

# Lifecycle Management for Business Process Variants

Manfred Reichert<sup>1</sup>, Alena Hallerbach<sup>2</sup>, and Thomas Bauer<sup>3</sup>

<sup>1</sup>University of Ulm, Ulm, Germany

<sup>2</sup>Daimler TSS GmbH, Ulm, Germany

<sup>3</sup>Neu-Ulm University of Applied Science, Neu-Ulm, Germany

**Abstract** This chapter deals with advanced concepts for the configuration and management of business process variants. Typically, for a particular business process, different variants exist. Each of them constitutes an adjustment of a master process (e.g., a reference process) to specific requirements building the process context. Contemporary Business Process Management tools do not adequately support the modeling and management of such process variants. Either the variants have to be specified in separate process models or they are expressed in terms of conditional branches within the same process model. Both methods can result in high model redundancies, which make model adaptations a time-consuming and error-prone task. In this chapter, we discuss advanced concepts of our Provop approach, which provides a flexible and powerful solution for managing business process variants along their lifecycle. Such variant support will foster more systematic process configuration as well as process maintenance.

## 1 Introduction

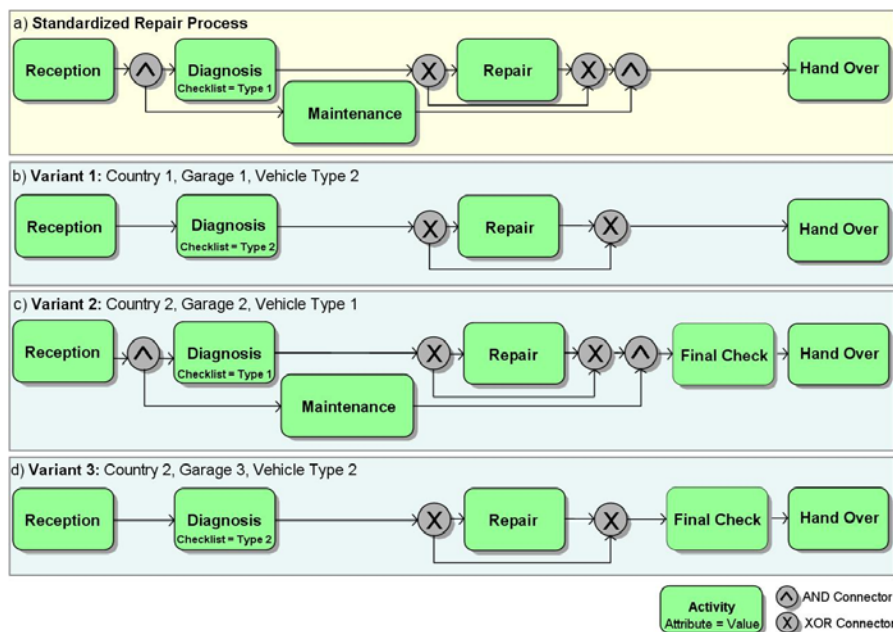
Process support is required in almost all business domains (Mutschler et al. [2008](#); Weber and Reichert [2012](#)). As examples, consider healthcare (Lenz and Reichert [2007](#)), automotive engineering (Müller et al. [2006](#)), and public administration (Becker et al. [2007](#)). Characteristic process examples from the automotive industry, for instance, include product change management (VDA [2005](#)), release management (Müller et al. [2006](#)), and product creation.

Usually, there exists a multitude of *variants* of a particular process model, whereby each of these variants is valid in a specific scenario or in the context of a particular business objective (Lohrmann and Reichert [2012](#)); i.e., the *configuration* of a particular *process variant* depends on concrete requirements building the *process context* (Hallerbach et al. [2008b](#)). Regarding release management, for example, we have identified more than twenty process variants depending on the considered product series, involved suppliers, or development phases. Similar observations can be made with respect to the product creation process in the automotive domain for which dozens of variants exist. Thereby, each variant is assigned to a particular product type (e.g., car, truck, or bus) with different organizational responsibilities and strategic goals, or varying in some other aspects.

In the following, we refer to the service process handling vehicle repair in a garage (cf. Fig. [1a](#)). Basically, this process works as follows: It starts with the reception of a vehicle. After a diagnosis is made, the vehicle is repaired (if necessary). During diagnosis and repair, the vehicle is maintained; e.g., oil and wiping water may be checked and refilled. The process completes when handing the repaired and maintained vehicle back to the customer. Depending on the process context, different variants of this process are required, whereas the

context is described by country- specific, garage-specific, and vehicle-type-specific variables. In our case studies, we have identified hundreds of such variants and we have learned that existing process modeling tools do not provide sophisticated support for modeling and maintaining such large number of process variants.

Figure 1b–d show three such variants of a vehicle repairs process. Variant 1, as depicted in Fig. 1b, assumes that the damaged vehicle requires a checklist of *Type 2* to perform the diagnosis. Therefore, activity *Diagnosis* is adapted by modifying its attribute *Checklist* to value “*Type 2*”. Additionally, the garage omits maintenance of the vehicle as this is considered as a special service not offered conjointly with the repair process. At the model level, this is realized by skipping activity *Maintenance*. As another example, consider Variant 2 as depicted in Fig. 1c. Due to country-specific legal regulations, a final security check is required, before handing over the vehicle back to the customer. Regarding this variant, the new activity *Final Check* has to be added when compared to the standardized process from Fig. 1a. Finally, Variant 3 will become relevant if a checklist of *Type 2* is required for diagnosis, the garage does not link maintenance to the repair process, and there are legal regulations requiring a final check (cf. Fig. 1d).



**Fig. 1** Variants of a standardized vehicle repair process (simplified view)

As can be seen from these simple examples, variants exist for many processes, and thus have to be adequately managed. This chapter presents selected concepts of the Provop (*PROcess Variants by OPTions*) approach for managing large collections of process variants. More precisely, Provop allows to configure relevant process variants out of one basic process model (Hallerbach et al. 2008a; Hallerbach et al. 2008c; Hallerbach 2010) and to manage them along their lifecycle. This chapter focuses on the technical issues, which become relevant in this context. Also very important, but out of the scope of this chapter, are governance issues (e.g.,

Who selects or enforces configurations? What does variant management mean for process ownership?).

The chapter is structured as follows: First, we present problems, which will arise if we do not treat variants as first class objects and only model them conventionally. Second, we describe key requirements with respect to process variant management. Then, we introduce our Provop approach and selected concepts for process variant management. Following this we compare Provop with other process configuration approaches and then discuss other relevant aspects. The chapter concludes with a summary and an outlook.

## 2 Dealing with Process Variants in Existing BPM Tools

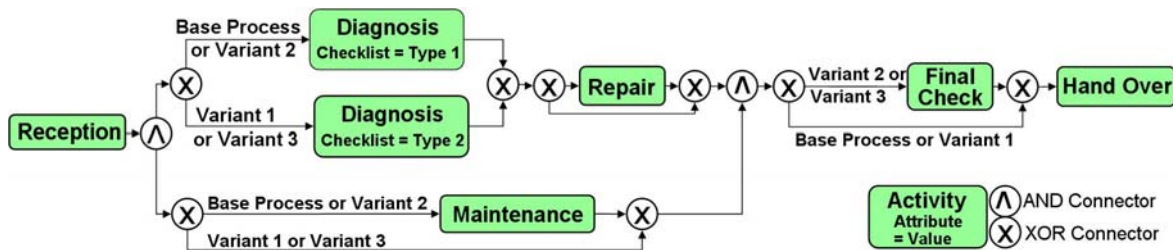
Solutions for managing variants in existing BPM tools can be divided into two approaches: the *multi-model* and the *single-model* approach.

*Multi-model approach.* In existing BPM tools, process variants often have to be defined and kept in separate process models as shown in Fig. [1](#). Typically, this results in highly redundant model data as the variant models are identical or similar for most parts. Furthermore, the variants cannot be strongly related to each other; i.e., their models are only loosely coupled (e.g., based on naming conventions). Furthermore, there is no support for (semi-) automatically combining existing variants to a new one; e.g., Variant 3 of our repair process (cf. Fig. [1d](#)) combines the adjustments made by Variant 1 and Variant 2, and applies them to the standardized process. However, it cannot be created out of the existing models of these two variants as there is no indication which model parts are variant-specific and which are common for all models.

This multi-model approach will therefore be only feasible if few variants exist or the variants differ to a large degree from each other. Considering the large number of variants occurring in practice, however, the aforementioned drawbacks increase modeling and maintenance efforts significantly. Particularly, the efforts for maintaining and changing process variants become high since more fundamental process changes have to be accomplished for each variant separately (e.g., due to changed or new legal regulations). This is both time-consuming and error-prone. As another consequence, over time models representing the variants more and more differ from each other; e.g., when optimizations are only applied to single variants without considering their relations to other ones (Weber and Reichert [2008b](#)). This, in turn, makes it a hard job for process designers to analyze, compare, and unify business processes and to implement the multiple variants within a common IT system. As conclusion, generally, modeling all process variants in separate models does not constitute an adequate solution for variant management.

*Single-model approach.* Another approach, frequently applied in practice, is to capture multiple variants in one single model using conditional branchings (i.e., XOR-/OR-Splits). Consider Fig. [2](#) as an example, which shows the repair process together with different variants (cf. Fig. [1a-d](#)). Each execution path in the model represents a particular variant. Therefore, branching conditions indicate which path belongs to which variant.

Generally, specifying all variants in one process model can result in a large model, which is difficult to comprehend and expensive to maintain. Note that in realistic scenarios there might be dozens to up to hundreds of variants of a particular process type (Weber et al. 2011; Li et al. 2011). As another drawback, variants are then mixed with “normal” process logic; i.e., branchings relevant for all process variants cannot be distinguished from the ones representing a variant selection. For example, our repair process includes a decision to only perform activity *Repair* if necessary. Therefore, on the model side, there is a conditional branching to either perform or skip the repair step. This branching is relevant for all discussed variants of the repair process; i.e., it is no variant-specific branching. However, the user cannot distinguish between normal and variant-specific branchings, unless there are special conventions to represent variant specific conditions or other model extensions used to mark a branching as normal or variant-specific. In summary, variants are neither transparent nor explicitly defined in this approach. As a consequence, the supporting IT system is unaware of the different process variants and only treats them as “normal” branchings within a single process model.



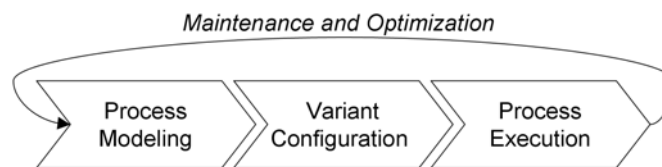
**Fig. 2** Process variants realized by conditional branches

*Discussion.* Neither the use of separate models for capturing process variants nor their definition in one model based on conditional branchings constitutes adequate methods. Both approaches do not treat variants as first class objects; i.e., the variant-specific parts of a process are maintained and hidden either in separate models (multi-model approach) or in control flow logic (single-model approach). Another drawback of these approaches is the lack of context-awareness. Contextual knowledge might only be integrated and used in terms of process meta-data or branching conditions. As the process context mainly influences variant configuration, however, this fundamental aspect has to be considered more explicitly.

Note that these limitations also apply to popular business process modeling tools like ARIS Business Architect or WBI Modeler. ARIS Business Architect (IDS Scheer 2008), for example, allows to create a new process variant by copying the respective model directory and its objects, resulting in high redundancy of model data. Though the derived variant objects refer to the original objects (denoted as *master objects* in ARIS) afterwards, changes of the latter are not propagated to the variants. In principle, this corresponds to the multi-model approach as described above. However, through the explicit documentation of relation structures (between original and variant objects) some improvement is achieved.

### 3 Requirements

We conducted several case studies not only in the automotive industry (Müller et al. [2006](#), VDA [2005](#)) but also in other domains like healthcare (Lenz and Reichert [2007](#)), to elaborate key requirements for the configuration, adaptation, and management of process variants. This strong linkage to practice was needed in order to realize a complete and solid approach for process variant management. The requirements we identified are related to different aspects including the modeling of process variants, their linkage to process context and context-driven configuration, their execution in workflow management systems (WfMS), and their continuous optimization to deal with evolving needs; i.e., we have to deal with requirements related to the whole process life cycle (Hallerbach et al. [2008c, e](#), Weber et al. [2006](#), Weber et al. [2009](#)). The standard process life cycle is depicted in Fig. 3. It consists of three phases, namely the design and modeling of the process, the creation of a particular process variant, and the deployment of this variant in a runtime environment. The process life cycle can be described as a (feedback) loop of these phases during which a process is continuously optimized and adapted (Weber et al. [2006](#), Weber et al. [2009](#)). The major requirements to be met are described in the following.



**Fig. 3** Process life cycle

*Modeling.* Efforts for modeling process variants should be kept as minimal as possible. Reuse of the variant models (or parts of them) has to be supported. In particular, it should be possible to create new variants by taking over properties from existing ones, but without creating redundant or inconsistent model data. Thus, the hierarchical structure of such “variants of variants” has to be adequately represented and should be easy to adapt.

*Variant configuration.* The configuration of a process variant (i.e., its derivation from a given master or base process) should be done automatically if possible. Therefore, the specific circumstances (i.e., the *process context*) under which this configuration takes place have to be considered. In particular, an elaborated procedure for context-aware, automated variant configuration is required. At the same time, consistency and correctness of the configured process variants have to be ensured throughout the entire process life cycle.

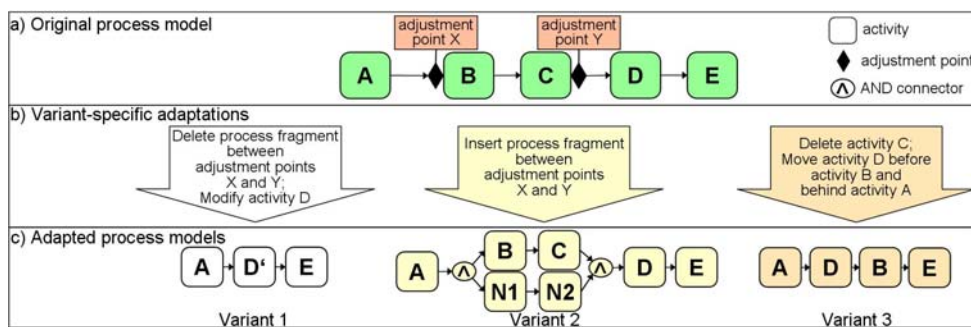
*Execution.* To execute a process variant, its model has to be interpreted by a workflow engine. In this context, it is important to keep information about the configured process variant and its relation to a master or base process (and to other variants) in the runtime system. To deal with dynamic changes of the process context, the runtime system should additionally allow to dynamically switch process execution from one variant to another if required (i.e., to reconfigure the corresponding process variant on-the-fly). Finally, if context information is only available during runtime, the specific variant will have to be determined (i.e., configured) at runtime as well.

*Maintenance and optimization.* To reduce maintenance efforts and cost of change, fundamental changes affecting multiple process variants should be conducted only once. As a consequence, all process variants concerned by the respective change should be adapted automatically and correctly.

There exist other requirements addressed by Provop, but not treated here. Examples include the consistency of configured variants, adequate visualization of the variants in all life cycle phases, and provision of intuitive user interfaces for variant configuration. In this chapter, we focus on the main requirements discussed above, covering the complete *process life cycle*.

## 4 The Provop Approach

In practice, process variants are often created by cloning and adjusting an existing process model of a particular type according to the given context. For example, regarding the three process variant models from Fig. 1b–d, one can notice that they can be derived from the standardized process as depicted in Fig. 1a by adding, removing, or modifying activities. Generally, every process model can be derived out of another one by adjusting it accordingly, i.e., by applying a set of change operations and change patterns, respectively, to it (Weber et al., 2008). Starting from this observation, Provop provides an *operational approach* for managing process variants based on a single process model (see Fig. 4a). In particular, process variants can be configured by applying a set of high-level change operations to a given process model. We denote the latter as *base process*.



**Fig. 4** Variant configuration by process model adaptation

In the following, we provide an overview of our Provop approach and describe it along the different phases of the process lifecycle.

### 4.1 Modeling

In the modeling phase, first of all, a base process, from which the different process variants can be derived through configuration, has to be defined. Following this, high-level change operations, which can be applied to this base process, are specified (Hallerbach et al. 2008a; Hallerbach et al., 2008d).



*Defining the base process.* Basic to the configuration of process variants is a base process, which serves as reference for the high-level change operations. When considering typical use cases as well as the overall process landscape in an enterprise, different policies for defining such base process are relevant. Basically, Provop supports the following ones:

- *Policy 1 (standard process):* Here, the base process represents a domain-specific standard or reference process. In the automotive domain, for example, such reference processes exist for Engineering Change Management. Usually, a standard process has to be adjusted to meet specific requirements; i.e., it must be possible to derive variants from it. Provop assists designers in correctly defining the necessary adjustments when configuring a process variant out of the reference process.
- *Policy 2 (most frequently used process):* If one process variant is used more frequently than others, it can be chosen as base process. This reduces configuration efforts in terms of the number of processes for which adjustments become necessary. Provop maintains statistics on the use of process variants to enable Policy 2. Generally, Policy 2 does not ensure that the average number of change operations needed to configure the variants out of the base process becomes minimal.
- *Policy 3 (minimal average distance):* When applying change mining to a collection of variants, we can derive a base model such that average distance between this model and its variants (i.e., the number of high-level operations needed to transform the base process into the process variant) becomes minimal (Li et al. [2008a](#)). Thus, configuration efforts can be reduced accordingly. For mining process variants, we utilize algorithms we developed in the MinAdept project (Li et al. [2008b](#)).
- *Policy 4 (superset of all process variants):* The base process is created by merging all variants into one process model using conditional branchings; i.e., the base process realizes a “superset” of all relevant variants. Consequently, every element that is part of at least one variant belongs to the base process as well. When deriving process variants, therefore, only DELETE operations have to be applied.
- *Policy 5 (intersection of all process variants):* The base process comprises only those elements that are part of all variants; i.e., the base process realizes a kind of “intersection” of relevant variants. Therefore, the base process covers the identical elements of the process variants. When deriving process variants, no DELETE operations have to be performed, but elements may have to be moved, modified, or inserted.

Policies 1–5 differ in one fundamental aspect: When using Policy 1 or 2, the respective base process serves a specific use case; i.e., it represents one process variant valid in a specific context. Policies 3–5, in turn, have been especially designed for configuring variants and thus do not necessarily represent a semantically valid process model. Which policy to choose mainly depends on the modeling scenario and the present process landscape; e.g., if a standard process already exists, Policy 1 will be recommended.

*Change operations.* A base process can be adjusted in different ways to configure a specific variant. Provop supports the following adaptation patterns: INSERT, DELETE, and MOVE process fragments, and MODIFY process element attributes. And fragments constitute connected process sub-graphs (including single activity nodes and edges respectively), which

not necessarily have a single entry and single exit. To refer to fragments and elements of the base process within such change operations, we use *adjustment points*, which correspond to the entry or exit of an activity or connector node (e.g., split and join nodes) of the base process.<sup>1</sup> Adjustment points are labeled with unique names. As example consider “adjustment point X” in Fig. 4, which corresponds to the entry of activity B.

<sup>1</sup>If only single elements are affected by a particular change operation, their process element IDs may be used alternatively.

**Table 1** Change operations (i.e., change patterns) supported by Provop





<b>1. INSERT-Operation</b>	
Symbol	
Purpose	Addition of <i>process fragments</i> (A process fragment consists of at least one process element, e.g., activity nodes or control edges).
Parameters	<ul style="list-style-type: none"> <li>• Process fragment to be added with entries and exits marked by adjustment points</li> <li>• Target position of the process fragment within the base process, marked by adjustment points for entries and exits</li> <li>• Mapping between entries and exits of the added fragment to the target position within the base process (i.e., mapping of the respective adjustment points)</li> </ul>
<b>2. DELETE-Operation</b>	
Symbol	
Purpose	Removal of process elements
Parameters	<ul style="list-style-type: none"> <li>• Process fragment to be deleted with entries and exits marked by adjustment points</li> <li>• Alternatively: deleting single elements by referring to their ID</li> </ul>
<b>3. MOVE-Operation</b>	
Symbol	
Purpose	Change execution order of activities
Parameters	<ul style="list-style-type: none"> <li>• Process fragment to be moved with entries and exits marked by adjustment points</li> <li>• Target position of the process fragment marked by adjustment points</li> </ul>
<b>4. MODIFY-Operation</b>	
Symbol	
Purpose	Change attributes of process elements
Parameters	<ul style="list-style-type: none"> <li>• Element ID</li> <li>• Attribute name</li> <li>• Value to be assigned</li> </ul>

Table 1 gives an overview of the change operations currently supported by Provop. Each entry describes the purpose of the respective operation, its parameters, and the symbol representing it. The formal semantics of respective change patterns is described in Rinderle-Ma et al. (2008). Note that Provop covers only a subset of the change patterns presented in Weber et al. (2007, 2008), which have turned out to be the most relevant ones needed for variant configuration in practice; i.e., we were able to capture the different scenarios



discussed in the introduction section based on these change patterns. It is also worth mentioning that Provop provides an extensible approach, to which other change patterns may be added later.

*Grouping change operations into options.* As the number of change operations required to configure all relevant variants might become large, Provop allows structuring multiple change operations by grouping them into the so-called *options*. This is useful, for example, if the same change operations are always applied in conjunction with each other when configuring certain variants. Think of, for example, the handling of a medical examination in the radiology unit of a hospital. While for ambulant patients no transport between ward and radiology room is required, basic patients first have to be transferred from the ward to the radiology unit and later back to the ward. To capture the latter variant, we need to add two activities at different positions of the respective base process. This can be achieved by defining the two insert operations and grouping them in one option.

*Constraint-based use of options.* Our case studies have revealed that options are often correlated in a structural or semantic manner. To capture this, Provop considers three types of relations between options, which can be explicitly defined by the user: dependency, mutual exclusion, and hierarchy.

- *Dependency:* When applying different options conjointly to the base process (e.g., due to semantic dependencies), the user can explicitly define a dependency relation between them. Dependency relations are directed; i.e., if relation “Option 1 depends on Option 2” holds, the inverse relation (i.e., “Option 2 depends on Option 1”) is not true.
- *Mutual exclusion,* in turn, is helpful to describe which options must not be used in conjunction with each other when configuring variants.
- *Hierarchy:* The definition of option hierarchies allows for the inheritance of change operations. If an option is selected to configure a particular variant and has an ancestor in the option hierarchy, the change operations defined by the ancestor options will be applied as well. This reduces the amount of change operations defined in options and also structures the options landscape; i.e., maintenance is improved.

When defining relations between options, generally, the designer does not only use one relation type but may also apply them in combination with each other as well. Provop allows for the combined use of multiple relations and ensures consistency of a set of relations applied in a given context. For example, contradictory relations (e.g., a mutual exclusion between an option and its parental option) must not be applied. Due to lack of space, we omit further details on how such contradicting constraints can be identified.

The ability to define explicit relations between different options eases their use significantly. Additionally, Provop excludes semantic errors when configuring a process variant, as we will discuss in the sequel.

*Context model.* Provop allows for context-aware process configurations; i.e., it allows for the configuration of a process variant by applying only those options relevant in the given *process*

*context* (Hallerbach et al. 2008b). This, in turn, necessitates a model capturing the process context. In Provop, such context model comprises a set of *context variables*. Each context variable represents one specific dimension of the process context, and is defined by a name and value range.

**Table 2** Context model of a vehicle repair process

Variable name	Range of values	Behavior
Vehicle type	Type 1, Type 2, Type 3, Type 4	Static
Maintenance	Yes, No	Static
Security level	low, medium, high	Static
Workload	low, medium, high	Dynamic

Table 2 shows an example of the context model defined for the vehicle repair process from Fig. 1. The depicted context variables do not only differ in their names and range of values but also in another important aspect. While some context variables are defined as *static*, others are classified as *dynamic*. For example, the value of the context variable *Workload* is raised or lowered from time to time according to the current workload of the garage (e.g., switching from “medium” to “high” if many new repair orders emerge at the same time). Thus, this variable is of dynamic nature, as its value may change during process execution. The context variable *Vehicle Type*, in turn, is static as the vehicle type is set once and does not change during the repair process.

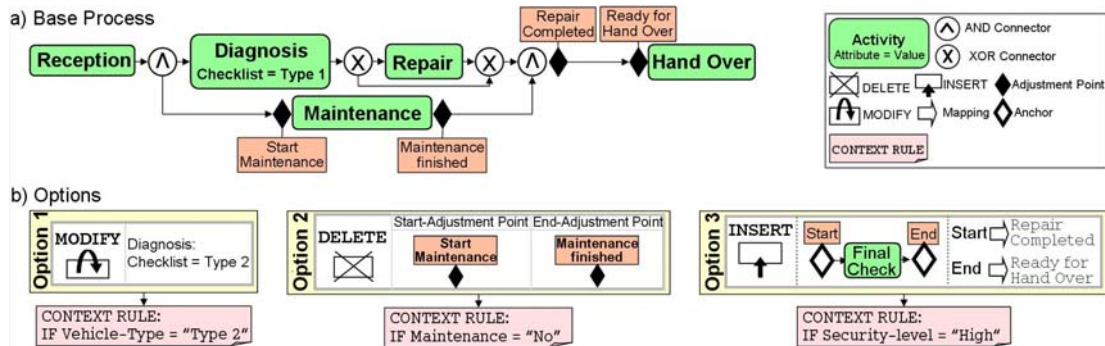
#### 4.2 Variant Configuration

In the configuration phase, the base process, the options defined for it, and the context model are used to configure the models of the different variants. More precisely, a particular variant is configured by applying a sequence of options and their corresponding change operations to the base process. We describe the steps needed for configuring a variant in Provop:

*Step 1: Select relevant options.* To configure a particular variant, usually, only a subset of the defined options is relevant. Therefore, as a first step in the configuration phase, the set of relevant options has to be identified. One possible approach is to ask users to manually select the relevant options. However, this would require sufficient knowledge about available options and their effects (i.e., change operations). In particular, if users have to choose among a large number of options, this approach will get error-prone (e.g., relevant options might be omitted or wrong ones chosen).

A more sophisticated approach is to select relevant options based on contextual knowledge. Rather than mapping already configured process variants to a context description, *context-aware process configuration* allows for the combination of the concepts provided by options and context models. In Provop, this linkage is realized by the use of *context rules*. Such rules, can be assigned to the options and make use of the defined *context model*. Regarding a given context, all options whose context rules evaluate to true, are applied to the base process and therefore determine the respective variant. As special case, the base process itself may serve

as variant (i.e., no option is applied). In Step 3, we describe the order in which the selected options are applied to the base process.



**Fig. 5** Example of context dependent options

Figure 5 illustrates how the three variants of the repair process (cf. Figure 1) are captured in Provop: The standardized process of Figure 1a is defined as the base process out of which the variants are configured. This base process contains several adjustment points (e.g., “Start Maintenance” at the entry of activity *Maintenance*). As mentioned, adjustment points may be referred to by options and their change operations. Furthermore, Figure 5b depicts three options: Option 1 performs a modification of activity *Diagnosis*. It will be applied if the type of the vehicle is of value *Type 2*. Option 2, in turn, will delete the maintenance activity if no maintenance of the vehicle is requested. Finally, Option 3 inserts a final security check activity in case of high security levels. The variants of Figure 1b–d can now be configured by applying a subset of these options to the base process. For example, if the context of a process variant is defined by the expression “*Vehicle-Type=Type 2 AND Maintenance=No AND Security-Level=Low*,” Options 1 and 2 will be applied resulting in Variant 1 (cf. Figure 1b).

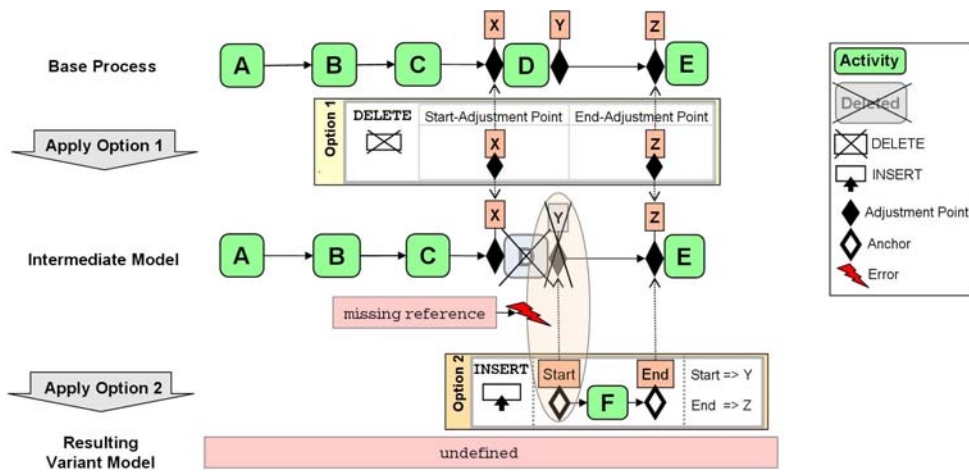
*Step 2: Evaluate relations between selected options.* As aforementioned, options may be related. Generally, for a sequence of options to be applied to the base process, compliance with explicitly defined constraints has to be ensured. For example, if a selected option depends on another one, not yet contained in the set of selected options, this set will have to be adjusted accordingly. Generally, this can be achieved either by adding missing options to the selection list or by removing the ones that cause the constraint violation. Another constraint violation will occur if the selection set comprises mutually excluding options. In this case, one of the conflicting options has to be removed by the user in order to restore consistency. In summary, option constraints are considered to ensure semantic correctness and consistency of the selected set of options at configuration time.

*Step 3: Determine the order in which options shall be applied.* Generally, selected options have to be applied in sequence; i.e., their order has to be specified when configuring a variant. A naïve approach would be to sort these options in the order they were created; e.g., by making use of their creation time stamps. Obviously, this approach will only make sense if the options and their change operations are commutative. Otherwise, unintended and inconsistent variant models can result, particularly when applying options in the wrong order. Figure 6 shows an example: After applying Option 1 to the base process, an intermediate model is

derived with activity D and adjustment point Y being deleted.<sup>2</sup> This model is now used as “reference model” for applying Option 2. In the present case, Option 2 cannot be applied as the adjustment point Y it refers to was deleted when applying Option 1. In order to avoid such inconsistencies, Provop allows defining the order in which selected options shall be applied. Furthermore, wrong option sequences, resulting in erroneous variant models afterwards, are excluded based on well-defined correctness criteria (see Step 5). Finally, by evaluating predefined sequencing constraints, a correct application order can be determined.

*Step 4: Applying options and their change operations.* After selecting the options and determining their order, their change operations are applied to the base process in order to configure the model of the respective variant. Generally, change operations have specific pre- and postconditions, which allow us to guarantee their correct application.<sup>3</sup> As one precondition, for example, process elements to which an operation refers have to be present in the respective model. Thus, the problem depicted in Fig. 6 would be recognized before applying the INSERT-operation of Option 2; i.e., Provop would disallow to apply the two options in the depicted order.

*Step 5: Checking consistency.* The variant models resulting from the sketched configuration procedure are supposed to be executed in the process enactment phase. Therefore, consistency and correctness of the models have to be guaranteed. In addition to the already described constraint-based selection approach (cf. Step 2), Provop validates the resulting models by checking the consistency and correctness of data and control flow. Unlike other variant configuration approaches (van der Aalst et al. 2008), Provop does not necessarily require a consistent and correct base process as starting point when configuring variants. This follows from the above described policies for defining the base process. Assume, for example, a base process being defined as intersection of its variants. If two variants have different activities to write a data object, read by a common activity, the base process would only contain the reading activity and thus be inconsistent in terms of data flow. Of course, Provop excludes such flaws for the configured variant models.



**Fig. 6** Syntactical error after applying options in wrong order

<sup>2</sup>Note that this example indicates that we need more advanced change support that considers the special semantics of adjustment points. Generally, the user should be able to define whether adjustment points may be deleted when applying certain change operations or shall be kept in the intermediate model. In the latter case, the deleted activities and nodes respectively are replaced by silent activities without associated actions. Generally, silent activities and adjustment points are removed after application of all selected options.

<sup>3</sup>For a formal semantics of respective change patterns, we refer to (Rinderle-Ma et al. [2008](#)).

### **4.3 Deployment and Execution**

After the configuration phase, the resulting variant model needs to be translated into an executable workflow model. Common tasks emerging in this context are to assign graphical user interfaces, to subdivide workflow activities into human and automated tasks, or to choose the right level of granularity for the workflow model. In Provop, we are focusing on problems arising in the context of variant management.

One major aspect concerns the *context-aware configuration* of the different variants. To also capture *context changes* during process instance execution, Provop supports *dynamic context variables*; i.e., variables whose values may change during process execution. When using dynamic context variables for defining a context rule of an option, the decision whether to apply the corresponding change operations or not has to be made at runtime. As a consequence, the respective process variant either cannot be completely configured when creating the process instance or it has to be reconfigured during runtime. To allow for the dynamic reconfiguration of a process instance of a variant model, Provop supports *variant branches*. Basic idea is to encapsulate the adjustments of single options within these variant branches. The split condition at a variant branching corresponds to the context rule of the option. Whenever process execution reaches a variant branch, the current context is evaluated. If the split condition evaluates to true, the variant branch will be executed, i.e., the change operations will be applied to the base process. Otherwise, the variant branch is skipped and therefore all adjustments of the option are ignored. Provop ensures the constraints regarding the use of options in the context of such dynamic reconfigurations as well. However, the handling of respective correctness issues is outside the scope of this chapter.

Figure [7](#) shows an example of a *variant branch definition* in conjunction with the INSERT operation.<sup>4</sup> If the workload of a garage is *high*, subcontractors will be commissioned to provide maintenance activities. Thus, Option 4 will be applied adding corresponding activities *Commissioning Sub-contractor* and *Support Maintenance* to the base process. As the context variable *Workload* is dynamic (cf. Table [2](#)), these activities are encapsulated in a variant branch (indicated by the encircled “less than” and “greater than” symbols). Furthermore, context rule of Option 4 is used as split condition. Whenever a variant branch is reached during process execution, corresponding context rules are evaluated. If they evaluate to true (cf. Figure [8a](#)), the variant branch will be executed; otherwise, it will be skipped (cf. Figure [8b](#)).

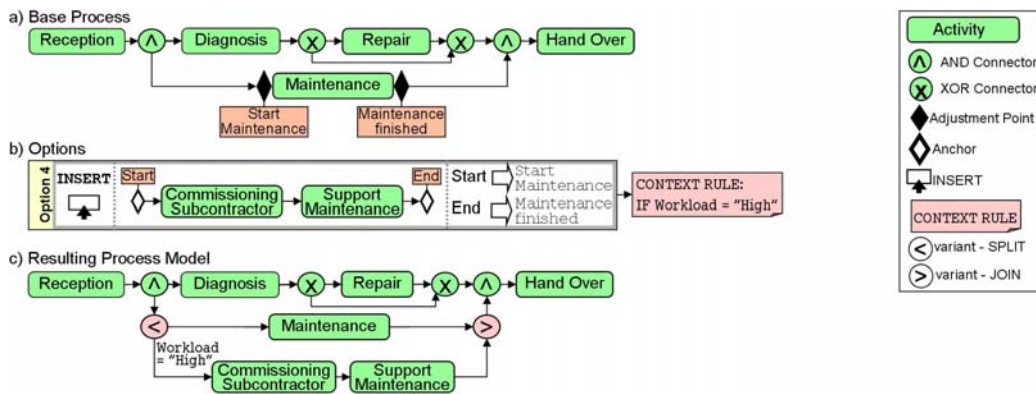


Fig. 7 Dynamic configuration of process variants

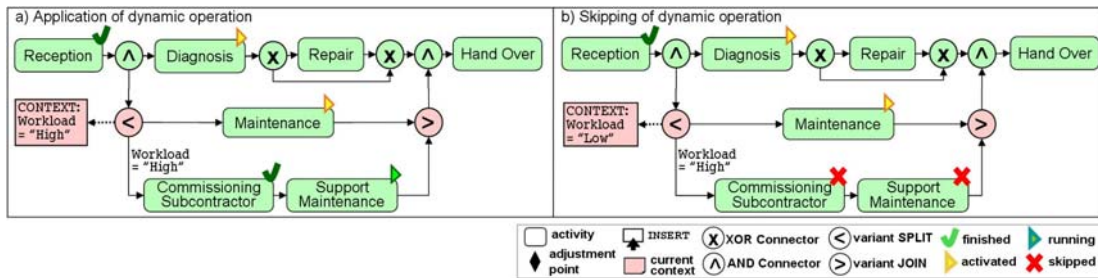


Fig. 8 Determine variant at runtime

<sup>4</sup>Note that every change operation supported by Provop requires specific considerations here.

#### 4.4 Maintenance and Optimization

When evolving base processes in Provop (e.g., due to organizational optimization efforts), all related process variants (i.e., their models) are reconfigured automatically. Thus, maintenance efforts can be significantly reduced. However, evolving and optimizing the base process may affect existing options, for example, when referred adjustment points are moved to a new position or are even deleted. Such problems are detected in Provop; e.g., by checking whether the definitions of existing options are affected by the adaptations of the base process model. Furthermore, solving those conflicts is largely automated.

#### 4.5 Proof-of-Concept Implementation

We implemented the described concepts in a powerful proof-of-concept prototype. When developing this prototype, we had to decide whether to realize a process configuration tool from scratch or to enhance an existing process modeling tool with respective configuration features. On one hand, the first option offers the flexibility to implement the presented concepts in a native and consistent way. On the other, it is accompanied by high development costs; e.g., basic functionality of the process modeling tool would have to be re-implemented



from scratch. Therefore, we decided to base our implementation on an existing process modeling tool and to enhance this tool with the process configuration facilities described. To be more precise, we decided to use the ARIS Business Architect (IDS Scheer 2008), which belongs to the ARIS Design Platform and constitutes a widespread tool supporting a variety of modeling notations (e.g., EPCs and BPMN) as well as use cases (e.g., modeling, analyzing and optimizing business processes).

The general limitations of commercial BPM tools with respect to the handling of process variability, which have been described in Section 2, apply to ARIS Business Architect as well. Basically, this tool allows creating new process variants by cloning (i.e., copying) existing process models (and their objects) and modifying them afterwards. However, this might result in model redundancies. Although the derived variant objects still refer to the original objects, which are called master objects in ARIS Business Architect, changes of the latter are not propagated to the variants.

Another decision we had to make when implementing the Provop prototype concerns the choice of the language for modeling base processes and change options as well as for representing the variant models resulting from the configurations applied. We first started with Event-Process-Chains (EPCs) as this notation is widely used in practice. However, EPCs do not offer grouping functions which become relevant in our context for grouping parameters of a particular change operation as well as for grouping multiple change operations into one change option. To enable grouping in ARIS Business Architect, in principle, model folders may be used as workaround. However, we decided to use the BPMN notation instead since it provides different grouping mechanism as required in our approach (cf. Figure 9).

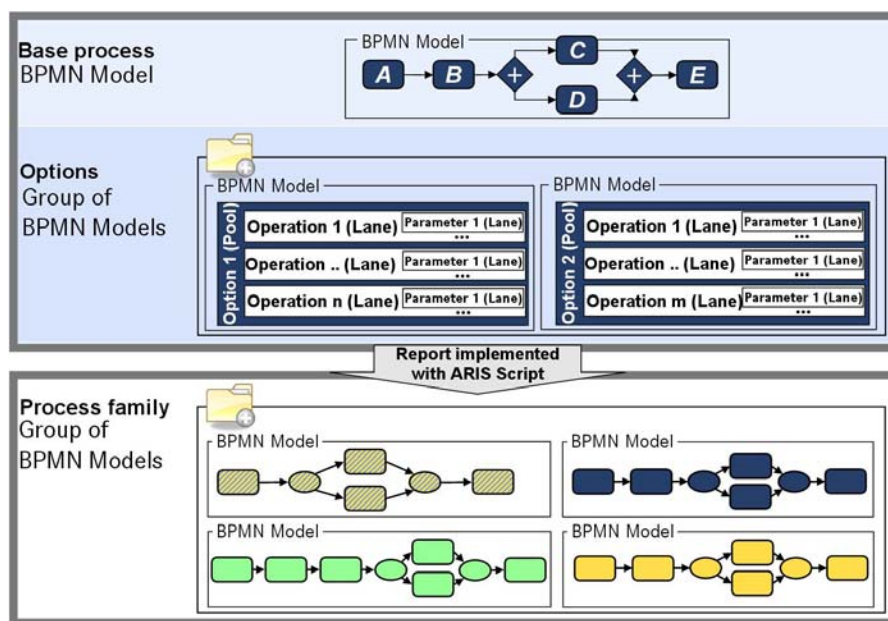
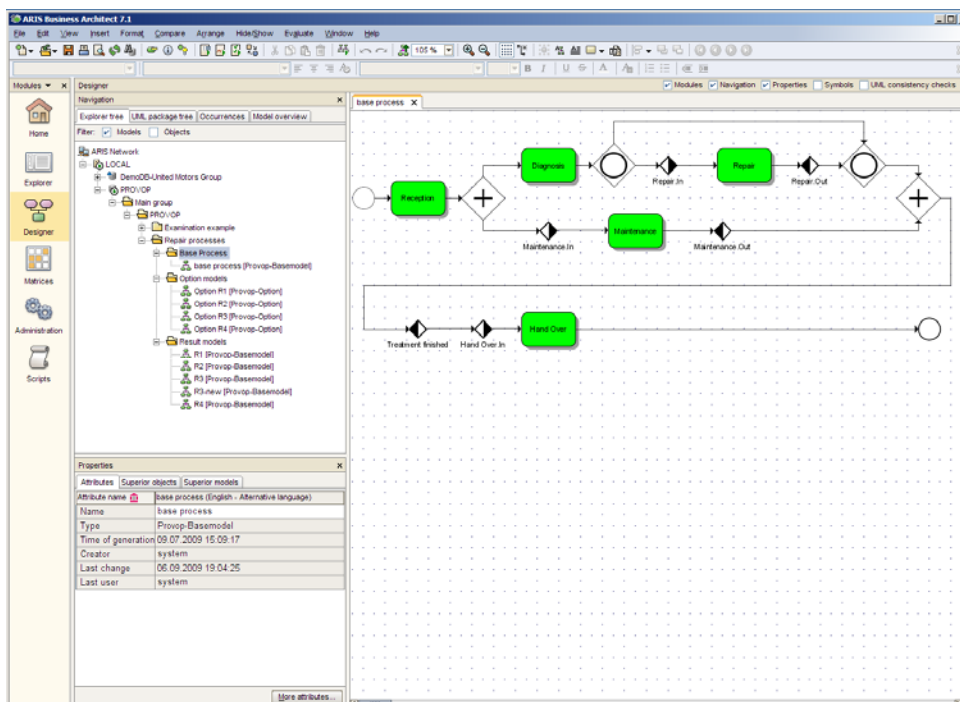


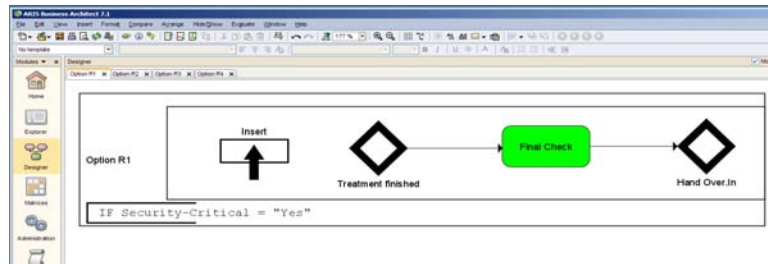
Fig. 9 Architecture of the Provop prototype

Figure 9 shows the overall architecture of our proof-of-concept prototype. Each change option is realized as a single BPMN model. Within these models the change operations corresponding to the respective option are encapsulated in pools, which correspond to graphical as well as logical containers. The relevant parameters of a particular change operation (e.g., the adjustment points marking a process fragment to be deleted) are specified by using lanes, which constitute sub-containers of a particular pool (or another lane respectively). A particular ARIS report, which we implemented using ARIS Script (i.e., JavaScript extended by specific functions), realizes the transformation of a base process model to a specific process variant. More precisely, for a base process represented as BPMN model, variant configuration can be started by selecting a set of options. Following this, the change operations of selected options are applied to the base process resulting in a new BPMN model, which then represents the configured process variant.

Figures 10 and 11 depict two screens of this proof-of-concept prototype. Thereby, new objects and symbols (e.g., operation types and adjustment points) were designed using the ARIS Symbol Editor. Figure 10 shows a base process, together with its adjustment points, as it can be modeled with Provop. In turn, Figure 11 depicts an option comprising exactly one change operation. More precisely, the depicted option allows inserting activity *Final Check* between the specified adjustment points of the base process if the corresponding context rule (i.e., Security-Critical = “Yes”) is satisfied.



**Fig. 10** Modeling a base process in Provop



**Fig. 11** Example of a change option in Provop

## 5 Comparing Provop with other Process Configuration Approaches

Generally, there is a great interest in capturing common process knowledge only once and re-using it in terms of *configurable process models* that represent a collection of related process variants (i.e., a process family). In the following, we make use of the process carried out when checking-in at an airport (Ayora et al. [2012a](#)) in order to illustrate commonalities and differences of existing approaches for capturing such variability (including Provop). We choose this process since it shows a high degree of variability; e.g., occurring 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).

In a systematic literature review, we identified 25 proposals (including Provop) dealing with the modeling and management of process variants (Ayora et al. [2013a](#); Ayora et al. [2013b](#)). Common to them is the extension of existing process modeling languages with variability-specific constructs that enable the creation of configurable process models. By treating variability as a first class citizen, these extensions help avoiding redundancies, fostering reusability, and reducing process modeling efforts (Torres et al [2012](#)). In particular, we identified the following language constructs commonly used by existing proposals capturing variability (including Provop) in addition to standard process modeling elements:

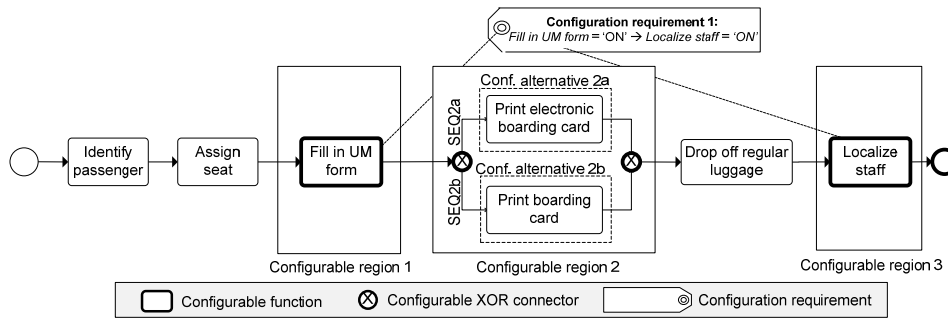
- *Language Construct LC1 (configurable region)*: A *configurable region* is a region in a configurable process model for which different *configuration choices* may exist depending on the application context, e.g., an airline may offer different ways of obtaining the boarding cards depending on the check-in type: printing a boarding card at the airline desk, download an electronic boarding card, or obtaining it via a 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., passengers with overweight luggage pay a fee.
- *Language Construct LC4 (configuration constraint)*. A *configuration constraint* is defined as a (structural) 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.

In the following, we describe a well-known approach for realizing configurable process models in more detail and compare it with Provop. More precisely, we consider *process models with configurable nodes*, which take a fundamentally different approach to realize and describe the variability-specific parts of a process model when compared to the Provop approach.

*Process models with configurable nodes*: 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 (Gottschalk et al. [2007](#); Gottschalk [2009](#); Gottschalk et al. [2009](#)). Basically, these proposals extend an existing process modeling language (e.g., EPC and YAWL) by adding configurable elements for explicitly representing variability in process families. In particular, E-EPCs provide support for the specification and customization of reference process models.

Figure [12](#) 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, and may take two different forms. First, the process 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 Figure [12](#)). Alternatively, the process fragment may consist of a *configurable function* (i.e., activity), e.g., configurable regions 1 and 3 in Figure [12](#), 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 process fragment that may be included as a branch between two *configurable connectors* (e.g., *Print electronic boarding card* in configurable region 2 in Figure [12](#)). Context conditions (LC3) are represented in C-EPC separately in a *questionnaire model* (Rosa et al. [2007](#)). Finally, a configuration constraint (LC4) may be specified in terms of 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 Figure [12](#) states that the inclusion of the function *Fill in UM form* implies the inclusion of the function *Localize staff*.

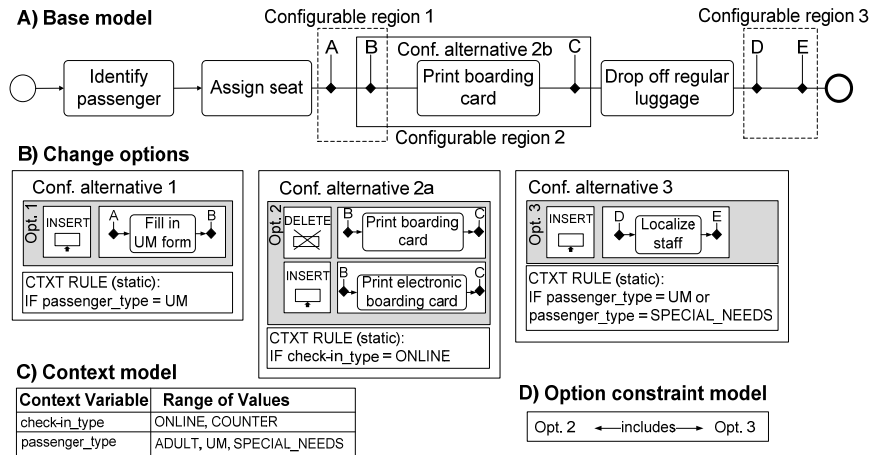


**Fig. 12** C-EPC configurable process model for the check-in process

A similar approach is presented in Gottschalk et al. (2007), which transfers the concepts for configuring a reference process model (i.e., to enable, hide, or block a configurable workflow element) to workflow models.

*Provop*: The Provop approach presented in this chapter constitutes a fundamentally different way of handling process families, which is based on the observation that process variants are often derived by adapting a pre-specified *base process model* to the given context through a sequence of structural adaptations.

Figure 13 illustrates how the process family dealing with the check-in process can be represented using Provop. Figure 13A shows the base process model from which the process variants may be derived. As discussed, in Provop, a configurable region (LC1) is specified by a 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 Figure 13). 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 Figure 13B). 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 in the *context model* (cf. Figure 13C). Finally, configuration constraints (LC4) are specified as constraints (e.g., mutual exclusion) 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. Figure 13C).



**Fig. 13** Provpn model for the check-in process

Basically, both approaches described enable reference process modeling (Rosemann and van der Aalst 2007; vom Brocke 2007). Usually, a reference process has recommending character, covers a family of process models, and may be customized in different ways to meet the needs of the specific application environment.

When comparing the two approaches, one can notice that they both realize the aforementioned language constructs for capturing process variability (i.e., LC1 – LC4), although this is accomplished in a completely different way. On one hand, proposals like C-EPC and C-YAWL represent a configurable process model (and process family respectively) in one artifact, capturing both the commonalities and particularities of the different process variants. Hence the configurable process model reflects all possible behavior. On the other, proposals such as the presented Provpn approach or the one suggested by Kumar and Wen (2012) propose a gradual construction of the process family by modifying the structure of a specific process variant (i.e., base process model) at specific points (i.e., variation points) through change operations. Both approaches have their pros and cons, and additional research is needed to learn which of them fits better in a given application environment.

In principle, the C-EPC approach constitutes an optimization of the *single model approach* introduced in Section 2. As opposed to Provpn, the suggested methods does not allow moving or adding model elements, or adapting element attributes when configuring a process variant out of a reference process model. Basically, the provided configuration support corresponds to the one of Policy 4 for which the chosen base process (i.e., reference process) constitutes the superset of all process variants. Obviously, in this specific scenario, only delete or optional delete operations (i.e., dynamic delete operations in Provpn) become necessary in order to configure a particular process variant out of a reference process model. However, Policy 4 is only one out of several configuration policies supported by Provpn; i.e., a base process can be defined in a more flexible way.

A qualitative comparison of these and other approaches supporting business process variability is provided by Torres et al. (2012). In particular, this work considers understandability issues that emerge when configuring concrete process variants with either



C-EPC or Provop. Furthermore, Ayora et al. ([2013a](#); [2013b](#)) presents a set of empirically evidenced change patterns for defining and changing configurable process models. Thereby, the proposed change patterns are based on the general language elements presented above (i.e., LC1 – LC4) and hence abstract from the concrete process configuration approach taken.

## 6 Further Issues and Related Work

This section discusses further aspects that should be considered by any framework enabling configurable process models.

*Capturing variability of process perspectives other than control flow.* The previous sections have focused on the variability of activities and their control flow, whereas the variability of other process perspectives (e.g., data and resources) has yet to be considered. To overcome this limitation, for example, La Rosa et al. ([2008](#); [2011](#)) suggest a configurable process modeling notation, which incorporates features for capturing resources, data, and physical objects involved in the performance of activities. Similarly, Provop considers variability of the information and organization perspective (Hallerbach [2010](#)).

*Ensuring soundness of configured process variants.* A big challenge for any process configuration approach is to ensure that configured process variants are *sound*. When considering the large number of process variants that may be configured out of a configurable process model, as well as the many syntactic and semantic constraints these process variants have to obey, this constitutes a nontrivial task. In particular, manually correcting potential errors would hamper any process configuration approach. Instead, efficient and automated techniques for ensuring the soundness of process variant models are required.

Van der Aalst et al. ([2010](#)) propose a formal foundation for incrementally deriving process variants from a configurable reference process model, while preserving correctness in respect soundness. Specifically, assuming the configurable reference process model itself is sound, the derived process variants are guaranteed to be sound as well. The underlying theory was developed in the context of Petri nets and then extended to EPCs.

To ensure the soundness of configured process variants, van der Aalst et al. ([2010b](#)) suggest a verification approach inspired by the operating guidelines used in the context of partner synthesis (Lohmann and Wolf, [2011](#)). For this purpose, the configuration process itself is viewed as external service. Using partner synthesis, a *configuration guideline* is computed that constitutes a compact characterization of all feasible configurations. In particular, this allows ruling out configurations that lead to soundness violations. The approach is generic and imposes no constraints on the configurable process models to which it may be applied. Moreover, all computations are done at design time (i.e., when defining a configurable process model) and not at configuration time; i.e., there is no need for repeatedly checking each individual configuration when configuring a process variant model. Thus, once the configuration guideline has been generated, the response time is instantaneous, thus encouraging the practical use of configurable process models.

Hallerbach et al. ([2009](#)) show how the soundness of process variants can be ensured in the context of Provop. Thereby, advanced concepts are introduced that enable a context- as well as constraint-based configuration of process variants. In particular, it is shown how respective information can be utilized to effectively ensure soundness of the configured process variants.

*Merging process variants.* Designing a configurable process model is usually not done from scratch, but rather by analyzing existing process variants. Hence, merging these variants constitutes an important task that is also particularly relevant in today's world of company mergers and organizational consolidations. Considering the large number of process variants that may exist in enterprises, however, manually merging process models would be a tedious, time consuming, and error-prone task. Instead, techniques are required for automatically merging process variants in order to derive a configurable process model.

Regarding approaches like C-EPC or C-YAWL, variant merging needs to meet the following requirements. First, the behavior of the produced process model should subsume that of the input variant models (via the union of these input models). Second, it should be possible to trace back from which process variants an element has originated (via annotations). Third, one should be able to derive each input process variant from the merged one (via variation points). La Rosa et al. (2010) present an algorithm producing a single configurable process model from a pair of process variant models. This algorithm works by extracting the common parts of the input process variants, creating a single copy of them, and then appending the differences as branches of configurable connectors. This way, the merged process model is kept as small as possible, while still capturing all the behavior of the two input models. Moreover, analysts are able to trace back from which model(s) a given element in the merged model originated. The algorithm has been prototypically implemented and tested based on process models from several application domains.

Regarding structural approaches like Provop, a family of algorithms for merging process variants has been suggested by Li et al. (2011). These algorithms discover a process model by mining a given collection of process variants. Thereby, the discovered process model has a minimum average weighted distance to the considered process variants. By adopting the discovered model as new reference process model, future process configurations become more efficient, since the efforts (in terms of changes to be applied) for deriving the variants will be reduced.

*Retrieval of process variants.* There exist approaches that provide support for the management and retrieval of separately modeled process variants (i.e., optimizations of the multi-model approach). For example, Lu and Sadiq (2006) allow storing, managing, and querying large collections of process variants within a process repository. Graph-based search techniques are used in order to retrieve process variants that are similar to a user-defined process fragment (i.e., the query is represented as graph). Obviously, this approach requires profound knowledge about the structure of stored processes, an assumption that does not always hold in practice. Variant search based on process metadata (e.g., the process context) is not considered.

*Run-time flexibility for process families.* Existing approaches for managing process variability focus on the modeling and configuration of process variants. However, Ayora et al. (2012) show that run-time configuration and re-configuration as well as the evolution of process variants are essential requirements as well. Effectively handling process variants in these lifecycle phases requires deferring certain configuration decisions to the run-time, dynamically re-configuring process variants in response to contextual changes, adapting process variants to emerging needs, and evolving process families over time. Ayora et al.

(2012) characterize these flexibility needs for process families, discuss fundamental challenges to be tackled, and provide an overview of existing proposals made in this context.

*Applying object-oriented concepts to deal with process variability.* Different work exists on how specialization can be applied to deal with process model variability taking advantage of the generative power of a specialization hierarchy (Wyner and Lee 2003; van der Aalst and Basten 2002). In the context of the MIT Process Handbook, for example, Wyner and Lee (2003) show how specialization is enabled for simple state diagrams and dataflow diagrams, respectively. For both kinds of diagrams, a corresponding set of transformation rules is provided that result in process specializations when being applied to a particular model. Similarly, van der Aalst and Basten (2002) discuss transformation rules to define specialization for process models based on Petri Nets. Finally, Wyner and Lee (2003) show how specialization can be used to generate a taxonomy of processes to facilitate the exploration of design alternatives and the reuse of existing designs. Obviously, specialization and process taxonomies also allow capturing process variants to some degree. As opposed to the discussed approaches, Provop follows an operational approach, which is independent of the underlying process meta-model. In addition, Provop provides comprehensive support for the context- and constraint-based configuration of process variants.

A similar contribution stems from the PESOA project (Bayer et al. 2005, Puhlmann et al. 2005), which provides basic concepts for variant modeling based on UML. More precisely, different variability techniques like inheritance, parameterization, and extension points are provided and can be used when describing UML models. As opposed to PESOA, the operational approach enabled by Provop provides a more powerful instrument for describing variance in a uniform and easy manner; i.e., no distinction between different variability mechanisms is required.

*Software variability.* Variants are relevant in many other domains as well, including product line engineering and software engineering. For example, fundamental characteristics of software variability have been described in Bachmann and Bass (2001). In particular, software variants exist in software architectures and software product lines (Becker et al. 2001, Halmans and Pohl 2003). In many cases, feature diagrams are used for modeling software systems with varying features. A similar approach is offered by the so-called plus-minus-lists known from variant management in bill- of-materials. Correctness issues are not considered in both cases.

## **7 Summary and Outlook**

We have described the Provop approach for configuring and managing process variants. Provop considers the whole process life cycle and supports variants in all phases. This includes advanced techniques for modeling variants in a unified way and within a single process model, but without resulting in too complex or large model representations. Based on well-defined change operations, on the ability to group change operations into reusable options and on the possibility to combine options in a constrained way, necessary adjustments of the base process can be easily and consistently realized when creating and configuring a variant.

We successfully applied Provop in several case studies in the automotive, healthcare and governmental domains. Thereby, we were able to demonstrate the applicability of the Provop framework as well as to elaborate its benefits (Hallerbach [2010](#)). As pointed out by Torres et al. ([2012](#)), however, empirical research is required to evaluate Provop and other process configuration approaches (e.g., C-EPC) in respect to understandability issues that emerge when configuring concrete process variants out of a configurable process model.

Issues related to the evolution and change of process families (Ayora et al. [2013a](#)) as well as the flexible execution of process variants (e.g., to dynamically switch between variants during runtime; see Ayora et al. ([2012](#))) need to be addressed in future work. Moreover, process variability needs to be considered for other process support paradigms like, for example, data- and object-centric modeling approaches (Künzle and Reichert [2012](#); Reichert [2012a](#)). Finally, proper techniques for visualizing process variant collections are required (Reichert and Weber [2012](#), Reichert [2012b](#)).

## References

1. Ayora C., Torres V., Reichert M., Weber B., Pelechano V.: Towards run-time flexibility for process families: open issues and research challenges. In: Proceedings BPM'12 Workshops, LNBIP 132, Tallinn, Estonia, pp. 477-488 (2012)
2. Ayora C., Torres V., Weber B., Reichert M., Pelechano V.: Enhancing modeling and change support for process families through change patterns. In: Proceedings 14th International Working Conference on Business Process Modeling, Development, and Support (BPMDS'13), LNBIP 147, Valencia, Spain, pp. 246-260 (2013)
3. Ayora C., Torres V., Weber B., Reichert M., Pelechano V.: Change patterns for process families. Technical report PROS-TR-2013-06, University of Valencia, Spain (2013)
4. Bachmann F., Bass L.: Managing variability in software architectures. In: Proceedings Symposium on Software Reusability, New York, ACM Press, pp. 126-132 (2001)
5. Bayer J., Buhl W., Giese C., Lehner T., Ocampo A., Puhmann F., Richter E., Schnieders A., Weiland J., Weske M.: PESOA: process family engineering – modeling variant-rich processes. TR 18/2005, Hasso-Plattner Institute, Potsdam, Germany (2005)
6. Becker M., Geyer L., Gilbert A., Becker K.: Comprehensive variability modeling to facilitate efficient variability treatment. In: Proceedings 4th International Workshop of Product Family Engineering (PFE'01), LNCS 2290, Bilbao, Spain, pp. 294-303 (2001)
7. Becker J., Lis L., Pfeiffer D., Räckers M.: A process modeling language for the public sector – the PICTURE approach. In: Wybrane Problemy Elektronicznej Gospodarki, pp. 271-281 (2007)
8. Gottschalk F., van der Aalst W.M.P., Jansen-Vullers M.H., La Rosa M.: Configurable workflow models. International Journal of Cooperative Information Systems 17(2),177-221 (2007)

9. Gottschalk F.: Configurable process models. PhD thesis, Eindhoven University of Technology, The Netherlands (2009)
10. Gottschalk F., Wagemakers T., Jansen-Vullers M., van der Aalst, W.M.P., La Rosa, M.: Configurable process models: experiences from a municipality case study. In: Proceedings CAiSE'09, LNCS 5565, Amsterdam, The Netherlands, pp. 486-500 (2009)
11. Hallerbach A., Bauer T., Reichert M.: Modellierung und Darstellung von Prozessvarianten in Provop. In: Proceedings Modellierung'08, Berlin, Germany, LNI P-127, pp. 41–56 (2008 a)
12. Hallerbach A., Bauer T., Reichert M.: Context-based configuration of process variants. In: Proceedings 3rd International Workshop on Context-Aware Business Process Management (TCoB'08), Barcelona, Spain, pp. 31–40 (2008 b)
13. Hallerbach A., Bauer T., Reichert M.: Managing process variants in the process life cycle. In: Proceedings 10th International Conference on Enterprise Information Systems (ICEIS'08), Barcelona, Spain, pp. 154–161 (2008 c)
14. Hallerbach A., Bauer T., Reichert M.: Issues in modeling process variants with Provop. In: Proceedings BPM'08 Workshops, LNBIP 17, Milan, Italy, pp. 56–67 (2008 d)
15. Hallerbach A., Bauer T., Reichert M.: Anforderungen an die Modellierung und Ausführung von Prozessvarianten. Datenbank Spektrum 24, 48–58 (2008e)
16. Hallerbach A., Bauer T., Reichert M.: Guaranteeing soundness of configurable process variants in Provop. In: Proceedings 11th IEEE Conference on Commerce and Enterprise Computing (CEC'09), Vienna, Austria, pp. 98-105 (2009)
17. Hallerbach A.: Management von Prozessvarianten. PhD thesis, University of Ulm, Germany (2010)
18. Halmans G., Pohl K.: Communicating the variability of a software-product family to customers. Software Syst Model 2(1),15–36 (2003)
19. IDS Scheer: ARIS platform method 7.1, Technical Handbook, IDS Scheer GmbH, Germany (2008)
20. Kumar A., Wen Y.: Design and management of flexible process variants using templates and rules. Computers in Industry 63(2), 112-130 (2012)
21. Künzle V., Reichert M.: Striving for object-aware process support: how existing approaches fit together. In: Proceedings 1st International Symposium on Data-driven Process Discovery and Analysis (SIMPDA'11), LNBIP 116, Campione d'Italia, Italy, pp. 169-188 (2012)
22. La Rosa M., Lux J., Seidel S., Dumas M., ter Hofstede A.H.M.: Questionnaire-driven configuration of reference process models. In: Proceedings 19th International Conference on Advanced Information Systems Engineering (CAiSE'07), LNCS 4495, Trondheim, Norway, pp. 424–438 (2007)
23. La Rosa M., Dumas M., ter Hofstede A.H.M., Mendling J., Gottschalk F.: Beyond control-flow: extending business process configuration to roles and objects. In: Proceedings ER'08 Conference, Barcelona, Spain, pp. 199–215 (2008)

24. La Rosa M., Dumas M., Uba R., Dijkman R.M.: Merging business process models. In: Proceedings OTM Conferences (1), LNCS 6426, pp. 96–113 (2010)
25. La Rosa M., Dumas M., ter Hofstede A.H.M., Mendling J.: Configurable multi-perspective business process models. *Information Systems* 36(2), 313–340 (2011)
26. Lenz R., Reichert M.: IT support for healthcare processes – premises, challenges, perspectives. *Data Knowl Eng* 61(1), 39–58 (2007)
27. Li C., Reichert M., Wombacher A.: Mining process variants: goals and issues. In: IEEE 5th International Conference on Services Computing (SCC'08), Honolulu, Hawaii, USA, pp. 573–576 (2008 a)
28. Li C., Reichert M., Wombacher A.: Discovering reference process models by mining process variants. In: Proceedings 6th International Conference on Web Services (ICWS'08), Beijing, China, IEEE Computer Society Press, pp. 45–53 (2008 b)
29. Li C., Reichert M., Wombacher A.: Mining business process variants: challenges, scenarios, algorithms. *Data & Knowledge Engineering* 70(5), 409–434 (2011)
30. Lohrmann M., Reichert M.: Modeling business objectives for business process management. In: Proceedings S-BPM ONE 2012 Scientific Research, LNBIP 104, Vienna, Springer, pp. 106–126 (2012)
31. Lohmann N., Wolf K.: Compact representations and efficient algorithms for operating guidelines. *Fundam Inform* 108(1–2), 43–62 (2011)
32. Lu R., Sadiq S.: On managing process variants as an information resource. Technical report No. 464, University of Queensland, Australia (2006)
33. Müller D., Herbst J., Hammori M., Reichert M.: IT support for release management processes in the automotive industry. In: Proceedings 4th International Conference on Business Process Management (BPM'06), LNCS 4102, Vienna, Austria, pp. 368–377 (2006)
34. Mutschler B., Reichert M., Bumiller J.: Unleashing the effectiveness of process-oriented information systems: problem analysis, critical success factors and implications. *IEEE Trans Syst Man Cyberm C* 38(3), 280–291 (2008)
35. Puhlmann F., Schnieders A., Weiland J., Weske M.: PESOA – variability mechanisms for process models. Technical Report, Hasso-Plattner-Institute, Potsdam, Germany (2005)
36. Reichert M., Kolb J., Bobrik R., Bauer T.: Enabling personalized visualization of large business processes through parameterizable views. In: Proceedings 27th ACM Symposium on Applied Computing (SAC'12), Trento, Italy, ACM Press, pp. 1653–60 (2012)
37. Reichert M.: Process and data: two sides of the same coin. In: OTM 2012 Proceedings Part I, 20th International Conference on Cooperative Information Systems (CoopIS'12), LNCS 7565, Rome, Italy, pp. 2–19 (2012)
38. Reichert M.: Visualizing large business process models: challenges, techniques, applications. In: Proceedings BPM'12 Workshops, LNBIP 132, Tallinn, Estonia, pp. 725–736 (2012)



39. Reichert M., Weber B.: Enabling flexibility in process-aware information systems: challenges, methods, technologies. Springer, Berlin-Heidelberg (2012)
40. Rinderle-Ma S., Reichert M., Weber B.: On the formal semantics of change patterns in process-aware information systems. In: Proceedings 27th International Conference on Conceptual Modeling (ER'08), LNCS 5231, Barcelona, Spain, pp. 279–293 (2008)
41. Rosemann M., van der Aalst W.M.P.: A configurable reference modeling language. *Information Systems* 32, 1–23 (2007)
42. Torres V., Zugal S., Weber B., Reichert M., Ayora C., Pelechano V.: A qualitative comparison of approaches supporting business process variability. In: Proceedings BPM'12 Workshops, Tallinn, Estonia, LNBIP 132, pp. 560-572 (2012)
43. van der Aalst W.M.P., Basten T.: Inheritance of workflows: an approach to tackling problems related to change. *Theor Comput Science* 270(1–2), 125–203 (2002)
44. van der Aalst W.M.P., Dumas M., Gottschalk F., ter Hofstede A.H.M, La Rosa M., Mendling J.: Correctness-preserving configuration of business process models. In: Proceedings Fundamental Approaches to Software Engineering, LNCS 4961, Budapest, Hungary, pp. 46–61 (2008)
45. van der Aalst W.M.P., Lohmann N., La Rosa M., Xu J.: Correctness ensuring process configuration: an approach based on partner synthesis. In: Proceedings BPM'10 Conference, LNCS 6336, pp. 95–111 (2010)
46. van der Aalst W.M.P., Dumas M., Gottschalk F., ter Hofstede A.H.M., La Rosa M., Mendling J.: Preserving correctness during business process model configuration. *Formal Asp Comput* 22(3–4), 459-482 (2010)
47. VDA: Engineering change management (ECM) – Part 1 Engineering Change Request (ECR) Version 1.1, Recommendation 4965 T1(2005)
48. vom Brocke J.: Design principles for reference modelling - Reusing information models by means of aggregation, specialization, instantiation, and analogy. In: Fettke P, Loos P (eds) Reference modelling for business systems analysis. Idea Group Publishing, Hershey, PA, pp. 47–75 (2007)
49. Weber B., Reichert M.: Refactoring process models in large process repositories. In: Proceedings 20th International Conference on Advanced Information Systems Engineering (CAiSE'08), LNCS 5074, Montpellier, France, pp. 124–139 (2008 b)
50. Weber B., Reichert M., Rinderle S., Wild W.: Towards a framework for the agile mining of business processes. In: Proceedings BPM'05 Workshops, LNCS 3812, pp. 191–202 (2006)
51. Weber B., Rinderle S., Reichert M.: Change patterns and change support features in process-aware information systems. In: Proceedings 11th International Conference on Advanced Information Systems Engineering (CAiSE'07), LNCS 4495, Trondheim, Norway, pp. 574–588 (2007)
52. Weber B., Rinderle S., Reichert M.: Change patterns and change support features – enhancing flexibility in process-aware information systems. *Data Knowl Eng* 66(3), 438–466 (2008)

53. Weber B., Reichert M., Wild W., Rinderle-Ma S.: Providing integrated life-cycle support in process-aware information systems. *Int J Cooper Inform Syst* 18(1),115–165 (2009)
54. Weber B., Reichert M., Mendling J., Reijers H.: Refactoring Large Process Model Repositories. *Computers in Industry* 62(5), 467-486 (2011)
55. Wyner G.M., Lee J.: Defining specialization for process models. In: Malone TW, Crowston K, Herman GA (eds) *Organizing business knowledge – the MIT process handbook*. MIT Press, Cambridge, MA, pp. 131–174 (2003)