

# Capturing Variability in Business Process Models: The Provop Approach

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

<sup>1</sup> Group Research and Advanced Engineering, Daimler AG,  
Ulm, Germany

{alena.hallerbach, thomas.tb.bauer}@daimler.com

<sup>2</sup> Institute of Databases and Information Systems, University of Ulm, Germany  
manfred.reichert@uni-ulm.de

**Abstract.** Usually, for a particular business process different variants exist. Each of them constitutes an adjustment of a reference process model to specific requirements building the process context. Contemporary process management tools do not adequately support the modeling of such process variants. Either the variants have to be specified as separate process models or they are expressed in terms of conditional branches within the same process model. Both methods often lead to redundancies making model adaptations a time consuming and error-prone task. In this paper we discuss selected concepts of the Provop approach for modeling and managing process variants. A particular process variant can be configured at a high level of abstraction by applying a set of well-defined change operations to a reference process model. In particular, this paper discusses advanced concepts for the design and modeling of such a reference process model as well as for the adjustments required to configure the different process variants. Altogether, Provop provides a flexible and powerful solution for process variant management.

**Key words:** Process Variant, Process Configuration, Process Adaptation, Process Reference Model, Process Design

## 1 Introduction

For several reasons companies are developing a growing interest in improving the efficiency and quality of their internal business processes and in optimizing their interactions with customers and business partners [1, 2]. During the last years we have seen an increasing adoption of business process management (BPM) tools by enterprises as well as emerging standards for business process specification and execution (e.g., BPMN, WS-BPEL) in order to meet these goals. Corresponding technologies (e.g., workflow systems, case handling tools) enable the definition, execution, and monitoring of the operational processes of an enterprise [3]. In connection with Web service technology, in addition, the benefits of business process management from within a single enterprise can be transferred to cross-organizational business processes as well [4].

A *business process model* captures the activities an organization has to perform to achieve a particular business goal. Thus, it implements a process type (e.g., handling of a credit request or medical treatment of a patient) by describing its activities as well as their execution constraints (i.e., control flow), resources required (e.g., humans

or computer systems), and information processed. For creating and maintaining such business process models there exists a multitude of tools like ARIS Business Architect [5], ADONIS [6], and WebSphere Business Modeler [7]. These tools support different modeling techniques including UML Activity Diagrams, Business Process Modeling Notation (BPMN), and Event-Process-Chains (EPCs).

## 1.1 Problem Statement

Process support is required in almost all business domains [1]. As examples consider healthcare [2], automotive engineering [8, 9], public administration [10], and the financial sector [11]. Characteristic process examples from the automotive industry, for instance, include product change management [12] and product release management [8].

When creating business process models, one of the fundamental challenges the process designers faces is to cope with business process variability. Typically, for a particular process type, a multitude of *process variants* exists, each of them being valid in a particular context; i.e., the configuration of a particular process variant depends on concrete requirements building the *process context*. Regarding release management for electric/ electronic components in a car [8], for example, we have identified more than twenty process variants depending on the product series, involved suppliers, or considered 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 this paper, we use the service process handling vehicle repair in a garage as running example (cf. Fig. 1a): The process starts with the reception of a vehicle. After a diagnosis is made, the vehicle is repaired if necessary. During its diagnosis and repair the vehicle is maintained; e.g., oil and wiper fluid are checked and refilled if necessary. The process ends when handing over the repaired and maintained vehicle 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-specific variables. When studying this case, we have identified hundreds of process variants, we learned that existing BPM tools do not provide sophisticated support for modeling and maintaining such large number of process variants. In particular, existing BPM tools do neither support transparent management of process variants nor their context-based creation.

Fig. 1b-1d exemplarily show three (simplified) variants of our vehicle repair process: Variant 1 (cf. Fig. 1b), assumes that the damaged vehicle requires a checklist of “Type 2” to perform the diagnosis. Therefore, activities `Diagnosis` and `Repair` are adapted by modifying their attribute `Checklist` to value “Type 2”. Additionally, the garage omits maintenance of the vehicle as this is considered as special service not offered conjointly with the repair process. At the model level this is realized by skipping activity `Maintenance`. As a second example consider Variant 2 (cf. Fig. 1c). Here, due to country-specific legal regulations, a final security check is required before handing over the vehicle back to the customer. Regarding this variant, new activity `Final Check` has to be added when compared to the standard process from Fig. 1a. Finally,

Variant 3 (cf. Fig. 1d) will be relevant if a checklist of “Type 2” is required for vehicle diagnosis and repair, the garage does not link maintenance to the repair process, and there are legal regulations requiring a final security check.

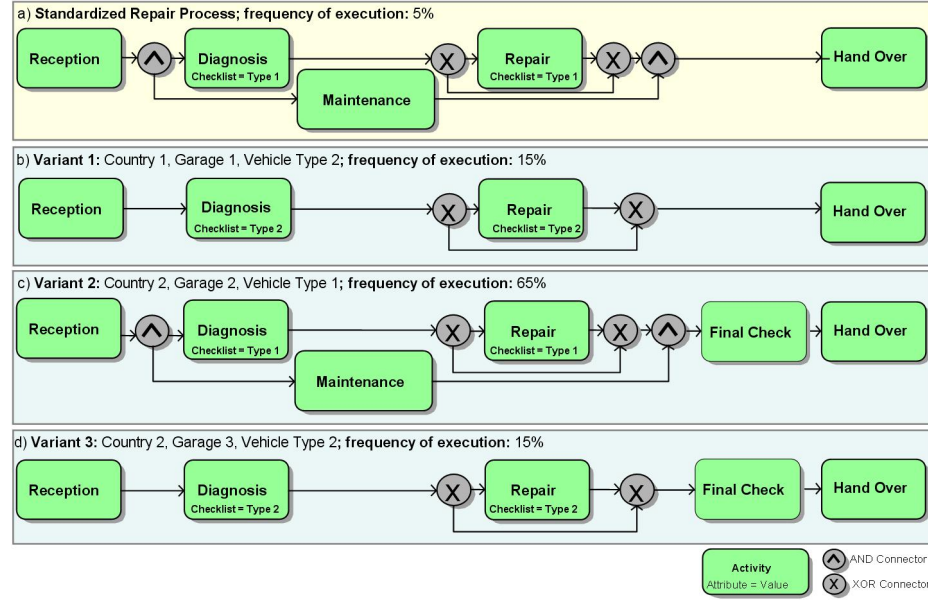


Fig. 1: Variants of a standardized vehicle repair process (simplified view)

Configuring process variants constitutes a non-trivial challenge when considering the high variability of business processes in practice as well as the many syntactical and semantical constraints the configured process variants have to obey depending on the given context. One challenge is to design a basic process model that can serve as reference for configuring a family of related process models. Another challenge is to design, model, and structure the adjustments that may be applied to configure the different process variants out of this basic process model.

## 1.2 Contribution

As motivated, variants exist for many business processes and should therefore be adequately managed. In previous work we have introduced the Provop framework (Process Variants by Options) and we have shown how information about the process context can be utilized for enabling automated configuration of process variants [9, 13, 14]. This paper picks up the Provop framework and extends it. We present selected concepts of the Provop approach for modeling and managing large collections of process variants in an adequate way. In particular, we deal with the following research questions:

- How to adequately design a *reference process* (denoted as *base process* in the Provop framework) out of which relevant process variants can be configured?

- How to pre-define the adjustments of a given base process, which may become necessary to configure relevant variant models? How to organize these changes in a way such that overall modeling and maintenance efforts can be reduced?
- How to support the evolution of a base process over time without invalidating existing variant models or introducing errors to them?
- How to integrate Provop concepts with contemporary process modeling tools?

Section 2 discusses characteristic problems which will arise if we do not treat variants as first class objects or model them conventionally. Section 3 summarizes basic concepts of the Provop framework. Sections 4 and 5 present sophisticated concepts supported by Provop for modeling process variants. Section 6 introduces our proof-of-concept implementation and Section 7 discusses related work. We conclude with a summary and outlook in Section 8.

## 2 Variant Management in Existing Tools and Requirements

We first discuss issues that emerge when modeling process variants based on existing BPM tools. Then we present requirements for modeling process variants, which we have identified in the context of our case studies we performed in the automotive and healthcare domains.

### 2.1 Conventional Variant Management

When using existing BPM tools, process variants are usually defined and maintained in separate process models as depicted in Fig. 1. We denote this straightforward solution as *multi-model approach*. It typically results in highly redundant model data as the variant models are identical or similar for most parts. Furthermore, the process variants are not strongly connected with each other; i.e., their process models are only loosely coupled (e.g., based on naming conventions). In particular, the multi-model approach does not provide any support for combining or merging existing variants to new ones [15].

When considering the large number of variants occurring in practice, the multi-model approach leads to significant modeling and maintenance efforts. Particularly, efforts for maintaining and changing process variants become high since more fundamental process changes (e.g., due to new or changed legal regulations) have to be separately accomplished for each individual variant model. This is both time-consuming and error-prone. As a consequence, variant models degenerate over time as optimizations are only applied to single variants without considering relations to other ones [16]. This, in turn, makes it a hard job for process designers to analyze, compare, and unify business processes.

Another approach frequently applied in practice is to capture multiple variants in one single process model using conditional branchings or labels [11, 17, 18]. We denote this solution as *single-model approach*. As example consider Fig. 2 which shows our basic repair process together with different variants (cf. Fig. 1a-d). Each execution path within the model then represents one particular variant. Generally, specifying all

variants in one process model results in a large model for a particular process family, which is difficult to comprehend and expensive to maintain.<sup>1</sup> As another drawback “normal” branchings cannot be distinguished from the ones representing a variant selection. Variants, therefore, are neither transparent nor explicitly defined. Consequently, the underlying system is unaware of the process variants and thus cannot provide sophisticated support for them; e.g., creating a view of the process model representing a particular process variant is not enabled unless the information about which process element (e.g., activity) is part of which particular process variant is explicitly included within the business process models [11].

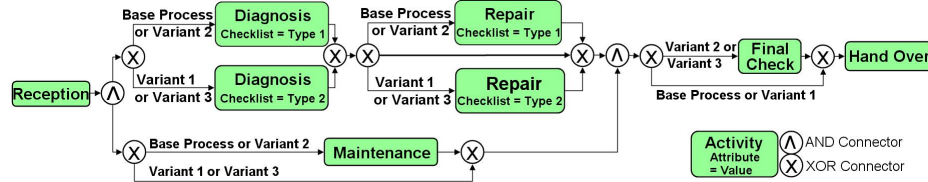


Fig. 2: Process variants realized by means of conditional branches

In summary, neither the use of separate process models for capturing the different variants nor the definition of these variants within one model (using conditional branches) constitutes a viable solution in many cases.

## 2.2 Requirements

We have conducted several case studies in the automotive industry [8, 12], but also in other domains like healthcare [2], to elaborate key requirements for the definition, adaptation, and management of process variants. This strong linkage to practice is required to realize a complete and solid approach for process variant management. The requirements we have identified are related to different aspects including the modeling of process variants, their linkage to process context, their execution in a workflow management system (WfMS), and their continuous optimization in order to deal with evolving needs; i.e., we have to elicit requirements related to the whole *business process lifecycle* [9, 19, 20, 21].<sup>2</sup> In our context, the process lifecycle consists of three major phases, namely the design and modeling phase, the variant configuration phase, and the execution or monitoring phase. It can be described as (feedback) loop of these phases during which a process is continuously optimized and adapted [20, 21].

**Modeling.** Modeling process variants should be intuitive and raise minimal efforts for process designers. Therefore, reuse of process (variant) models has to be supported when creating new process variants. In particular, it should be possible to create new

<sup>1</sup> Note that in realistic cases there may be dozens up to hundreds of variants derived from the same reference process.

<sup>2</sup> In this paper we focus on major requirements with respect to the modeling of process variants.

variants by inheriting properties from existing ones, but without creating redundant or inconsistent model data. The hierarchical structure of such “variants of variants” has to be adequately represented and should be easy to adapt.

**Configuration.** In order to relieve variant designers the configuration of a particular process variant should be accomplished in an automated way. Therefore, the specific circumstances (i.e., the process context) in which configuration takes place have to be considered. In particular, a context-aware variant configuration procedure is required. Thereby, one key requirement is to guarantee soundness of all configurable process variants throughout their entire lifecycle.

**Execution.** To execute a particular process variant, its model has to be interpreted by a workflow engine during runtime. In this context, it is important to maintain information about the selected process variants in the runtime system as well. Ideally, the runtime system should allow to dynamically switch process execution from one variant to another (e.g., when the process context changes). Such dynamic variant switches are by far not trivial when considering correctness and consistency issues as well.

**Maintenance.** Finally, in order to reduce both maintenance efforts and costs of change, fundamental process changes affecting multiple process variants should be conducted only once. Consequently, all process variants concerned by a particular change should be adapted automatically.

### 3 The Provop Approach

This section summarizes basics of the Provop framework, which are necessary for understanding the following sections.

#### 3.1 Backgrounds

Business processes are defined by a particular *process type* (e.g., handling a customer request or car repair in a garage) which is represented by a *process schema*. Often, for a particular process type, several process schemes exist, which represent different *variants* of this type. In this paper, we visualize a process schema as directed graph which comprises a set of nodes representing *process activities* or *control connectors* (e.g., AND-Split, XOR-Join), and a set of edges (i.e., control flow edges and data flow edges). Thereby, *control flow edges* describe precedence relations between activities, whereas *data flow edges* correspond to a read or write access of an activity to a data object. Note that Provop has not been developed with a specific process formalism in mind, but shall provide an overall conceptual framework for variant modeling and management. We further assume that a process schema can be changed by applying a sequence of (high-level) change operations (e.g., to add, delete or move activities) to it. We define process schema change and process variants as follows:

**Definition 1 (Process Change and Process Variant)**

Let  $\mathcal{P}$  denote the set of possible process schemes and  $\mathcal{C}$  the set of possible process changes. Let  $S, S' \in \mathcal{P}$  be two process schemes, let  $\Delta \in \mathcal{C}$  be a process change, and let  $\sigma = \langle \Delta_1, \Delta_2 \dots \Delta_n \rangle \in \mathcal{C}^*$  be a sequence of process changes. Then:

- $S[\Delta]S'$  iff  $\Delta$  is applicable to  $S$  and  $S'$  is the process schema resulting from the application of  $\Delta$  to  $S$ .
- $S[\sigma]S'$  iff  $\exists S_1, S_2, \dots, S_{n+1} \in \mathcal{P}$  with  $S = S_1, S' = S_{n+1}$ , and  $S_i[\Delta_i]S_{i+1}$  for  $i \in \{1, \dots, n\}$ . We also denote  $S'$  as variant of  $S$ .

**3.2 An Operational Approach for Variant Management**

When studying process variants in practice one can observe that they are often derived from an existing process schema; i.e., variant models are not created from scratch, but constitute an adjustment of an existing process model (cf. Def. 1).

The single-model approach introduced in Section 2.1 is one extreme form of reusing existing models. Here new variants are embedded in the original process model using conditional branchings. When following a multi-model approach, in turn, process variants are created by “cloning” a process model and adjusting it to the specific context. The latter approach is illustrated by Fig. 3: Here the original process model consists of a sequence of five activities (cf. Fig. 3a). Based on this original model three process variants are derived by performing variant-specific adjustments (cf. Fig. 3b+c). For example, Variant 1 can be configured by deleting activities B and C from the original process model and by updating attributes of activity D in this model. Variant 2, in turn, results when adding the sequence comprising activities N1 and N2 in parallel to the one consisting of activities B and C. Note that the latter can be realized by referring to adjustment points X and Y when adding the new sequence. Finally, Variant 3 can be configured by deleting activity C and by moving activity D behind activity A and before activity B.

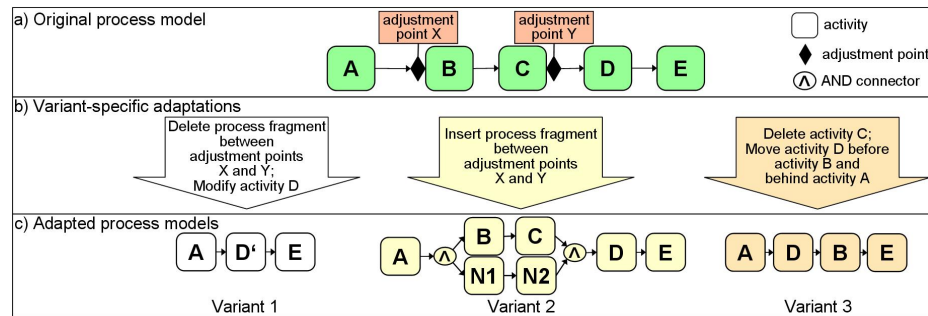


Fig. 3: General approach of variant modeling

Generally, a particular process model can be derived out of another one by adjusting it accordingly, i.e. by applying a set of change operations to it [22]. Starting from this

observation, we developed the Provop approach which provides comprehensive support for configuring variants out of a common process model. In the following we denote the original process model out of which the different variants can be configured as *base process* (cf. Def. 2).

**Definition 2 (Base Process, Options, and Process Family)**

Let  $\mathcal{P}$  be the set of process models and  $\mathcal{C}$  the set of possible process changes. Let further each process change  $\Delta \in \mathcal{C}$  be represented as parameterized change operation. Then:

- A base process  $S = (N, E, AP, \dots) \in \mathcal{P}$  is a process model with node set  $N$  and edge set  $E$ . Additionally, it is associated with a set of adjustment points  $AP \subseteq \text{Identifiers} \times N \times \{pre, post\}$ . Thereby, adjustment point  $ap = (id, n, pos) \in AP$  either corresponds to the entry ( $pos = pre$ ) or exit ( $pos = post$ ) of node  $n \in N$ . Adjustment points can be used as reference points when defining possible changes on the base process.
- Let  $S = (N, E, AP, \dots) \in \mathcal{P}$  be a base process. An associated option  $o = (oid, \sigma)$  then corresponds to a sequence of process changes (i.e.,  $\sigma = \langle \Delta_1, \Delta_2 \dots \Delta_n \rangle$ ) that may be applied to  $S$  (or to a variant derived from it). Thereby,  $\Delta_k$  ( $k = 1 \dots n$ ) refers to corresponding adjustment points or process elements (i.e., nodes and edges).
- Let  $S = (N, E, AP, \dots) \in \mathcal{P}$  be a base process and let  $O$  be the set of all options defined for it. Let further  $\Sigma = \langle o_1, \dots, o_n \rangle$  be a sequence of options with  $\{o_1, \dots, o_n\} \subseteq O$  and  $S[\Sigma]S'$ . Then we denote  $S'$  as process variant, which is configurable out of  $S$  by applying option sequence  $\Sigma$ . Further, we denote the total set of process variants that may be configured out of a base process with associated option set as process family.

A base process can be adjusted in different ways to configure a specific process variant. Currently, Provop supports the following change patterns (i.e., parameterizable change operations): INSERT fragment, DELETE fragment, MOVE fragment, and MODIFY attribute. While the first three patterns may be applied to a model fragment (i.e., a connected subgraph)<sup>3</sup>, the latter pattern can be used to modify the value of a process element attribute. More details on the Provop change patterns are provided by Table 3 (see Appendix). Fig. 5 gives an example of how change operations are applied in Provop. Finally, a formal semantics of these change patterns can be found in [23].

In Provop, an activity or connector node of the base process may be associated with *adjustment points* which correspond to its entry or exit respectively (cf. Fig. 3a). This, in turn, enables designers of process-specific adaptations to refer to selected process fragments. By the use of explicit adjustment points we can restrict the regions to which adaptations (e.g., insertion of a fragment) may be applied when configuring the variants. As Provop change operations have been designed to change process elements like activities and control flow edges rather than adjustment points, the latter constitute more robust reference points within the base process when compared to activity nodes.

As the number of change operations required to configure all relevant variants might become large, Provop allows to structure multiple change operations by grouping them into so called *options* (cf. Def. 2). This will be useful, for example, if the same change

<sup>3</sup> Note that a fragment does not necessarily need to have a single entry and single exit point.



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 outpatients no transport between ward and radiology room is required, inpatients first have to be transferred from the ward to the radiology unit and later back from the radiology unit to the ward. To capture the latter variant we need to add two activities at different positions of the base process. This can be achieved by first defining the two insert operations and then grouping them in one option. Generally, a particular process variant can be configured by applying several options to the given base process. As a special case no options may be applied, i.e. the base process itself may constitute a process variant.

### 3.3 Variant Management along the Process Lifecycle

Provop covers the whole process lifecycle as illustrated by Fig. 4. In the *modeling phase* a base process is defined out of which different process variants shall be configurable. For the latter purpose, the designer has to specify a set of reusable change operations which either explicitly refer to process elements (via element identifiers) or implicitly to a particular process fragment (through the use of adjustment points). As example consider Variant 1 from Fig. 3c. It can be configured out of the base process depicted in Fig. 3a by deleting the sequence comprising activities B and C, and by modifying attributes of activity D. The deletion of the activity sequence can be realized through a DELETE operation which refers to the fragment between adjustment points X and Y (cf. Fig. 3a). Finally, another design task is to group change operations into reusable operation sequences (i.e., options) as described before.

In the *configuration phase* the user selects a sequence of options (and corresponding operations respectively) and applies it to the base process in order to configure the desired process variant. (Note that the operations within options are ordered as well.) In the following we denote the set of process models that are configurable out of a base process as *process family*. Typically, a particular variant of such a process family is needed in a specific context. For example, the concrete variant of the process handling vehicle repair in a garage is determined by the specific country, vehicle type, and garage. To capture such context dependencies, Provop allows for the *context-aware* configuration of business process variants [24]; i.e., the context-based selection and application of relevant options to the base process. Following this approach variant configuration can be accomplished automatically making use of the given process context.

In the *execution phase* a configured process variant model is transformed into an executable workflow model, which is then deployed to the runtime environment of a workflow management system. Provop provides additional flexibility during the execution of process variants. In particular, it allows users to dynamically switch execution between different process variants during runtime. This becomes necessary, for example, in order to adequately deal with dynamic context changes (for details we refer to [9]).

Finally, by enabling adaptations of a given base process and its options, respectively, a process family may evolve over time. By enabling corresponding changes in an *optimization phase* Provop covers the whole process lifecycle.

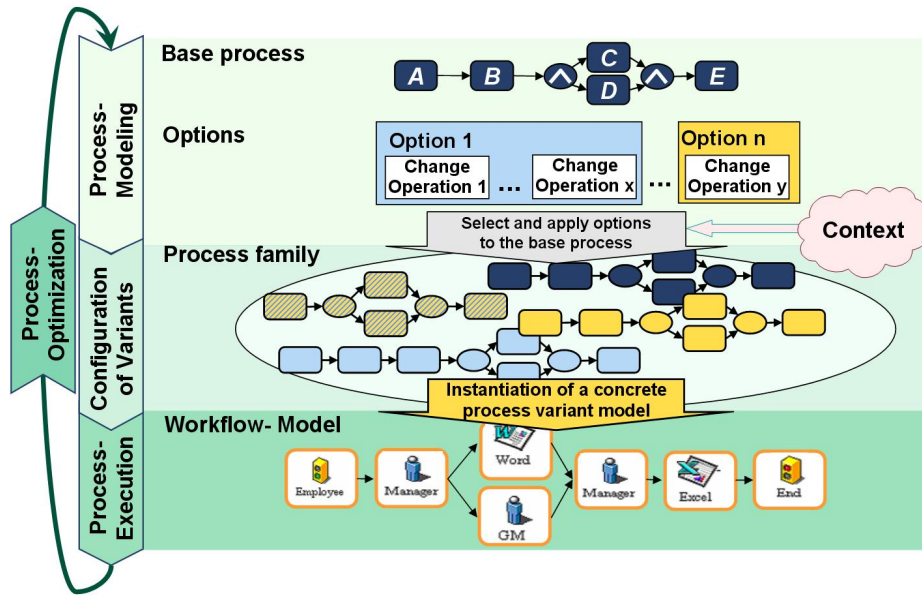


Fig. 4: The Provop process variant lifecycle

### 3.4 Example

Fig. 5 gives an example of a base process with three corresponding options. Further, the depicted base process comprises four adjustment points (with unique labels) to which defined options may refer. For example, Option 3 refers to adjustment points *Repair Completed* and *Ready for Hand Over*, which then serve as anchors (i.e., reference points) for the activity to be inserted by this option.<sup>4</sup> When applying Options 1 and 2 together to the depicted base process, for example, we obtain Variant 1 (cf. Fig. 1b). The model of Variant 2 (cf. Fig. 1c) may be created by applying Option 3 solely. Finally, Variant 3 (cf. Fig. 1d) results when first applying Option 2 and then Option 3 to the base process from Fig. 5a. In summary, Fig. 1 shows the process family configurable out of the base process and the corresponding options depicted in Fig. 5.

### 3.5 Discussion

The main advantage of the Provop approach in comparison to the previously introduced conventional approaches is the treatment of variants as first class objects. The latter can be considered as result of the use of explicit change operations and adjustment points when configuring the process variants. By contrast, in conventional approaches variants are either hidden in separate process models (multi-model approach) or are hard-wired in the control flow logic (single-model approach).

<sup>4</sup> If only single elements are affected by a particular change their identifier may be referred instead. This applies to Option 1 in the given example.

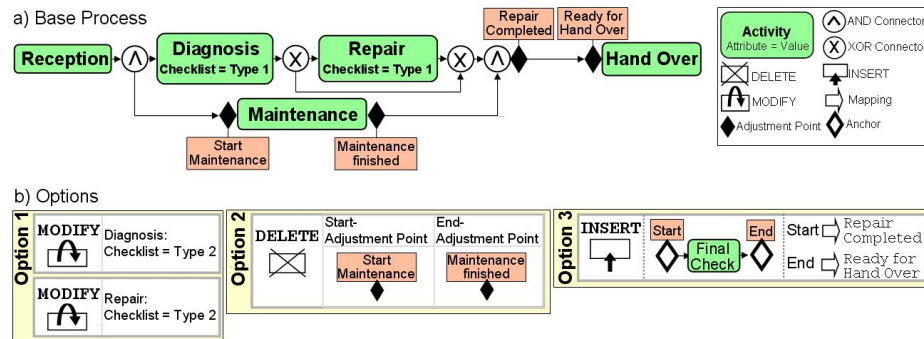


Fig. 5: A base process (a) and options (b) in Provop

Another crucial advantage of the Provop approach constitutes its ability to configure process variants based on the given process context. Finally, Provop provides a comprehensive approach offering solutions for all phases of the process variants lifecycle.

In Section 4 we discuss advanced issues that emerge when designing the base process of a process family; e.g., what policy to apply for this design, how to choose adjustment points, or how to evolve the base process over time. Following this, Section 5 presents sophisticated issues which are relevant for the design of options. The latter concern, for example, the granularity of options, constraints for the use of options, and soundness of configurable process variants.

## 4 Designing the Base Process of a Process Family

As prerequisite for configuring the variants of a process family we need to design its base process. The model of the designed base process then serves as reference for the options capturing possible model adjustments.

### 4.1 Policies for Designing a Base Process

In order to be able to support all characteristic use cases for process variant management and to adequately cover the broad spectrum of business processes we can find in companies today, Provop provides different policies for designing a base process:

- **Policy 1 (standard process):** Using this policy, the base process corresponds to a domain-specific standard or reference process. In the automotive domain, for example, such a standard process exists for Engineering Change Management [12]. Usually, a standard process has to be configured (i.e., adjusted) in order to meet the specific requirements of the given application context.
- **Policy 2 (most frequently used process variant):** If a particular process variant is used more frequently than others we can choose it as base process. This, in turn, reduces configuration efforts in terms of the number of processes for which adaptations become necessary. The process variants depicted in Fig. 1, for example, are weighted

considering execution frequencies. As can be seen the standard process (cf. Fig. 1a) has lowest frequency. Opposed to this, Variant 2 has highest frequency. Therefore, Variant 2 is chosen as base process when applying Policy 2. Note that Policy 2 does not mean that the average number of change operations needed to configure the variants of a process family (i.e., the average edit distance between the base process and the variants [25]) becomes minimal. However, this can be achieved when applying Policy 3.

- **Policy 3 (minimal average distance):** Through the structural mining of a collection of process variants, we can discover a (base) process model for which the average distance to the variant models becomes minimal. In the MinAdept project we are developing corresponding mining algorithms [26, 27, 28]. The algorithm presented in [27], for example, has polynomial complexity, which enables scalability for large collections of complex variant models. In [27] we have also shown that the discovered base process is better (i.e., requires lower average number of operations to configure the variants) than the process models we can discover through conventional process mining algorithms (e.g., Alpha algorithms, Multi-Phase mining, or Genetic Mining [29]). – Overall goal of Policy 3 is to learn from process adaptations and to merge the resulting model variants into a generic process model in the best possible way.
- **Policy 4 (superset of all process variants):** The base process is created by merging all known variants in one process model using conditional branches; i.e., the base process realizes a “superset” of all relevant variants. Consequently, every element that is part of at least one process variant belongs to the base process as well. When configuring process variants out of the base process, therefore, only DELETE operations have to be applied; e.g., the process model depicted in Fig. 2 constitutes the superset of all variants from Fig. 1 and may therefore be defined as base process.
- **Policy 5 (intersection of all process variants):** When applying this policy the base process will only comprise those elements that are part of all known process variants; i.e., the base process realizes an “intersection” of these variants. When configuring process variants no DELETE operations become necessary, but elements may have to be moved, modified or inserted. Regarding our example from Fig. 1, intersection of the variants would contain activities `Reception` and `Hand Over` (cf. Fig. 6).



Fig. 6: Intersection of the process family

As opposed to existing approaches for variant modeling [18, 26, 30, 31], Provop does not presume a specific policy for designing a base process of a process family and reference process, respectively. Instead, it provides more freedom to process designers and enables all five design policies. Note that Policies 1-5 differ in one interesting aspect: When using Policy 1 or 2 the respective base process is created to realize a specific use case; i.e., the base process represents one possible process variant valid in a specific context. Policies 3-5, in turn, are especially designed for configuring variants and thus

do not necessarily represent a semantically valid process model. Which policy is most favorable in a given context mainly depends on the given modeling scenario and the considered process landscape. For example, if a standard process model already exists, Policy 1 will be recommended to the designer.

## 4.2 Defining Adjustment Points

When creating a base process model for a process family another challenge is to define its adjustment points; i.e., explicit positions within the base process to which the parameters of the defined change operations may refer.<sup>5</sup> In order to better control possible adjustments when configuring process variants, Provop requires explicit specification of adjustment points. We recommend designers to use “business-relevant” reference positions within the base process. In Fig. 5, for example, adjustment point `Repair Completed` corresponds to the completion of activity `Repair`.

## 4.3 Evolving a Base Process and its Options

Provop targets at full lifecycle support for process variants. Thus, the controlled evolution of a base process and its corresponding options constitutes a fundamental task to be accomplished. When adapting a base process we have to ensure that corresponding process variants or – more precisely – the options needed to configure them, are adapted accordingly. In the following we exemplarily describe problems that arise when evolving a base process and we show how Provop deals with them.

**Potential problems caused by the evolution of a base process.** Assume that the base process depicted in Fig. 7a shall be transformed into the one shown in Fig. 7b. This change can be accomplished, for example, by removing activity `Maintenance` and by changing the order of the adjustment points `Ready for Hand Over` and `Repair Completed` (e.g., to correct wrong placement of these adjustments points) (cf. Fig. 7b). Consider further Options 4 and 5 (cf. Fig. 7c), which refer to the original base process from Fig. 7a. Option 4, modifies the duration attribute of activity `Maintenance` and therefore cannot be applied to the newly derived base process from Fig. 7b. (Note that activity `Maintenance` no longer exists in the corresponding process model). Option 5 (cf. Fig. 7c), in turn, inserts activity `Final Check` between the two adjustment points `Ready for Hand Over` and `Repair Completed`. As the order of these adjustment points is switched in the adapted base process, an insertion of activity `Final Check` would lead to an undesired cycle; i.e., when adding activity `Final Check` the corresponding branch would lead to a backward jump in the model of the new base process; after completing activity `Final Check`, execution of the base process would continue (i.e., activity `Hand Over` would be activated), while the backward jump would re-start execution of activity `Final Check`. This, in turn, would lead to an infinite loop and thus violate soundness (cf. Section 5.4) of the process variant.

---

<sup>5</sup> Note that change operations may not only refer to adjustments points, but to process element identifiers as well.

As a consequence, when updating a base process, applicability of all defined options as well as soundness of resulting process variant models (i.e., the process family) have to be guaranteed. For this purpose, Provop enables a number of checks that guarantee soundness of each configurable process variant model (see Section 5.4).

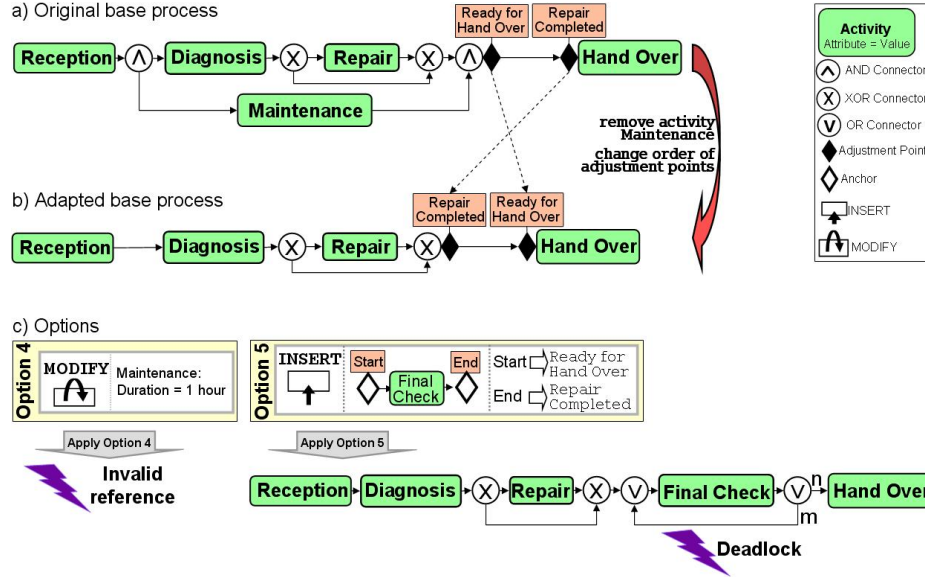


Fig. 7: Problems caused by base process adaptations

**Prohibiting problems caused by the evolution of a base process.** To avoid orphaned references after deleting elements of the base process, execution errors, and other potential problems, possible sources of errors are automatically detected in the Provop correctness checking framework (cf. Section 5.4). Furthermore, Provop provides elaborated concepts to exterminate potential problems associated with the evolution of a base process. For example, a DELETE operation can be based either on process element identifiers or on adjustment points. This, in turn, may lead to different effects when adapting a base process: Fig. 8a shows a base process model that evolves to the one depicted in Fig. 8b by adding activity *Final Check*. Here, Options 6 and 7 (cf. Fig. 8c) refer to the original base process. Option 6 performs a DELETE operation whose definition refers to the identifiers of single elements. When applying it to the adapted base process from Fig. 8b the resulting process model would contain activity *Final Check* (cf. Fig. 8c). Option 7, in turn, is based on adjustment points; i.e., it deletes every process element between adjustment points *Start Treatment* and *Finish Treatment*. As activity *Final Check* is inserted between these adjustment points it is deleted as well when applying Option 7 to the (already) modified base process. In fact, this would lead to the same process model as if Option 7 had been applied to the original base process.

When applying a DELETE operation, Provop allows users to choose between these two behaviors by either listing element identifiers or adjustment points. Note that in connection with base process evolution, the use of adjustment points constitutes a robust solution in respect to semantical and structural correctness.

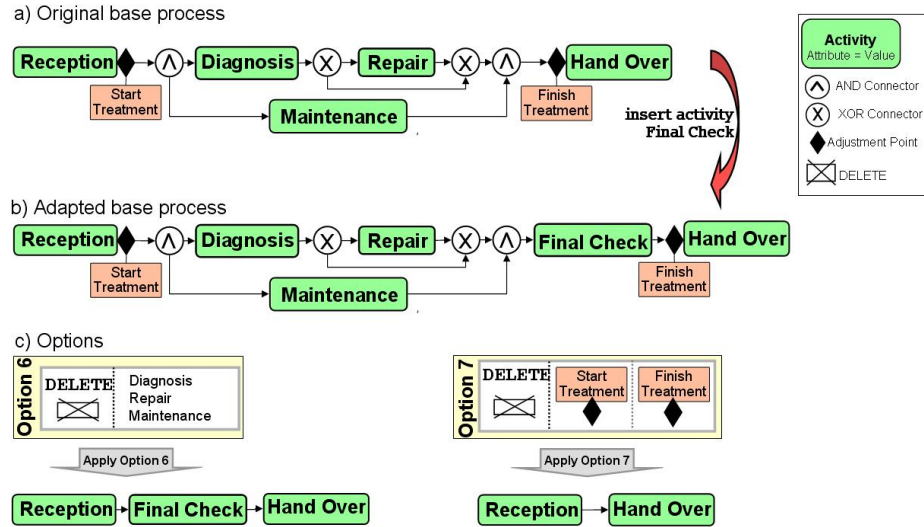


Fig. 8: Deleting elements either using their identifiers or adjustment points

#### 4.4 Using multiple Base Processes for Configuring Variants of a Process Family

In certain scenarios, the variants of a particular process family may differ to a large degree. One approach to deal with these differences is to configure the process variants out of different base processes; i.e., each of these base processes then covers a specific sub-class of variants of the considered process family. This minimizes the number of change operations required for configuring process variants and also enables a better structuring of the process landscape. One disadvantage coming with the use of multiple base processes for a particular process family, however, are higher modeling and maintenance efforts. In particular, inconsistencies and errors may result if a change affects all base processes. Hence, problems are similar to the ones which emerge when modeling variants separately (cf. Section 2.1). – Provop enables the use of multiple base processes, but will recommend process designers to only apply this approach if few classes of related process variants, which significantly differ from each other, can be identified.

## 5 Designing and Modeling the Options of a Process Family

So far, we have considered modeling issues related to the design of a base process for a particular process family. When defining corresponding adaptation (i.e. options) additional issues emerge. One question we have to answer is whether a conditional branch is only variant-specific or shall be considered in all variant models (and therefore be included in the base process). Furthermore, when grouping operations into reusable options, granularity selection becomes an issue. Another challenge is to deal with the many semantical and syntactical dependencies that may exist between different options. Finally, the explicit modeling of option constraints also becomes necessary to ensure semantical correctness of configurable process variants.

### 5.1 Variant-specific versus Normal Conditional Branchings

When modeling processes and their variants we have to decide which control flow alternatives are variant-specific and which ones shall be common for all process variants. For example, our vehicle repair process from Fig. 1a includes a conditional branching to decide whether or not the vehicle shall be repaired. The corresponding decision is required either to get approval from the vehicle owner with respect to expected costs or to omit the repair activity. Defining this particular scenario in terms of two separate process models (i.e., one with activity `Repair` and one without this activity) does not constitute a viable solution in the given context since both alternatives are relevant for all variants of our process family (cf. Fig. 1b-d). In particular, the decision to either omit activity `Repair` or to perform it cannot be made before executing activity `Diagnosis`. Therefore, such common decisions should be modeled as normal conditional branchings within the base process.

### 5.2 Granularity of Options

As sketched in Section 3, Provop allows to combine several options to configure a specific process variant. When defining process variants, therefore, the designer has to decide how to group change operations into options. Thereby aspects like reuse of options (and change operations respectively) as well as maintainability have to be considered.

As one extreme we can design fine-grained options with only one change operation per option. Such options can be easily combined to derive specific process variants. The number of options, however, then increases significantly. Coarse-grained options, in turn, group a larger number of change operations. An extreme would be to group all change operations needed to configure a specific process variant into one option. This, in turn, would lead to redundant definitions of those change operations relevant for multiple variants (and consequently to partially overlapping and redundant options).

The difference between fine- and coarse-grained options can be demonstrated using the base process and its variants from Fig. 1. Table 1 shows how the different change operations can be grouped into options when either applying the fine- or coarse-granularity approach. Note that neither fine nor coarse granularities are optimal in the given example, as fine granularity leads to four options and coarse granularity, in turn, to high redundancy with respect to the used change operations.



Table 1: Granularity of options

Fine Granularity:	Coarse Granularity:	Optimal Granularity:
Option 1: MODIFY Diagnosis	Option 1: MODIFY Diagnosis MODIFY Repair DELETE Maintenance	Option 1: MODIFY Diagnosis MODIFY Repair
Option 2: MODIFY Repair	Option 2: INSERT Final Check	Option 2: DELETE Maintenance
Option 3: DELETE Maintenance	Option 3: DELETE Maintenance INSERT Final Check	Option 3: INSERT Final Check
Option 4: INSERT Final Check		
# of Options = 4 # of Operations = 4	# of Options = 3 # of Operations = 6	# of Options = 3 # of Operations = 4

To support meaningful grouping of change operations into options, Provop defines several guidelines. In particular, one is to avoid redundant definitions of change operations within different options. Therefore, change operations should be grouped into one option if they are always used together; e.g., due to semantical interdependencies. Thereby, the number of options is reduced without affecting reuse. Considering this guideline the change operations of our example are grouped into the options shown in the third column of Table 1 (as well as in Fig. 5b): The adjustment for activity `Diagnosis` is always applied in conjunction with a change of the `Repair` activity. Therefore these two change operations can be grouped into one option. The other change operations, in turn, are modeled in separate options. As result we obtain three options without any redundancy.

### 5.3 Option Constraints

Generally, some of the options defined for a particular base process may be correlated; i.e., there may exist semantical and structural interdependencies between them. To capture this, Provop supports different kinds of option constraints (cf. Table 2):

- **Implication:** If different options shall be applied conjointly to the base process (e.g. due to semantical dependencies) the designer may explicitly define an implication constraint between them. Implication constraints are always directed. Fig. 9 depicts an example which visualizes a set of option constraints as partially ordered constraint network. Assume that the application of Option 1 always implies the use of Option 5 in order to configure a semantically valid variant model. Therefore, we have defined a corresponding implication constraint between these two options visualized as directed arrow in the constraint network. Such an arrow expresses that the source option of the arrow *implies* (i.e., requires) the application of its target option as well. As implication constraints are directed the reverse conclusion (i.e., *Option 5 implies Option 1*) is generally not true.

- **Mutual Exclusion:** Mutual exclusion, is helpful to describe which options must never be used in conjunction with each other when configuring a specific process variant. Provop visualizes the mutual exclusion constraint as bi-directed arrow between options. In our example from Fig. 9, Options 5 and 7 mutually exclude each other and therefore must not be applied together to the base process.
- **Application Order:** Options always have to be applied in sequence to a corresponding base process. As default Provop supports a low-level ordering mechanism based on time-stamps of options. As users may also define non-commutative options, however, this low-level ordering is enhanced by the provision of an explicit ordering constraint. For example, if both Option 3 and Option 6 (cf. Fig. 9) are selected for variant configuration, Option 6 will be always applied before Option 3 to the base process due to the defined ordering constraint. Note that such ordering constraint rather supports the correct application of options than correct option selection.
- **Hierarchy:** The definition of a hierarchy constraint between options allows to combine the constraint types `implication` and `application order`. If an option is selected to configure a particular process variant and it has an ancestor in the option hierarchy (e.g., Option 6 in Fig.9 constitutes a child of Option 1), the change operations defined in the ancestor options will be applied as well (i.e., implication constraint). Thereby, a parental option is always applied before its child options (i.e., application order constraint). By introducing such hierarchy constraints, we can reduce the number of constraints to be defined.
- **At-Most-n-Out-Of-m-Options:** Each variant of the process family has been created by applying at most  $n$  options out of  $m$  specified ones. For example, an `At-Most-2-Out-Of-3-Options` constraint is defined for Options 3, 6, and 7 from Fig. 9. When configuring a process variant none, one or two of these three options may be applied to the base process.

Table 2: Option Constraints

Constraint Type	Visualization	Expression
Implication		$\neg(\text{Option}_x \wedge \neg \text{Option}_y)$
Mutual Exclusion		$\neg(\text{Option}_x \wedge \text{Option}_y)$
Order of Application		$\text{Option}_x \Rightarrow \text{Option}_y$
Hierarchy		$\neg(\text{Option}_x \wedge \neg \text{Option}_y) \wedge (\text{Option}_x \Rightarrow \text{Option}_y)$
at least $n$ out of $m^*$		$\min_n(\text{Option}_1, \dots, \text{Option}_m)$
at most $n$ out of $m^*$		$\max_n(\text{Option}_1, \dots, \text{Option}_m)$
exactly $n$ out of $m^*$		$\min \max_n(\text{Option}_1, \dots, \text{Option}_m)$

$* n \leq m, n \in \mathbb{N}_0, m \in \mathbb{N}$

When defining relevant constraints for a set of options the designer usually does not only refer to one kind of constraint, but may want to consider different constraint types. Provop therefore enables the combined modeling of constraints as depicted in Fig. 9. For this purpose, the user first selects relevant options (Option 1, Option 3, Option 5,

Option 6, and Option 7 in the depicted example) and then defines the constraints (of same or different type) between them. (The constraints can be chosen from a palette (cf. Fig. 9)). Furthermore, Provop always ensures that the defined constraint network is consistent. For example, a mutual exclusion constraint between an option and its parental option would be disallowed. Corresponding consistency checks can be realized using an LTL-checker or some other model checking technique [34]. – In summary, the explicit definition of option constraints eases the usage of options and contributes to avoid semantical errors when configuring a particular process variant.

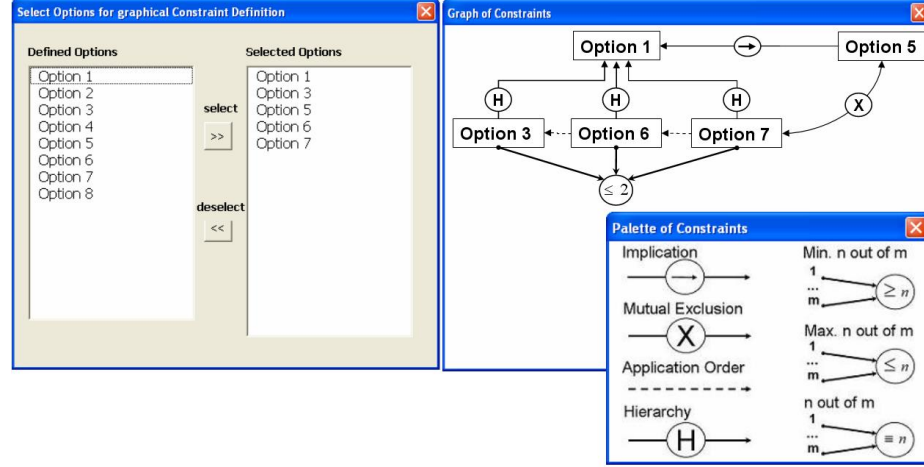


Fig. 9: Visualization of option constraints

#### 5.4 Ensuring Soundness of Configurable Process Variants

As aforementioned Provop targets at variant support along the whole process lifecycle. In particular, configured variants shall be executable in a workflow management system (cf. Fig. 4). Therefore, structural as well as behavioral soundness have to be guaranteed for the variant models before deploying them to the execution environment. In the following we sketch relevant issues emerging in this context and briefly discuss how we ensure soundness of variant models in the Provop framework. For further details we refer to [32, 33].

**Motivation.** Provop targets at *correctness by construction*; i.e., we want to ensure structural as well as behavioral soundness of all configurable process variants already at design time. One naive approach for accomplishing this would be to apply all possible combinations of options (i.e., all subsets of predefined adjustments) to the given base process and to check soundness for each of the resulting variant models. However, this would be too expensive in practical settings. As example consider the scenario from Fig. 5 for which three options exist. Taking into account that options are not commutative in general, we would have to check for 16 different option permutations whether

or not their application to the base process results in a sound variant model. Obviously, for more complex scenarios with dozens up to hundreds of options this is not a viable solution approach.

**Limiting the number of variant models to be checked.** Obviously, one challenge is to reduce the number of variant models for which we have to check soundness. As discussed in Section 3, Provop provides support for the context-based configuration of process variants. For this purpose each defined option is associated with a *context rule*, which is evaluated at configuration time based on the data building the process context. For a particular process context, exactly those options are selected and applied to the considered base process whose context rule evaluates to true. Furthermore, as described in Section 5.3, different kinds of constraints may exist between the selected options, which further reduces the possible permutations of a given option set. In summary, when utilizing information about context dependencies as well as option constraints, we can decrease the number of possible option permutations and thus reduce the number of (configurable) variants for which we have to check soundness.

**At which level do we need to check for soundness?** Assume now that we have to check soundness for a process variant that can be configured out of an ordered set of options. One possibility for this is to start the configuration procedure with a sound model of the base process and to enforce soundness after each applied option and change operation respectively. Consequently, the application of a set of operations and thus a set of options would result in a sound process model again. This rather rigid approach necessitates precise pre- and postconditions for the different change operations and corresponding checking routines. Another approach is to first apply the options in the given order to the base process and then to check soundness of the resulting process model afterwards. Provop follows the second approach since it provides more flexibility to the process designer (see [32, 33] for details).

**Do we need to start with a sound base process model?** As discussed Provop supports different policies regarding the design of the base process of a process family. Unlike existing approaches for process configuration, we do also not necessarily require a correct base process model as starting point for variant configuration. As example consider Fig. 10a where Variant 1 describes a car-specific and Variant 2 a bus-specific process variant. If we define the base process as "intersection" of these two variant models (cf. Policy 5), we obtain the process model depicted at the left bottom of Fig. 10a. It comprises activities CA1, CA2, and CA3, but does not contain the car- or bus-specific activity. Interestingly, this process model is not sound since the data element read by activity CA3 is neither written by CA1 nor by CA2. However, this missing soundness of the base process will not be any problem if we can ensure that always one of the two depicted variants is configured (i.e., each configurable variant model is sound). Enforcing a correct base process is therefore not required in the given scenario and also not appropriate. For example, assume that we choose the model of Variant 1 as base process (cf. Fig. 10b). Then visibility constraints might be violated. (Note that Variant 1 would then be visible to the process owner of Variant 2, who additionally must be able to track the adjustments of Variant 1 in order to correctly evolve Variant 2 over time).

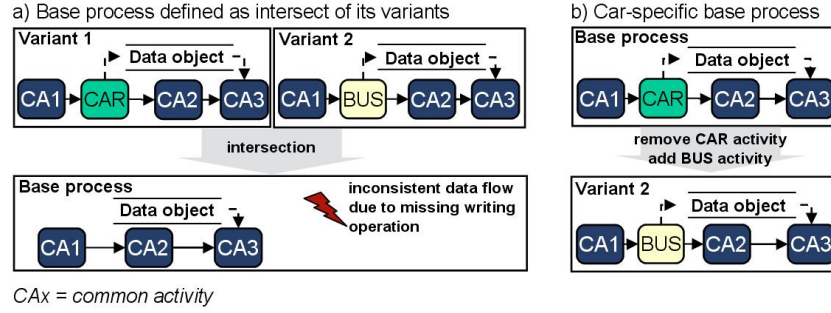


Fig. 10: Inconsistent base process (a) with an exemplary correctness scenario (b)

**Overview of the Provop correctness checking framework.** Guaranteeing soundness of configurable process variants is accomplished in several steps in the Provop framework (cf. Fig. 11). In Step 1, we identify the number of possible context descriptions for which corresponding process variants are needed. By only considering valid context descriptions we reduce the number of (configurable) variant models for which soundness has to be checked. In Step 2, for each valid context description, the corresponding option set (i.e., adjustments of the base process) is determined. Step 3 then checks whether or not the calculated option sets comply with the defined option constraints (cf. Section 5.3). If an option set is not valid an error will be reported to the designer. Otherwise, Steps 4 and 5 apply each potential option set to the base process and check whether or not the resulting process variant model is sound. Results of Steps 4 and 5 are logged in a report. For technical details and algorithms we refer to [32, 33].

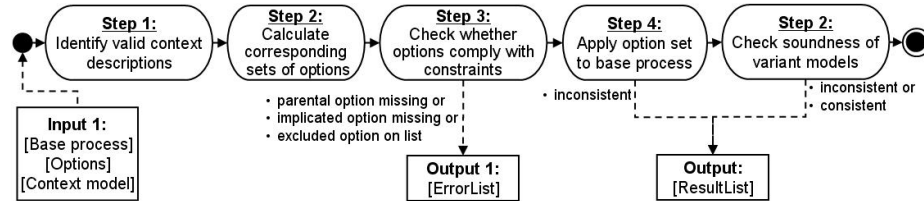


Fig. 11: Overview of the Provop procedure for guaranteeing soundness

## 6 Proof-of-Concept Prototype

When developing the proof-of-concept prototype for Provop, we had to decide whether or not to implement a new BPM tool from scratch or to enhance an existing one. On the one hand a newly developed tool offers the flexibility to implement the presented concepts in the best possible way, on the other hand this would come at the price of high development costs and long development times (e.g., basic functionality of the process

modeling tool would have to be implemented from scratch). Therefore we decided to use an existing BPM tool and to enhance it with facilities for process variant configuration and management. More precisely, for implementing our proof-of-concept prototype, we decided to use the *ARIS Business Architect*[5] which belongs to the ARIS Design Platform. ARIS Business Architect is a widespread tool that supports a variety of modeling notations (e.g., EPCs and BPMN), and that is widely used for modeling, analyzing and optimizing business processes.

The general limitations of commercial BPM tools with respect to variant modeling (cf. Section 2.2) also apply to the current version of ARIS Business Architect. The tool enables creation of new process variants by cloning (i.e., copying) an existing model repository, together with its objects, and by modifying them afterwards. This, in turn, results in high redundancies of model data. Though the derived variant objects still refer to the original objects afterwards (denoted as *master objects* in ARIS Business Architect), changes of the latter are not propagated to the variants. However, through the explicit documentation of relations (between original and variant objects) some improvement has been achieved when compared to other BPM tools.

Another decision we had to make when implementing the Provop proof-of-concept prototype concerns the choice of the process meta model (and modeling notation respectively) for representing base processes and options as well as the variant models resulting from corresponding configurations. We first started with Event-Process-Chains (EPCs) as this notation is widely used in our company. Unfortunately, EPCs do not offer a grouping functionality which is highly relevant in our context for grouping parameters of a particular change operation as well as for grouping multiple change operations into one 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. Fig. 12).

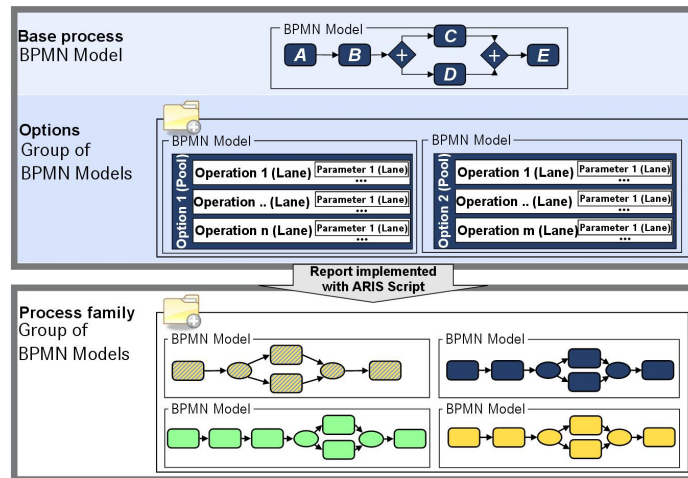


Fig. 12: Architecture of the Provop prototype

Fig. 12 shows the overall architecture of our proof-of-concept prototype. Here, each option is realized as a single BPMN model. Within these models corresponding operations of an option are encapsulated in so-called “pools”, which correspond to graphical and logical containers. The relevant parameters of a particular change operation (e.g., 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., Java-Script extended by specific functions), realizes the transformation of a base process 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.

Fig. 13 provides a screen of the Provop proof-of-concept prototype. It depicts an option comprising two operations. Operation 1 corresponds to an insertion, whereas Operation 2 deletes a fragment located between two adjustment points (represented as filled diamonds in Fig. 13). New objects and symbols (e.g., operation types and adjustment points) have been designed using the *ARIS Symbol Editor*.

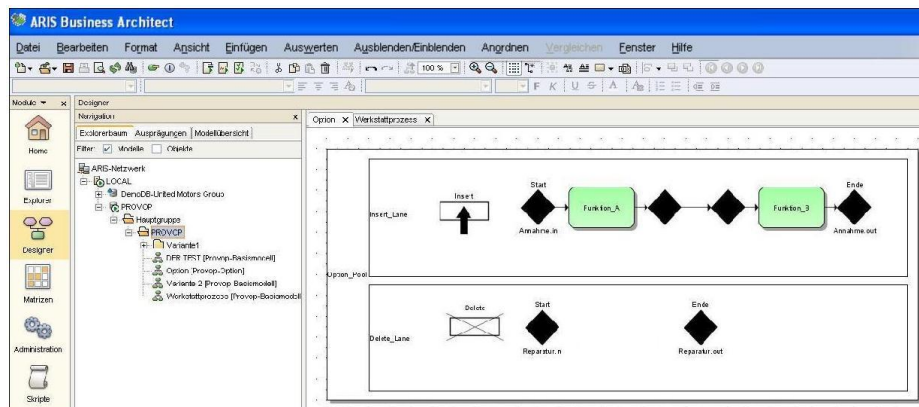


Fig. 13: Provop prototype implemented in ARIS

## 7 Related Work

Though the adequate support of process variants is highly relevant for practice, only few approaches for process variant management exist.

There exists adaptive process management technology that enables dynamic process changes during runtime; i.e., authorized users may dynamically adapt the structure (i.e., the schema) and or state of running process instances (e.g., by adding, deleting or moving activities) [22, 35, 36, 37]. Obviously, this runtime flexibility results in a

multitude of process variants, of which each represents one particular case (i.e., process instance) [27, 28]. The approach described in [38] additionally provides support for the management and retrieval of the resulting process instance variants. In particular, it becomes possible to store, manage, and query 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 process instances, an assumption which does not always hold in practice. Variant search based on process meta data (e.g., the process context) is not considered.

An important area related to variant management is reference process modeling. Usually, a reference process has a recommending character, covers a family of process models, and can be customized in different ways to meet specific needs. Configurable event process chains (C-EPCs), for example, provide support for both the specification and the customization of reference process models [17, 39]. When modeling a reference process, EPC functions (and decision nodes) can be annotated to indicate whether they are mandatory or optional. Such information is then considered when configuring the C-EPCs. A similar approach is presented in [18]. Here the concepts for configuring a reference process model (i.e., to enable, hide or block a configurable workflow element) are transferred to workflow models. Similar to Provop, these approaches allow to define constraints (denoted as "requirements") regarding the application of different adjustments of the reference process (e.g., two activities either may have to be deleted together from the reference process or none of them). As opposed to Provop, it neither is allowed to move or add model elements nor to adapt element attributes when configuring a variant. The work presented in [40] shows how reference process models can be configured incrementally and in a way that ensures correctness of the configured process variants, both with respect to syntax and behavioral semantics. As opposed to Provop, this approach always requires that the used reference process model (on which configurations are based) is sound. Finally, [11] introduces an approach to aggregate multiple process models (i.e., process variants) into one single model using aggregated EPCs. For this purpose, functions and events of the aggregated EPC are annotated with simple labels, which can then be used to identify those process elements relevant for extracting a particular process model out of the aggregated EPC. The proposed approach focuses on ease of use and improved maintainability of multiple, highly similar process models.

In principle, all these approaches constitute optimizations of the single model approach introduced in Section 2.1. As opposed to Provop, the suggested methods neither allow to move or add model elements nor to adapt element attributes when configuring a variant out of a reference or aggregated process model. Basically, the provided configuration support corresponds to Policy 4 where the chosen base process (i.e., reference process) constitutes the superset of all process variants. Obviously, in this specific scenario we only need to provide delete operations in order to configure a particular variant out of a reference process model. Note that Policy 4 constitutes only one out of several configuration policies supported by Provop; i.e., a base process can be defined in a more flexible way when using Provop.



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 [30, 41]. In the context of the MIT Process Handbook [30], for example, it is shown how specialization can be enabled for simple state diagrams and dataflow diagrams respectively. For both kinds of diagrams a corresponding set of transformation rules is provided that results in process specializations when being applied to a particular model. Similarly, [41] discusses transformation rules to define specialization for process models based on Petri Nets. Finally, [30] shows 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 to capture 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, we provide comprehensive support for context- and constraint-based configuration of process variants.

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 [42]. In particular, software variants exist in software architectures and software product lines [43, 44]. In many cases, feature diagrams are used for modeling software systems with varying features. A similar approach is offered by plus-minus-lists known from variant management in bill-of-materials. Correctness issues, identified as highly relevant in Provop and thus considered in our concepts, are not considered in both cases.

Another contribution stems from the PESOA project [45, 46], which provides basic concepts for variant modeling in connection with UML. 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 provided by Provop enables a more powerful way for describing variance in a uniform and easy manner; i.e., no distinction between different variability mechanisms is required.

Finally, [47] presents an approach which goes beyond control flow and extends business process configuration to roles and objects. At this point, Provop focuses on control and data flow, but considers process element attributes as well. Note that, for example, the role of an user can be treated as element attribute and thus can be updated by applying the Provop `modify` operation.

## 8 Summary and Outlook

In this paper we have introduced the Provop framework for modeling and managing large collections of business process variants. In particular, we discussed selected issues related to the modeling of a reference process model (i.e., a base process model) as well as the adjustments becoming necessary to configure this reference process to different process variants in an effective and manageable way. We have shown that Provop does not presume a particular policy for designing such a reference process model, but supports different policies in this context. The same applies with respect to the definition of options and model adjustments respectively. Thus, users have a high degree of flexibility when designing the ingredients for configuring the variants of a process

family. This, in turn, allows us to apply the Provop framework in different context and to a large spectrum of processes from different application domains. Furthermore, we have investigated advanced issues which are relevant for process variant management in practice. These include the controlled evolution of a process family, the context- and constraint-based configuration of process variants, and the correctness of configurable variant models. Finally, the provision of a proof-of-concept prototype that demonstrates basic Provop concepts adds to the completeness of our work.

One of the challenges on which we will have to work in more detail concerns the flexible execution of variants; i.e., to allow for the dynamic reconfiguration of process variants at runtime. Challenging issues in this context are to guarantee soundness of dynamically reconfigured process models, to ensure compliance with semantical constraints, and to accomplish such dynamic switches between variants in a way comprehensible to users. Finally, we will conduct additional case studies in industry in order to validate Provop concepts based on the realized prototype. In particular, we are going to apply the Provop approach to document the multitude of process variants that can be found in computer-aided engineering in the automotive domain.

## References

1. Mutschler, B., Reichert, M., Bumiller, J.: Unleashing the effectiveness of process-oriented information systems: Problem analysis, critical success factors and implications. *IEEE Transactions on Systems, Man, and Cybernetics (Part C)* **38** (2008) 280–291
2. Lenz, R., Reichert, M.: IT Support for Healthcare Processes - Premises, Challenges, Perspectives. *Data and Knowledge Engineering* **61** (2007) 39–58
3. Mutschler, B., Weber, B., Reichert, M.: Workflow management versus case handling: Results from a controlled software experiment. In: *Proc. of the 23rd Annual ACM Symposium on Applied Computing (SAC'08), Special Track on Coordination Models, Languages and Architectures*, ACM Press (2008) 82–89
4. Rinderle-Ma, S., Reichert, M., Jurisch, M.: Equivalence of web services in process-aware service compositions. In: *Proc. of the 7th IEEE Conference on Web Services (ICWS'09)*, IEEE Computer Society Press (2009)
5. IDS Scheer AG: ARIS Platform Method 7.1. (2008) [www.ids-scheer.com](http://www.ids-scheer.com).
6. BOC: The Business Process Management Tool ADONIS. (2007) (in German).
7. IBM: IBM WebSphere Business Modeller, Version 6.1. (2007)
8. Müller, D., Herbst, J., Hammori, M., Reichert, M.: IT Support for Release Management Processes in the Automotive Industry. In: *Proc. of the 4th Int. Conf. on Business Process Management (BPM'06)*. (2006) 368–377
9. Hallerbach, A., Bauer, T., Reichert, M.: Managing process variants in the process lifecycle. In: *Proc. of the 10th Int. Conf. on Enterprise Information Systems (ICEIS'08)*. (2008) 154–161
10. Becker, J., Lis, L., Pfeiffer, D., Räckers, M.: A process modeling language for the public sector - the picture approach. *Wybrane Problemy Elektronicznej Gospodarki* (2007) 271–281
11. Reijers, H.A., Mans, R.S., van der Toorn, R.A.: Improved model management with aggregated business process models. *Data Knowledge and Engineering* **68** (2009) 221–243
12. VDA Recommendation 4965 T1: Engineering Change Management (ECM) - Part 1: Engineering Change Request (ECR) Version 1.1 (2005)

13. Hallerbach, A., Bauer, T., Reichert, M.: Issues in modeling process variants with Provop. In: Proc. of the 4th Int. Workshop on Business Process Design (BPD'08), Workshop held in conjunction with Business Process Management (BPM'08) conference. LNBIP 17, Springer (2009) 54–65
14. Hallerbach, A., Bauer, T., Reichert, M.: Configuration and management of process variants. In Rosemann, M., Brocke, J.V., eds.: Handbook on Business Process Management, Springer-Verlag (2009) (forthcoming).
15. Wahler, K., Küster, J.M.: Predicting coupling of object-centric business process implementations. In: Proc. of the 6th Int. Conf. on Business Process Management (BPM'08). LNCS 5240, Springer (2008) 148–163
16. Weber, B., Reichert, M.: Refactoring process models in large process repositories. In: Proc. of the 20th Int. Conf. on Advanced Information Systems Engineering (CAiSE'08). LNCS 5074 (2008) 124–139
17. Rosemann, M., van der Aalst, W.: A Configurable Reference Modeling Language. Information Systems **32** (2007) 1–23
18. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H., la Rosa, M.: Configurable Workflow Models. In: Int. Journal of Cooperative Information Systems. (2007)
19. Hallerbach, A., Bauer, T., Reichert, M.: Anforderungen an die Modellierung und Ausführung von Prozessvarianten. Datenbank Spektrum **24** (2008) 48–58 (in German).
20. Weber, B., Reichert, M., Rinderle, S., Wild, W.: Towards a framework for the agile mining of business processes. In: Proc. of the 1st Int. Workshop on Business Process Intelligence (BPI'05) in conjunction with Business Process Management (BPM'05). LNCS 3812, Springer (2006) 191–202
21. Weber, B., Reichert, M., Wild, W., Rinderle-Ma, S.: Providing integrated life cycle support in process-aware information systems. Int. Journal of Cooperative Information Systems (IJCIS) **18** (2009) 115–165
22. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data and Knowledge Engineering **66** (2008) 438–466
23. Rinderle-Ma, S., Reichert, M., Weber, B.: On the formal semantics of change patterns in process-aware information systems. In: Proc. of the 27th Int. Conf. on Conceptual Modeling (ER'08). LNCS 5231, Springer (2008) 279–293
24. Hallerbach, A., Bauer, T., Reichert, M.: Context-based configuration of process variants. In: Proc. of the 3rd Int. Workshop on Technologies for Context-Aware Business Process Management (TCoB'08). (2008) 31–40
25. Li, C., Reichert, M., Wombacher, A.: On measuring process model similarity based on high-level change operations. In: Proc. of the 27th Int. Conf. on Conceptual Modeling (ER'08). LNCS 5231, Springer (2008) 248–264
26. Li, C., Reichert, M., Wombacher, A.: Mining process variants: Goals and issues. In: Proc. of the 5th IEEE Conf. on Services Computing (SCC 2008), IEEE Computer Society Press (2008)
27. Li, C., Reichert, M., Wombacher, A.: Discovering reference process models by mining process variants. In: Proc. of the 6th Int. Conf. on Web Services (ICWS'08), IEEE Computer Society Press (2008) 45–53
28. Li, C., Reichert, M., Wombacher, A.: What are the problem makers: Ranking activities according to their relevance for process changes. In: Proc. of the 7th Int. Conf. on Web Services (ICWS'09), IEEE Computer Society Press (2009)
29. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data and Knowledge Engineering **47** (2003) 237 – 267





30. Wyner, G., Lee, J.: Defining specialization for process models. In Malone, T., Crowston, K., Herman, G., eds.: *Organizing Business Knowledge - The MIT Process Handbook.*, MIT Press (2003) 131–174
31. Rosa, M.L., Dumas, M., ter Hofstede, A.H.M.: Modelling business process variability. Technical Report 13358, QUT ePrints Technical Report (2008)
32. Hallerbach, A., Bauer, T., Reichert, M.: Guaranteeing soundness of configurable process variants in Provop. In: 11th IEEE Conf. on Commerce and Enterprise Computing (CEC'09) (accepted for Publication).
33. Hallerbach, A., Bauer, T., Reichert, M.: Correct Configuration of Process Variants in Provop. Technical Report UIB-2009-03, Uni Ulm (2008)
34. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Control to Users. PhD thesis, Technical University of Eindhoven (2008)
35. Reichert, M., Rinderle, S., Kreher, U., Dadam, P.: Adaptive process management with ADEPT2. In: Proc. of the Int. Conf. on Data Engineering (ICDE'05), IEEE Computer Society Press (2005) 1113–1114
36. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. *Data and Knowledge Engineering* **50** (2004) 9–34
37. Weber, B., Sadiq, S., Reichert, M.: Beyond rigidity - dynamic process lifecycle support: A survey on dynamic changes in process-aware information systems. *Computer Science - Research and Development* **23** (2009)
38. Lu, R., Sadiq, S.: Managing process variants as an information resource. LNCS 4102. Springer (2006) 426–431
39. Rosa, M.L., Lux, J., Seidel, S., Dumas, M., ter Hofstede, A.H.M.: Questionnaire-driven Configuration of Reference Process Models. In: Proc. of the 19th Int. Conf. on Advanced Information Systems Engineering (CAiSE'07). LNCS 4495, Springer (2007)
40. van der Aalst, W.M.P., Dumas, M., Gottschalk, F., ter Hofstede, A.H.M., Rosa, M.L., Mendling, J.: Correctness-preserving configuration of business process models. In: Proc. of the 11th Int. Conf. on Fundamental Approaches to Software Engineering (FASE'08). LNCS 4961, Springer (2008) 46–61
41. van der Aalst, W.M.P., Basten, T.: Inheritance of workflows: an approach to tackling problems related to change. *Theoretical Computer Science* **270** (2002) 125–203
42. Bachmann, F., Bass, L.: Managing Variability in Software Architectures. In: Proc. of 2001 Symp. on Software Reusability, New York, ACM Press (2001) 126–132
43. Becker, M., Geyer, L., Gilbert, A., Becker, K.: Comprehensive Variability Modeling to Facilitate Efficient Variability Treatment. In: Proc. of the 4th Int. Workshop on Product Family Engineering (PFE'01). (2001)
44. Halmans, G., Pohl, K.: Communicating the Variability of a Software-Product Family to Customers. *Software and System Modeling* **2** (2003) 15–36
45. Bayer, J., Buhl, W., Giese, C., Lehner, T., Ocampo, A., Puhlmann, F., Richter, E., Schnieders, A., Weiland, J., Weske, M.: PESOA - Process Family Engineering - Modeling Variant-rich Processes. Technical Report 18/2005, Hasso-Plattner-Institut, Potsdam (2005)
46. Puhlmann, F., Schnieders, A., Weiland, J., Weske, M.: PESOA - Variability Mechanisms for Process Models. Technical Report 17/2005, Hasso-Plattner-Institut, Potsdam (2005)
47. 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: Proc. of the 27th Int. Conf. on Conceptual Modeling (ER'08). LNCS 5231, Springer (2008) 199–215

## 9 Appendix

### 9.1 Change Operations supported by Provop

Table 3 gives an overview of the change operations currently provided by Provop. For each operation we use a particular symbol as notation.

Table 3: Overview of the Provop change operations

<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>