Ayora Esteras, C.; Torres Bosch, MV.; De La Vara González, JL.; Pelechano Ferragud, V. (2016). Variability management in process families through change patterns. Information and Software Technology. 74:86-104. doi:10.1016/j.infsof.2016.01.007.

Variability Management in Process Families through Change Patterns

Clara Ayora[a,1], Victoria Torres[a], Jose Luis de la Vara[b], Vicente Pelechano[a]

[a] *Centro de Investigación en Métodos de Producción de Software, Universitat Politècnica de València, Camino de Vera s/n, 46022 València, Spain, Phone: +34 963 87 70 07, Ext. 83533*

[b] *Computer Science Department, Carlos III University of Madrid, Avda. Universidad 30, 28911 Leganés (Madrid), Spain, Phone:+34 916 24 91 15*

Abstract

Context: The increasing adoption of process-aware information systems together with the high variability in business processes has resulted in collections of process families. These families correspond to a business process model and its variants, which can comprise hundreds or thousands of different ways of realizing this process. Managing process variability in this context can be very challenging, labor-intensive, and error-prone, and new approaches for managing process families are necessary.

Objective: We aim to facilitate variability management in process families, ensure process family correctness, and reduce the effort needed for such purposes.

Method: We have derived a set of change patterns for process families from variability-specific language constructs identified in the literature. For validation, we have conducted a case study with a safety standard in which we have measured the number of operations needed to model and evolve the variability of the standard with and without the patterns.

Results: We present 10 change patterns for managing variability in process families and show how they can be implemented. The patterns support the

Email addresses: cayora@pros.upv.es (Clara Ayora), vtorres@pros.upv.es (Victoria Torres), jvara@inf.uc3m.es (Jose Luis de la Vara), pele@pros.upv.es (Vicente Pelechano)

[1] Corresponding author.

modeling and evolution of process families and ensure process family correctness by automatically introducing and deleting modeling elements. The case study results show that the application of the defined change patterns can reduce the number of operations when modeling a process family by 34% and when evolving it by 40%.
Conclusions: The application of the change patterns can help in effectively modeling and evolving large and highly-variable process families. Their application can also considerably reduce variability management effort.

Keywords: business process modeling, business process variability, process-aware information system, process family, change patterns

---

1. Introduction

Process-aware information systems (PAISs) manage, execute, and analyze the business processes of an enterprise (e.g., sales business processes) based on explicitly specified *process models* [55, 19]. The increasing adoption of PAISs during the last decade has resulted in large process model repositories [54, 15], which usually comprise collections of related *process model variants* (*process variants* for short) [40]. Process variants pursue the same or a similar business objective (e.g., product sale) and can have activities (and their ordering constraints) in common. Nevertheless, process variants differ in their *application context*, such as the regulations to comply with in different countries, and some activities may be relevant only for certain contexts [10, 15, 40]. All the context factors causing process variability are typically known at design time [40, 17].

A collection of related process variants can be referred to as process family [40]. In practice, such a family may comprise hundreds or thousands of process variants. *Hallerbach et al.* describe a process family from automotive that comprises over 900 process variants [26] and *Li* reports on over 90 variants for medical examinations [31]. Finally, check-in procedures at airports are similar irrespective of the airport or airline, but variations exist depending on the type of check-in (e.g., online or at the counter) or passenger (e.g., unaccompanied minors) [10]. Example 1 describes the check-in process, discussing its different sources of variability. We will use this process as a running example throughout the paper.

Example 1 (Check-in process). Numerous variations exist for this process depending on different factors. For example, variability is caused by the type of passenger (e.g., unaccompanied minors and handicapped people might require extra assistance and special seats). Another source of variability includes the flight destination (e.g., accommodation information is required when traveling to the USA). Finally, depending on the type of luggage (e.g., bulk or overweight luggage), the process may differ because an extra fee has to be paid.

Figures 1 and 2 show six simplified variants of the check-in process represented in BPMN (*Business Process Modeling Notation*) [12]. The variants have been modeled in collaboration with domain experts. While these process variants share commonalities (colored in grey), they also show differences. *Variants 1* and *2* (cf. Fig. 1) presume that the check-in is done online by the passenger, who is identified and assigned in a seat. *Variant 1* describes the case of a passenger flying from Europe to the USA, which requires accommodation information as well as filling in the electronic form for travel authorization (i.e., ESTA form). An electronic boarding card is printed and the passenger drops off the luggage at the business class counter. The payment of an extra fee at the ticket office is required in *Variant 2* due to luggage overweight. For *Variant 3*, the check-in is done at the self-servicing machine and the luggage is dropped off at the fast bag-drop counter. Check-in for these three process variants becomes available 23 hours before departure.

In contrast, *Variants 4-6* (cf. Fig. 2) represent the check-in process accomplished at the counter at the airport. *Variant 4* describes the check-in for an unaccompanied minor (UM). A special seat is assigned, an extra form is filled in, and a copy of the boarding card is required for the relative accompanying the minor to the gate. *Variant 5* refers to a handicapped passenger requiring extra assistance to accompanying him, whereas *Variant 6* corresponds to the check-in process of a passenger carrying bulk luggage. In these three process variants, check-in may only be performed at maximum 3 hours before departure, once the counters are opened. The boarding card is printed in paper format.

1.1. Problem Statement

Modeling and evolving process families and ensuring their correctness can be very challenging mainly due to their size and heterogeneous application

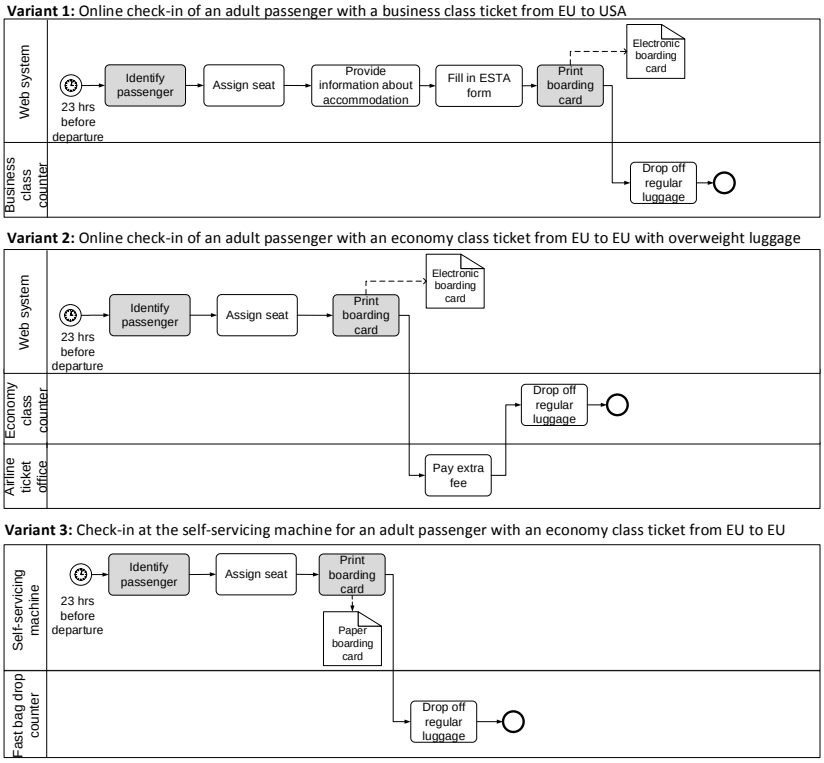

Figure 1: Variants of the check-in process (1)

context (e.g., type of passenger, flight destination, and type of luggage) [40]. This has resulted in the development of approaches that support process variability along the process lifecycle, such as C-EPC (Configurable *Event-driven Process Chain*) [23]. These approaches enable the analysis, design, configuration, enactment, diagnosis, and evolution of process families, and specify process variants by means of configurable process models that represent a complete process family [10]. By treating variability as a first class citizen, configurable process models contribute to avoiding model redundancies, fostering model reusability, and reducing modeling efforts [40]. Fig. 3 shows a configurable process model for the check-in process family, created with C-EPC (introduced in Sect. 2.2).

When using process variability approaches, PAIS engineers need assistance because they have to manually model and manage all the elements

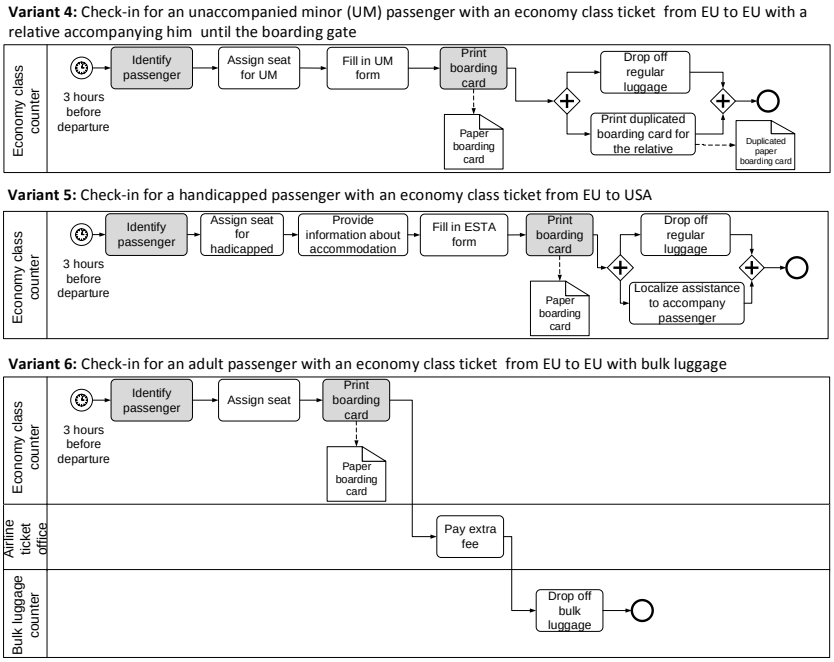

Figure 2: Variants of the check-in process (2)

and dependencies of a configurable process model individually [26]. In the model of Fig. 3, a PAIS engineer needs to represent that the activity *Print duplicated boarding card for the relative* for the relative is allowed only if a seat for UM has been assigned before. Modeling such constraints manually with the primitives currently offered by the existing process variability approaches can be tedious and error-prone, especially when a process family comprises a high number of variants and with many dependencies. There is a lack of approaches to deal with this variability in an explicit manner, especially at a level of abstraction higher than the one provided by the existing process variability approaches [10].

The use of modeling patterns (reusable solutions to a commonly occurring problem [53]) is a promising way to address these issues. For example, *adaptation patterns* have been proposed for creating and managing (individual) process models [53]. These patterns allow creating and modifying process models and ensure *correctness by construction*. They also provide systematic means for realizing change operations and aim to reduce modeling efforts

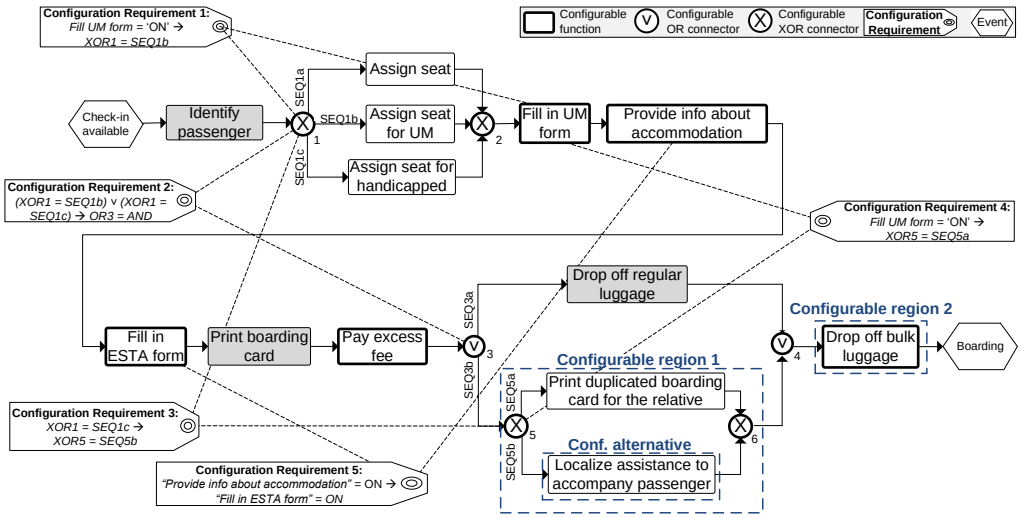

Figure 3: Configurable process model for the check-in process in C-EPC notation

[16]. However, these patterns do not deal with process variability in an explicit manner [7] and hence are not accurate for process family management [9].

1.2. Contribution

The major contributions of this paper are twofold:

1. *A set of 10 change patterns specifically tailored for managing variability in process families.* Our change patterns for process families (hereafter referred to as *CP4PF*) enable the modeling and evolution of the variability of a configurable process model, and are intended to reduce the effort needed for such purposes. CP4PF have been derived from *variability-specific language constructs* used in the literature to capture variability within a process family. The patterns allow the insertion, deletion, and modification of these constructs and hence CP4PF can be implemented in process variability approaches supporting such constructs. As an example, we present the implementation of CP4PF in C-EPC. This implementation allows us to show that the proposed patterns support process variability management and can ensure process family correctness by providing systematic means for introducing and deleting modeling elements. For example, a pattern can facilitate the insertion of the function *Drop off bulk luggage* in the model of Fig. 3. A

PAIS engineer would indicate the position of the function in the model, and the implementation of the pattern would automatically take all the rest of necessary actions for its correct insertion.

2. *Evidence of feasibility and effort reduction for variability management of real process families by applying CP4PF.* We have conducted a case study with a safety standard with a high degree of variability. Its results show that CP4PF reduce the effort for modeling and evolving a highly-variable process family in comparison with state-of-the-art approaches. To the best of our knowledge, it is the largest case study that has been conducted on process variability management, the first one that has dealt with process family evolution, and the first one that has compared the results of different approaches.

This paper extends the work presented in [9], where we described only three of the patterns as a proof of concept of their potential for managing process variability. According to the variability-specific language constructs identified in [10], this set of three patterns has been revised and extended in this paper to properly support all the constructs. As a result, we have obtained a set of ten patterns, which are presented and implemented in detail in this paper. We have also put the patterns into practice in a case study on a real scenario with a high degree of variability. Empirically validating the patterns is crucial to assess their benefits.

The remainder of the paper is organized as follows. Section 2 presents the background of CP4PF. Section 3 describes CP4PF. Section 4 reports on the case study for validating CP4PF. Finally, Section 5 reviews related work and Section 6 concludes the paper with a summary and outlook.

2. Background

This section presents the background that we use in Sect. 3 to define CP4PF and thus is necessary to understand the definition of the patterns. More concretely, we describe adaptation patterns, C-EPC, and the variability-specific language constructs that serve as a basis for our CP4PF. Adaptation patterns are used as a basis for implementing some of our CP4PF, we use C-EPC for illustrating how the defined patterns can be implemented in a process variability approach, and variability-specific language constructs have been used as a basis for pattern derivation.

2.1. Adaptation patterns

Adaptation patterns (APs) [53] allow users to structurally change a process model using high–level change operations (e.g., to insert a process fragment between two nodes) instead of low level change primitives (e.g., add or delete node). Defining a set of pre- and post–conditions for the operations, APs allow a PAIS to guarantee soundness when applying the respective operations [53]. In addition, they have served as basis for implementing changes at different phases of the process lifecycle (e.g., process model creation [25] and configuration [26]).

There are 14 APs. They can be applied along to the entire process lifecycle (i.e., process analysis, design, configuration, enactment, diagnosis, and evolution) and are well suited for realizing process changes at both design and runtime. In particular, AP1 and AP2 allow inserting and deleting process fragments (e.g., insert/delete activity), respectively. In turn, moving and replacing fragments is supported by AP3 (Move Process Fragment), AP4 (Replace Process Fragment), AP5 (Swap Process Fragment), and AP14 (Copy Process Fragment). AP6 and AP7 allow adding or removing levels of hierarchy, AP8-AP12 support adaptations of control dependencies: embedding process fragments in loops (AP8), parallelizing process fragments (AP9) or embedding them in a conditional branch (AP10), and adding/removing control dependencies (AP11, AP12). Finally, AP13 allows changing transition conditions.

Although APs are well suited for creating and managing (individual) process models, they are not sufficient to cope with the complexity that process variability introduces [9]. Thus, our CP4PF complement APs by addressing the variability-specific needs of process families.

2.2. Configurable EPC

A possible way of realizing a configurable process model is to enrich a process model with *configurable nodes*. A modeling language supporting this approach is C-EPC, which extends EPC by introducing configurable elements [23]. We select this approach for implementing our CP4PF because it is well established and there exists a mature tool support for it [10]. C-EPC is also the most common process variability approach in the literature [10].

For better illustrating C-EPC, we refer to the check in process depicted in Fig. 3. Configurable nodes are depicted with a thicker line. We do not add intermediate events between functions in order to keep the size of the

configurable process model as small as possible. This helps us mitigate possible undesirable effects on understandability and likelihood of errors due to model size [33]. In addition, practitioners recommend not to include events between functions in EPC for the sake of simplicity [5]. Thus, configurable nodes correspond to the variable process fragments that have one single entry and single exit (SESE fragment). They may have two different forms. On the one hand, a SESE fragment may consist of a splitting *configurable connector*, immediately followed by a set of branches representing configuration options, and a joining configurable connector (e.g., Configurable region 1 in Fig. 3). Alternatively, a SESE fragment may correspond to a *configurable function* (e.g., Configurable region 2 in Fig. 3). Depending on the context, such functions may be configured as ON (the function shall be kept in the configured process model), OFF (the function shall not be included in the configured process model), or OPT (the function shall be conditionally included in the configured process model deferring the decision about its execution to enactment time). In turn, SESE fragments representing the different configuration options are included as branches between two configurable connectors (e.g., *Localize assistance to accompany passenger* in Fig. 3). Further, the application context of each process variant is represented in a *questionnaire model* [29] (cf. Fig. 4).

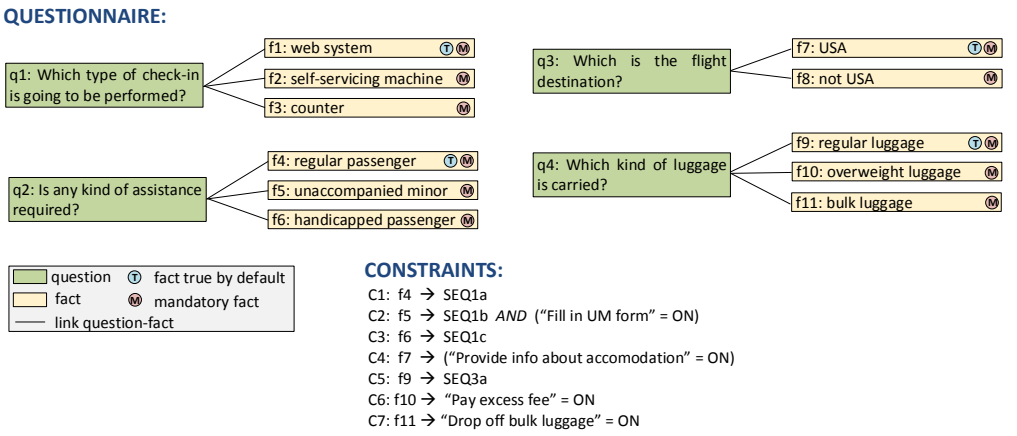

Figure 4: Questionnaire model and associated constraints for the check-in process

This questionnaire comprises a set of *questions* that capture the application context in terms of domain aspects (e.g., type of check-in). Each question

refers to a set of *facts* (e.g., check-in in a web system or in a self-servicing machine). Additionally, a set of *constraints* in the form of propositional logic expressions can be defined over the facts. They represent the mapping between the questionnaire model and the configurable process model. Thus, as the questions are answered, the selected facts, in combination with the defined constraints determine which actions should be performed to configure the configurable process model (i.e., which configurations are taken in each configurable node). Finally, semantic constraints with respect to the configuration of configurable functions and connectors (e.g., mutual exclusion, inclusion) may be specified in terms of *configuration requirements* linked to the configurable nodes. For example, *Configuration Requirement 1* in Fig. 3 states that the configurable function *Fill in UM form* is only included if SEQ1b is selected in XOR1 (i.e., activity *Assign seat for UM* is selected).

2.3. Language Constructs for Process Variability

We derived our CP4PF based on the *variability-specific language constructs* that are used to capture the variability within a process family. These constructs were identified conducting a systematic literature review. It included 34 different process variability approaches [10], where five variability-specific language constructs were identified: *configurable region, configuration alternative, configuration context condition, configuration constraint,* and *configurable region resolution time*. These constructs capture the expressiveness needed to represent process variability in a configurable process model. Although other studies might use different terminology (e.g., *configurable region* vs. *variation point*) and realize the constructs in different ways, these five languages constructs are the most prevalent in the literature on process variability. Thus, we take these constructs as a basis for defining the CP4PF. In the following, we present the identified constructs and illustrate how they are realized with C-EPC. Further details on the systematic literature review and its results can be found in [10].

Configurable Region Language Construct (LC1). A configurable region corresponds to an area of a configurable process model for which different configuration choices exist, depending on the application context. A configurable region in C-EPC may be specified by process fragments with exactly one entry and one exit (i.e., SESE fragment) delimited by two configurable connectors (e.g., Configurable region 1 in Fig. 3), or it may be a configurable function (e.g., Configurable region 2 in Fig. 3).

Configuration Alternative Language Construct (LC2). A configuration alternative corresponds to a particular configuration choice that may be selected in the context of a specific configurable region (LC1). In C-EPC, a configuration alternative is specified by a SESE fragment that may be included as a branch between two configurable connectors (e.g., *Localize assistance to accompany passenger* in Configurable region 1 in Fig. 3).

Configuration Context Condition Language Construct (LC3). A configuration context condition defines the conditions under which a particular configuration alternative (LC2) of a configurable region (LC1) shall be selected. Regarding C-EPC, configuration context conditions are captured as facts in the questionnaire model (e.g., f1 in Fig. 4).

Configuration Constraint Language Construct (LC4). A configuration constraint is defined as a restriction regarding the selection of configuration alternatives (LC2). The constraints are based on semantic restrictions to ensure the proper use of the defined configuration alternatives (e.g., exclusion or inclusion relationships). In C-EPC, a configuration constraint is specified by a configuration requirement. This requirement is linked to the configurable nodes that delimit the configurable region to which the respective configuration alternatives belong. For example, *Configuration Requirement 4* in Fig. 3 states that if the configurable function *Fill in UM form* is included in the model, then SEQ5a in XOR5 is chosen.

Configurable Region Resolution Time Language Construct (LC5). The configurable region resolution time allows process designers to distinguish between configurable regions (LC1) whose configuration depends on either the initial or the current context of a process instance (i.e., configuration or enactment time). In C-EPC, configurable region resolution time is supported in the configurable functions since they can be configured to OPT, deferring their configuration to enactment time when the context information is available (e.g., configurable function *Pay excess fee* in Fig. 3).

3. CP4PF: Change Patterns for Process Families

Taking as a basis the variability-specific language constructs introduced above, this section presents the set of change patterns for modeling and evolv-

ing process families. Since adaptation patterns have been effectively applied in (individual) process models previously [53], we follow this perspective and provide patterns to deal with process variability in an explicit manner (i.e., based on the variability-specific constructs). In addition, our patterns address process variability at a level of abstraction higher than the one provided by the existing process variability approaches [10] (e.g., C-EPC). In fact, we intend to provide a set of generic patterns that can be implemented in any of these approaches [8].

For deriving the patterns, we applied the three basic operations over the identified variability-specific language constructs: *insertion, deletion,* and *modification* (cf. Table 1). As a result, we obtained four patterns to add variability-specific language constructs to a configurable process model (CP1, CP3, CP5, and CP8), four patterns to remove them (CP2, CP4, CP6, and CP9), and two patterns to modify them (CP7 and CP10). We do not consider patterns for modifying configurable regions, configuration alternatives, and configuration constraints. These modifications can be realized combining other change patterns and existing adaptation patterns. For example, modifying a configuration alternative may be implemented applying patterns CP3 and CP4. However, we defined CP7 and CP10 in order to modify in a more efficient way the variability-specific language constructs they affect (i.e., configuration context condition and configurable region resolution time).

CP1: Insert Configurable Region
CP2: Delete Configurable Region
CP3: Insert Configuration Alternative in a Configurable Region
CP4: Delete Configuration Alternative from a Configurable Region
CP5: Insert Configuration Context Condition of a Configuration Alternative
CP6: Delete Configuration Context Condition of a Configuration Alternative
CP7: Modify Configuration Context Condition of a Configuration Alternative
CP8: Insert Configuration Constraint between Configuration Alternatives
CP9: Delete Configuration Constraint between Configuration Alternatives
CP10: Modify Configurable Region Resolution Time

Table 1: CP4PF - change patterns for process families

We describe CP4PF in detail in Figs. 4–13. For each pattern, we provide a name, a brief description, a description of the problem addressed, an illustrative example, and the design choices (indicating pattern variants). For

example, CP1 (cf. Sect. 3.1) presents three design choices: (1) insert a configurable region as a new region with a set of new configuration alternatives, (2) insert it by transforming a commonality into a configuration alternative (i.e., a common process fragment now is only applied in a specific process variant), or (3) insert it by transforming a set of commonalities into a set of configuration alternatives. To demonstrate that the patterns–despite their intended generic nature–still cover the essence of process variability, we show their implementation in C-EPC. For example, for each design choice for CP1, we indicate how it can be implemented in C-EPC. This implementation in C-EPC guarantees *correctness-by-construction* in terms of structure and behavior (i.e., modeling elements are not introduced incorrectly) [20, 55]. For example, we guarantee that deadlocks are not introduced. In addition, for some cases pattern implementation is based on the use of adaptation patterns (e.g., implementation of design choice 1 in Fig. 5). This allows us to promote reuse between the new patterns and the existing ones. We further provide implementation details distinguishing between (i) configurable functions and (ii) configurable connectors since both allow representing configurable regions in C-EPC. In addition, we provide information about the parameters needed for each pattern. For example, realizing CP1 requires (1) the precise position in the configurable process model where the configurable region shall be inserted and (2) the configuration alternatives to be inserted in the configurable region (if needed). This information is highlighted in gray in the figures.

3.1. CP1: Insert Configurable Region

Description: A configurable region is added in a configurable process model.

Problem addressed: At a certain position in the configurable process model, different configuration alternatives that exist are not reflected in the configurable process model so far. Hence, a configurable region covering these configuration alternatives shall be added.

Example: The way how boarding cards are handled depends on the type of check-in (e.g., paper-based vs. electronic boarding cards). Assume that the configurable process model has not considered these configuration alternatives yet. Hence, a configurable region needs to be added to reflect this variability.

Design choices (DC):

(DC1) Insertion as a new configurable region with up to n configuration al-

ternatives (n $\geq$ 0)
(DC2) Insertion as a new configurable region by transforming a common process fragment into a configuration alternative
(DC3) Insertion as a new configurable region by transforming existing process fragments into a set of configuration alternatives

Implementation in C-EPC:

For DC1, CP1 is realized by
1. applying adaptation pattern AP1 (Insert Process Fragment) to insert the configurable region using either (i) a configurable function (i.e., configurable function *Print paper boarding card*) or (ii) two configurable connectors (i.e., split and join) at the precise position where the configurable region should be located (i.e., after activity *Assign seat*), and
2. applying repeatedly CP3 (Insert Configuration Alternative in a Configurable Region) to insert a process fragment representing the configuration alternative (only relevant for configurable connectors), i.e., the configuration alternative is added as a branch between the two configurable connectors delimiting the configurable region (i.e., activity *Print paper boarding card*).

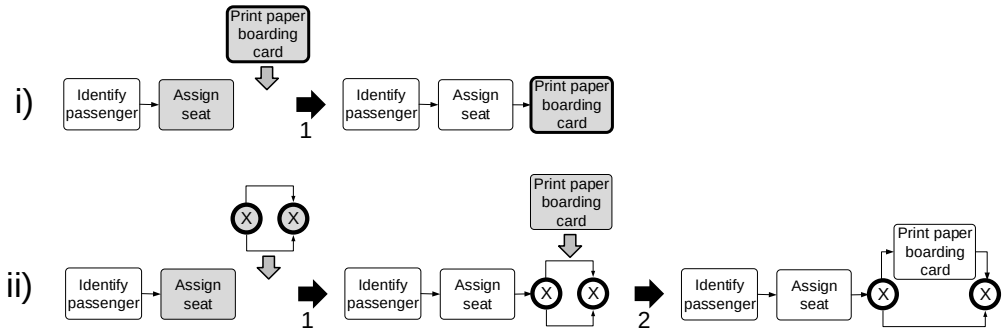

Figure 5: CP1: Design choice 1 implemented in C-EPC

For DC2, CP1 is realized by
1. applying adaptation pattern AP1 (Insert Process Fragment) to insert the configurable region using either (i) a configurable function (i.e., configurable function *Assign seat* or (ii) two configurable connectors (i.e., split and join) at the precise position where the configurable region should be located (i.e., after activity *Assign seat*),
2. applying adaptation pattern AP2 (Delete Process Fragment) to delete

the common process fragment from its current position (i.e., activity *Assign seat*), and
3. applying CP3 (Insert Configuration Alternative in a Configurable Region) to re-insert the common process fragment as a configuration alternative of the configurable region (only relevant for configurable connectors), i.e., the alternative is added as a branch between the two configurable connectors delimiting the configurable region (i.e., activity *Assign seat*).

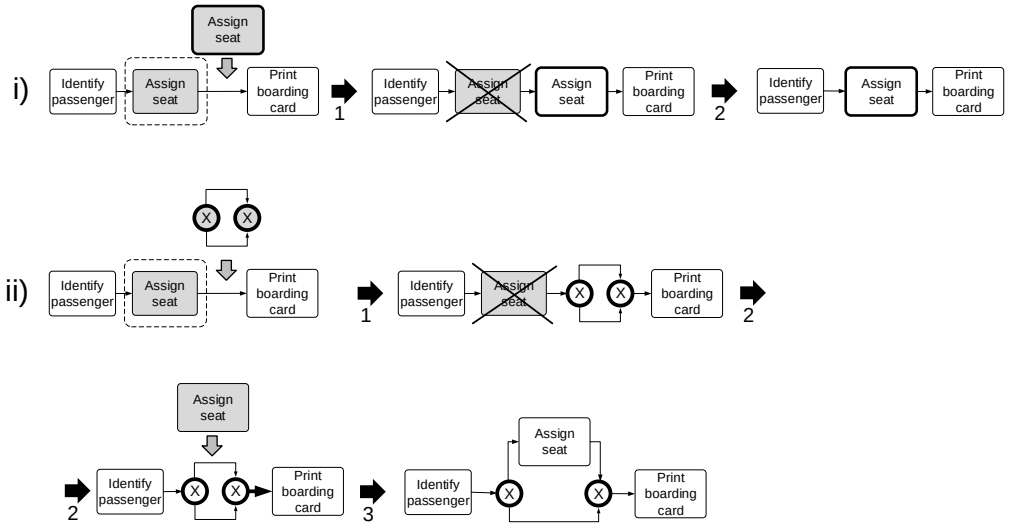

Figure 6: CP1: Design choice 2 implemented in C-EPC

For DC3, CP1 is realized by
1. applying adaptation pattern AP1 (Insert Process Fragment) to insert the configurable region (only relevant for configurable connectors) at the precise position where the configurable region should be located (i.e., after the join XOR gateway),
2. applying adaptation pattern AP2 (Delete Process Fragment) to delete the existing process fragment from its current position, and
3. applying repeatedly CP3 (Insert Configuration Alternative in a Configurable Region) once per configuration alternative to re-insert the existing process fragments as configuration alternatives of the configurable region, i.e., each process fragment is added as a branch between the two configurable

connectors delimiting the configurable region (i.e., activity *Print paper boarding card* is inserted as one alternative and activity *Pring electronic boarding card* as another one).

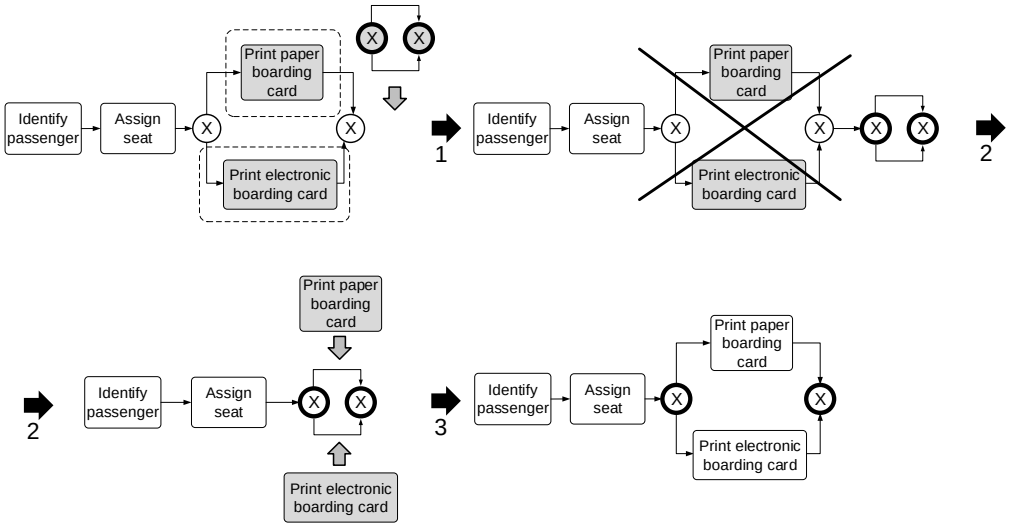

Figure 7: CP1: Design choice 3 implemented in C-EPC

Parameters:
– the position in the configurable process model where the configurable region shall be inserted
– the configuration alternative(s) to be added to the configurable region

3.2. CP2: Delete Configurable Region

Description: A configurable region of a configurable process model is deleted.
Problem addressed: A configurable region is no longer needed and thus it is deleted.
Example: Assume that a configurable region, capturing the variability for obtaining a boarding card, exists (i.e., paper vs. electronic document). However, in order to save money, the airline now only offers the electronic-based boarding card (i.e., other configuration alternatives are no longer offered) and hence the configurable region is no longer needed.

Design choices (DC):
(DC1) Deletion by removing all the configuration alternatives
(DC2) Deletion by keeping exactly one of the configuration alternatives (i.e., the configuration alternative remains as a common process fragment)
(DC3) Deletion by keeping the set of configuration alternatives
Implementation in C-EPC:
For DC1, CP2 is realized by
1. applying repeatedly change pattern CP4 (i.e., Delete Configuration Alternative in a Configurable Region) to delete each existing configuration alternative; i.e., once per configuration alternatives (only relevant for configurable connectors, i.e., activity *Print paper boarding card*), and
2. applying adaptation pattern AP2 (Delete Process Fragment) to delete the configurable region in form of either (i) a configurable function or (ii) two configurable connectors (i.e., split and join).

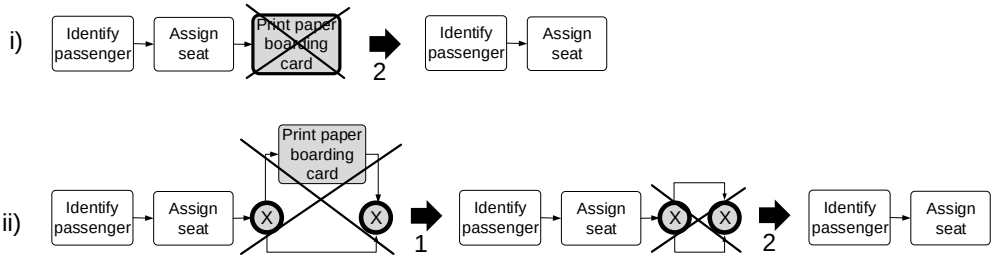

Figure 8: CP2: Design choice 1 implemented in C-EPC

For DC2, CP2 is realized by
1. applying repeatedly CP4 (Delete Configuration Alternative in a Configurable Region) to delete the existing configuration alternatives of the configurable region (only relevant for configurable connectors, i.e., activity *Assign seat*),
2. applying adaptation pattern AP2 (Delete Process Fragment) to delete the configurable region in form of either (i) a configurable function or (ii) two configurable connectors (i.e., split and join), and
3. applying adaptation pattern AP1 (Insert Process Fragment) to re-insert the remaining configuration alternative as a (common) process fragment in the exact position where the configurable region was located (i.e., activity

Assign seat).

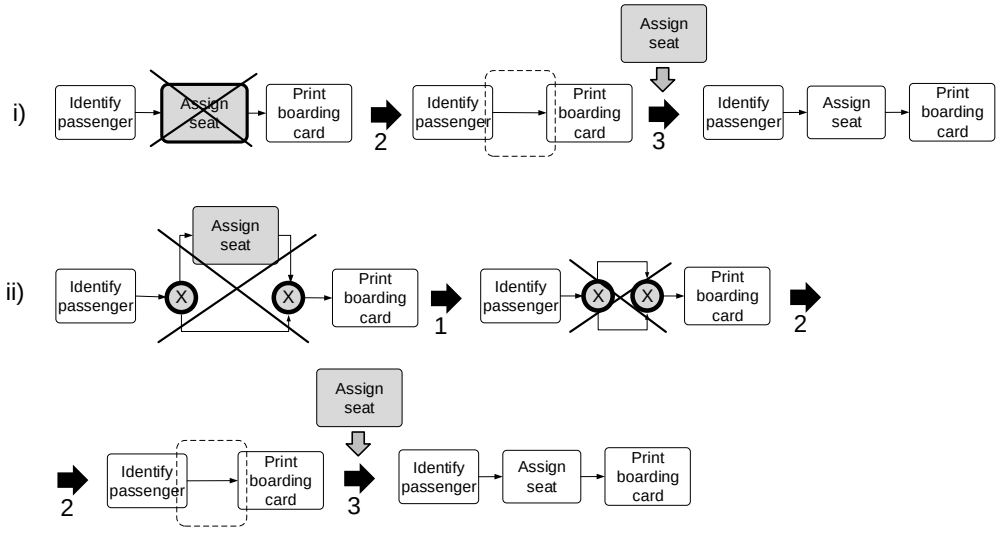

Figure 9: CP2: Design choice 2 implemented in C-EPC

For DC3, CP2 is realized by
1. applying adaptation pattern AP2 (Delete Process Fragment) to delete the existing process fragment (including the configurable region and its alternatives) from its current position,
2. applying adaptation pattern AP1 (Insert Process Fragment) to re-insert at the precise position where the configurable region was located a process fragment consisting of a two non-configurable connectors, and
3. applying repeatedly adaptation pattern AP1 (Insert Process Fragment) to re-insert the deleted configuration alternatives as branches between the two recently added non-configurable connectors (i.e., activity *Print paper boarding card* is inserted as one branch and activity *Print electronic boarding card* as another one).

Parameters:
– the configurable region to be deleted
– the configuration alternative(s) that should be kept

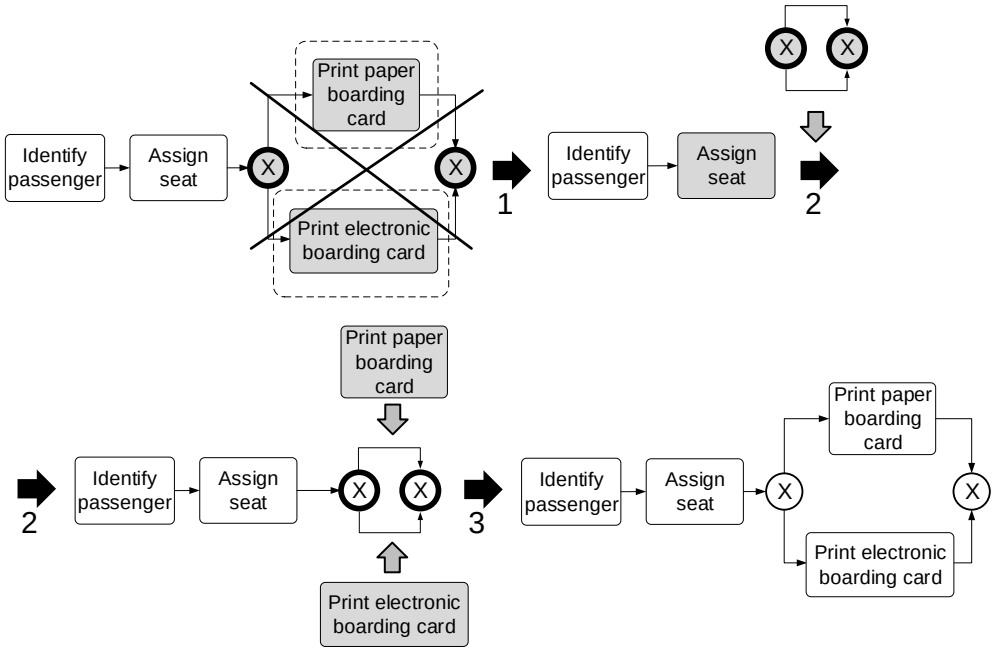

Figure 10: CP2: Design choice 3 implemented in C-EPC

3.3. CP3: Insert Configuration Alternative in a Configurable Region

Description: A configuration alternative is added in a specific configurable region of a configurable process model.

Problem addressed: For a specific configurable region of the configurable process model, existing configuration alternatives do not cover all possible configuration choices so far.

Example: Assume that a configurable region, capturing the variability for obtaining a boarding card, exists (i.e., paper vs. electronic document). Assume further that the airline now wants to offer the possibility of obtaining the boarding card for smart phones as well. Thus, an alternative shall be added to this configurable region.

Implementation in C-EPC:

CP3 is realized by applying adaptation pattern AP1 (Insert Process Fragment) to insert the process fragment representing the configuration alter-

native, i.e., the configuration alternative is added as a branch between the two configurable connectors delimiting the configurable region (i.e., activity *Print smart boarding card*).

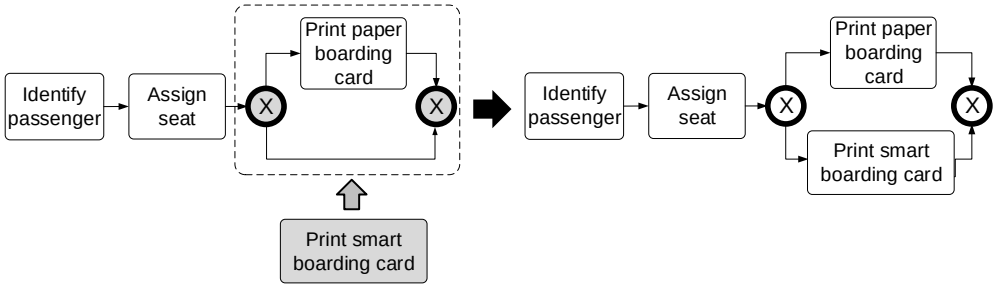

Figure 11: CP3 implemented in C-EPC

Parameters:
– the configurable region to which the configuration alternative belongs
– the configuration alternative to be inserted

3.4. CP4: Delete Configuration Alternative from a Configurable Region

Description: A configuration alternative is removed in a specific configurable region of a configurable process model.
Problem addressed: A configuration alternative is no longer needed and thus it is deleted.
Example: Assume that a configurable region capturing the variability for obtaining a boarding card exists (i.e., paper vs. electronic document). Assume further that for economic reasons, the airline does not offer paper-based boarding cards anymore allowing only electronic and mobile phone ones. Thus, the configuration alternative printing a paper boarding card is no longer needed.
Implementation in C-EPC:
CP4 is realized by applying adaptation pattern AP2 (Delete Process Fragment) to delete the process fragment representing the configuration alternative, i.e., the configuration alternative is deleted as a branch between the two configurable connectors delimiting the configurable region (i.e., activity *Print paper boarding card*). If the configuration alternative is associated with

configuration requirements, these may be deleted as well by applying CP9 (Delete Constraint between Configuration Alternatives), i.e., Configuration requirement 1.

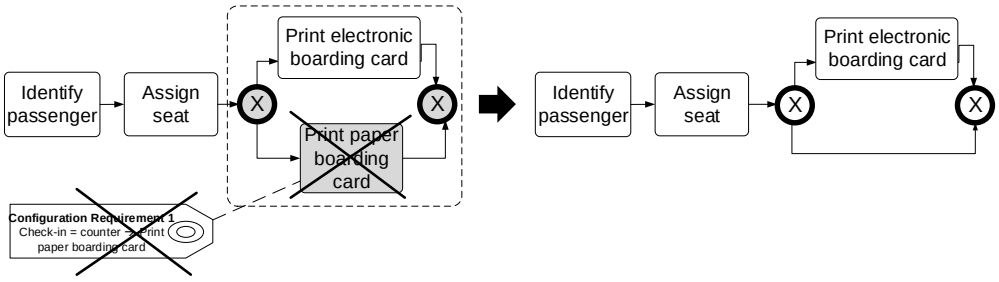

Figure 12: CP4 implemented in C-EPC

Parameters:
– the configurable region to which the configuration alternative belongs
– the configuration alternative to be deleted

3.5. CP5: Insert Configuration Context Condition of a Configuration Alternative

Description: A context condition related to a configuration alternative of a configurable region is added to define when the configuration alternative shall be selected.
Problem addressed: A context condition is added to a configurable process model to specify the condition under which a particular configuration alternative shall be selected.
Example: A passenger who carries bulk luggage must pay an extra fee (where bulk luggage refers to the new context condition).
Implementation in C-EPC:
Since the configuration context conditions are included in a separate questionnaire model, CP5 is realized by adding a new fact to the question referred to the application context of the condition. If the questionnaire model does not include a question for the specific application context, a new question should be added. In addition, if the new fact implies new constraints, these should be included as well.

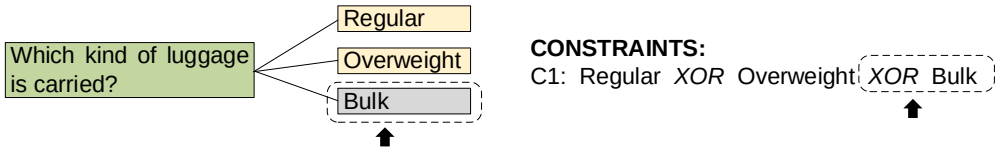

Figure 13: CP5 implemented in C-EPC

Parameters:
– the context condition to be inserted

3.6. CP6: Delete Configuration Context Condition of a Configuration Alternative

Description: A context condition related to a configuration alternative of a configurable region of a configurable process model is deleted.
Problem addressed: A configuration context condition is no longer needed for selecting a configuration alternative in a configurable region and thus it is deleted.
Example: VIP passengers do not have to pay a fee for luggage overweight so far. However, the airline decides that from now on all passengers must pay such fee.
Implementation in C-EPC:
Since the configuration context conditions are included in a separate questionnaire model, CP6 is realized by removing an existing fact from the question referred to the application context of the condition. If the question of the removed fact is not used by any other configuration alternatives, the question should be removed as well. In addition, the constraints referred to the removed fact should be removed as well.

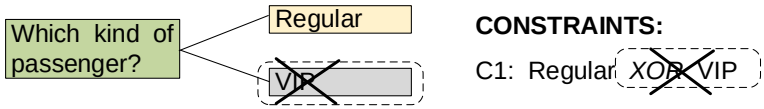

Figure 14: CP6 implemented in C-EPC

Parameters:
– the context condition to be deleted

3.7. CP7: Modify Configuration Context Condition of a Configuration Alternative

Description: A context condition related to a configuration alternative of a configurable region of a configurable process model is modified.
Problem addressed: A context condition is no longer adequate and shall be modified in the configurable process model.
Example: The payment of an extra fee is required when luggage weight exceeds over 20kg. Due to new business goals, this is changed and the extra fee is only required when the luggage weights more than 15kg.
Implementation in C-EPC:
Since the configuration context conditions are included in a separate questionnaire model, CP7 is realized by modifying the fact or the constraint referred to the application context of the alternative.

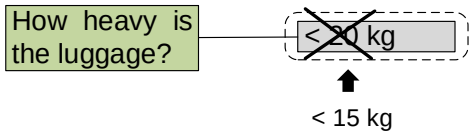

Figure 15: CP7 implemented in C-EPC

Parameters:
– the context condition to be modified

3.8. CP8: Insert Configuration Constraint Between Configuration Alternatives

Description: A constraint regarding the selection of configuration alternatives from one or more configurable regions is added to the configurable process model.
Problem addressed: The selection of configuration alternatives can only be done under certain conditions.
Example: When unaccompanied minors are travelling, a duplicated boarding card is printed to the relative who accompany them to the boarding gate,

i.e., an inclusion constraint exists.

Implementation in C-EPC:

CP8 is realized by inserting a configuration requirement, which is then linked to the involved configurable nodes (either configurable functions or connectors) that delimit the configurable region of the configuration alternatives to be constrained.

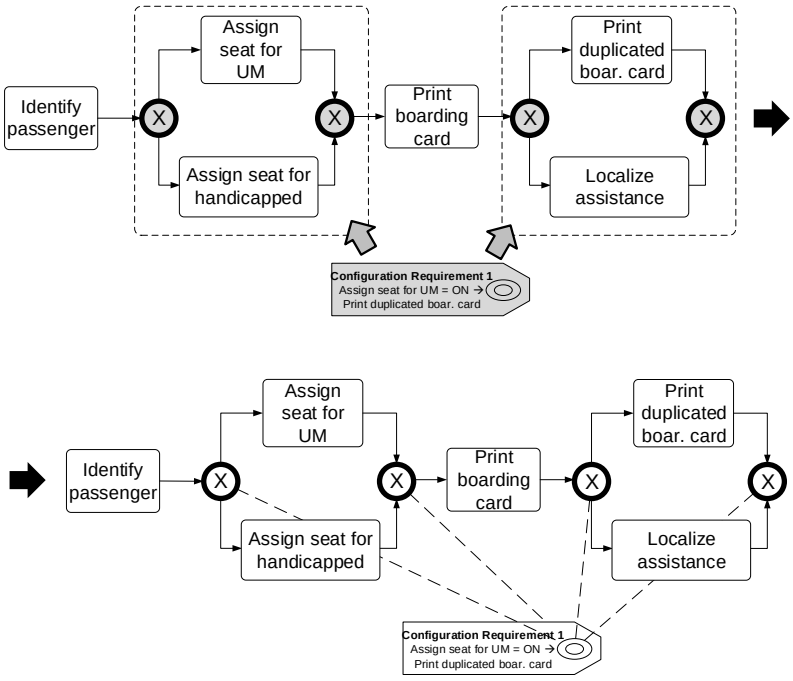

Figure 16: CP8 implemented in C-EPC

Parameters:

– the configuration region to which the alternatives whose selection will be constrained

– the configuration constraint to be inserted

3.9. CP9: Delete Configuration Constraint Between Configuration Alternatives

Description: A constraint between two or more configuration alternatives from one or more configurable regions is deleted.

Problem addressed: A constraint between two or more configuration alternatives is no longer needed and thus it is deleted.
Example: When unaccompanied minors are travelling, a relative accompany them to the boarding gate (i.e., inclusion constraint). Due to emerging legal regulations, from now on the staff from the airline shall accompany them, i.e., the inclusion constraint is no longer needed.
Implementation in C-EPC:
CP9 is realized by deleting a configuration requirement, which is linked to the configurable nodes that delimit the configurable region of the configuration alternatives to be constrained.

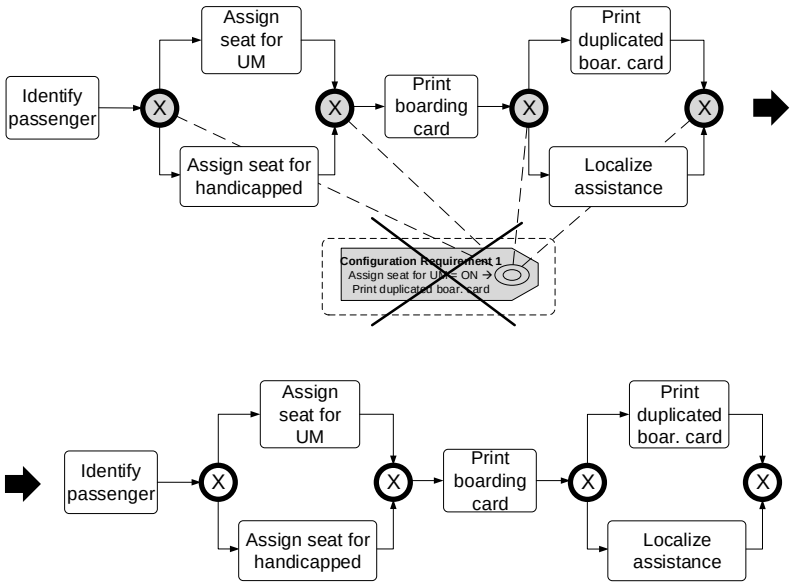

Figure 17: CP9 implemented in C-EPC

Parameters:
– the configuration constraint to be deleted

3.10. CP10: Modify Configurable Region Resolution Time

Description: In a configurable process model, the resolution time of a configurable region is modified.

Problem addressed: The resolution time of a configurable region in a configurable process model is no longer adequate and is modified.
Example: Passengers travelling to US should fill in the ESTA form. However, due to new regulations, if the passenger already travelled to the US in smaller period than six months with the same airline, the latter may decide that the ESTA form is not needed again. This means that the activation of the activity "Fill in ESTA form" depends on the passenger and the airline (i.e., activity "Fill in ESTA form" becomes optional).
Implementation in C-EPC:
Since resolution time is only supported by the optionality of configurable functions (i.e., OPT configuration), CP10 is implemented by modifying to OPT the configuration requirements that restricts the configuration of the configurable function; e.g., from ON to OPT. The constraints of the questionnaire model referred to the function which resolution time has been changed should be adapted accordingly.

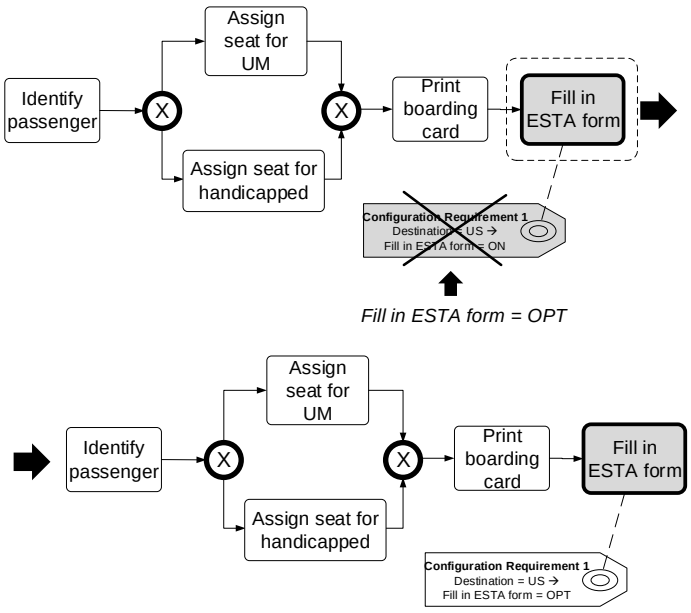

Figure 18: CP10 implemented in C-EPC

Parameters:
– the configurable region whose resolution time is modified

3.11. Discussion

In absence of an established method to deal with process variability in an explicit manner, we provide a set of change patterns that enable the modeling and evolution of process families. In addition, these patterns are intended to reduce the effort needed for such purposes and ensure process family correctness. In this context, in the following we discuss the completeness, generalizability and application order of our patterns.

Regarding the completeness of the proposed patterns, we ground the patterns on a set of variability-specific language constructs obtained from a large-scale systematic literature review. As a consequence of this methodological choice, our patterns describe how process variability is supported in the literature. Therefore, CP4PF is complete with regards to their support for such constructs. We have not specified patterns for modifying configurable regions, configuration alternatives, and configuration constraints because they can be realized by combining other change patterns and existing adaptation patterns. However, CP7 and CP10, which correspond to the combination of other patterns for some approaches (e.g., Provop, cf. Sect. 5), have been defined in order to modify in a more efficient way the variability-specific language constructs that they affect (i.e., configuration context condition and configurable region resolution time). As a consequence, our pattern set is not minimal.

Regarding the generalizability of CP4PF, our systematic review identified the variability-specific language constructs in 34 different approaches for process family management. Although these approaches use different terminology (e.g., *configurable region* vs. *variation point*) and may realize the language constructs in different ways, the five languages constructs are widely used in the literature on process variability. Using this set of constructs as a basis, we can ensure that the proposed patterns are expressive enough to model and evolve process families in such approaches. That is, CP4PF can be implemented in any of these approaches for the constructs that they support.

When modeling with only CP4PF, the application order of the patterns is implicitly determined by the type of operation (insertion, modification, and deletion) and the constructs of each pattern. For example, configuration alternatives cannot be inserted if configurable regions have not been inserted previously. The same happens when inserting configuration context conditions and constraints, they need the previous insertion of configuration

alternatives. Fig. 19 shows the application dependence graph between the patterns.

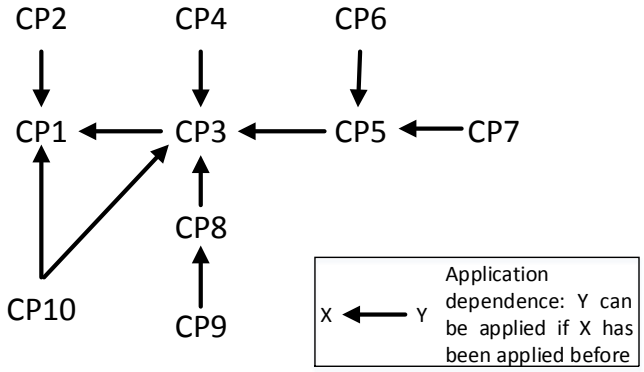

Figure 19: Application dependence graph between CP4PF

Process variability management distinguishes between design-time and runtime variability [30, 40]. Variability at design time deals with choices whose resolution is known before process execution. For example, in the check-in process, the way a boarding card could be printed (i.e., electronic or paper format) is resolved when the process varint to execute is configured (selecting) out of the process family (e.g., Variant 1 in Fig. 1). Variability at runtime deals with choices that are only resolved once a specific process variant instance (i.e., a concrete case) is being executed. For example, whether or not a passenger carries overweight luggage is not known until he arrives at the counter. In addition, variability at runtime may refer to the dynamic resolution of a concrete region of a process variant based on runtime data (e.g., late modeling) [10]. CP4PF allow managing both types of variability because the different choices are foreseen at design time, and hence can be represented in a configurable process model independently of when they are resolved. On the contrary, CP4PF do not manage ad-hoc changes supported in adaptive PAISs [40]. Usually, ad-hoc changes correspond to unplanned dynamic changes which are not foreseen before process execution.

Finally, CP4PF are also intended to reduce the effort needed for modeling and evolving configurable process models. This is analyzed in the following section through a case study.

4. Validation

This section reports on a case study conducted with a safety standard for validating CP4PF. We have selected this empirical method because it is a well-established and widely accepted approach for studying phenomena in their real life context [42], including for software and systems engineering research [49, 43]. Case studies are usually classified as flexible research and aim to provide new knowledge from and about actual situations. Their conclusions are based on evidence collected in a planned and consistent manner. We followed the guidelines and procedures proposed in [43].

The case study allows us to validate the *feasibility* of our proposed CP4PF in a real scenario as well as analyzing the *effort* of applying them. Case studies with similar or the same purposes (i.e., feasibility and effort analyses) can be easily found in the literature on e.g. business process modeling [34, 10] and system assurance modeling [37, 35].

In the following, we present the context of the case study, the research questions, the case selection and data collection procedure, the case study results, and a discussion of these results. Finally, we discuss the validity of our case study.

4.1. Context

A *safety–critical system* is one whose failure can lead to injury or death to people or damage to the environment [18]. These systems are subject to rigorous assurance processes so that they do not pose undue risks. These processes are usually based on *safety standards* whose compliance with must be shown. An example of this type of standards is IEC 61508 [22], which deals with functional safety of electrical, electronic, and programmable electronic systems. IEC 61508 is a generic standard that has been used as basis for sector-specific ones (e.g., automotive).

Safety standards indicate requirements to fulfill, artifacts to create, and activities to execute in a system's lifecycle. The standards also recommend *techniques* that represent alternative ways to reach the standards' objectives. Table 2 shows an example of the techniques recommended for a software development. The content of the figure is based on the software architecture design phase of IEC 61508-3.

Safety standards indicate when a technique should be used. For example, in IEC 61508 the techniques are assigned to *safety integrity levels* (SIL), which represent the relative level of risk reduction. IEC 61508 defines four

Technique		SIL 1	SIL 2	SIL 3	SIL 4
1	Dynamic reconfiguration	--	--	NR	NR
2a	Structured methods	--	R	HR	HR
2b	Formal methods	--	R	HR	HR
3	Modular approach	HR	HR	HR	HR

Table 2: Example of the techniques extracted from a safety standard

SILs, being SIL 1 the lowest risk-reduction and SIL 4 the highest one. In addition, the standard provides a *recommendation* regarding the use of a technique: *highly recommended* (HR), *recommended* (R), *no recommendation* for or against being used (- -), and *not recommended* (NR). In Fig. 2 the technique *Dynamic reconfiguration* has no recommendation for SIL 1 and SIL 2, while it is not recommended for SIL 3 and SIL 4. As a rule of thumb, the use of HR techniques is compulsory and NR techniques must not be used. Finally, IEC 61508 indicates the alternative use of some techniques to specify that only one of them might be used (e.g., techniques 2a and 2b in Fig. 2).

The variability associated to safety standard compliance (hereafter referred to as variability of a safety standard) is high and complex. For example, IEC 61508-3 recommends around 150 techniques, which in combination with the number of SILs, the given recommendations, and the existence of alternative techniques, result in thousands of possible ways of applying the standard. For example, the technique *Structured methods* has three different recommendations in Fig. 2. In addition, *Structured methods* is only used if the technique *Formal methods* is not used (i.e., alternative techniques). These variations even increase if it is considered that, for example, some systems might not use a HR technique because of the specific characteristics of the systems (e.g., code automatically generated). Safety standards do not and cannot provide a unique algorithm for combining the techniques that will be correct for any application of the standard [22].

In order to facilitate their application, safety standards can be represented with models [35]. Practitioners use process models [36] and commercial tools for representing safety standards' processes with BPMN [3, 51], including the modeling of techniques as BPMN activities [6]. In addition, specific models for safety assurance have been proposed during the last years. For example, *SafetyMet* is a metamodel for specifying how to comply with a safety standard

[18]. Fig. 20 shows an excerpt of this metamodel. In general terms, and using Fig. 2 as a reference, the class *Reference Applicability* is used to represent a row of the table, and the class *Reference Technique* is used to specify a technique (e.g., *Structured methods*). The classes *Reference Criticality Level* and *Reference Applicability Level* are used to specify the SILs (SIL 1 - SIL 4) and the types of recommendation (e.g., HR), respectively, whereas the class *Reference Criticality Applicability* is used to link a technique with a recommendation for a specific SIL (e.g. technique *Structured methods* has a HR recommendation for SIL 4). The class *Reference Requirement* can be used to specify that some techniques are alternative to each other (e.g., techniques 2a and 2b in Fig. 2).

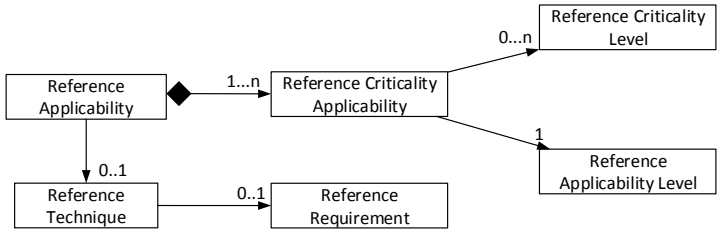

Figure 20: Excerpt of the SafetyMet metamodel (adapted from [18])

Finally, new versions of safety standards can be released for including emerging techniques as well as making adjustments; e.g., IEC 61508 has two versions: 1998 and 2010. As a consequence, existing representations of the standards must be evolved in order to represent the changes.

4.2. Research Questions

The goal of the case study was to analyze the application of CP4PF in a real and industrial modeling scenario. For such a purpose, we focused on the variability of safety standards. In particular, we formulate the following research questions:

- RQ1: Is the application of CP4PF a *feasible approach* for modeling the variability of a safety standard?

This question refers to whether CP4PF can be used effectively for (1) creating a configurable process model representing the variability of a safety

standard (e.g., when to use a technique) and (2) evolving this model in accordance to the changes introduced by a new version of the standard.

- RQ2: Does the application of CP4PF reduce the *effort* for modeling the variability of a safety standard?

This question is based on the level of effort spent throughout the creation and evolution of the configurable process model representing a safety standard when compared to state-of-the-art approaches. Effort is an important factor for determining if CP4PF can be successfully adopted. If the effort for modeling and evolving a configurable process model using CP4PF is higher than with the approaches currently used (e.g., BPMN), then CP4PF adoption will be hindered.

4.3. Case Selection and Data Collection

The subject of the case study is the IEC 61508 standard. We chose IEC 61508 because it is a general standard that is applied in different sectors and for different systems. Among its parts, we chose part 3, which deals with software development. For answering **RQ1**, both the first (1998) and the second version (2010) were taken into account. More precisely, we used CP4PF for creating a configurable process model representing the variability of the IEC 61508-3:1998 and used them again for evolving the resulting model in order to comply with the 2010 version. We used C-EPC and applied therefore CP4PF using their implementation in this notation (cf. Sect. 3).

Regarding **RQ2**, we used the *number of operations* as the effort metric. The main advantages of operation measurement for effort analysis over, for instance, time measurement are that: (1) it allowed us to compare the modeling effort with different approaches without having to engage experts (in each approach) with similar levels of expertise and modeling skills; (2) finding experts that can spend the time necessary to create large models can be extremely difficult, especially if the experts must meet the above conditions; (3) it avoided threats to validity related to the modelers' fatigue in creating large models, and; (4) the results were more reliable since the operations could be measured twice and the outcome would be the same. In particular, we compared the number of operations needed with CP4PF with the number of operations needed to create and evolve a IEC 61508 model using two state-of-the-art approaches: BPMN and the SafetyMet metamodel. We chose BPMN because it is the *de-facto* standard for process modeling [12]

and it is used in industry for modeling safety standards [3, 51]. We chose SafetyMet because it is a generic metamodel specifically targeted at representing safety standards [18]. In addition, to determine the exact impact that CP4PF have in the representation of the standard in a configurable process model, we also compared the number of operations applying CP4PF with the number of operations needed to create and evolve the same configurable process model in C-EPC but without applying the patterns. In summary, we compared the number of operations using four approaches:

- Create and evolve a C-EPC model using CP4PF
- Create and evolve a C-EPC model adding/deleting/modifying language primitives (e.g., configurable connector)
- Create and evolve a BPMN model adding/deleting/modifying language primitives (e.g., activity)
- Create and evolve a SafetyMet model adding/deleting/modifying language primitives (e.g., Reference Technique)

Data collection involved two main activities. The first one consisted in creating the models representing the variability of the standard IEC 61508-3:1998 using the four approaches. In turn, the second activity consisted in evolving the four resulting models to comply with the 2010 version. For both activities, we measured the number of operations needed for creating/evolving the models. More precisely, we considered four types of actions:

1. *Insert a modeling element* (e.g., a link)
2. *Insert a named modeling element* (e.g., a BPMN activity) or *apply an insert pattern* (e.g., CP1)
3. *Delete a modeling element* (e.g., a link) or *apply a delete pattern* (e.g., CP2)
4. *Modify a modeling element* (e.g., rename a BPMN activity, rearrange a link) or *apply a modify pattern* (e.g., CP10)

We measured the first, third, and fourth type of action as one operation, and the second one as two operations (i.e., one operation for the insertion and another for writing a name). We made this difference in order to reflect that both inserting an element and naming it require a higher effort than only inserting, deleting, or modifying it. We considered that no distinction was

necessary for the aspects common to all the types of actions (e.g., indication of element location, when either selecting an element for modification or deletion or specifying where to insert it).

Prior to data collection, we defined how the variability of the standard can be systematically represented with the selected approaches. Fig. 21 shows the configurable process model of the techniques presented in Fig. 2 with the C-EPC notation.[2]

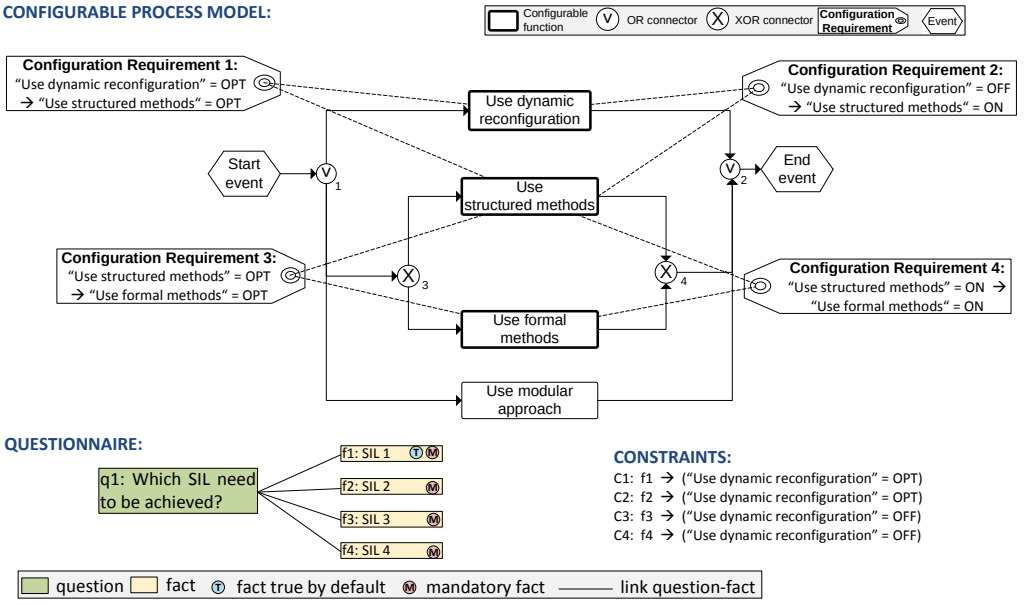

Figure 21: C-EPC model of the techniques presented in Fig. 2 and the associated questionnaire and constraints

Techniques were represented depending on the provided recommendations. Techniques with two (or more) recommendations were defined as configurable functions, which can be configured to ON (i.e., the function is included), OFF (i.e., the function is not included), or OPT (i.e., the function is optionally included) depending on the SIL to achieve and the recommendation of use. Thus, the SILs corresponded to the application context of the configurable functions. For a given SIL, we considered that

[2]The name of each technique has been adapted to the format "verb + object" in order to make them process-oriented [33].

- HR techniques must be used. Thus, the configurable function should be configured to ON.
- R techniques and techniques with no recommendation (- -) are optional. Thus, the configurable function should be configured to OPT.
- NR techniques must not be used. Thus, the configurable function is configured to OFF.

However, techniques with only one recommendation for all the SILs were represented as regular functions; i.e., no configurations were needed (e.g., function *Use modular approach* in Fig. 21). The concrete configuration of each function for each SIL was defined in the configuration requirements. Since the application context in C-EPC is represented in a questionnaire model (cf. Fig. 21), configuration requirements were defined based on previous configurations [29]. For example, Configuration Requirement 3 states that the configurable function *Use formal methods* is configured based on the configuration of the function *Use structured methods*. Finally, we defined that alternative techniques (e.g., 2a and 2b of Fig. 2) should be represented using XOR connectors (e.g., XOR 3 and 4 in Fig. 21). We used OR connectors to represent that there is not predefined precedence order in the use of the techniques (e.g., OR 1 and 2 in Fig. 21). In IEC 61508-3, the use of the techniques is provided in different tables, which are associated to specific lifecycle activities. For example, Fig. 2 shows the techniques for the activity software architecture design. Thus, for each table provided in the standard, we created a configurable process model for representing the variability of the corresponding lifecycle activity.

Fig. 22 shows how the content of Fig. 2 can be represented with BPMN. In line with the way of modeling presented in [6], we modeled each technique as an activity. In turn, we used XOR gateways to differentiate among the recommendations. For example, XOR 3 and 4 in Fig. 22 are used to fork the sequence flow into two paths, one for each type of recommendation for the activity (i.e., technique) *Use dynamic reconfiguration*. SILs are then used as conditions of these gateways to decide which path should be taken. For example, the path of SIL 3 and 4 in XOR3 is taken when the technique is NR and thus must not be used. In the case of R techniques and techniques with no recommendation (- -), we modeled them using XOR gateways in order to represent that the techniques might be used or not (e.g., XOR gateways 5 and 6 in Fig. 22). Like in C-EPC, alternative techniques are represented

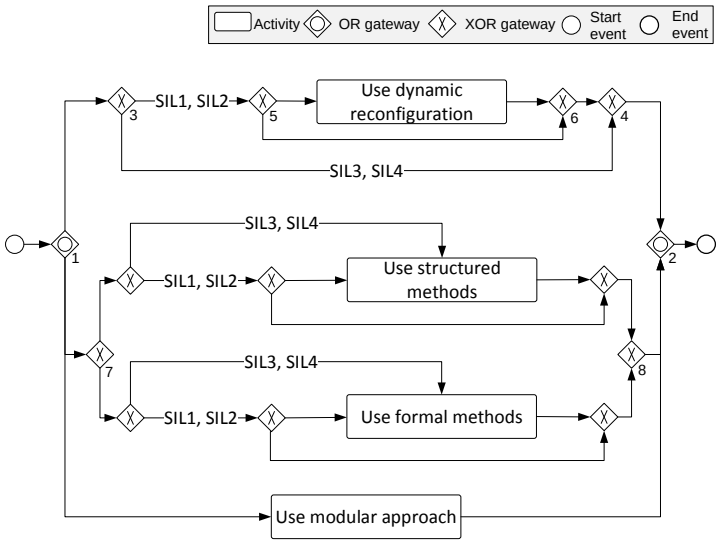

Figure 22: BPMN model of the techniques presented in Fig. 2

using XOR gateways (e.g., XOR 7 and 8 in Fig. 22). Finally, we used OR gateways again to represent that there is no precedence order in the use of the techniques. We created a BPMN model for each table of the standard, as we did with C-EPC.

Finally, regarding the SafetyMet metamodel, we represented the variability of the IEC 61508 using the classes and relationships of the metamodel (cf. Fig. 23).

During the data collection, the first author was the main responsible for systematically creating and evolving the models and measuring the operations. Nonetheless, she did not create the models alone. The third author iteratively validated the resulting models, as well as the measurement results. He has wide knowledge on safety assurance and certification (e.g., [35, 36]), and is co-creator of SafetyMet [18]. The rest of authors also reviewed the collected data.

4.4. Results

In this section, we describe the outcomes of the case study.

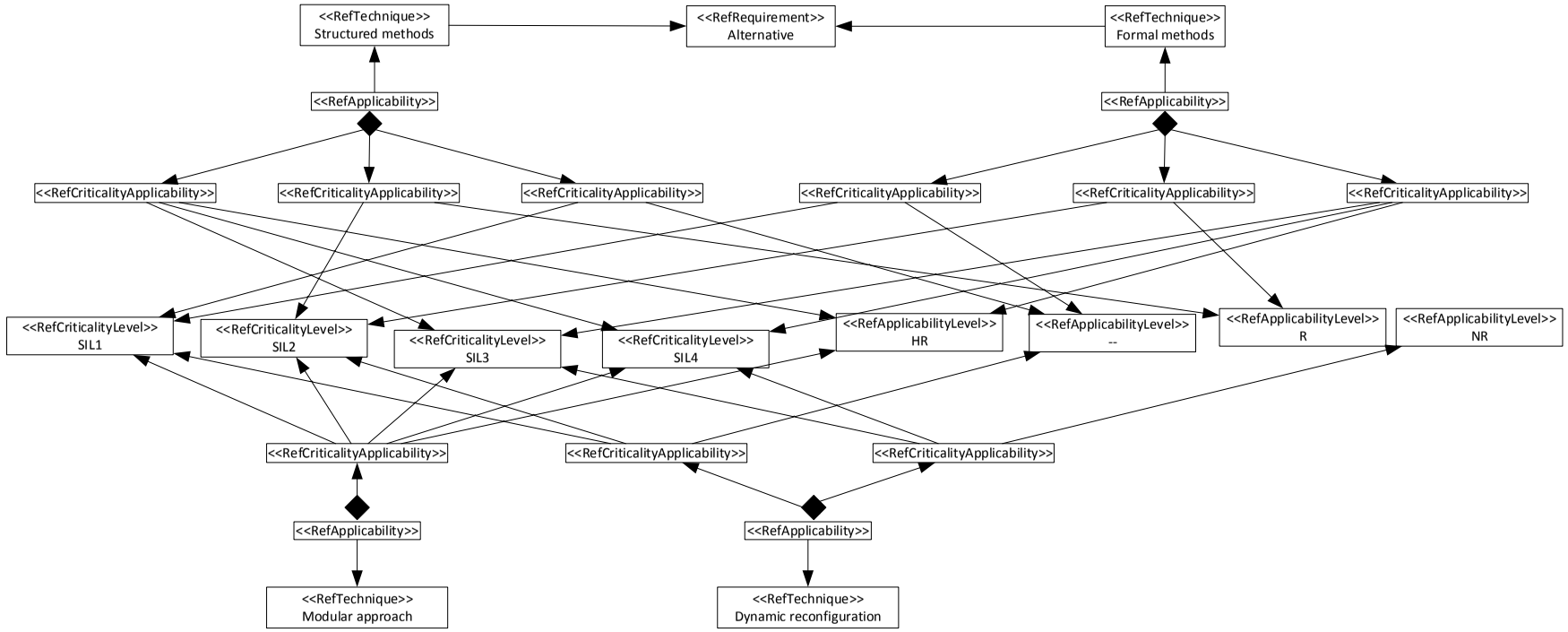

Figure 23: SafetyMet model of the techniques presented in Fig. 2

4.4.1. Results for Model Creation

Since the IEC 61508 documents are under copyright, we refrained from sharing the created models. We are not allowed to publish, for example, the content of their tables. However, for illustration purposes, we detail in Table 3 the results obtained for representing the table presented in Fig. 2 using the four approaches (i.e., the results for obtaining the models presented in Figs. 16-18). More concretely, Table 3 shows the total number of modeling elements for the resulting models as a way to show their scale. In addition, it shows the number of actions needed for creating the models. Since we modeled from scratch, we mostly used insertions of modeling elements, insertions with name, and insert patterns. The total number of operations is the result of adding the number of inserted elements plus two times the number of insertions of named elements or the application of insert patterns. For example, for creating the configurable process model in C-EPC in combination with CP4PF, we needed 15 insertions of modeling elements (2+2+1+1+9), one insertion of a function, 11 applications of insert patterns (3+4+4), and four applications of modify patterns to change the constraints of the questionnaire model associated to the facts. This resulted in a total of 43 operations (15+1x2+11x2+4=43). Regarding the application of CP4PF, CP1 automatically introduces sequence flows and CP5 automatically introduces a question of the questionnaire and the respective constraints (cf. Sect. 3).

Table 4 summarizes the results of the creation of the models representing IEC 61508-3:1998. A total of 119 techniques were represented. For creating the configurable process model with CP4PF, four patterns were used. We applied CP1 once per technique represented, CP5 once per existing SIL, CP8 for inserting the configuration requirements needed to specify the configuration of each configurable function, and CP7 for modifying the constraints of the questionnaire. In total, we needed 295 pattern applications.

4.4.2. Results for Model Evolution

Table 5 shows the differences between the 1998 and 2010 versions of IEC 61508-3. Thus, we evolved the created models to represent these changes.

Table 6 summarizes the results of this evolution. More precisely, it shows the number of the elements of the evolved models as well as the operations

Approach	Total of modeling elements	Type of actions				Total of operations
		1. Insert element	2. Insert named element / apply INSERT pattern	3. Delete modeling element / apply DELETE pattern	4. Modify modeling element / apply MODIFY pattern	
C-EPC + CP4PF *(model presented in Fig. 16)*	47	- 2 XOR connectors - 2 OR connectors - 1 start event - 1 end event - 9 sequence flows	- 1 function - 3 app. of CP1 - 4 app. of CP5 - 4 app. of CP8	----------	- 4 app. of CP7	43
C-EPC *(model presented in Fig. 16)*	47	- 2 XOR connectors - 2 OR connectors - 1 start event - 1 end event - 12 sequence flows - 8 requirement connectors - 4 links question/facts	- 1 function - 3 configurable functions - 4 configuration requirements - 1 question - 4 facts - 4 constraints	----------	- 4 modification constraints	68
BPMN *(model presented in Fig. 17)*	48	- 12 XOR gateways - 2 OR gateways - 1 start event - 1 end event - 22 sequence flows	- 4 activities - 6 labeled sequence flows	----------	----------	58
SafetyMet *(model presented in Fig. 18)*	57	- 4 Reference Applicability - 9 Reference Criticality Applicability - 31 links	- 4 Reference Technique - 1 Reference Requirement - 4 Reference Criticality Level - 4 Reference Applicability Level	----------	----------	70

Table 3: Summary of the results for representing the table of Fig. 2

needed for the evolution. With CP4PF, we applied four patterns, resulting in a total number of 181 pattern applications. More precisely, we applied CP1 once per new technique with two different recommendations, CP2 once per technique with two different recommendations deleted, CP8 for adding the configuration constraints of the techniques introduced, and CP9 for modifying the constraints in the questionnaire model.

4.4.3. Synthesis of the Results

Table 7 synthesizes the results of the case study. As shown, in the creation of the models, the use of CP4PF in combination with C-EPC reduces the number of operations in a 19.1% with respect to using only C-EPC, a 34.5% with respect to BPMN, and a 33.3%, with respect to SafetyMet. For evolving the models, the use of CP4PF in combination with C-EPC reduces the number of operations in a 25.1% with respect to only C-EPC, a 40.4% with respect to BPMN, and a 33.1%, with respect to SafetyMet.

4.5. Discussion

In this section, we discuss the results of the case study focusing on answering the research questions.

Approach	Total of modeling elements	Type of actions				Total of operations
		1. Insert element	2. Insert named element / apply INSERT pattern	3. Delete modeling element / apply DELETE pattern	4. Modify modeling element / apply MODIFY pattern	
C-EPC + CP4PF	841	- 62 XOR connectors - 38 OR connectors - 19 start event - 19 end event - 288 sequence flows	- 46 functions - 73 app. of CP1 - 4 app. of CP5 - 70 app. of CP8	----------	- 148 app. of CP7	960
C-EPC	841	- 62 XOR connectors - 38 OR connectors - 19 start event - 19 end event - 361 sequence flows - 140 requirement connectors - 4 links question/facts	- 46 functions - 73 configurable functions - 70 configuration requirements - 1 question - 4 facts - 4 constraints	----------	- 148 constraint modifications	1187
BPMN	1203	- 282 XOR gateways - 38 OR gateways - 19 start event - 19 end event - 582 unlabeled sequence flows	- 119 activities - 144 labeled sequence flows	----------	----------	1466
SafetyMet	1305	- 119 Reference Applicability - 216 Reference Criticality Applicability - 835 links	- 119 Reference Technique - 8 Reference Requirement - 4 Reference Criticality Level - 4 Reference Applicability Level	----------	----------	1440

Table 4: Summary of the results for creating the models representing the IEC 61508-3:1198 standard

Techniques introduced	63
Techniques deleted	30
Techniques renamed	17
Techniques with different recommendations	5

Table 5: Differences between the IEC 61508-3:1998 and IEC 61508-3:2010 versions

Regarding **RQ1** (*Is the application of CP4PF a feasible approach for modeling the variability of a safety standard?*), we could successfully model and evolve a configurable process model for IEC 61508-3 using CP4PF. We needed 295 pattern applications for creating the configurable process model and 181 for evolving it. The main challenge was to define how the information of the standard had to be represented with C-EPC. More precisely, we had to decide how to capture all the variability in a C-EPC configurable process model and at the same time how to ensure model understandability and accuracy. For example, we discussed how to properly represent the techniques and the associated recommendations in the most suitable manner (e.g., represent techniques as configurable functions). For evolving the configurable process model, we also had to determine the impact that the

Approach	Total of modeling elements	Type of actions				Total of operations
		1. Insert element	2. Insert named element / apply INSERT pattern	3. Delete modeling element / apply DELETE pattern	4. Modify modeling element / apply MODIFY pattern	
C-EPC + CP4PF	1137	- 44 XOR connectors - 140 sequence flows	- 18 functions - 45 app. of CP1 - 72 app. of CP8	- 20 app. of CP2 - 10 functions - 18 XOR connectors	- 44 app. of CP7 - 17 function rename	563
C-EPC	1137	- 44 XOR connectors - 185 sequence flows - 144 requirement connectors	- 18 functions - 45 configurable functions - 72 configuration requirements	- 20 configurable functions - 10 functions - 18 XOR connectors	- 44 constraint modifications - 17 function rename	752
BPMN	1558	- 181 XOR gateways - 325 unlabeled sequence flows	- 63 activities - 92 labeled sequence flows	- 30 activities - 82 XOR gateways	- 17 activity rename	945
SafetyMet	1701	- 63 Reference Applicability - 117 Reference Criticality Applicability - 443 links	- 63 Reference Technique - 7 Reference Requirement	- 30 Reference Technique - 30 Reference Applicability - 1 Reference Requirement	- 17 technique rename	841

Table 6: Summary of the results for evolving the created models

Approach	Model creation			Model evolution		
	Number of modeling elements	Number of operations	% of effort reduction	Number of modeling elements	Number of operations	% of effort reduction
C-EPC + CP4PF	841	960	na	1137	563	na
C-EPC	841	1187	19.1%	1137	752	25.1%
BPMN	1203	1466	34.5%	1558	945	40.4%
SafetyMet	1305	1440	33.3%	1701	841	33.1%

Table 7: Synthesis of the results of the case study

differences between both versions of the standard had on the already created configurable process model. For example, we needed to decide how to reflect in the configurable process model that a recommendation had changed for a specific SIL (e.g., adding new configuration requirements).

For modeling and evolving the configurable process model, we used a total of six change patterns (CP1, CP2, CP5, CP8, CP9, and CP10) out of the 10 defined. More precisely, we applied patterns referred to four variability-specific language constructs: configurable region, configuration context condition, configuration constraint, and configurable region resolution time. We did not apply patterns related to configuration alternatives (i.e., CP3 and CP4) because we decided to model techniques as configurable functions, which implicitly define the configuration alternatives (i.e., ON, OFF, OPT). In addition, we did not use patterns for deleting or modifying the application

context (i.e., CP6 and CP7) because the application context (i.e., SILs) did not change between the versions of the standard.

Regarding **RQ2** (*Does the application of CP4PF reduce the effort for modeling the variability of a safety standard?*), we consider that the results show that the application of CP4PF can significantly reduce the effort for modeling the variability of a safety standard. When compared to state-of-the-art approaches (i.e., BPMN and SafetyMet metamodel), CP4PF in combination with C-EPC can reduce up to 34.5% the number of operations for creating a configurable process model of the variability of a safety standard, and up to 40.4% for evolving it. We acknowledge that CP4PF were used in combination with a variability-specific approach that deals with process variability in an explicitly manner (i.e., C-EPC). This might be advantageous, for example, in respect to BPMN, since it was not conceived to explicitly deal with process variability. In addition, we need to consider that the SafetyMet metamodel was conceived for being compliant with any safety standard, supporting any kind of information they may include (e.g., guidelines and techniques explanations). This adds a set of extra modeling actions (e.g., inserting a *Reference Applicability* element) that need to be done but are not needed for representing the selected information of IEC61508-3. However, even when compared to modeling with only C-EPC, the application of the change pattern is clearly advantageous to us (over 19% reduction in the number of operations). In this sense, we consider that most of the benefit comes from using CP4PF, not from the process variability approach.

Finally, the results for RQ2 coincide with the benefits that we envisioned when defining CP4PF. The case study allowed us to determine the actual extent to which the application of CP4PF can reduce the effort for modeling a process family in a real scenario. This is mainly due to the fact that CP4PF can automatically insert or delete several modeling elements with a single modeling action (i.e., pattern application).

4.6. Validity

Like any other modeling situation, modeling IEC 61508-3 comprised decisions about how to create the models (e.g., modeling gateways in pairs). In addition, one author was the main responsible for creating the models. These factors affect *internal validity*. To mitigate possible threats, we decided prior to data collection how to systematically represent the variability of the standard with the selected approaches. The obtained models were also validated.

Regarding *conclusion validity*, CP4PF were implemented and applied with only one process variability approach (C-EPC). The results of the case study and thus the conclusions drawn could differ when implementing CP4PF with other approaches (e.g., Provop [26]).

Single case studies as the one conducted (with only one safety standard) pose threats to *external validity*. However, we expect similar results for standards with similar characteristics (e.g., other safety standards, and especially those based on IEC 61508). We also believe that variability management of process families from other domains can benefit from CP4PF. Since the implementation of CP4PF with C-EPC automatically introduces or deletes modeling elements, we may expect an effort reduction when modeling and evolving other process families.

5. Related Work

In addition to the background of the paper (cf. Sect. 2), research on *process variability modeling* and *workflow patterns* is closely related to our work.

First, in the context of process families, a way for defining process variability is by *restricting* the behavior captured in a configurable process model [30, 10]. For example, in C-EPC, configuration requirements constraint the behavior of configurable functions and connectors (e.g., defining mutual exclusions or inclusions). The same applies for C-YAWL where process variants can be configured by applying *hiding and blocking operators*, which restrict different execution paths by making them unobservable or disable [2]. In turn, the PESOA approach includes a set of *annotations* attached to the activities that may be subject to variation [39]. In addition, the application context of these variable activities is specified in terms of *features* attached to them. Accordingly, process variants are configured by selecting (restricting) the features that refer to each variant.

Another way for defining process variability is via *extension or modification* of the behavior captured in configurable process model [30, 10]. For example, in Provop, a pre-specified *base process model* is structurally adjusted to the given application context through a sequence of model changes (e.g., delete, insert, and move process fragments) [26]. In turn, the ADOM approach, *cardinality attributes* are used to annotate the elements of a configurable process model in order to specify their available number of instantiations (i.e. how many times an activity can be used in a process variant).

Then, process configuration is achieved by instantiating elements with multiple cardinality, and adding application-specific elements [41]. Finally, there are also approaches that apply *business rules* (i.e., change artifacts) to configure process variants from a process template [27]. Our patterns deal with process variability at a level of abstraction higher than the one provided by these approaches. CP4PF are intended to be implemented in any of these approaches as we did with C-EPC.

Empirical evaluations of process variability approaches have also been conducted [10]. Case studies are the most frequent method and have been conducted in different domains such as egovernment [24], logistic [32], risk management [48], and retail [38]. To the best of our knowledge, the case study conducted in this work is the largest one in the context of process variability. The high variability presented in safety standards (e.g., possible combinations of techniques and recommendations) has resulted in a configurable process model with over 1000 elements. In addition, we did not only create a configurable process model, but also evolved it to meet changing requirements. Finally, unlike other case studies, we also compared the results of CP4PF with how process variability can be managed with other approaches (i.e., BPMN and SafetyMet metamodel). Thus, we could analyze the suitability of CP4PF in terms of effort reduction.

Regarding other types of evaluations in the context of process variability, the *Goal/Question/Metric method* is used to evaluate how good the design of a configurable process model is [4]. In turn, similarity metrics to measure the complexity (e.g., size) of a configurable process model are used in [52]. Mapping patterns to compare different process variability approaches in terms of complexity (e.g., size of resulting models) are also used in [11]. This paper complements these evaluations because we have analyzed the effort reduction of applying CP4PF in a real scenario.

In the context of software process lines, there exist approaches dealing with variability in software processes. For example, [50] describes and analyzes different methods for modeling variability in processes that are built from a set of core assets. More concretely, it analyzes SPEM, vSPEM, feature models, and OVM in terms of their expressiveness for modeling software variability, as well as their understandability and related tools. However, these approaches were not deeply analyzed since their focus is not on business process variability.

Second, regarding workflow patterns, they have been defined for analyzing the expressiveness of process modeling languages. These patterns cover

different perspectives such as control flow [1], data [44], resources [45], time [28], and exceptions [46]. Pattern compounds [25] are similar to adaptation patterns (cf. Sect. 2.1) and enable context-sensitive selection and pattern composition during process modeling. However, and as adaptation patterns, *workflow patterns* are not sufficient for effectively dealing with process families. They do not consider variability-specific needs introduced by process families and hence are complementary to CP4PF.

Finally, the management of process families through patterns has also been proposed in [47]. It presents a set of theoretical patterns for promoting the reuse of activities, resources, and data in configurable process models. However, these patterns have not been obtained in a systematic way and lack support for the entire set of variability-specific language constructs (cf. Sect. 2.3). Our work is broader in the sense that it covers more the variability needs of process families. In addition, CP4PF have been empirically-grounded and applied in a real scenario through a case study.

6. Summary and Outlook

Managing the variability of process families can be very difficult in practice due to their size and complexity, and means that facilitate the creation and evolution of configurable process models are necessary. Such means should also aim to be cost-effective, reducing modeling and evolution effort and being able to ensure model correctness.

Our work complements existing work on patterns for creating and modifying process models and on process variability management by introducing 10 patterns for modeling and evolving process families. Our *change patterns for process families* were derived from the five variability-specific constructs used for capturing process variability. To show that our patterns–despite their intended generic nature–are specific enough to manage process variability, we have presented their implementation in C-EPC, a well-known process variability approach.

In addition, we conducted a case study with the process family of a safety standard (IEC 61508-3). The case study results allow us to show the feasibility of the patterns and their suitability in terms of effort reduction. When compared to other state-of-the-art approaches, the defined patterns were able to reduce the effort needed for modeling a process family by 34% and for evolving it by 40%.

Our future work includes extending the patterns for aspects different to control flow (e.g., data and resources). We also aim to identify complementary patterns for covering other phases of the process lifecycle (e.g., runtime variability). Finally, we would like to conduct further empirical studies in order to determine the outcome of using our patterns in other application domains with a high degree of variability (e.g., healthcare). We would also like to perform controlled experiments with subjects in order to analyze how users apply the patterns and their perceptions of this application.

Acknowledgements
This work has been developed with the financial support of Spanish Ministry of Economy and Competitiveness under the project SMART-ADAPT TIN2013-42981-P. We also want to thank Barbara Weber and Manfred Reichert for their valuable input and feedback on the design and development of the set of change patterns for process families.

References

[1] van der Aalst, W.M.P., ter Hofstede, A.H.M., Barros, B.: Workflow patterns. Distributed and Parallel databases 14(3), pp. 5–51, (2003).

[2] van der Aalst, W.M.P., Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems 30(4), pp. 245–275, (2005).

[3] ADONIS: Community Edition 3.0 `http://www.adonis-community.com/` Accessed: October 2015.

[4] Alférez, G. H., Pelechano, V., Mazo, R., Salinesi, C., Diaz, D.: Dynamic adaptation of service compositions with variability models. Journal of Systems and Software 91, pp. 24–47, (2013).

[5] ARIS Community: Basic rules on EPC modeling `http://www.ariscommunity.com/users/rbaureis/2010-03-22-basic-rules-epc-modelling` Accessed: October 2015.

[6] Atego Process Director `http://www.atego.com/products/atego-process-director/` Accessed: October 2015.

[7] Ayora, C., Torres, V., Reichert, M., Weber, B., Pelechano, V.: Towards run-time flexibility for process families: open issues and research challenges. In Proc. BPM Workshops'12, pp. 477–488, (2012).

[8] Ayora, C., Torres, V., Weber, B., Reichert, M., Pelechano, V.: Change patterns for process families. Technical Report, PROS-TR-2012-06, http://www.pros.upv.es/technicalreports/PROS-TR-2012-06.pdf, (2012).

[9] Ayora, C., Torres, V., Weber, B., Reichert, M., Pelechano, V.: Enhancing modeling and change support for process families through change patterns. In Proc. BPMDS'13, pp. 246-260, (2013).

[10] Ayora, C., Torres, V., Weber, B., Reichert, M., Pelechano, V.: VIVACE: A framework for the systematic evaluation of variability support in process-aware information systems. Information and Software Technology 57, pp. 248–276, (2015).

[11] Baier, T., Pascalau, E., Mendling, J.: On the suitability of aggregated and configurable business process models. In Proc. BPMDS'10, pp. 108–119, (2010).

[12] Business Process Model and Notation, version 2.0. Object Management Group (OMG). http://www.bpmn.org/ Accessed: October 2015.

[13] Capability Maturity Model Integration (CMMI). Carnegie Mellon University. www.cmmiinstitute.com Accessed: October 2015.

[14] Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q. 13(3), pp. 319–340, (1989).

[15] Dijkman, R., La Rosa, M., Reijers H.A.: Managing large collections of business process models - Current techniques and challenges. Computers in Industry 63(2), pp. 91–97, (2012).

[16] Döhring, M., Zimmermann, B., Karg, L.: Flexible workflows at design- and runtime using BPMN2 adaptation patterns. In Proc. BIS'11, pp. 25–36, (2011).

[17] de la Vara, J.L., Ali. R., Dalpiaz, F., Sánchez, J., Giorgini, P.: COMPRO: A methodological approach for business process contextualisation. In Proc. OTM'10, pp. 132–149, (2010).

[18] de la Vara, J.L., Panesar-Walawege, R.K.: SafetyMet: A metamodel for safety standards. In Proc. MODELS'13, pp. 69–86, (2013).

[19] Dumas, M., van der Aalst, W.M.P., Hofstede, A.H.M.: Process-aware information systems: bridging people and software through process technology. John Wiley & Sons Publishers, (2005).

[20] Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of business process management, Springer, (2013).

[21] Ericsson, K.A., Simon, H.A.: Protocol analysis: verbal reports as data. MIT Press, (1980).

[22] IEC: Functional safety of electrical/electronic/programmable electronic safety-related systems (IEC 61508), (2010).

[23] Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M. H.: Configurable process models - A foundational approach. Reference modeling, Physica-Verlag HD, pp. 59–77, (2007).

[24] Gottschalk, F., Wagemakers, T.A.C., Jansen-Vullers, M.H., van der Aalst, W.M.P., La Rosa, M.: Configurable process models: experiences from a municipality case study. In Proc. CAiSE'09, pp. 486–500, (2009).

[25] Gschwind, T., Koehler, J., Wong, J.: Applying patterns during business process modeling. In Proc. BPM'08, pp. 4–19, (2008).

[26] Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the Provop approach. Journal of Software Maintenance and Evolution: Research and Practice 22(6–7), pp. 519–546, (2010).

[27] Kumar, A., Yao, W.: Design and management of flexible process variants using templates and rules. International Journal Computers in Industry 63(2), pp. 112–130, (2012).

[28] Lanz, A., Weber, B., Reichert, M.: Time patterns for process-aware information systems. Requirements Engineering Journal 19(2), pp. 113–141, (2014).

[29] La Rosa, M., van der Aalst, W.M.P., Dumas, M., Hofstede, A.H.M.: Questionnaire-based variability modeling for system configuration. Software and System Modeling 8(2), pp. 251–274, (2009).

[30] La Rosa, M., van der Aalst, W.M.P., Dumas, M., Milani, F.P.: Business process variability modeling : A survey, (2013).

[31] Li, C.: Mining process variants: challenges, techniques, examples. PhD Thesis. University of Twente. Netherlands, (2010).

[32] Lönn, C.M., Uppström, E., Wohed, P., Juell-Skielse, G.: Configurable process models for the Swedish public sector. In Proc. CAiSE'12, pp. 190–205, (2012).

[33] Mendling, J., Reijers, H.A, van der Aalst, W.M.P: Seven process modeling guidelines (7PMG). Information & Software Technology 52(2), pp. 127–136, (2010).

[34] Moreno-Montes de Oca, I., Snoeck, :., Reijers, H.A., Rodríguez-Morffi, A.: A systematic literature review of studies on business process modeling quality. Information and Software Technology 58, pp. 187–205, (2015).

[35] Nair, S., de la Vara, J.L., Sabetzadeh, M., Briand, L.: An extended systematic literature review on provision of evidence for safety certification. Information and Software Technology 56(7), pp. 689–717, (2014).

[36] Nair, S., de la Vara, J.L., Sabetzadeh, M., Falessi, D.: Evidence management for compliance of critical systems with safety standards: A survey on the state of practice. Information and Software Technology 60, pp.1–15, (2015).

[37] Panesar-Walawege, R.K., Sabetzadeh, M., Briand, L.: Supporting the verification of compliance to safety standards via model-driven engineering: Approach, tool-support and empirical validation. Information and Software Technology 55(5), pp. 836–864, (2013).

[38] Pascalau, E., Rath, C.: Managing business process variants at eBay. In Proc. BPM'10, pp. 91–105, (2010).

[39] Puhlmann, F., Schnieders, A., Weiland, J., Weske, M.: Variability mechanisms for process models. Technical report, BMBF-Project, (2006).

[40] Reichert. M., Weber, B.: Enabling flexibility in process-aware information systems: challenges, methods, technologies. Springer, (2012).

[41] Reinhartz-Berger, I., Sturm, A.: Enhancing UML models: a domain analysis approach. Journal on Database Management (special issue on UML Topics) 19, 1, pp. 74–94, (2008.)

[42] Robson, C.: Real world research: A resource for social scientists and practitioner-researchers (Vol. 2). Oxford: Blackwell, (2002).

[43] Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14, pp. 131–164, (2009).

[44] Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow data patterns. Technical Report FIT-TR-2004-01. Queensland Univ. of Technology, (2004).

[45] Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow resource patterns. Technical Report WP 127. Eindhoven University of Technology, (2004).

[46] Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Workflow exception patterns. In Proc. CAiSE'06, pp. 288-302, (2006).

[47] Sbai, H., Fredj, M., Kjiri, L.: Towards a process patterns based approach for promoting adaptability in configurable process models. In Proc. ICEIS'13, pp. 382–387, (2013).

[48] Scherer, R., Sharmak, W.: Process risk management using configurable process models. In Proc. IFIP AICT'11, pp. 341–348, (2011).

[49] Shull, F., Singer, J., Sjøberg, D.: Guide to advanced empirical software engineering. Ed. Forrest Shull. Vol. 93. Germany: Springer, (2008).

[50] Simmonds, J., Bastarrica, M., Silvestre, L., Quispe, A.: Analyzing methodologies and tools for specifying variability in sSoftware processes. Technical Report TR/DCC-2011-12. Universidad de Chile, (2011).

[51] The Stages process management system `http://stages.methodpark.com/` Accessed: October 2015

[52] Vogelaar, J.J.C.L., Verbeek, H.M.W., Luka, B., Aalst, W.M.P.: Comparing business processes to determine the feasibility of configurable models: A case study. In Proc. BPM'12 Workshops, pp. 50–61, (2012).

[53] Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - Enhancing flexibility in process-aware information systems. Data and Knoweldge Engineering 66(3), pp. 438–466, (2008).

[54] Weber, B., Reichert, M., Reijers, H.A., Mendling, J.: Refactoring large process model repositories. Computers in Industry 62(5), pp. 467–486, (2011).

[55] Weske, M.: Business process management: concepts, languages, architectures. Springer-Verlag Berlin Heidelberg Publisher, (2012).

[56] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wessln, A.: Experimentation in Software Engineering. Springer-Verlag Berlin Heidelberg, (2012).