UNIVERSIDAD POLITECNICA DE VALENCIA

PROS Centro de Investigación en Métodos de Producción de Software

Informe Técnico / Technical Report

Change Patterns for Process Families

Clara Ayora, Victoria Torres, Barbara Weber, Manfred Reichert, Vicente Pelechano

Ref. #:	PROS-TR-2013-06				
Title:	Change Patterns for Process Families				
Author (s):	Clara Ayora, Victoria Torres, Barbara Weber, Manfred Reichert, Vicente Pelechano				
Corresponding author (s):	cayora@pros.upv.es				
Document version number:	1.0	Final version:	Yes	Pages:	28
Release date:	February 28th, 2013				
Key words:	Process variability, process families, patterns, process change				

Change Patterns for Process Families[1]

Clara Ayora[1], Victoria Torres[1], Barbara Weber[2], Manfred Reichert[3], and Vicente Pelechano[1]
[1] Universitat Politècnica de València
cayora, vtorres, pele@pros.upv.es
[2] University of Innsbruck, Austria
barbara.weber@uibk.ac.at
[3] University of Ulm, Germany
manfred.reichert@uni-ulm.de

Abstract. The increasing adoption of process-aware information systems (PAISs), together with the variability of business processes (BPs), has resulted in large collections of related process model variants (i.e., process families). To effectively deal with process families, several proposals (e.g., C-EPC, Provop) exist that extend BP modeling languages with variability-specific constructs. While fostering reuse and reducing modeling efforts, respective constructs imply additional complexity and demand proper support for process designers when creating and modifying process families. Recently, generic and language-independent adaptation patterns were successfully introduced for creating and evolving single BP models. However, they are not sufficient to cope with the specific needs for modeling and evolving process families. This paper suggests a complementary set of generic and language-independent change patterns specifically tailored to the needs of process families. When used in combination with existing adaptation patterns, change patterns for process families will enable the modeling and evolution of process families at a high-level of abstraction. Further, they will serve as reference for implementing tools or comparing proposals managing process families.

Keywords: Process Variability, Process Families, Patterns, Process Change

1. Introduction

The increasing adoption of process-aware information systems (PAISs) has resulted in large process model repositories [25,6]. Since business process (BP) models usually may vary, existing repositories often comprise large collections of related *process model variants* (*process variants* for short) [24]. Usually, such process variants have common parts and pursue same or similar business objective, but at the same time differ regarding the application context in which

[1] This work has been developed with the support of MICINN under the Project EVERYWARE TIN2010-18011

they are used [12,25], e.g., countries' regulations, services delivered, or customer categories [23,6,24]. We denote such collections of related process variants as *process families*. In large companies, a process family might comprise dozens or hundreds of process variants [23]. For example, a process family for vehicle maintenance may comprise more than 900 variants with country-, garage-, and vehicle-specific differences [13]. In turn, [21] reports on a process family comprising more than 90 variants for planning and handling medical examinations. Designing and implementing each process variant model from scratch and maintaining it separately would be too inefficient and costly. Thus, there is a great interest in capturing common process knowledge only once and re-using it in terms of *configurable process models* representing the complete process family.

Motivated by the shortcomings of traditional BP modeling approaches [13], proposals exist for dealing with process families, e.g., [26,13,2]. Common to them is the extension of BP modeling languages with variability-specific constructs that enable the creation of configurable process models. By treating variability as first class citizen, these extensions help avoiding redundancies, fostering reusability, and reducing modeling efforts. However, introducing variability-specific constructs implies additional complexity concerning the modeling language. To make these proposals amenable for industrial strength use, the quality of created models becomes crucial. In turn, this necessitates proper support for process designers when creating and modifying process families.

In [32], a language-independent and empirically grounded set of adaptation patterns is proposed allowing for the creation and modification of single BP models [31]. Adaptation patterns not only allow creating and modifying BP models at a high level of abstraction, fostering model quality by ensuring correctness-by-construction, but also provide systematic means for realizing change operations optimized for a specific modeling language as well as comparing existing approaches in respect to BP flexibility [7]. Further, adaptation patterns have served as basis for implementing changes in different stages of the process lifecycle; e.g., model creation [30,10], process configuration [13], process instance change [5,9,22], model evolution [5,17], model refactoring [33], change reuse [3], model comparison [16], and change analysis [11].

While adaptation patterns are well suited for creating and modifying single BP models, they are not sufficient to cope with the specific needs for dealing with process families [4]. In the vein of adaptation patterns, this paper suggests a complementary set of generic, language-independent change patterns specifically tailored towards the needs of process families. Used in combination with the existing adaptation patterns, *change patterns for process families* will enable the modeling as well as evolution of process families at a high level of abstraction. In particular, they will serve as reference for specific language-dependent implementations, build the foundation for realizing changes along the BP lifecycle, and foster the comparison of existing proposals for BP variability.

Change patterns have been obtained after performing a systematic literature review looking specifically at variability-specific constructs used by proposals for BP variability. Since the proposed patterns are meant to be generic and language-independent, we abstract from approach-specific particularities. However, to ensure that the proposed patterns—despite their generic nature—are specific enough to cover existing proposals, we apply them to two well-known proposal dealing with process families, i.e., C-EPC and Provop.

The remainder of this report is structured as follows. Sect. 2 discusses related work and Sect. 3 presents the systematic literature review. In Sect. 4, we present the variability-specific language constructs obtained from the latter. Sect. 5 presents nine change patterns for process families. Sect. 6 provides a discussion and Sect. 7 concludes the paper.

2. Related Work

Closely related to our work is research on adaptation patterns, workflow patterns, and process variability.

Adaptation patterns (AP) [31] allow structurally changing process models using high-level change operations instead of low level change primitives (e.g., add or delete node). They can be applied along to the entire process lifecycle and do not have to be pre-planned, i.e., the region to which adaptation patterns may be applied can be chosen dynamically. Hence, adaptation patterns are well suited for realizing process changes at both build- and run-time. AP1 and AP2 allow inserting and deleting process fragments. 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 (AP9) or embedding them in a conditional branch (AP10), and adding/removing control dependencies (AP11, AP12). Finally, AP13 allows changing transition conditions. This paper complements adaptation patterns, which cover the basic use cases for creating and modifying process models, with a set of patterns covering variability needs in process families.

Workflow patterns were introduced for analyzing the expressiveness of process modeling languages. Patterns cover different perspectives like control flow [1], data [27], resources [28], time [18], and exceptions [29,20]. Further, [10] describes a set of pattern compounds, similar to adaptation patterns, allowing for the context-sensitive selection and pattern composition during process modeling. However, these patterns are not sufficient for effectively modeling and modifying process families. They do not consider variability-specific constructs introduced by process families and hence are complementary to our change patterns.

Proposals dealing with BP variability exist for modeling, configuring [26, 13], and maintaining process families; e.g., [15] provides a set of language-specific operators to adapt process variants at runtime based on software product line concepts. In [7], a combination of workflow-, rule-, and event-modeling is presented to customize process variants for a given execution context. Unlike these proposals, change patterns provide language-independent means to model and evolve process families at a high level of abstraction. Finally, there are refactoring techniques [33] to remove redundancies among process variants in large process model repositories.

3. Research Methodology

The *goal* of this paper is to provide a set of generic and language-independent patterns for modeling and evolving process families. We first present the research methodology we employed for identifying these patterns. To ensure that the latter are expressive enough to deal with the specific needs of process families, as basis, we identified the set of variability-specific language constructs frequently used by existing proposals to capture the variability within a process family. More precisely, we conducted a *systematic literature review* (SLR) [14] using the following procedure: (1) formulation of the research question, (2) description of a search strategy for finding relevant papers, (3) identification of inclusion and exclusion criteria, and (4) analysis of the data obtained. The main research question to be answered by the SLR is "*What variability-specific language constructs are provided by existing proposals for modeling BP variability and process families respectively?*". For this, we selected the following search string (considering different synonyms):

('process family' OR 'configurable process model' OR 'process model collection' OR 'reference process model' OR 'configurable workflow') OR 'process variant' OR 'business process variability' OR ('process configuration' OR 'process model configuration')

This search string was applied to relevant data sources: ACM Digital Library, IEEE Xplore Digital Library, Science Direct - Elsevier, SpringerLink, Wiley Inter Science, and World Scientific. Overall, these libraries include the proceedings of the most relevant conferences and journals in the area of BP management; e.g., *Data & Knowledge Engineering Journal*, *Information Systems Journal*, *Conference on Business Process Management* (BPM), *Conference on Advanced Information Systems Engineering* (CAiSE), and *Working Conference of Business Process Modeling, Development, and Support* (BPMDS). To apply the search string appropriately, it was adjusted where necessary (e.g., plural forms). As an additional data source, we considered the references of the identified papers.

A paper was included in the SLR (i.e., inclusion criterion) if and only if its title, abstract, and content is related to process families, either from a theoretical or practical perspective. On the contrary, papers were excluded (i.e., exclusion criterion) if their focus was not related to process families (e.g., software product lines). Papers describing the same proposal were removed and only the most complete version was included. We did not use any restriction concerning the publication date and only papers written in English were included. Finally, we only included proposals for which an implementation or evaluation exists.
Our SLR resulted in a total of 4960 papers, which were manually reviewed based on their titles and abstracts. In total, 25 papers passed this filtering and were further analyzed. To identify the language constructs commonly used in BP proposals (and serving as basis for our change patterns), we first create a list of candidate constructs. Thereby, we relied on our experience with process families [4,31,33]. Then, we analyzed the 25 identified papers to find empirical evidence for our candidate variability-specific language constructs and iteratively refined the initial list. Finally, only those constructs for which enough empirical evidence exists were included in the final list of variability-specific constructs.

Although proposals use different terminology and realize constructs in different ways, the SLR revealed that they essentially support the same language constructs for dealing with BP variability. We identified four variability-specific language constructs commonly shared by the 25 proposals: *configurable region, configuration alternative, context condition, and configuration constraint* (see Sect. 4.1 for details). Configurable regions are supported by 20 of the 25 proposals and configuration alternatives by 22 proposals. Context conditions are covered by 16 proposals while 15 proposals support the definition of configuration constraints. Additional language constructs we identified (e.g., *configurable region resolution time*) are only considered by few proposals (<3) and are therefore not included in our final list of variability-specific language constructs (for further details on the SLR see[2]).

The final list of four variability-specific language constructs was then used as a basis for the change patterns. Since the proposed patterns are meant to be generic and language-independent, we abstracted from approach-specific particularities (cf. Sect. 4). Thereby, we focused on the *control flow* perspective since the SLR showed that this is the perspective mostly addressed by existing proposals. To ensure that the proposed patterns—despite their generic nature—are specific enough to cover existing proposals, we applied the respective patterns to two well-known proposal dealing with process families (cf. Sect. 5).

[2] https://pros.webs.upv.es/bpvar/SLR/BPvariability.rar

4. Coping with Variability in Business Process Families

This section describes the variability-specific language constructs obtained from the SLR and introduces two representative proposals. For the latter, we will show in Sect. 5 how the change patterns can be realized. For illustration purpose, we make use of the process carried out when checking-in at an airport. We chose this process since it shows a high degree of variability; e.g., variability occurs due to the type of check-in (e.g., online, or at a counter), which also determines the type of boarding card (e.g., electronic vs. paper-based). Other sources of variability include the type of passenger (e.g., unaccompanied minors requiring extra assistance) and the type of luggage (e.g., overweight luggage).

4.1 Coping with Variability in Business Process Families

The SLR described in Sect. 3 has revealed that the following language constructs are commonly used by existing proposals to capture variability (although their concrete realization might differ) in addition to standard process modeling constructs (e.g., activities and gateways). These language constructs form the basis of the *change patterns for process families* (see Sect. 5).

- **Language Construct LC1 (Configurable Region).** A *configurable region* is a region in a configurable process model for which different configuration choices may exist depending on the application context, e.g., the airline may offer different ways of obtaining the boarding cards depending on how the check-in is accomplished: printing a boarding card at the airline desk, download an electronic boarding card at home, or obtaining it via mobile phone.

- **Language Construct LC2 (Configuration Alternatives)**. A *configuration alternative* is defined as a particular configuration choice that may be selected for a specific configurable region, e.g., there exist different types of boarding card: paper-based, electronic, or in the mobile phone.

- **Language Construct LC3 (Context Condition)**. A *context condition* defines the conditions under which a particular configuration alternative of a configurable region shall be selected, e.g., only passengers carrying an overweight luggage may have to pay an excess fee.

- **Language Construct LC4 (Configuration Constraint).** A configuration constraint is defined as a restriction of the selection of configuration alternatives of the same or different configurable regions. Respective constraints are based on semantic restrictions to ensure the proper use of configuration alternatives, e.g., staff members need to be localized when unaccompanied minors are travelling.

4.2 Proposals Dealing with Process Families

The SLR described in Sect. 3 identified 25 proposals for dealing with process families. In the following, we describe two of them in more detail and explain how the variability-specific language constructs introduced in Sect. 4.1 have been realized by these proposals. Sect. 5 will then apply the identified change patterns to these two proposals to demonstrate that the proposed patterns are indeed generic. As representatives, we select two proposals that are (1) well established and highly cited, and (2) take fundamentally different approaches to realize the variability-specific language constructs. This way we want to ensure that the proposed patterns are general enough to cover very distinct proposals, but still specific enough to cover their essence.

C-EPC. A possible way of specifying a configurable process model is by means of *configurable nodes*. Modeling languages supporting this approach include, for example, C-EPC and C-YAWL [8]. Basically, these proposals extend an existing BP modeling language by adding configurable elements for explicitly representing variability in process families. In the following, we take C-EPC [26] as representative of this approach since it constitutes a well-known proposal. Fig. 1 illustrates the configurable process model as C-EPC for the check-in process. Configurable nodes are depicted with a thicker line. A configurable region (LC1) in C-EPC is specified by a process fragment of the configurable process model with exactly one entry and one exit (i.e., SESE fragment), and may take two different forms. First, the SESE fragment may consist of a splitting *configurable connector*, immediately followed by a set of branches representing configuration alternatives, and a joining *configurable connector*; i.e., the configurable connectors delimit the configurable region (e.g., Configurable region 2 in Fig. 1). Alternatively, the SESE fragment may consist of a *configurable function* (e.g., Configurable region 1 and 3 in Fig. 1), which may be configured as ON (i.e., the function is kept in the model), OFF (i.e., the function is removed from the model), or OPT (i.e., a conditional branching is included in the model deferring the decision to run-time). In turn, a configuration alternative (LC2) is specified by a SESE fragment which may be included as a branch between two configurable connectors (e.g., *Print electronic boarding card* in Configurable region 2 in Fig. 1). Context conditions (LC3) are represented in C-EPC separately in a questionnaire model [19], which is not considered in this paper. Finally, a configuration constraint (LC4) is specified by a *configuration requirement* linked to the configurable nodes that delimit the configurable region to which the respective configuration alternatives belong (e.g., Configuration requirement 1 in Fig. 1 states that the inclusion of the function *Fill in UM* form implies the inclusion of the function *Localize staff*).

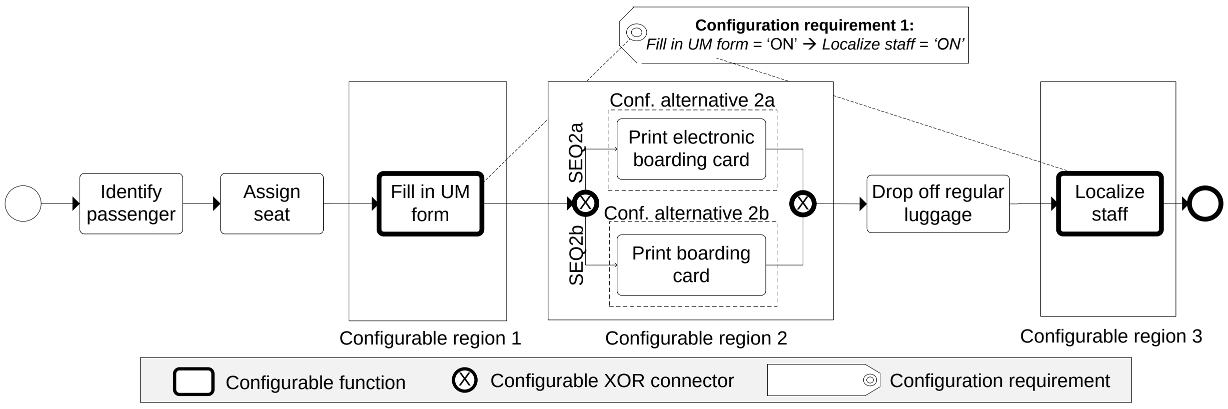


Fig. 1. C-EPC configurable process model for the check-in process

Provop. Another way of handling process families is based on the observation that process variants are often derived by adapting a pre-specified *base process model* (*base process*, for short) to the given context through a sequence of structural adaptations. The Provop proposal follows this approach [13]. We choose it since it provides advanced tool support for adapting a base process and for ensuring syntactical and semantical correctness of process variants derived. Fig. 2 illustrates how the process family dealing with the check-in process can be represented using Provop. The top of Fig. 2 shows the *base process* model from which the process variants may be derived. In Provop, a configurable region (LC1) is specified by a SESE fragment of the *base process*, delimited by two *adjustment points*; i.e., black diamonds (e.g., Configurable region 1 comprises the process fragment delimited by adjustment points A and B in Fig. 2). In turn, a configuration alternative (LC2) is specified by a *change option* that includes (1) the list of *change operations* modifying the base process at a specific configurable region and (2) a *context rule* that defines the context conditions under which the change operations shall be applied (e.g., Opt. 1 in Fig. 2). Context conditions (LC3) are specified by context rules which include a set of context variables and their values specifying the conditions under which a configuration alternative (i.e., a change *option*) shall be applied (e.g., Opt. 2 is applied if the check-in type is online). All context variables and their allowed values are gathered all together in the *context model* (cf. Fig. 2C). Finally, configuration constraints (LC4) are specified as constraints (e.g., mutual exclusion, inclusion) between two *change options* in the *option constraint model*; e.g., if Opt. 2 is applied then Opt. 3 has to be applied as well (cf. Fig. 2C).

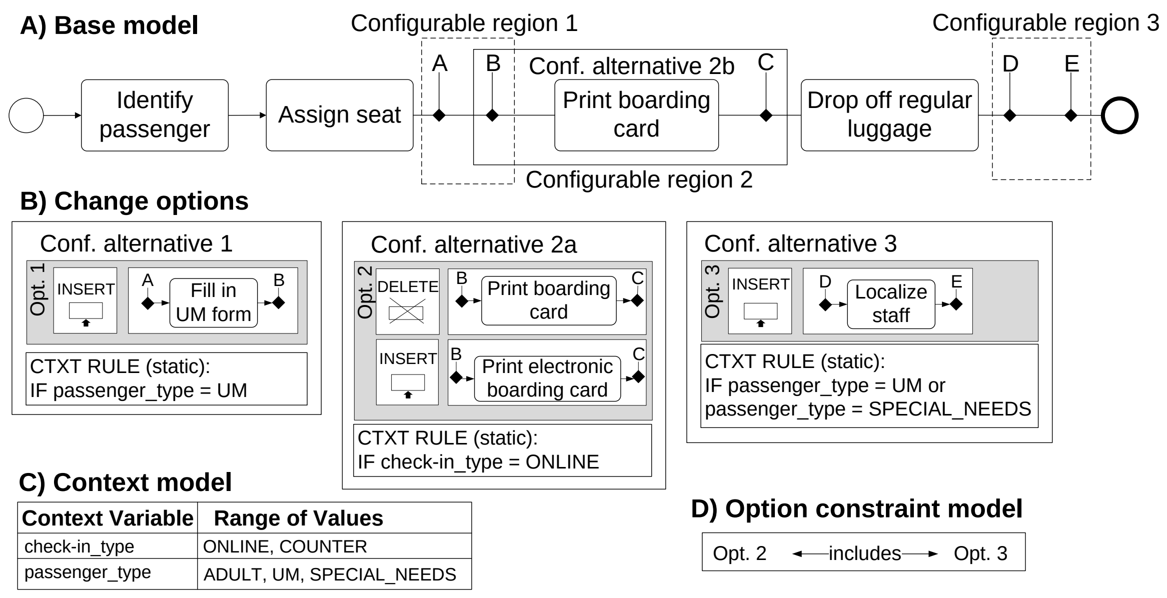


Fig. 2. Provop model for the check-in process

5. Coping with Variability in Business Process Families

This section presents nine change patterns we consider as relevant for dealing with changes in process families. These patterns refer to the four variability-specific language constructs we obtained from our systematic literature review. They are generic in the sense that they abstract from proposal-specific details. Moreover, they intend to be complete regarding the control flow perspective and cover all changes related to commonly used variability-specific language constructs. Further, we suppose that the change patterns will be combined with adaptation patterns to allow for the modeling and evolution of process families at a high level of abstraction. As illustrated in Table 1, we divide the change patterns into three categories: *insertion*, *deletion*, and *modification* of variability-specific parts of a configurable process model.

CP1: INSERT Configurable Region
CP2: DELETE Configurable Region
CP3: INSERT Configuration Alternative IN a Configurable Region
CP4: DELETE Configuration Alternative IN a Configurable Region
CP5: INSERT Context Condition OF a Configuration Alternative
CP6: DELETE Context Condition OF a Configuration Alternative
CP7: MODIFY Context Condition OF a Configuration Alternative
CP8: INSERT Configuration Constraint BETWEEN Configuration Alternatives
CP9: DELETE Configuration Constraint BETWEEN Configuration Alternatives

Table 1. Change patterns for process families

All change patterns, except CP7, allow adding (removing) variability-specific language constructs to (from) a configurable process model, representing the process family. In turn, pattern CP7 allows changing the conditions under which a configuration alternative is selected. To keep the pattern set minimal, we do not consider patterns for *modifying* configurable regions, configuration alternatives, and configuration constraints. These modifications can be realized based on the combined use of change patterns and adaptation patterns. For example, modifying a configuration alternative may be implemented applying patterns CP3 and CP4 (INSERT/DELETE Configuration Alternative IN a Configurable Region), which, in turn, make use of respective adaptation patterns. Further, adding or removing process fragments which are shared by all process variants (i.e., commonalities), may be realized using adaptation patterns AP1 and AP2 (INSERT/DELETE Process Fragment).

For each of the change patterns, we provide a name, a brief description, an illustrative example, a description of the problem addressed, and corresponding design choices (indicating pattern variants). For example, CP1 (i.e., INSERT a Configurable Region) presents three design choices (cf. Fig. 3): insert a configurable region as a new process region with a set of new configuration alternatives, inserting it by transforming a commonality into a configuration alternative (i.e., a common process fragment now is only applied in a specific application context), or by transforming a set of commonalities into a set of configuration alternatives. To demonstrate that the patterns—despite their generic nature—still cover the essence of different proposals for BP variability, we apply them to C-EPC and Provop, and show how they can be realized in their context. For example, regarding CP1 (cf. Figs. 3-4), for each design choice, we indicate for both C-EPC and Provop how CP1 can be implemented using adaptation patterns. Further, note that for C-EPC we provide implementation details distinguishing between (i) *configurable functions* and (ii) *configurable connectors* since both allow representing configurable regions. 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 indicating how change patterns may be realized.

Pattern CP1: INSERT Configurable Region

Description: In a configurable process model, a configurable region 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.

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

Design choices: Three different design choices (DCs) exist:

DC1) Insertion as a new configurable region with up to *n* configuration alternatives (n ≥ 0)

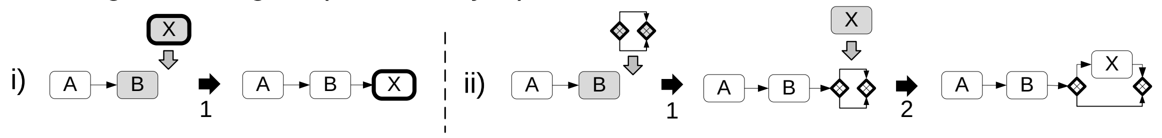
DC2) Insertion as a new configurable region by transforming a common process fragment into a configuration alternative

DC3) Insertion as a new configurable region by transforming existing process fragments into a set of configuration alternatives

Implementation in C-EPC:

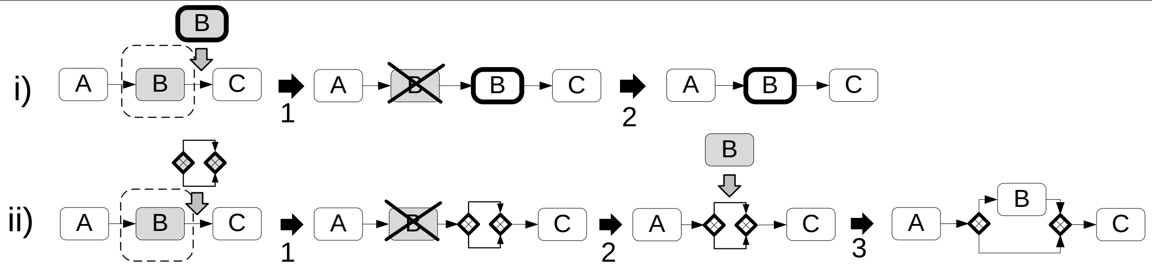
For DC1, CP1 is realized by

1. applying adaptation pattern AP1 (i.e., *INSERT Process Fragment*) to insert the configurable region using either (i) a *configurable function* 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 B),
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 X).



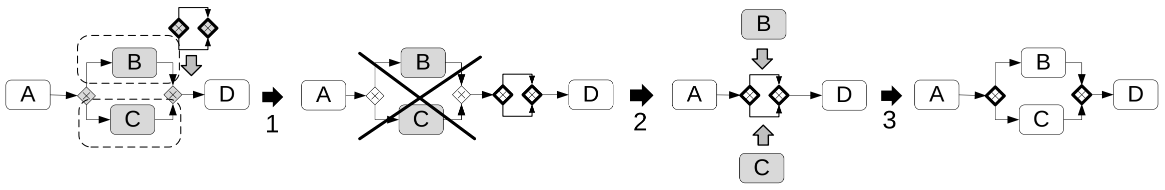
For DC2, CP1 is realized by

1. applying adaptation pattern AP1 (i.e., *INSERT Process Fragment*) to insert the configurable region using either (i) a *configurable function* 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 B),
2. applying adaptation pattern AP2 (i.e., *DELETE Process Fragment*) to delete the common process fragment from its current position (i.e., activity B), 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 configuration alternative is added as a branch between the two *configurable connectors* delimiting the configurable region (i.e., activity B).



For DC3, CP1 is realized by

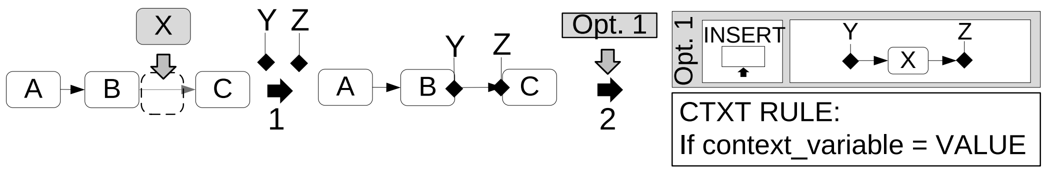
1. applying adaptation pattern AP1 (i.e., *INSERT Process Fragment*) to insert the configurable region (only relevant for *configurable connectors*) at the precise position where the configurable region should be located (i.e., after the join XOR gateway),
2. applying adaptation pattern AP2 (i.e., *DELETE 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 branch of the process fragment is added as a branch between the two *configurable connectors* delimiting the configurable region (i.e., activity B is inserted as one alternative and activity C as another one).



Implementation in Provop:

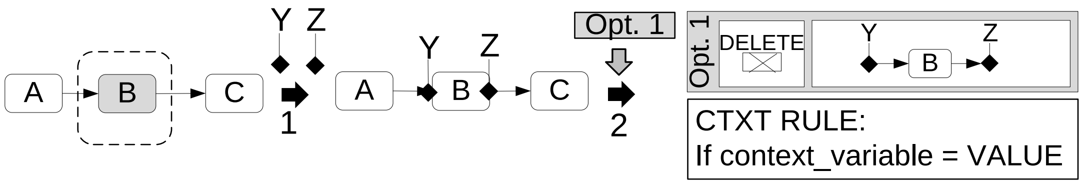
For DC1, CP1 is realized by

1. inserting two *adjustment points* (i.e., Y and Z) in the *base process* and
2. applying repeatedly CP3 (*INSERT Configuration Alternative IN a Configurable Region)* once for each new configuration alternative to define respective *change options* (i.e., Opt. 1).



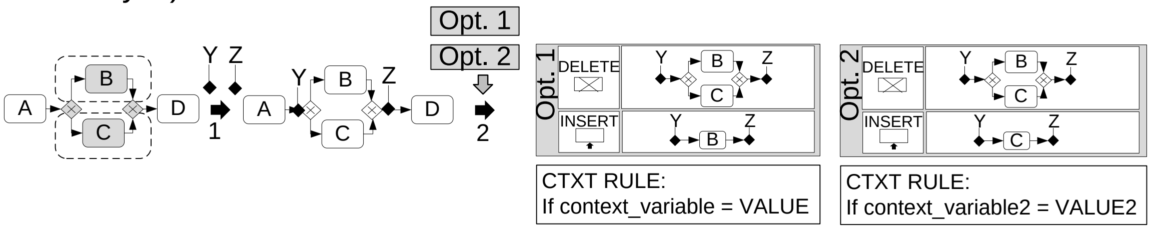
For DC2, CP1 is realized by

1. inserting two *adjustment points* (i.e., Y and Z) embedding an existing process fragment of the *base process* (i.e., activity B) and
2. applying CP3 (*INSERT Configuration Alternative IN a Configurable Region)* to define a configuration alternative in terms of a *change option* inserting the existing process fragment into/removing the existing process fragment under certain conditions from the *base process* (i.e., *Opt. 1*).

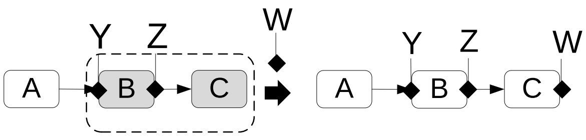


For DC3, CP1 is realized by

1. inserting two *adjustment points* (i.e., Y and Z) embedding an existing process fragment of the *base process* (i.e., the process fragment becomes optional) and
2. applying repeatedly CP3 (*INSERT Configuration Alternative IN a Configurable Region)* to define the set of configuration alternatives in terms of *change options* inserting/removing existing process fragments under certain conditions from the *base process* (i.e., one option for activity B and another one for activity C).



If *adjustment points* already exist at the entry or exit of the new configurable region (e.g., as part of another configurable region) these may be reused instead.



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

Fig. 3. CP1 (INSERT Configurable Region)

Pattern CP2: DELETE Configurable Region

Description: In a configurable process model, a configurable region shall be deleted.

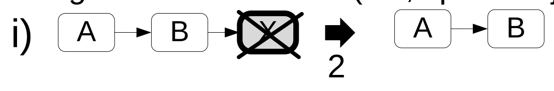
Example: Assume that a configurable region, capturing the variability for obtaining a boarding card, exists (i.e., paper vs. electronic document). However, in order to save money, the airline now only offers the electronic-based boarding card (i.e., other configuration alternatives are no longer offered)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

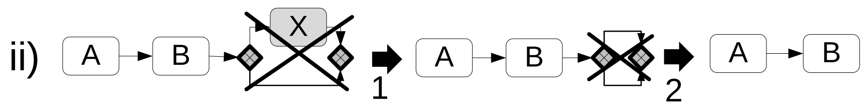
Problem: A configurable region is no longer needed and thus it shall be deleted.

Design choices: Three different design choices (DCs) exist:
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, CP1 is realized by

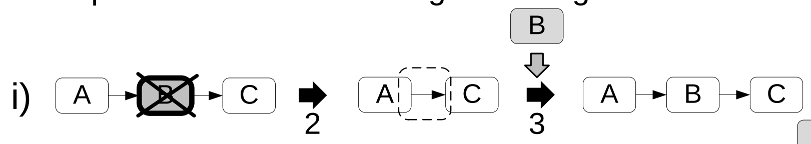
1. applying repeatedly change pattern CP4 (i.e., *DELETE Configuration Alternative IN a Configurable Region*) to delete each existing configuration alternative; i.e., once per configuration alternatives (only relevant for *configurable connectors,* i.e., activity X),
2. applying adaptation pattern AP2 (*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).

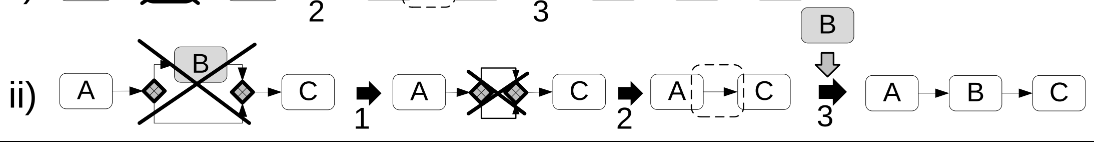




For DC2, CP1 is realized by

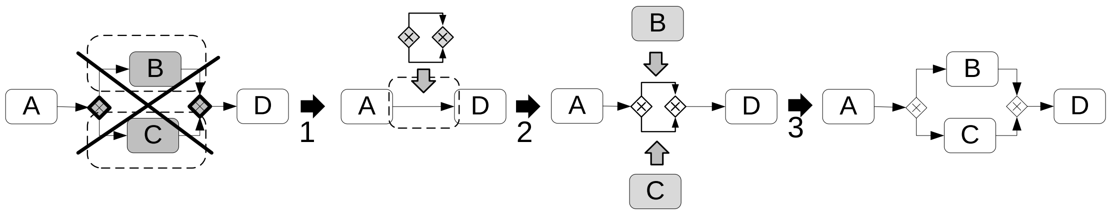
1. applying repeatedly CP4 (*DELETE Configuration Alternative IN a Configurable Region)* to delete the existing configuration alternatives of the configurable region (only relevant for *configurable connectors*, i.e., activity B),
2. applying adaptation pattern AP2 (i.e., *DELETE Process Fragment*) to 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 (i.e., *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 B).





For DC3, CP1 is realized by

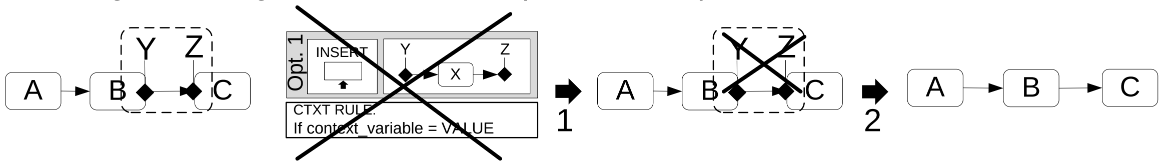
1. applying adaptation pattern AP2 (i.e., *DELETE Process Fragment*) to delete the existing process fragment (including the configurable region and its alternatives) from its current position,
2. applying adaptation pattern AP1 (i.e., *INSERT Process Fragment*) to re-insert at the precise position where the configurable region was located a process fragment consisting of a two non-configurable connectors, and
3. applying repeatedly adaptation pattern AP1 (i.e., *INSERT Process Fragment*) to re-insert the deleted configuration alternatives as branches between the two recently added non-configurable connectors (i.e., activity B is inserted as one branch and activity C as another one).



Implementation in Provop:

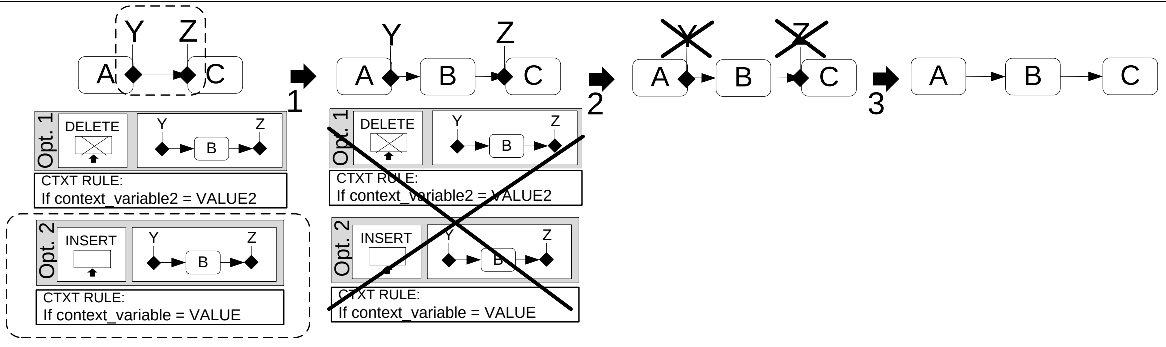
For DC1, CP1 is realized by

1. applying repeatedly CP4 (*DELETE Configuration Alternative IN a Configurable Region)* once for each configuration alternative defined by respective *change options* (i.e., *Opt. 1*) and
2. deleting the two *adjustment points* in the *base model* that delimit the configurable region to be deleted (i.e., Y and Z).



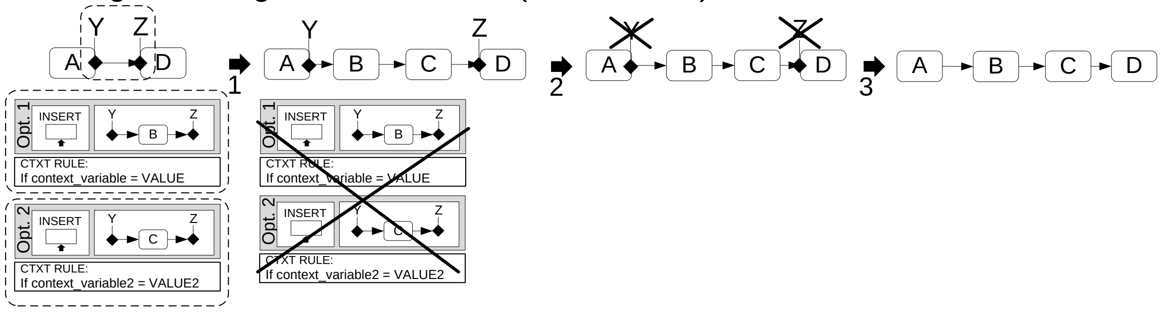
For DC2, CP1 is realized by

1. applying the *change option* defining the configuration alternative to be kept (i.e., *Opt. 2*),
2. applying repeatedly CP4 (*DELETE Configuration Alternative IN a Configurable Region)* once for each configuration alternative defined by respective *change options* (i.e., *Opt. 1* and *Opt. 2*), and
3. deleting the two *adjustment points* in the *base model* that delimit the configurable region to be deleted (i.e., Y and Z).

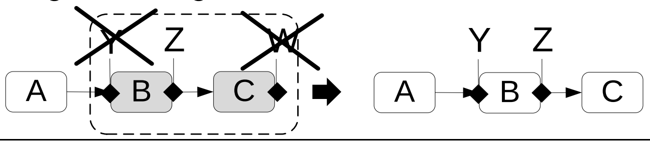


For DC3, CP1 is realized by

1. applying the *change options* defining the configuration alternatives that should be kept (i.e., *Opt. 1* and *Opt. 2*),
2. applying repeatedly CP4 (*DELETE Configuration Alternative IN a Configurable Region)* once for each configuration alternative to define respective *change options* (i.e., *Opt. 1* and *Opt. 2*), and
3. deleting the two *adjustment points* in the *base model* that delimit the configurable region to be deleted (i.e., Y and Z).



If an *adjustment point* (to be deleted) constitutes the entry or exit of another configurable region, it must not be deleted.



Parameters:

- the configurable region to be deleted
- the configuration alternative(s) that should be kept

Fig. 4. CP2 (DELETE Configurable Region)

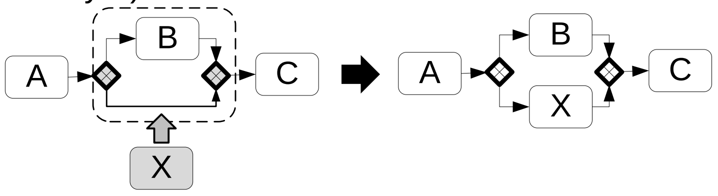
Pattern CP3: INSERT Configuration Alternative IN a Configurable Region

Description: In a configurable process model, a configuration alternative shall be added to a specific configurable region.

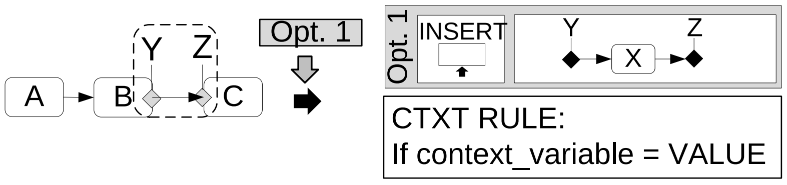
Example: Assume that a configurable region, capturing the variability for obtaining a boarding card, exists (i.e., paper vs. electronic document). Assume further that the airline now wants to offer the possibility of obtaining the boarding card for smart phones as well. Thus, a alternative shall be added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

Problem: For a specific configurable region of the configurable process model, existing configuration alternatives do not cover all possible configuration choices and hence an additional configuration alternative shall be added.

Implementation in C-EPC: CP3 is realized by applying adaptation pattern AP1 (i.e., *INSERT Process Fragment*) to insert the process fragment representing the configuration alternative, i.e., the configuration alternative is added as a branch between the two *configurable connectors* delimiting the configurable region (i.e., activity X)[3].



Implementation in Provop: CP3 is realized by defining a *change option* consisting of a sequence of *change operations* and a *context rule.*



Parameters:

- the configurable region to which the configuration alternative belongs
- the configuration alternative to be inserted

Fig. 5. CP3 (INSERT Configuration Alternative IN a Configurable Region)

[3] Note that we do not consider here configuration alternatives referred by *configurable functions* (i.e., ON, OFF, OPT) since they are implicitly inserted when inserting a *configurable function* as a configurable region.

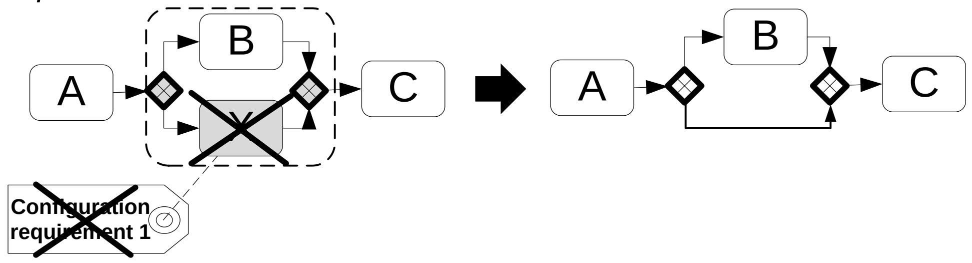
Pattern CP4: DELETE Configuration Alternative IN a Configurable Region

Description: In a configurable process model, a configuration alternative of a specific configurable region shall be deleted.

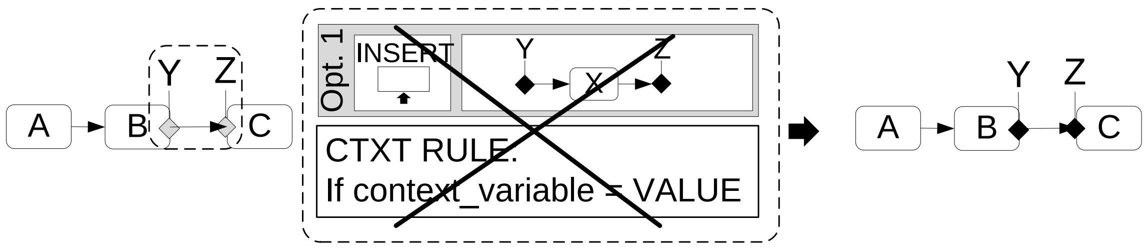
Example: Assume that a configurable region capturing the variability for obtaining a boarding card exists (i.e., paper vs. electronic document). Assume further that for economic reasons, the airline does not offer paper-based boarding cards anymore allowing only electronic and mobile phone ones. Thus, the configuration alternative printing a paper boarding card is no longer needed.

Problem: A configuration alternative is no longer needed and thus shall be deleted.

Implementation in C-EPC: CP4 is realized by applying adaptation pattern AP2 (i.e., *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 X)[4]. 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*.



Implementation in Provop: CP4 is realized by deleting a *change option* consisting of a sequence of *change operations* and a *context rule.*



Associated *context rules* may be deleted as well if they are no needed anymore in other *context rules*. This can be done applying CP6 (*DELETE Context Conditions OF a Configuration Alternative*). In addition, if the change option deleted is the only one referring to the configurable region, the latter may be deleted as well applying CP2 (*DELETE Configurable Region*). Furthermore, associated option constraints defined in the option constraint model may be deleted as well by applying CP9 (*DELETE Constraint BETWEEN Configuration*

[4] Note that we do not consider here configuration alternatives referred by *configurable functions* (i.e., ON, OFF, OPT) since they are implicitly deleted when deleting a *configurable function* as a configurable region.

Alternatives).

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

Fig. 6. CP4 (DELETE Configuration Alternative IN a Configurable Region)

Pattern CP5: INSERT Context Condition OF a Configuration Alternative

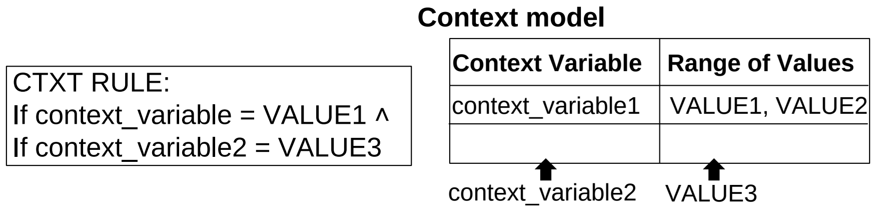
Description: In a configurable process model, a context condition related to a configuration alternative of a configurable region shall be added to define when the configuration alternative shall be selected

Example: A passenger who carries luggage exceeding 20kg must pay an extra fee (where *luggage exceeding 20kg* refers to the new context condition).

Problem: A context condition shall be added to a configurable process model to specify the condition under which a particular configuration alternative of a configurable process model shall be selected.

Implementation in C-EPC: CP5 is not applicable since context information is captured separately in the *questionnaire model*, which is not considered in this work.

Implementation in Provop: CP5 is realized by adding a context rule. If the context rule includes context variables (or values) not yet defined in the *context model*, the latter must be updated accordingly, i.e., context variables (or values) are inserted.



CTXT RULE:
If context_variable = VALUE1 ^
If context_variable2 = VALUE3

Context model

Context Variable	Range of Values
context_variable1	VALUE1, VALUE2

Parameters:
- the context condition to be inserted

Fig. 7. CP5 (INSERT Context Condition OF a Configuration Alternative)

Pattern CP6: DELETE Context Condition OF a Configuration Alternative

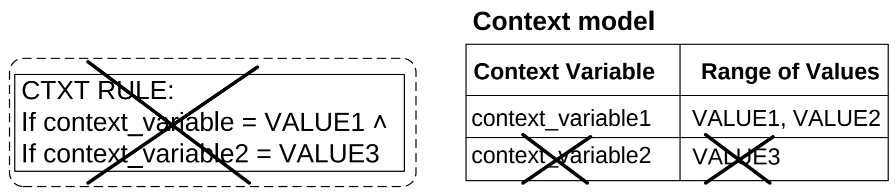
Description: In a configurable process model, a context condition of a configuration alternative is deleted

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.

Problem: A context condition is no longer needed for selecting a configuration alternative in a configurable region and thus shall be deleted.

Implementation in C-EPC: CP6 is not applicable since context information is captured separately in the *questionnaire model*, which is not considered in this work.

Implementation in Provop: CP6 is realized by deleting a context rule. If the context rule includes context variables (or values) not used by any other *context rule,* the *context model* must be updated accordingly, i.e., context variables (or values) are deleted.



~~CTXT RULE:~~
~~If context_variable = VALUE1 ∧~~
~~If context_variable2 = VALUE3~~

Context model

Context Variable	**Range of Values**
context_variable1	VALUE1, VALUE2
~~context_variable2~~	~~VALUE3~~

Parameters:
- the context condition to be deleted

Fig. 8. CP6 (DELETE Context Condition OF a Configuration Alternative)

Pattern CP7: MODIFY Context Condition OF a Configuration Alternative

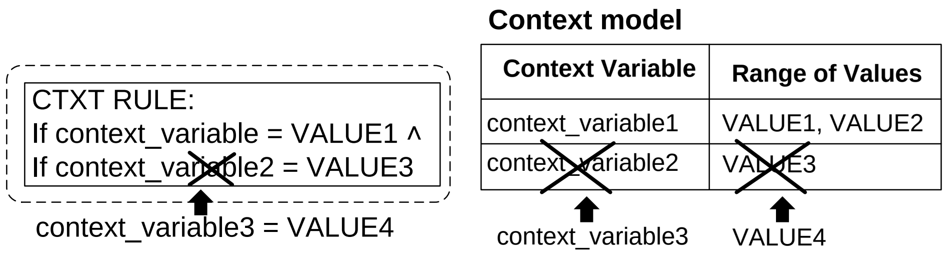
Description: In a configurable process model, a context condition is modified.

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**.**

Problem: A context condition is no longer adequate and shall be modified in the configurable process model.

Implementation in C-EPC: CP7 is not applicable since context information is captured separately in the *questionnaire model*, which is not considered in this work.

Implementation in Provop: CP7 is realized by modifying a *context rule* either inserting or deleting its context variables, or modifying their values. If new context variables (or values) are inserted, the *context model* should be updated accordingly. On the contrary, if deleted context variables (or values) are not used in other *context rules,* they are deleted from the *context model*.



Parameters:

- the context condition to be modified

Fig. 9. CP7 (MODIFY Context Condition OF a Configuration Alternative)

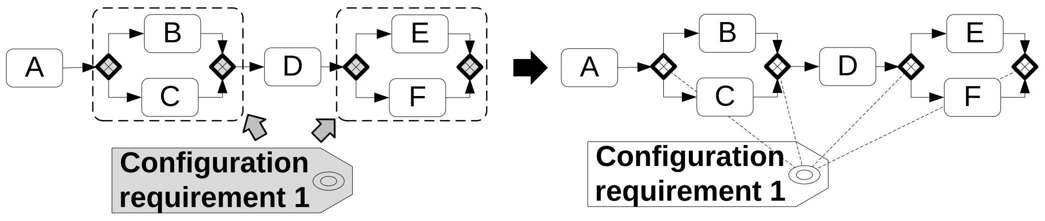
Pattern CP8: INSERT Configuration Constraint BETWEEN Configuration Alternatives

Description: In a configurable process model, a constraint regarding the use of configuration alternatives from one or more configurable regions shall be added.

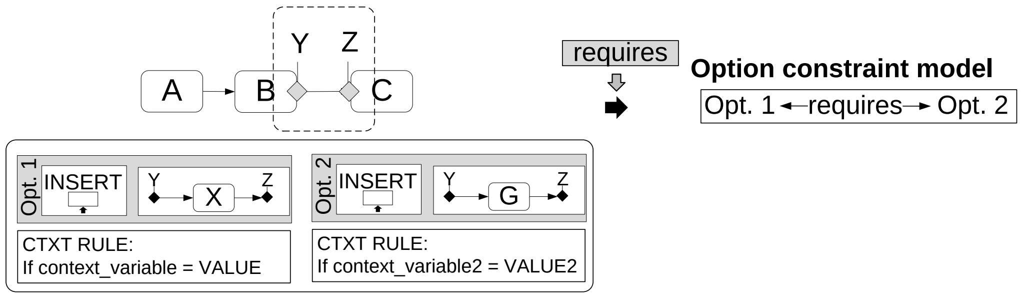
Example: When unaccompanied minors are travelling, extra staff is required to accompany them to the boarding gate, i.e., an inclusion constraint exists.

Problem: The use of configuration alternatives needs to be constrained in a configurable process model and thus a configuration constraint shall be added.

Implementation in C-EPC: CP8 is realized by inserting a *configuration requirement,* which is then linked to the configurable nodes that delimit the configurable region to which the respective configuration alternatives to be related belong.



Implementation in Provop: CP8 is realized by adding a constraint regarding the use of *change options* in the *option constraint model* [5].



Parameters:

- the configuration region to which the alternatives whose use shall be constrained belong
- the configuration constraint to be inserted

Fig. 10. CP8 (INSERT Configuration Constraint BETWEEN Configuration Alternatives)

[5] We only consider configuration constraints between *change options* since configuration constraints between *change operations* are implicitly specified by their definition

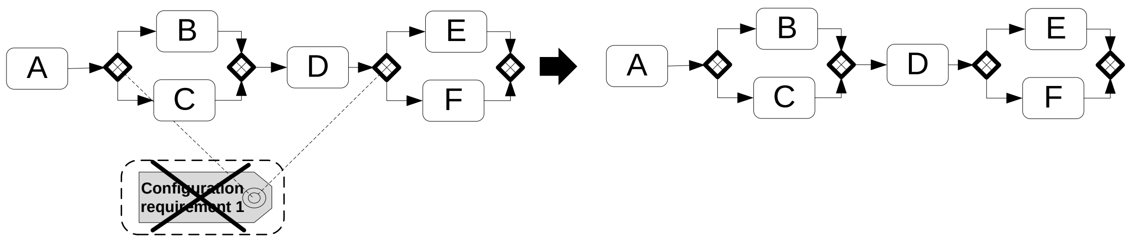
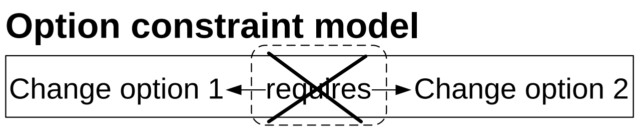
Pattern CP9: DELETE Configuration Constraint BETWEEN Configuration Alternatives
Description: In a configurable process model, a constraint between two or more configuration alternatives from one or more configurable regions shall be deleted.
Example: When unaccompanied minors are travelling, extra staff is required to accompany them to the boarding gate (i.e., inclusion constraint). Due to emerging legal regulations, from now on their relatives shall accompany them, i.e., the inclusion constraint is no longer needed.
Problem: A constraint between two or more configuration alternatives is no longer needed and thus it shall be deleted.
Implementation in C-EPC: CP9 is realized by deleting a *configuration requirement,* which is linked to the configurable nodes that delimit the configurable region to which the respective configuration alternatives belong.  **Implementation in Provop**: CP9 is realized by deleting a constraint between two *change options* in the *option constraint model* [6].  Option constraint model Change option 1 requires Change option 2
Parameters: - the configuration constraint to be deleted

Fig. 11. CP9 (DELETE Configuration Constraint BETWEEN Configuration Alternatives)

[6] We only consider configuration constraints between *change options* since configuration constraints between *change operations* are implicitly specified by their definition

6. Discussion

Even though—as shown by the systematic literature review—existing proposals use different terminology and realize the constructs in different ways, they essentially support the same variability-specific language constructs. Similar to adaptation patterns, we expect that the change patterns have the potential to speed up the creation as well as modification of configurable process models. In addition, like adaptation patterns, the *change patterns for process families* may therefore serve as benchmark for evaluating change support in existing languages and tools dealing with process families as well as for facilitating their systematic comparison by providing a frame of reference. To substantiate these claims, we plan to conduct empirical studies testing the impact of the proposed patterns on both the creation and evolution of configurable process models. Moreover, in a similar vein than adaptation patterns, the proposed change patterns may serve as a reference for realizing changes in different stages of the process family life cycle, e.g., modeling, maintenance, and evolution.

As with every research, our work is subject to limitations. A first one concerns the completeness of the proposed patterns. We tried to accommodate this by grounding patterns on a SLR covering 25 different proposals for process families and by using variability-specific language requirements commonly occurring as basis for our patterns. As a consequence, we are confident that the proposed pattern set is complete in the sense that it allows modeling and modifying process families. However, we cannot state with certainty that the identified patterns set is sufficiently large to address all potential use cases regarding the modeling and change of process families in the most efficient way. For this, empirical studies on the practical use of the patterns are needed. Closely related to this are considerations regarding the language-independent nature of the proposed patterns. Using commonly occurring variability-specific constructs as a basis, we can ensure that the proposed patterns are expressive enough to model and modify process families. To ensure that the patterns are also specific enough to cover the particularities of the different proposals, we applied them to two commonly used and entirely different proposals for process families. To strengthen the validation of the patterns, they will be applied to other proposals in future work. Moreover, the focus of the proposed patterns is currently on variability-specific constructs regarding the *control flow* perspective. Variability regarding additional perspectives like data or resources has not been considered so far.

The proposed patterns have been described in an informal way. To obtain unambiguous pattern descriptions and ground pattern implementation as well as pattern-based analysis on a sound basis, a formal semantics is needed. This formalization should be independent from any process meta model and thus allow implementing the patterns in a variety of process support tools.

7. Conclusions and Outlook

We proposed nine patterns for dealing with changes in process families. We complement existing work on patterns for creating and modifying BP models by introducing a set of generic and language-independent patterns that cover the specific needs of process families. The patterns are based on variability-specific language constructs. To demonstrate that they still cover the essence of existing proposals managing BP variability, we applied them to two representative proposals. Used in combination with adaptation patterns, change patterns for process families allow modeling and evolving process families at an abstract level. In future work, we will develop a prototype based on which we will conduct experiments to measure the efforts of handling variability in process families. We will study the impact of patterns on modeling process families as well as on changing either at design or run-time.

References

1. van der Aalst, W.M.P., ter Hofstede, A., Barros, B.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003).
2. Aiello, M., Bulanov, P., Groefsema, H.: Requirements and Tools for Variability Management. In Porc. IEEE 34th Annual Computer Software and Applications Conference Workshops, 245-250 (2010).
3. Aghakasiri, Z., Mirian-Hosseinabadi, S.H.: Workflow change patterns: Opportunities for extension and reuse. In Proc. SERA'09, 265-275 (2009).
4. 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, 477–488 (2012).
5. Dadam, P., Reichert, M.: The ADEPT project: a decade of research and development for robust and flexible process support. Com Sci - R&D 23, 81–97 (2009).
6. Dijkman, R., La Rosa, M., Reijers H.A: Managing large collections of business process models - Current techniques and challenges, Comp in Ind 63(2), 91–97 (2012).
7. Döhring, M., Zimmermann, B., Karg, L.: Flexible workflows at design- and runtime using BPMN2 adaptation patterns. In Proc. BIS'11, 25–36 (2011).
8. Gottschalk, F.: Configurable process models. Ph.D. thesis, Eindhoven University of Technology, The Netherlands (2009).
9. Grambow, G., Oberhauser, R., Reichert, M.: Contextual injection of quality measures into software engineering processes. Intl J Adv in Software 4, 76-99 (2011).
10. Gschwind, T., Koehler, J., Wong, J.: Applying patterns during business process modeling. In: Proc BPM'08, 4–19 (2008).

11. Günther, C.W., Rinderle, S., Reichert, M., van der Aalst, W.M.P.: Change mining in adaptive process management systems. In Proc. CoopIS'06, 309–326 (2006).
12. Hallerbach, A., Bauer, T., Reichert, M.: Context-based configuration of process variants. In Proc. TCoB'08, 31–40 (2008).
13. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the Provop approach. J of Software Maintenance 22(6–7), 519–546 (2010).
14. Kitchenham, B., Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering, Technical Report EBSE/EPIC–2007–01 (2007).
15. Kulkarni, V, Barat, S., Roychoudhury, S.: Towards business application product lines. In Proc. MoDELS'12, 285–301 (2012).
16. Küster, J., Gerth, C., F¨orster, A., Engels, G.: Detecting and resolving process model differences in the absence of a change log. In Proc. BPM'08, 244–260 (2008).
17. Küster, J., Gerth, C., Engels, G.: Dynamic computation of change operations in version management of business process models. In: ECMFA'10, 201-216 (2010).
18. Lanz, A., Weber, B., Reichert, M.: Time patterns for process-aware information systems. Requirements Engineering, 1–29 (2012).
19. La Rosa, M., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Questionnaire-based variability modeling for system configuration. Software and System Modeling 8(2), 251–274 (2009).
20. Lerner, B.S., Christov, S., Osterweil, L.J., Bendraou, R., Kannengiesser, U., Wise, A.: Exception Handling Patterns for Process Modeling. IEEE Transactions on Software Engineering 36(2), 162-183 (2010).
21. Li, C., Reichert, M., Wombacher, A.: Mining business process variants: Challenges, scenarios, algorithms. Data Knowledge & Engineering 70(5), 409–434 (2011).
22. Marrella, A., Mecella, M., Russo, A.: Featuring automatic adaptivity through workflow enactment and planning. In Proc. CollaborateCom'11, 372-381 (2011).
23. Müller, D., Herbst, J., Hammori, M., Reichert, M.: IT support for release management processes in the automotive industry. In Proc. BPM'06, 368–377 (2006).
24. Reichert, M., Weber, B.: Enabling flexibility in process-aware information systems: challenges, methods, technologies. Springer (2012).
25. Reinhartz-Berger, I., Soffer, P., Sturm, A.: Organizational reference models: supporting an adequate design of local business processes. IBPIM 4(2), 134–149 (2009).
26. Rosemann, M., van der Aalst, W.M.P.: A configurable reference modeling language. Information Systems 32(1), 1–23 (2007).
27. Russell, N., ter Hofstede, A., Edmond, D., van der Aalst, W.: Workflow data patterns. Technical Report FIT-TR-2004-01, Queensland Univ. of Techn. (2004).

28. Russell, N., ter Hofstede, A., Edmond, D., van der Aalst, W.: Workflow resource patterns. Technical Report WP 127, Eindhoven Univ. of Technology (2004).
29. Russell, N., van der Aalst, W.M.P., Hofstede, A.: Workflow Exception Patterns. Advanced Information Systems Engineering 4001, 288-302 (2006).
30. Smirnov, S., Weidlich, M., Mendling, J., Weske, M.: Object-sensitive action patterns in process model repositories. In: Proc. BPM10 Workshops, 251-263 (2010).
31. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - Enhancing flexibility in process-aware information systems. Data Knowledge & Engineering 66, 438-466 (2008).
32. Weber, B. Sadiq, S. Reichert, M. Beyond rigidity - dynamic process lifecycle support. Computer Science 23, 47–65 (2009).
33. Weber, B., Reichert, M., Reijers, H.A., Mendling, J.: Refactoring large process model repositories. Computers in Industry 62(5), 467–486 (2011).