Issues in Modeling Process Variants with Provop

Alena Hallerbach[1], Thomas Bauer[1], and Manfred Reichert[2]

[1] Group Research and Advanced Engineering, Daimler AG, Wilhelm-Runge-Straße, Ulm, Germany
{alena.hallerbach,thomas.tb.bauer}@daimler.com
[2] Database and Information Systems Group, University of Ulm, Germany
manfred.reichert@uni-ulm.de

Abstract. For a particular business process, typically, different variants exist. Each of them constitutes an adjustment of a basic process (e.g. a reference process) to specific requirements building the process context. Contemporary business process management (BPM) tools, however, do not adequately support the modeling and management of process variants. Either the variants have to be specified by separate process models or they are expressed in terms of conditional branches within the same process model. Both methods can lead to high model redundancies, which make model adaptations a time consuming and error-prone task. In this paper we discuss advanced modeling concepts of our Provop approach, which provides a flexible and powerful solution for modeling and managing process variants. With Provop, a particular process variant can be configured at a high level of abstraction by applying a set of well-defined change operations to a basic process model.

Key words: process variant management, process configuration, process reference models, process design methods and methodologies

1 Introduction

Usually, a business process model captures the activities an organization has to perform to achieve a particular goal. More precisely, it implements a process type (e.g., for handling a credit request or for declaring travel costs) by describing process activities as well as their execution constraints (i.e., control flow), resources required (e.g., humans or IT systems), and information processed. For creating and maintaining such process models, there exist tools like ARIS Business Architect [1], ADONIS [2], and IBM WebSphere Business Modeler [3]. These tools, in turn, support different modeling methods like UML Activity Diagrams, BPMN or Event-Process-Chains (EPCs). When modeling business processes several objectives exist. As example consider improved process transparency. By its model-based documentation, process information is provided in a more transparent and unified way to users. As another advantage, process models can be analyzed and simulated resulting in further optimizations. However, modeling, analyzing, and optimizing processes constitute only one side of process management. The other one is to implement and execute these processes, e.g., based on workflow management systems (WfMSs). For this purpose, executable workflow models have to be provided. Based on such models, a WfMS controls the execution of process activities and assigns them to user worklists during runtime [4].

Process support is required in almost all business domains (e.g., healthcare [5], automotive engineering [6, 7], or public administration). Characteristic process examples from the automotive industry, for instance, include product creation, product change management, and release management [6]. Typically, respective processes occur in a multitude of *variants*, whereby each of these variants is valid in a particular context; i.e., the selection of a concrete process variant depends on concrete requirements building the *process context*. When having a closer look at the product creation process, for example, different process 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.

Similar considerations can be made for the process *handling vehicle repair in a garage* as depicted in 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 wiping water will be 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. Fig. 1b-1d show examples of such process variants: Variant 1 as depicted in Fig. 1b assumes that the cause of a vehicle defect can be detected very fast and repair can be done quickly as well; e.g., a flat tire constitutes an obvious break down problem and does not need detailed studies of the flashed software. Therefore the activities `Diagnosis` and `Repair` are shortened by modifying the attribute *duration*. Additionally, the customer wants to omit maintenance of his vehicle, as this shortens the process and reduces costs. At the model level this is indicated by skipping activity `Maintenance`. As another example consider Variant 2 as depicted in Fig. 1c: Because of legal regulations a specific security check is required before handing over the vehicle to the customer. Regarding this variant, a new activity `Final Check` is inserted when compared to the standardized process from Fig. 1a. Finally, Variant 3 will result if a `Final Check` is required and the customer wants to omit `Maintenance` activities (cf. Fig. 1d).

This paper presents selected concepts of the Provop (PROcess Variants by OPtions) approach for managing large collections of such process variants in a single process model. In particular, we deal with specific modeling issues which complement our previous work [7, 8]. Section 2 presents problems which will arise if we do not treat variants as first class objects or only model them conventionally. Section 3 describes basic Provop concepts. In Section 4 and 5 we address advanced concepts for variant modeling in Provop. Section 6 discusses related work. The paper concludes with a summary and an outlook in Section 7.

2 Challenges and Requirements

We first discuss issues that emerge when modeling process variants based on conventional BPM tools. Afterwards, we introduce major requirements for modeling process variants, which we identified during case studies we had performed in the automotive domain.

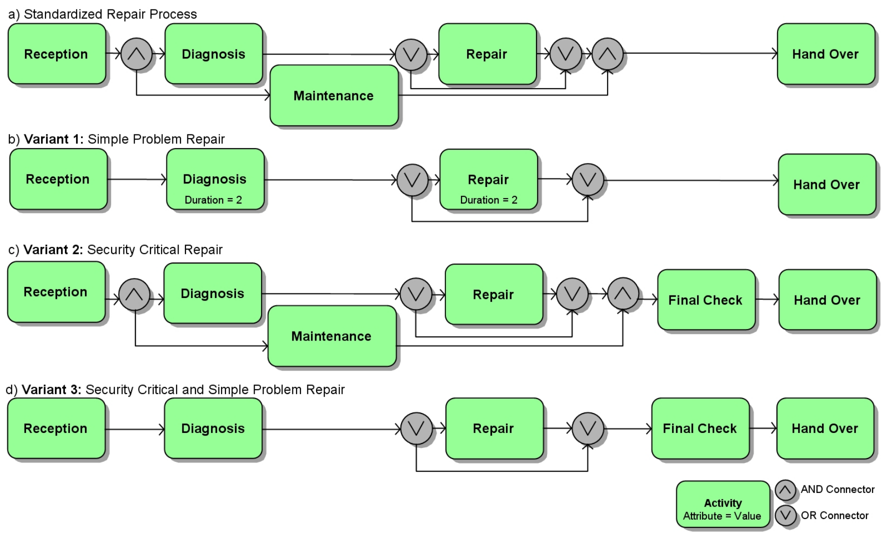

Fig. 1: Variants of a Standardized Vehicle Repair Process (Simplified View)

Conventional Variant Management: In existing BPM tools, process variants usually have to be defined and kept in separate process models as shown in Fig. 1. This typically 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) and there is no support for (semi-) automatically combining existing variants to a new one. Considering the large number of variants occurring in practice, these 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 individual variant (e.g., due to new or changed legal regulations). 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 others. This, in turn, makes it a hard job for process designers to analyze, compare, and unify business processes. Additionally, IT systems providing support for multiple process variants are difficult to realize. As a conclusion, modeling all process variants in separate models does not constitute an adequate solution for variant management.

Another straightforward approach frequently applied in practice is to capture multiple variants in one single process model based on conditional branching. As example consider Fig. 2 which shows the repair process together with its different variants (cf. Fig. 1a-d). Each execution path in the model represents a particular variant with branching conditions being correlated to the variant context. Generally, specifying all variants in one process model can result in large models, which are difficult to comprehend and expensive to maintain.[1] As another drawback, to a large degree, variants are then mixed with "normal" process logic; i.e., regular branching conditions cannot be distinguished from the ones representing a variant selection. Therefore, neither variants are transpar-

[1] Note that in realistic cases there might be dozens to hundreds of variants of one base process.

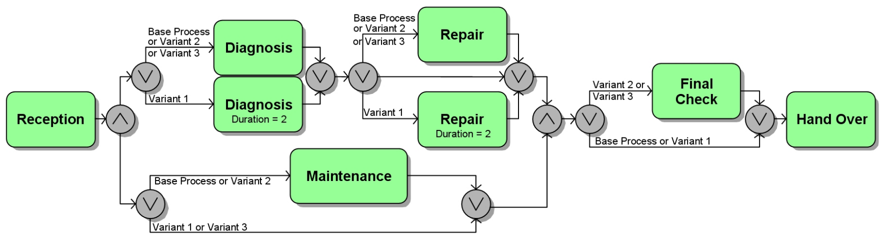

Fig. 2: Process Variants realized by Conditional Branches

ent nor they are explicitly defined. As a consequence the underlying system is not aware of process variants.

In summary, neither the use of separate models for capturing process variants nor their realization based on conditional branches within a single process model (as described above) constitute adequate methods.

Requirements: We conducted several case studies in the automotive industry as well as in other domains (e.g. healthcare) 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 identified are related to different aspects including the modeling of process variants, their linkage to process context, their execution in a WfMS, and their continuous optimization to deal with evolving needs; i.e., we have to respect requirements related to the whole *process life cycle*.[2]

Modeling process variants should be intuitive and come with minimal efforts for process designers (Req. 1). Therefore, reuse of both process fragments and process models (of the different process variants) has to be supported (Req. 2). In particular, it should be possible to create new variants by inheriting properties from existing ones, but without creating redundant or inconsistent model data (Req. 3). The hierarchical structure of such "variants of variants" has to be adequately represented and should be easy to adapt (Req. 4). To reduce both maintenance efforts and costs of change, fundamental process changes affecting multiple process variants should be performed only once. Consequently all variants concerned by such a change should be adapted automatically (Req. 5).

3 The Provop Approach

Basically, a process variant can be created by "cloning" a given process model and by adjusting it. Consequently the process model of a variant is similar to the one it is derived from. For example, the process variants depicted in Fig. 1b- 1d constitute adjustments of the process model shown in Fig. 1a to particular process contexts and are therefore similar. Provop utilizes this similarity and allows to configure the variants

[2] In this paper we focus on major requirements to be met when modeling process variants. There exist further requirements addressed by Provop, which we cannot mention here.

out of a common model - the *base process*; i.e., a process variant results when applying a set of high-level change operations to such base processes.

A base process can be adjusted in different ways to configure a specific process variant. Provop supports the following change operations: `INSERT`, `DELETE`, and `MOVE` *process fragment*s and `MODIFY` *process element attributes*. 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. Adjustment points are labeled with unique names; e.g., "Start Maintenance" at the entry of activity *Maintenance* (cf. Fig.3). If only single elements are affected by a particular change operation their process element IDs may be used alternatively. Furthermore, to deal with more complex process adjustments, Provop allows to group change operations into reusable sets. We denote these sets as *options*. A particular process variant can then be configured by applying one or more options to a base process, i.e., by performing all change operations defined by these options. At this point, we assume that the combined application of a set of options to a sound base process results in a sound process model again. Generally, we have to restrict possible combinations of options to ensure this (cf. Section 5).

Fig. 3 presents the basic elements of the Provop modeling approach. Fig. 3a presents a base process, whereas Fig. 3b depicts three predefined options that can be applied to this base process in order to derive a particular process variant. Furthermore, to labeled adjustment points (i.e., black filled diamonds) of a base process, the change operations may refer. The modeled options specify what kinds of adjustments have to be made (`INSERT`, `DELETE`, `MOVE`, and `MODIFY`). Application of Options 1 and 2, for example, results in Variant 1 as depicted in Fig. 1b. The model of Variant 2 (cf. Fig. 1c) can be created by applying Options 3. Finally, Variant 3 results when applying Options 2 and 3 to the base process.

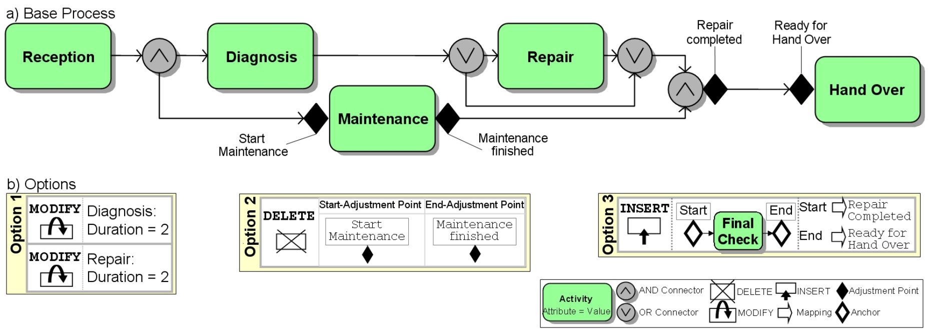

Fig. 3: The Provop Constructs Base Process (a) and Options (b)

Provop aims at the support of the whole process life cycle [7]. In previous work, we have shown how to model and visualize a base process and its options [9]. Furthermore, we have given insights into the context-based configuration of variants by applying a context-dependent set of options to the base process [8]. In the following we focus on additional issues emerging in the context of variants modeling.

4 Designing a Base Process

Basic to the configuration of process variants of a particular type is a respective base process, which serves as reference. Several aspects become relevant when designing such base processes, including its nature and the definition of adequate adjustment points.

Policies for defining a base process: When considering typical use cases as well as the overall process landscape different policies for defining a base process exist (cf. Req. 1). Basically, one of the following policies can be chosen in the context of Provop:

- **Policy 1 (Standard Process):** The base process represents a domain-specific standard or reference process. In the automotive domain in which we conducted our case studies, for example, such a reference process exists for Engineering Change Management [10]. Usually, such a standard process has to be adjusted to meet specific requirements; i.e., it should be possible to derive different variants from a standard process.
- **Policy 2 (Most Frequently Used Process):** If one particular process variant is used more frequently than others, it can be chosen as base process. This will reduce configuration efforts in terms of the number of processes for which adjustments become necessary. Generally, this policy does not guarantee 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 techniques for structural mining to a collection of process variants [11], we can derive a base model such that the average distance between this model and its variants becomes minimal. Thus, configuration efforts can be reduced.
- **Policy 4 (Superset of all Process Variants):** The base process is created by merging all variants into one process model by using alternative branches; i.e., the base process realizes a "superset" of all relevant process variants. Consequently, every element that is part of at least one process 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 process variants given in any order; i.e., the base process realizes a kind of "intersection" of relevant process 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 interesting aspect: When using Policy 1 or 2 the respective base process is created to serve 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 to follow is mainly influenced by the modeling situation and the existing process landscape (e.g., if a standard process model already exists Policy 1 will be recommended).

Fixing adjustment points: When a base process model is created another challenge is to define its adjustment points; i.e. explicit positions within the base process to which the high-end change operations may refer [3] In Provop, both the entry and the exit of activities or connector nodes from the base process can serve as adjustment points. Defining too many adjustment points for a base process, however, would lead to complex process models. Therefore, Provop asks for an explicit specification of valid adjustment points. We recommend designers to use "business-relevant" positions within the base process (e.g., the adjustment point `Repair completed` describing the end of activity `Repair` as depicted in Fig. 3).

Evolving the base process: Another challenge concerns the evolution of base processes for which corresponding options exist. For example, assume that the base process depicted in Fig. 4a is adapted by removing Activity `D` and by swapping Activities `A` and `B` (cf. Fig. 4b). Assume further that Options 1 and 2 (cf. Fig. 4c) had been defined for the original base process before it was adapted: Option 1 modifies the duration of Activity `D`. Obviously, this change operation cannot be applied to the adapted base process from Fig. 4b as Activity `D` no longer exists. Consequently, the process variant cannot be derived correctly. Option 2 inserts Activity `F` using adjustment points `X` and `Y`. As the order of these adjustment points is changed in the adapted base process, insertion of `F` would lead to a deadlock causing cycle: When inserting an activity a parallel branch comprising the inserted activity is created and connected to the base process at given adjustment points. When inserting Activity F the parallel branch leads to a backward jump in the process model: After the execution of Activity `A` the control-edges `m` and `n` are signaled as true. Therefore, the execution of the base process is continued (i.e., Activity `C` is activated), while the backward jump starts execution of Activities `B` and `A` again. This leads to a serious execution error at runtime.

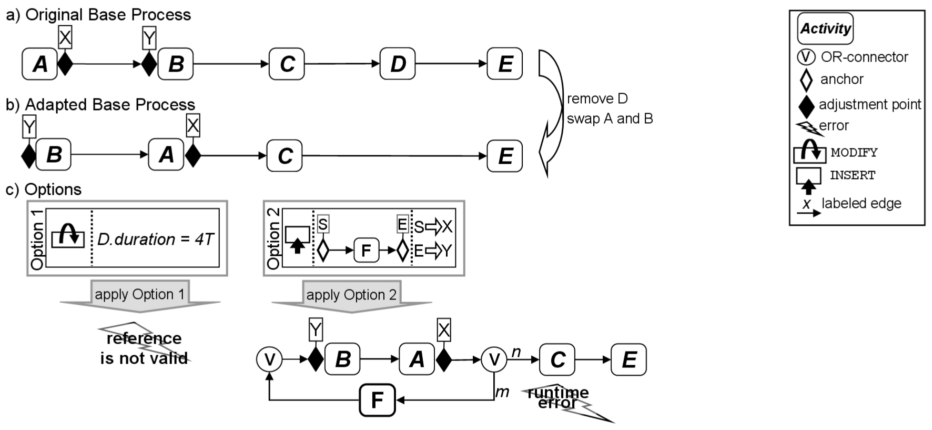

Fig. 4: Problems caused by Base Process Adaptations

[3] Note that change operations are not restricted to adjustments points but may alternatively use process element ids, that are not visualized explicitly in the base process.

To avoid references to missing objects, execution errors and other problems, possible sources of errors have to be automatically detected by the system (e.g., by checking whether existing options and base model adaptations are conflicting).

Provop provides elaborated concepts to support the adaptation of base processes. For example, a `DELETE`-operation either can be defined based on element IDs or by using adjustment points. This, in turn, may lead to different effects when adapting a base process: Fig. 5a shows a base process that evolves to the model depicted in Fig. 5b by adding Activity `F`. Options 1 and 2 (cf. Fig. 5c) were defined before adapting this base process. Option 1 performs a `DELETE`-operation whose definition refers to IDs of single elements. When applying it to the adapted base process from Fig. 5b the resulting process model would contain Activity `F` (cf. Fig. 5c). Option 2, in turn, is based on adjustment points; i.e., it deletes every process element between adjustment points `X` and `Y`. As Activity `F` is inserted between adjustment points `X` and `Y` it will be deleted when applying Option 2 to the adapted base process. In fact, this leads to the same process model as if Option 2 would have been applied to the original base process. When modeling a `DELETE`-operation Provop allows users to choose between these two behaviors.

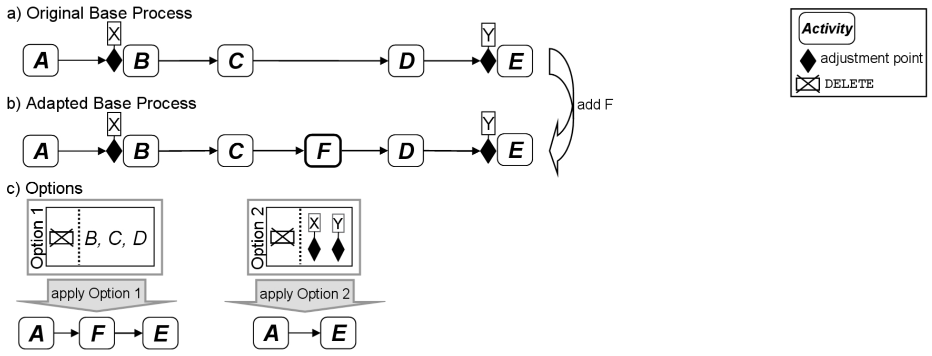

Fig. 5: Deleting Elements using their IDs or Adjustment Points

By adapting the base process all process variants can be reconfigured automatically (Req. 5). Thus, maintenance efforts are significantly reduced.

Using multiple base processes for a particular process type: In certain scenarios, the variants of a particular process type may differ to a large degree. To deal with these differences, the process variants can be configured out of different base processes; i.e., each of these base processes then covers a specific class of variants. This minimizes the number of change operations required for configuring process variants and allows to better structure the process landscape (cf. Req. 1). One disadvantage coming with the use of multiple base processes for a particular process type are the higher modeling and maintenance efforts. In particular, inconsistencies and errors will result if a change affects all base processes. Hence, problems are similar to the ones which emerge when modeling variants separately (cf. Sec. 2). Therefore, multiple base processes shall be only used if classes of process variants can be identified that differ to a large degree, i.e. they have no common parts or structures.

5 Designing and Modeling Options

So far, we have considered modeling issues related to base processes. When defining options for a base process several additional issues occur. In particular, the process designer has to decide whether to use first class objects (i.e. options) to describe a particular variant or whether this shall be accomplished using conditional branches within the model of the base process. Options can be created using different modeling methods. Thereby, the granularity of options becomes an issue. In the following we cope with these aspects in more detail.

Modeling variants based on change operations vs. use of conditional branches: When modeling processes and their variants in an approach like Provop one has to decide which control flow alternatives are variant-specific and which ones are common for all process variants. For example, the repair process from Fig. 1a includes one decision point (i.e., conditional branching) to decide whether the vehicle can be repaired at the agreed cost limit or whether the repair shall be omitted. If these two alternatives are modeled as separate process variants, only one variant of the repair process will comprise a repair activity. This is not meaningful as both alternatives are relevant for all repair process variants. In particular, the decision to either omit the Activity `Repair` or to perform it cannot be made before executing Activity `Diagnosis`. Therefore such decisions should be modeled in a conditional branch.

Granularity of options: Provop allows to combine several options to derive a specific process variant (cf. Req. 2 and 3). When defining process variants the designer therefore has to decide how to group change operations into options. Thereby aspects like reuse of options as well as maintainability have to be considered.

As one extreme we can have fine-grained options with only one change operation. 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 of a specific process variant into one option. Such partially redundant options are difficult to reuse, but the configuration of a process variant becomes more transparent and easy to track.

The difference between fine- and coarse-grained options can be demonstrated using the process and its variants from Fig. 1. Fig. 6 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 to high redundancy with respect to the change operations. To support a 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. Doing so, the number of options can be reduced without affecting reuse. Considering this guideline the change operations of our example are grouped into the options shown in the third column of Fig. 6 (and in Fig. 3b): The

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`		

Fig. 6: Granularity of Options

adjustment for Activity `Diagnosis` is always applied in conjunction with a modification of the `Repair` activity. Therefore these two change operations can be grouped into one option. The other change operations are modeled in separate options. As result we obtain three options without any redundancy.

Relations between options: Our case studies have revealed that options are often correlation. Provop considers three types of relations between options: *dependency*, *mutual exclusion*, and *hierarchy* (cf. Req. 3 and 4).

- **Dependency:** If different options are applied conjointly to the base process (e.g. because of semantical dependencies) the user can explicitly define a dependency relation between those options. Dependency relations are directed; i.e., if the relation *"Option 1 depends on Option 2"* has been defined the reverse conclusion (i.e., *Option 2 depends on Option 1*) is not true.
- **Mutual Exclusion:** Mutual exclusion, in turn, is helpful to describe which options must never be used in conjunction when configuring a specific process variant.
- **Hierarchy:** The definition of a hierarchy of options allows for the inheritance of change operations. More precisely, if an option is selected to configure a particular process variant and the option has an ancestor in the option hierarchy the change operations defined in the ancestor options will be applied as well. This reduces the amount of change operations defined in options and structures the options landscape.

Generally, when defining relations between options the designer does not only use one type of relation, but may apply them in combination with each other as well. Thus, Provop supports the combined use of several relations and ensures consistency of a set of relations applied in the same context. For example, it is not allowed to define contradicting relations (e.g., an mutual exclusion between an option and its parental option).

The presented concepts for defining relations between options ease the use of multiple options. Additionally, semantical errors are avoided when configuring a process variant. Therefore, whenever relations are known they should be defined explicitly.

6 Related Work

Though the support of process variants is highly relevant for practice, only few approaches for variant management exist. In particular, there is no comprehensive solution for the adequate modeling of multiple variants within a single process model.

There exist approaches which provide support for the management and retrieval of separately modeled process variants. As an example, [12] allows 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 stored processes, an assumption which does not always hold in practice. Variant search based on process metadata (e.g., the process context) is not considered.

An important area related to variant management is *reference process modeling*. Usually, a reference process has recommending character, covers a family of processes, and can be customized in different ways to meet specific needs. *Configurable* EPCs (C-EPCs), for example, provide support for both the specification and the customization of reference process models [13, 14]. When modeling a reference process, EPC functions (and decision nodes) can be textually annotated to indicate whether they are mandatory or optional. Respective information is then considered when configuring the C-EPCs. This approach is restricted to control flow and does only allow for the configuration of single elements (i.e., it is not possible to mark a complete branch as mandatory or optional in one step). It is also not possible to move or add model elements or to adapt element attributes like in Provop. As compared to reference process models, the basic process in Provop can be modeled without any restriction; i.e., it neither needs to be defined with a specific use case in mind nor does it constitute a recommendation for all processes of a given process type.

Variants are also important in software engineering and fundamental characteristics of software variability have been described [15]. In particular, software variants exist in software architectures and software product lines [16, 17]. In many cases, feature diagrams are used for modeling software systems with varying features. Another contribution stems from the PESOA project [18, 19], 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 followed 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.

7 Summary and Outlook

We have described a methodology using the Provop approach for managing process variants. Provop considers the whole process life cycle by supporting 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 in reusable options, and on the possibility to combine options in a constrained way, necessary adjustments of the basic process can be easily and consistently realized when creating or configuring a process variant.

In future research we will detail Provop concepts and apply it in industrial context. One of the challenges we have to tackle concerns the flexible execution of variants; i.e., to allow for dynamic switches between variants during runtime. Finally, a detailed case study based on a prototype that implements the Provop approach will be conducted.

References

1. IDS Scheer: ARIS Platform Method 7.0. (2006)
2. BOC: The Business Process Management Tool ADONIS. (2007) (in German).
3. IBM: IBM WebSphere Business Modeller, Version 6.1. (2007)
4. Weske, M.: Business Process Management - Concepts, Languages, Architectures. Springer (2007)
5. Lenz, R., Reichert, M.: IT Support for Healthcare Processes - Premises, Challenges, Perspectives. Data and Knowledge Engineering **61**(1) (2007) 39–58
6. 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. (2006) 368–377
7. Hallerbach, A., Bauer, T., Reichert, M.: Managing Process Variants in the Process Life Cycle. In: Proc. 10th Int. Conf. on Enterprise Information Systems. (2008)
8. Hallerbach, A., Bauer, T., Reichert, M.: Context-based Configuration of Process Variants. In: Proc. 3rd Int. Workshop on Context-Aware Business Process Management. (2008)
9. Hallerbach, A., Bauer, T., Reichert, M.: Modeling and Visualization of Process Variants in Provop. In: Proc. of Modellierung, Berlin (2008) (in German).
10. VDA Recommendation 4965 T1: Engineering Change Management (ECM) - Part 1: Engineering Change Request (ECR) Version 1.1 (2005)
11. Li, C., Reichert, M., Wombacher, A.: Mining Process Variants: Goals and Issues. In: IEEE 5th Int'l Conf. on Services Computing (SCC 2008), IEEE Computer Society Press (2008)
12. Lu, R., Sadiq, S.: On Managing Process Variants as an Information Resource. Technical Report No. 464, Uni of Queensland (2006)
13. Rosemann, M., van der Aalst, W.: A Configurable Reference Modeling Language. Information Systems **32** (2007) 1–23
14. Rosa, M.L., Lux, J., Seidel, S., Dumas, M., ter Hofstede, A.: Questionnaire-driven Configuration of Reference Process Models. In: Proc. of the 19th Int. Conf. on Advanced Information Systems Engineering. (2007)
15. Bachmann, F., Bass, L.: Managing Variability in Software Architectures. In: Proc. of 2001 Symp. on Software Reusability, New York, ACM Press (2001) 126–132
16. Halmans, G., Pohl, K.: Communicating the Variability of a Software-Product Family to Customers. Software and System Modeling **2**(1) (2003) 15–36
17. Becker, M., Geyer, L., Gilbert, A., Becker, K.: Comprehensive Variability Modeling to Facilitate Efficient Variability Treatment. In: Proc. 4th Int. Workshop of Product Family Engineering. (2001)
18. 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)
19. Puhlmann, F., Schnieders, A., Weiland, J., Weske, M.: PESOA - Variability Mechanisms for Process Models. Technical Report 17/2005, Hasso-Plattner-Institut, Potsdam (2005)