select little fries
- VP6 $\Rightarrow$ want to order a drink
- VP7 $\Rightarrow$ select regular drink.

Following the aforementioned requirements, Fig. 16 show the derived process model of "order a burger". We use the link intermediate event of type throw to

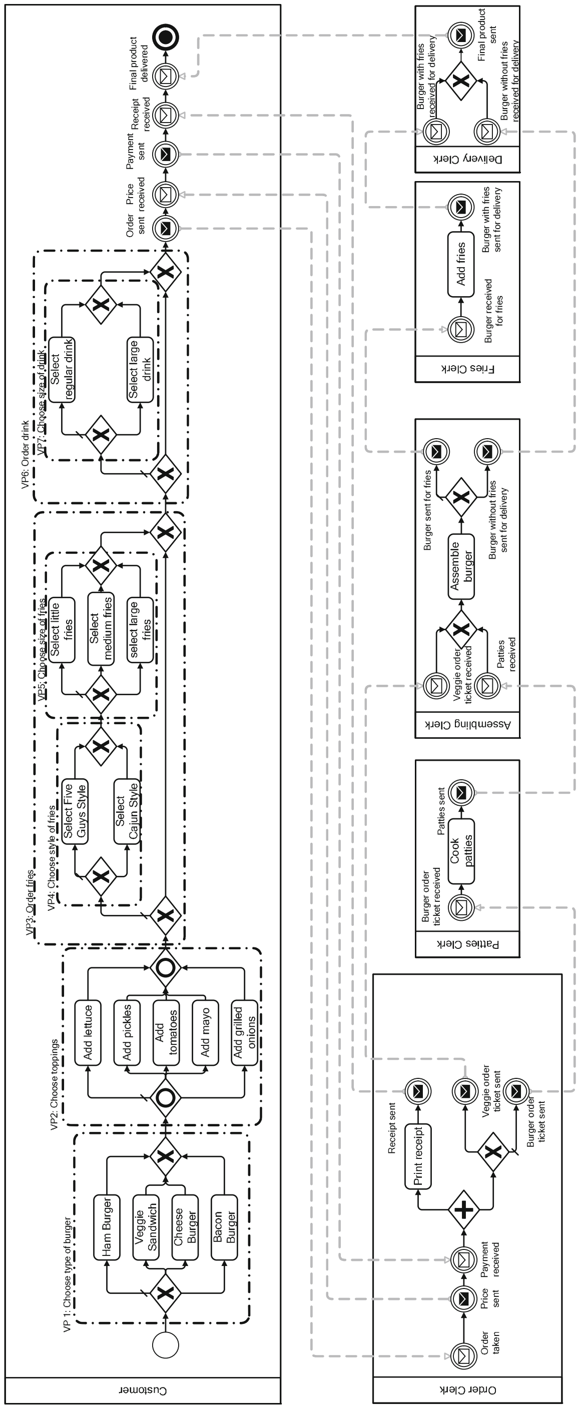


Fig. 15 "Order a burger" variable business process at design-time

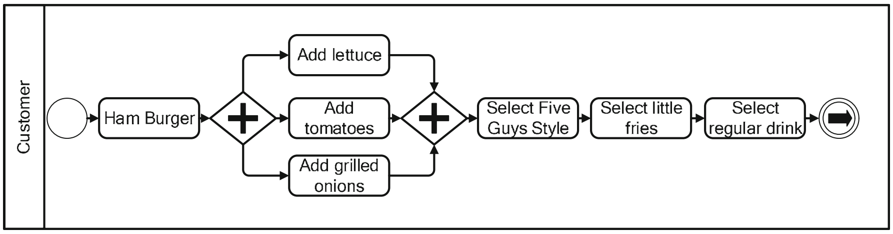


Fig. 16 “Order a burger” VBP at run-time

indicate that the process continues, however, we do not cover it in the diagram since the rest does not hold variable partitions. Note, the inclusive gateway in the variable partition whose variability objective at design-time was “choose toppings” changed to a parallel gateway as explained in the derivation pattern in Sect. 3.4.

5 Validation

The impact of the variability design patterns on an improved business process performance metrics (i.e., time, cost, quality and flexibility) stems from the business process improvement patterns. Introduced by Reijers and Liman Mansar (2005) and Forster (2006), then synthesized by Shtub and Karni (2010, p. 223, 224, 225) and employed to some extent by Dumas et al. (2013), the business process improvement patterns summarize the impact of 43 atomic actions on the performance metrics of an improved business process. The patterns are grouped into five views; process (17 patterns), object (4 patterns), organizational (9 patterns), informatics (2 patterns), IT(4 patterns) and environment (7 patterns).

The BPI patterns are atomic. Each of which describes how the performance metrics of a business process will behave with regard to a particular change. The variability design patterns we propose are a series of applying five business process improvement patterns. These actions lead to a variable partition. The BPI patterns we employ are grouped in Table 6. They belong to three views; process,

Table 6 Business process improvement patterns used in the variability design patterns (+: increase, −: decrease, x: no impact)

BPI pattern code (view)	Description	Time	Quality	Cost	Flexibility
1 (Process)	Add a process step or action	+	+	+	+
2 (Process)	Eliminate a process step or action	−	x	−	x
22 (Organizational)	Increase specialization of process performer	−	+	+	−
24 (Organizational)	Assign flexible roles to perform processes	+	−	+	+
40 (Environment)	Integrate processes by having a common performer	−	+	−	+

organization and environment. Whether data-based or event-based, the variability design patterns are built as follows:

1. Pattern 40 (*integrate processes by having a common performer*): it is about merging processes with commonalities.
2. Pattern 1 (*add a process step or action*): it covers adding a variation point as well as a flow-control data-based gateway (see the architectures in Figs. 10 and 11).
3. Pattern 24 (*assign flexible roles to perform processes*): the data-based or event based gateway serving as a variation point offers a wide range of flexible alternatives for the process performer. Flexibility can be measured by the work of Rolón et al. (2009). It computes how many alternatives a configurable business process can offer. Having multiple alternatives in the configurable business process model implies fulfilling a wide range of business requirements. For $\boldsymbol{F}$ and $\boldsymbol{n}$ being respectively the flexibility measure function and the number of outgoing branches of a divergent gateway (in our case we use XOR and OR), we have:

$$F(XOR) = n \tag{1a}$$

$$F(OR) = 2^n - 1 \tag{1b}$$

4. Pattern 22: (*increase the specialization of the process performer*): each variant in the variation partition (see the architectures in Figs. 10 and 11) increases the specialization of the process performer in the same process flow.
5. Pattern 2 (*eliminate a process step or an action*): at this level the repetitive process steps after the merger are omitted to alleviate the process flow.

Ideally, the improved business process should exhibit a decrease in the time and cost of execution as well as an increase in the quality of the end product/service and the flexibility of the structure. These metrics go hand in hand as portrayed with the Devil's Quadrangle Reijers and Liman Mansar (2005). Although, it may happen that a performance metric is sacrificed in exchange for another as indicated in pattern 22. This is usually the call of the process-centric organization to prioritize one/some metric(s) over other(s).

6 Conclusion, discussion and perspectives

Variability is a key concept to efficiently deal with business processes whose business rules and/or objectives are similar to one another is some ways yet differ in others. After reviewing the literature and highlighting its shortcomings, we define variable business process, variable partition and variability objective. Then, we introduce two types of variability patterns (*product patterns*: variability design patterns and *process patterns*: configuration and derivation patterns). The variability design patterns are a series of BPI patterns that act on the performance metrics of the improved business process while the configuration/derivation patterns make the steps of building a variable business process free from confusion and ambiguity.

With regard to the work done in this paper, we describe a novel formalism for efficiently improving business processes by means of variability. The presented technique facilitates the overall sine qua nons we extract and draw attention to, and, eschews the limitations we highlight about prior work. By adopting the path we pave in the direction to variability patterns, we avoid all the issues discussed at the end of Sect. 2 that may pop up while employing the stereotyped diagrams. The variability patterns grant a systematic reuse of the partitions, a simplification of the modeling task, a production of structured process models as well as a straightforward procedure to narrow down the workflow with regard to each business situation. Still, we extend BPMN v2.0 by defining a new inclusive event-based gateway and underline its importance in handling the event-based workflow. Overall, the configuration pattern along with the business rules generate a configurable business process while the derivation pattern along with the business requirements generate a derived business process that itself will generate an optimal outcome. Equally we have to say, we did center attention on the specificity of variability along the abstraction levels, on how it mutates in the cycle of process models and on how it should be dealt with in each level. As a result, we provide a complete business process improvement technique based on the concept of variability. Thus, we hope we do add some knowledge to this chosen area of work.

As a way to evolve our approach, we intend first on proposing a standardized approach of programming the variability patterns. This, we believe, will lead us to extract more criteria and enlarge our approach to deal with other sectors such as health monitoring or education.

References

Ambler Scott W (1998) Process patterns: building large-scale systems using object technology. Cambridge University Press, Cambridge

Conte A, Fredj M, Hassine I, Giraudin JP, Rieu D (2002) A tool and a formalism to design and apply patterns. In: Object-oriented information systems. Springer, pp 135–146

Derguech W, Gao F, Bhiri S (2012) Configurable process models for logistics case study for customs clearance processes. In: Business process management workshops. Springer, pp 119–130

Dijkman R, Dumas M, Van Dongen B, Käärik R, Mendling J (2011) Similarity of business process models: metrics and evaluation. Inf Syst 36(2):498–516

Dumas M, La Rosa M, Mendling J, Reijers HA (2013) Fundamentals of business process management. Springer, New York

Dumas M, Garcıa-Banuelos L, Polyvyanyy A (2010) Unraveling unstructured process models. In: Business Process Modeling Notation: Second International Workshop, BPMN 2010, Proceedings, vol 67. Springer, Potsdam, 13–14 October, 2010, p 1

Forster F (2006) The idea behind business process improvement: toward a business process improvement pattern framework. BPTrends, April, pp 1–13

Galster M, Avgeriou P (2011) Handling variability in software architecture: problems and implications. In: Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on IEEE, pp 171–180

Gamma E, Helm R, Johnson R, Vlissides J (1994) Design patterns: elements of reusable object-oriented software. Pearson Education, Boston

Gschwind T, Koehler J, Wong J (2008) Applying patterns during business process modeling. In: Business process management, Springer, pp 4–19

Hallerbach A, Bauer T, Reichert M (2010) Capturing variability in business process models: the provop approach. J Softw Maint Evol: Res Pract 22(6–7):519–546

Jeston J, Nelis J (2014) Business process management. Routledge, London

Khan A, Kastner C, Koppen V, Saake G (2011) The pervasive nature of variability in soc. In: Frontiers of Information Technology (FIT), 2011 IEEE, pp 69–74

La Rosa M, Dumas M, Ter Hofstede AHM (2008) Modelling business process variability

Mejia Bernal JF, Falcarin P, Morisio M, Dai J (2010) Dynamic context-aware business process: a rule-based approach supported by pattern identification. In: Proceedings of the 2010 ACM Symposium on applied computing, ACM, pp 470–474

Milani F, Dumas M, Matulevičius R (2012) Identifying and classifying variations in business processes. In: Enterprise, business-process and information systems modeling. Springer, pp 136–150

OMG (2011) Business process model and notation 2.0. Technical report, Object Management Group, Washington DC, USA

Ouyang C, Dumas M, Aalst WM, Ter Hofstede AHM, Mendling J (2009) From business process models to process-oriented software systems. ACM Trans Softw Eng Methodol (TOSEM) 19(1):2

Park J, Yeom K (2011) A modeling approach for business processes based on variability. In: Software Engineering Research, Management and Applications (SERA), 2011 9th International Conference on IEEE, pp 211–218

Recker J, Rosemann M, van der Aalst WMP, Jansen-Vullers M, Dreiling A (2007) Configurable reference modeling languages. In: Reference modeling for business systems analysis. Idea Group Publishing, London, pp 22–46

Recker J, Rosemann M, van der Aalst WMP, Jansen-Vullers M, Dreiling A (2007) Configurable reference modeling languages. In: Reference modeling for business systems analysis. Idea Group Publishing, London, pp 22–46

Reijers Hajo A, Mansar S Liman (2005) Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. Omega 33(4):283–306

Rolón E, Cardoso J, García F, Ruiz F, Piattini M (2009) Analysis and validation of control-flow complexity measures with bpmn process models. In: Enterprise, business-process and information systems modeling. Springer, pp 58–70

Santos E, Castro J, Sanchez J, Pastor O (2010) A goal-oriented approach for variability in bpmn. In: Proceedings of the 13th Workshop on Requirements Engineering-WER, pp 17–28

Santos E, Pimentel J, Castro J, Sánchez J, Pastor O (2010) Configuring the variability of business process models using non-functional requirements. In: Enterprise, business-process and information systems modeling, pp 274–286

Schnieders A, Puhlmann F (2006) Variability mechanisms in e-business process families. In: 9th International Conference on Business Information Systems (BIS 2006), vol 85, pp 583–601

Shtub A, Karni R (2010) ERP: the dynamics of supply chain and process management. Springer Science & Business Media, heidelberg

Sinnema M, Deelstra S (2007) Classifying variability modeling techniques. Inf Softw Technol 49(7):717–739

van Der Aalst WM, Ter Hofstede AHM, Kiepuszewski B, Barros AP (2003) Workflow patterns. Distrib Parallel Databases 14(1):5–51

van Dongen B, Dijkman R, Mendling J (2008) Measuring similarity between business process models. In: Advanced information systems engineering. Springer, pp 450–464

van Eijndhoven T, Iacob ME, Ponisio ML (2008) Achieving business process flexibility with business rules. In: Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE, pp 95–104

Weidmann M, Koetter F, Kintz M, Schleicher D, Mietzner R (2011) Adaptive business process modeling in the internet of services (abis). In: ICIW 2011. The Sixth International Conference on Internet and Web Applications and Services, pp 29–34