# Managing Process Variants in the Process Life Cycle

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

[1] Group Research and Advanced Engineering, Daimler AG,
Wilhelm-Runge-Straße, Ulm, German
{Alena_Janice.Buettler, Thomas.tb.Bauer}@Daimler.com
[2] Information Systems Group, University of Twente, The Netherlands
M.U.Reichert@ewi.utwente.nl

**Abstract.** When designing process-aware information systems, often variants of the same process have to be specified. Each variant then constitutes an adjustment of a particular process to specific requirements building the process context. Current Business Process Management (BPM) tools do not adequately support the management of process variants. Usually, the variants have to be kept in separate process models. This leads to huge modeling and maintenance efforts. In particular, more fundamental process changes (e.g., changes of legal regulations) often require the adjustment of all process variants derived from the same process; i.e., the variants have to be adapted separately to meet the new requirements. This redundancy in modeling and adapting process variants is both time consuming and error-prone. This paper presents the Provop approach, which provides a more flexible solution for managing process variants in the process life cycle. In particular, process variants can be configured out of a basic process following an operational approach; i.e., a specific variant is derived from the basic process by applying a set of well-defined change operations to it. Provop provides full process life cycle support and allows for flexible process configuration resulting in a maintainable collection of process variants.

## 1 INTRODUCTION

The flow of activities an organization has to perform to achieve a specific goal is often captured in a process model. Usually, each model implements one process type (e.g., for handling a credit request or travel cost declaration) by describing process activities and their execution constraints, resources needed (e.g., humans or IT systems), and information processed. For creating and managing process models there exist tools like ARIS Business Architect [IDS Scheer AG, 2006] and WBI Modeler [IBM Corporation, 2007].

When modeling processes several objectives are in the focus. As example consider improved process transparency. By the model-based documentation of business processes respective information is provided in a more transparent and unified manner to users. As another advantage process models can be analyzed and simulated resulting in further optimizations of the business processes [Scheer, 1998]. However, modeling, analyzing, and optimizing processes is only one side of the coin. The other is to implement and execute these processes, e.g., based on Workflow Management Systems (WfMS). For this purpose, executable workflow models have to be provided. Based on
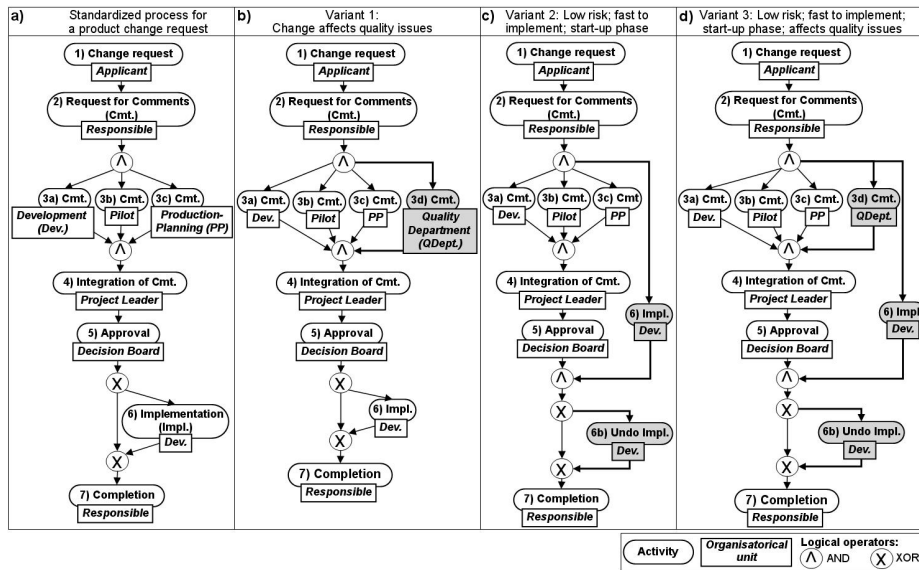
Fig. 1: Variants of a Standardized Product Change Process

such models the WfMS controls the execution of process activities and allocates them to user worklists during runtime [Dumas et al., 2005, Leymann and Roller, 1999, Weske, 2007].

Process support is needed in almost all business domains. Characteristic process examples from the automotive industry include product creation, change management, and release management. All these processes have to be modeled with a specific goal in mind. Depending on the given process context, in addition, different variations of a basic process are needed. Having a closer look at the product creation process, for example, different process variants exist. Thereby, each variant is connected to a particular product type (e.g., car, truck, or bus) with different organisatorical responsibilities and strategic goals, or varying in some other aspects.

Similar considerations can be made for a product change process as depicted in Figure 1a: The process starts with a change request (Activity 1). The person responsible for coordinating changes in the respective domain then requests comments from the departments that might be affected by the change (Activities 2, 3a, 3b, and 3c). After all comments are received an integrated change document is created (Activity 4). This document is then passed to the decision board which either approves the requested change or disapproves it (Activity 5). In case of approval the development department gets the permission to implement the change (Activity 6). Otherwise this step is skipped. The process ends by logging and filing the change request (Activity 7). Depending on the process context, different variations of this process are needed. Figure 1b-1d show examples of three possible process variants: The one depicted in Figure 1b additionally considers quality critical issues; i.e., the quality department is involved in the commenting process. At the model level this is realized by inserting an additional activity (Activity 3d) when compared to the original process from Figure 1a. Figure 1c shows a

process variant for which the change request is fastened. Particularly for changes with low risks and implementation times, which are requested during start-up phase, the development department starts implementing the change without waiting for approval. If the decision board refuses approval later, change implementation will have to be undone. At the model level this can be simply realized by moving Activity 6 from its original position to a position parallel to the commenting activities and by conditionally inserting the Undo activity (Activity 6b). Finally, the variant shown in Figure 1d will be required if the change affects quality critical issues, but can be fastened anyway. This variant constitutes a combination of the two variants from Figure 1b and 1c. Thus, the process inherits all adjustments from these two variants; i.e., an additional comment is requested from the quality department and early implementation of the change (i.e., without waiting for approval) is possible.

In existing approaches, process variants usually have to be defined and kept in separate process models as shown in Figure 1. This results in a huge amount of 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). Finally, there is no support for (semi-)automatically combining existing variants in order to create 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 are increasing over time since more fundamental process changes (e.g., due to new or changed legal regulations) might have to be accomplished for each individual variant. This is both time-consuming and error-prone. As a consequence, process variant models degenerate over time as optimizations are only applied to single variant models without considering the relations to other variants. This, in turn, makes it a hard job for process designers to analyze, compare, and unify business processes. In particular, IT systems providing integrated support for different process variants are difficult to realize.

In this paper we present the Provop (PROcess Variants by OPtions) approach for managing large collections of process variants in one model. The paper is organized as follows: Section 2 discusses major requirements for managing process variants in the process lifecycle. Section 3 presents basic concepts of the Provop approach in detail. In Section 4 we discuss related work. This paper concludes with a summary and an outlook in Section 5.

## 2   REQUIREMENTS

We conducted several case studies in the automotive industry, but also other domains (e.g., healthcare), to elaborate key requirements for the definition, adaptation, and management of process variants. This strong linkage to practice was needed in order to realize a complete and solid approach for process variant management. The requirements we identified are related to different aspects including the modeling of process variants, their linkage to process context, their execution in WfMS, and their continuous optimization to deal with evolving needs; i.e., we have to deal with requirements related to the whole *process life cycle*. The standard process life cycle is depicted in Figure 2.

It consists of three phases, namely the design and modeling of the process, the selection or configuration of a particular process variant, and the deployment of this variant in the runtime environment. The process life cycle can be described as a (feedback) loop of these phases during which a process is continuously optimized and adapted. The main requirements to be met are as follows (cf. Table 1):

**Modeling.** Efforts for modeling process variants should be kept minimal. Therefore, reuse of both process fragments and process models (of the different process variants) has to be supported. In particular, it should be possible to create new 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.

To reduce both maintenance efforts and costs of change, fundamental process changes affecting multiple process variants should be conducted only once. As a consequence all process variants concerned by the respective change should be adapted automatically. Finally, sophisticated visualization support is needed to enable selective views on process variants. This should allow for the comparison of variants as well. In this
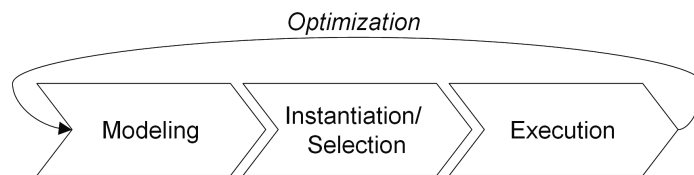


Fig. 2: Process Life Cycle

Table 1: Requirements for Process Variant Support

| Lifecycle Phase 1: Modeling | |
|---|---|
| Req 1.1 | Intuitive modeling of process variants and the relations between them |
| Req 1.2 | Minimized modeling & maintenance efforts |
| Req 1.3 | Easy analysis and comparison of variants |
| **Lifecycle Phase 2: Instantiation** | |
| Req 2.1 | Context-aware configuration or selection of variants |
| Req 2.2 | Consistency of configured variants |
| **Lifecycle Phase 3: Execution** | |
| Req 3.1 | Supporting the execution of variants by WfMS |
| Req 3.2 | Selecting variants at runtime |
| Req 3.3 | Switching variants during runtime to adjust to context changes |
| **Lifecycle Phase 4: Optimization** | |
| Req 4.1 | Capturing best practices in variant design |
| Req 4.2 | Evolving processes without making existing variants obsolete |

context, switching between different visualizations constitutes another requirement.

**Instantiation and Selection.** The selection of a process variant in a particular context should be done automatically. Therefore the specific circumstances (i.e., the process context) in which this selection takes place has to be considered. In particular, an elaborated context-aware variant selection process is required. Another challenge is to ensure consistency and correctness of all selectable process model variants throughout the entire process life cycle.

**Execution.** To execute a process variant, its model has to be interpreted by a workflow engine during runtime. In this context, it is important to keep information about the selected process variant and its relation to the basic process (and other variants) in the runtime system as well. Another challenge is to deal with dynamic changes of the process context. In the context of such changes the conditions under which a particular variant was originally selected might become obsolete. Ideally, the runtime system should allow to dynamically switch process execution from one variant to another if required. Such dynamic variant switches are by far not trivial when considering correctness and consistency issues as well.

**Optimization.** Generally, a large collection of related variants can be derived from a basic process model. In principle, each variant corresponds to a number of adaptations applied to this basic process. Since it is a complex task to decide which process parts shall be captured by the basic process and which ones are variant-specific, related process variants should be analyzed from time to time based on advanced process mining techniques. As a result it might turn out, for example, that it is more favorable to pull up certain variant-specific adaptations to the level of the basic process. Thus, the basic process evolves over time without making defined process variants obsolete.

There exist other requirements addressed by Provop, but not mentioned so far. Examples include the consistency of configured process variants, adequate visualization of process variants for all life cycle phases, and provision of intuitive user interfaces. Due to lack of space we omit respective issues in this paper.

## 3 THE PROVOP APPROACH

This section provides an overview of the Provop approach for process variant management. As Provop supports all phases of the process life cycle, we describe our approach along these phases.

### 3.1 Modeling

**Basic Process.** The basic idea behind Provop is to capture all process variants in a single process model. To achieve this Provop utilizes a major characteristic of process variant models, namely their similarity to the original process model they were derived from. In Provop we denote this original process as *basic process.* This can be both an existing

process model or a newly created one (cf. Figure 1a). Different policies for modeling the basic process are conceivable: On the one hand the basic process can be defined for a specific use case, e.g., the most frequently executed variant of a process family. On the other hand the basic process may be defined without a specific use case in mind.

**Change Operations.** Related variants are logically kept within the model of the basic process. More precisely, the different variants are represented by a set of change operations describing the difference between the basic process model and the respective variant model. The following change operations are provided in this context (cf. Table 2 and Figure 3a): INSERT, DELETE, and MOVE *process fragments* as well as MODIFY *process element attributes*. Each of these change operation types is represented by a special symbol (cf. Table 2). Further, each change operation needs a set of parameters as input for its correct execution. For example, the INSERT operation of "Option 1" in Figure 3b requires the position at which the respective process fragment shall be added to the basic process. In this case, entry node *S* and exit node *E* of the process fragment to be added are mapped to the adjustment points *AND1.out* and *AND2.in* in the basic process model.

**Options.** To define more complex adjustments, multiple change operations can be grouped in a single object called *option*. Thus, an option consists of an unambiguous name and

Table 2: Change Operations in Provop

| 1. INSERT operation | |
|---|---|
| **Purpose** | addition of process elements |
| **Parameters** | process fragment or element to be added; entry/ exit of the fragment to be added; mapping between entry/ exit of the fragment to adjustment points (labeled position in basic process model) |
| **Symbol** | ⬆ |
| **2. DELETE operation** | |
| **Purpose** | removal of process elements |
| **Parameters** | adjustment points to mark entry and exit of a process fragment for deletion (or IDs of single elements) |
| **Symbol** | ⊠ |
| **3. MOVE operation** | |
| **Purpose** | change execution order of activities |
| **Parameters** | process fragment of the basic process marked by adjustment points; target position of the process fragment |
| **Symbol** | → |
| **4. MODIFY operation** | |
| **Purpose** | change attributes of process elements |
| **Parameters** | element ID; attribute name; value to be assigned |
| **Symbol** | ⋀ |

a set of change operations. Figure 3 illustrates this approach taking our example from Figure 1. (Note that activity names are abbreviated by step numbers here.) The standard product change request process from Figure 1a is now defined as basic process. The variants from Figure 1b-1d are described in terms of change operations grouped to options. By applying one of the two options to the basic process the different variants can be derived: The application of "Option 1" from Figure 3b to the basic process model from Figure 3a results in Figure 1b, the application of "Option 2" produces the process model shown in Figure 1c. Provop additionally supports the combined use of these two options to create a third process variant (cf. Req 1.2); i.e. the combination of "Option 1" and "Option 2" leads to the model depicted in Figure 1d.

**Visualization of Options.** To support variant modeling sophisticated visualization concepts are needed (cf. Req 1.3). In particular, the positioning of options relative to the basic process model constitutes a challenge when displaying both the basic process and the options at the same time. As the change operations of a particular option refer to the basic process model, the points of adjustment can be used as anchor for positioning the option. Generally, options can be visualized in several ways. One approach is to show all information of the option as depicted in Figure 3b. Another one is to enable user-defined selection of the information to be visualized.

**Option Relations.** After modeling relevant options, different kind of relations between them can be defined in order to constrain their use (cf. Req 1.1). The relations supported in Provop are as follows: *dependency*, *mutual exclusion*, *execution order constraints*, and *hierarchy*. *Dependency* means that the respective options always are either jointly applied to the basic process or none of them is used when configuring a particular process variant. *Mutual exclusion*, in turn, allows to reduce the possible combinations of options that can be applied to the basic process model. Thus, the configurable process variants can be constrained. Two options mutually exclude each other, for example, if they constitute variations of each other, e.g., both options might add the same activity to the basic process model, but at different positions, thus leading to different variants. As one option might insert an activity whose attributes are changed by a second one, the
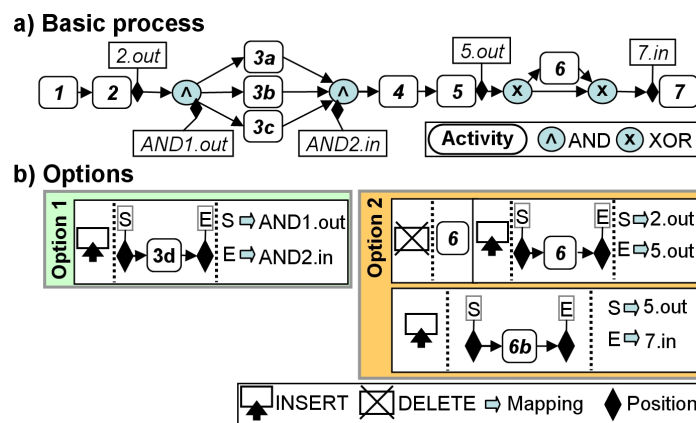


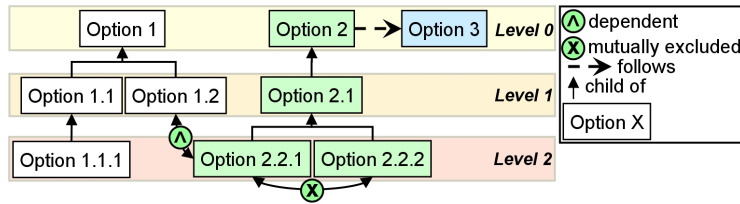Fig. 3: Modeling Process Variants in Provop

Fig. 4: Graphical Visualization of Option Relations

*execution order* of these options becomes crucial. Therefore, Provop allows specifying orders in which options can be applied to the basic process. Finally the *hierarchy* of options constitutes a combination of the relation *dependency* and *execution order*. More precisely, if a child option shall be applied to the basic process model, the corresponding parent options will have to be applied as well. To prevent inconsistencies due to non-determinism parent options are always applied before their child options.

Provop allows to represent the described option relations graphically as depicted in Figure 4: Every relation type uses a particular symbol or arrow; i.e., all relations between options can be represented in a unified and easy to handle manner.

**Context-aware Process Configuration.** As discussed option relations are useful when defining variants (i.e., when a user selects a particular option all dependent options are selected as well, while mutually excluded options are not considered). In addition, Provop supports context-aware process configuration; i.e., it allows for the configuration of a process variant by applying only those options relevant in a given application or process context (cf. Req 2.1). In a first step the process context has to be defined by utilizing *context variables* with a given range of value. Provop distinguishes between static and dynamic context variables. *Static context variables* are set once and their value is then fixed throughout process execution (e.g., product type). The value of *dynamic context variables*, in turn, may change during process execution (e.g., development phase). As this might invalidate the conditions based on which a process variant was configured, Provop enables dynamic variant changes as well; i.e., we allow to switch the execution of a process instance from one variant to another in order to adopt to context changes. An example of a process context definition is given in Figure 5a. The context variables introduced in Figure 1 are listed with their range of values and their mode (static/ dynamic).

**Process Context Constraints.** Sometimes there are constraints describing a relation between particular context variables. For example, if a requested product change is of high costs, its risks will be high as well. As these relations can get very complex, Provop allows for the definition of formal rules following an `IF THEN ELSE` logic (cf. Figure 5b). The relations between context variables can be represented graphically. Constraints are represented by arrows connecting the context variables and leading to a context graph (cf. Figure 5c).

**Context Rules.** A process context is defined to connect options with process variant configurations. For this purpose, context rules are defined and assigned to the options as depicted in Figure 6. Here, "Option 1" is relevant if the requested change affects quality issues (i.e., *quality = high*). In turn, "Option 2" is relevant for product changes

of low risks and implementation time. Further, it is constrained to product changes in the start-up phase of product development.

### 3.2 Selection and Instantiation

In the selection and instantiation phase the basic process model, the defined options, and the context model are used to configure the models of the different variants. A single variant is created by applying a number of options and their related operations to the basic process.

**Step 1: Select Options.** When configuring a process variant the relevant options are identified either explicitly or implicitly. In the former case the user directly selects the options manually from a given list. In the latter case the options that are relevant for configuring a particular variant are selected implicitly based on the current values of the context variables; i.e., an option will be selected if all context rules associated with it evaluate to "true".

**Step 2: Evaluate Relations.** After a set of options is selected their relations are checked. Extensions of the option set will have to be made if dependent options are missing. It is also possible that the set of options selected so far contains mutually excluding options. In this case the user is notified about the inconsistency and has to remove one of the conflicting options. In summary, option relations are considered to ensure process consistency.

**Step 3: Apply Options.** After defining and evaluating the relevant set of options, the related change operations are applied to the model of the basic process. First, options with static context variables are applied resulting in a process model of a particular process variant. Second, options with dynamic context variables are applied. The latter results in a process model representing a set of variants. The decision which variant is chosen then depends on the dynamic context to be defined.

**Step 4: Check for Consistency.** The application of several options in combination with each other constitutes a challenge. In certain cases, change operations might be

**a)** Context definition

| Context variables | Range of Values | Static/ dynamic |
|---|---|---|
| Quality | low, medium, high | static |
| Risk | low, medium, high | static |
| Time | low, medium, high | static |
| Phase | advanced development, development, start-up, production | dynamic |

**b)** Relations

IF Phase = production THAN Risk = high

IF Quality = high THAN Time in {medium, high}

IF Time = high and Quality = high THAN Risk = high

**c)** Context graph



Fig. 5: Context modeling in Provop

```
Option 1:            Option 2:
CONTEXT_RULE:        CONTEXT_RULE:
IF quality = high    IF risk = low
                     AND implementation time = low
                     AND phase = start-up
```

Fig. 6: Context rules

redundant or even conflicting; i.e., the application of all options then might result in a variant model with deadlocks or data inconsistencies. To avoid the latter comprehensive consistency checks are provided by Provop (cf. Req 2.2).

### 3.3 Deployment and Execution

After the selection and instantiation phase the resulting variant model needs to be translated into an executable workflow model (cf. Req 3.1), e.g., specified with WS-BPEL [OASIS, 2007]. Common problems emerging in this context are GUI assignments, distinction between human and automated tasks, or choice of the right level of granularity for process models. In Provop we are focusing on problems arising with variant management and their resolution. For several reasons we retain the information about options and contexts created in the previous phases in the runtime system as well. One particular reason for this is the presence of dynamic context variables, which necessitate the ability to switch between variants during runtime (cf. Req 3.2 and 3.3). Due to lack of space we omit further details here.

### 3.4 Optimization

Provop allows to evolve and optimize the basic process without making the defined options obsolete (cf. Req 4.2). In particular, the modeled options are checked against the new basic process model. If an option is affected by changes of the basic process, e.g., because an adjustment point has been moved to a new position, this option will be updated accordingly. In some cases, an option might be omitted, because its changes have been transferred to the basic process as "best practice" (cf. Req 4.1).

## 4 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 are approaches which provide support for the management and retrieval of separately modeled process variants. As an example take the work done by [Lu and Sadiq, 2006]. It allows storing, managing and querying large collections of process variants within a process repository. Graph-based search techniques are used in order to retrieve those process variants which are similar to a user-defined process fragment (i.e., the query is represented as a process graph). Obviously, this approach requires profound knowledge about the stored process structures, an assumption which does not always
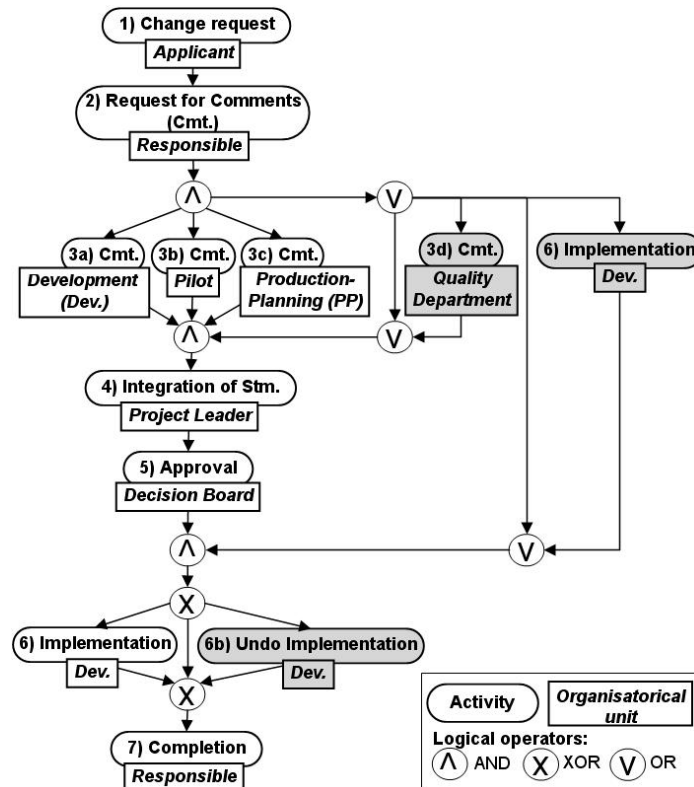
Fig. 7: Naive Approach for Variant Modeling

hold in practice. Variant search based on process metadata (e.g., the process context) is not considered.

A straightforward approach frequently applied in practice is to capture multiple variants within a single process model, but without treating the variants as first class objects as in Provop [IDS Scheer AG, 2006, IBM Corporation, 2007]. Usually, specifying all variants in one process model results in huge models, which are difficult to comprehend and costly to maintain. As example consider Figure 7 which shows the change request process from Figure 1a together with its different variants as depicted in Figure 1b-1d. Thereby, every execution path in the model represents a particular variant with the branching conditions indicating which variant to be selected during runtime; i.e., the relation between variants and process context is captured by these branching conditions. Following this naive approach, the resulting variants are to a large degree hidden within the process logic. As "normal" branching conditions cannot be distinguished from the ones representing contextual conditions (for variant selection), no views for particular process variants can be created.

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 [Schütte, 1997]. *Con-*

*figurable Event-Process-Chains* (C-EPCs), for example, provide support for both the specification and the customization of reference process models [Rosemann and van der Aalst, 2007, Rosa et al., 2007]. 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. As one drawback 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). It is also not possible to move or add model elements or to adapt element attributes like we do in Provop. As compared to reference process models, the basic process in Provop can be modeled without any restriction; i.e., it needs not to be defined with a specific use case in mind nor it constitutes 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 [Bachmann and Bass, 2001]. In particular, software variants exist in software architectures and software product lines [Halmans and Pohl, 2003, Becker et al., 2001]. In many cases feature diagrams are used for modeling software systems with varying features. Another contribution in this context stems from the PESOA project [Bayer et al., 2005, Puhlmann et al., 2005], which provides basic concepts for variant modeling based on UML. More precisely, different variability techniques like inheritance, parameterization, and extension points are provided and can be used when describing UML models of different type. 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.

## 5   SUMMARY AND OUTLOOK

We have described the Provop approach for managing process variants. Provop considers the whole process life cycle by supporting variants in all life cycle 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 correctly and easily realized when creating or configuring a process variant. Provop allows representing the objects and data needed in this context in a compact and efficient manner. Further, it offers advanced tool support for visualizing and comparing process variants. Finally, Provop allows for the dynamic configuration of process variants based on the given process context; i.e., the change operations needed to create the respective process variant are dynamically selected based on contextual information. Note that this also allows to dynamically switch between different variants during runtime. Altogether, developing and maintaining process variants in an integrated way becomes much easier with the techniques introduced in this paper.

In future research we will detail the Provop approach. Of the challenges we have to tackle one concerns the correct combination of options when creating a variant. The set

of options to be applied to the basic process to create a specific process variant might consist of options with dissent and redundant change operations (e.g., two options add the same activity to a process schema, but at different positions at the basic process). Sophisticated techniques are needed to prevent errors (e.g., deadlocks) or other consistency problems (e.g., concerning data consistency) due to such conflicting changes.

# References

[Bachmann and Bass, 2001] Bachmann, F. and Bass, L. (2001). Managing variability in software architectures. In *Proc. of the 2001 Symp. on Software Reusability*, pages 126–132, New York. ACM Press.

[Bayer et al., 2005] Bayer, J., Buhl, W., Giese, C., Lehner, T., Ocampo, A., Puhlmann, F., Richter, E., Schnieders, A., Weiland, J., and Weske, M. (2005). PESOA - process family engineering - modeling variant-rich processes. Technical Report 18/2005, Hasso-Plattner-Institut, Potsdam.

[Becker et al., 2001] Becker, M., Geyer, L., Gilbert, A., and Becker, K. (2001). Comprehensive variability modeling to facilitate efficient variability treatment. In *Proc. of the 4th Int. Workshop od Product Family Engineering*.

[Dumas et al., 2005] Dumas, M., van der Aalst, W., and ter Hofstede, A., editors (2005). *Process-aware information systems*. Wiley, Department of Information Studies, Uni of California, Los Angeles Los Angeles, CA 90095-1520.

[Halmans and Pohl, 2003] Halmans, G. and Pohl, K. (2003). Communicating the variability of a software-product family to customers. *Software and System Modeling*, 2(1):15–36.

[IBM Corporation, 2007] IBM Corporation (2007). *IBM WebSphere Business Modeller, Version 6.0.2*. IBM Corporation.

[IDS Scheer AG, 2006] IDS Scheer AG (2006). *ARIS Platform Method ARIS 7.0*.

[Leymann and Roller, 1999] Leymann, F. and Roller, D. (1999). *Production Workflow: Concepts and Techniques*. Prentice Hall PTR.

[Lu and Sadiq, 2006] Lu, R. and Sadiq, S. (2006). On managing process variants as an information resource. Technical Report No. 464, School of Information Technology & Electrical Engineering and University of Queensland, Brisbane.

[OASIS, 2007] OASIS (2007). *Web Services Business Process Execution Language Version 2.0*. OASIS.

[Puhlmann et al., 2005] Puhlmann, F., Schnieders, A., Weiland, J., and Weske, M. (2005). PESOA - variability mechanisms for process models. Technical Report 17/2005, Hasso-Plattner-Institut, Potsdam.

[Rosa et al., 2007] Rosa, M. L., Lux, J., Seidel, S., Dumas, M., and ter Hofstede, A. (2007). Questionnaire-driven configuration of reference process models. In *Proc. of CAISE 2007*.

[Rosemann and van der Aalst, 2007] Rosemann, M. and van der Aalst, W. (2007). A configurable reference modelling lanugage. *Information Systems*, 32:1–23.

[Scheer, 1998] Scheer, A.-W. (1998). *ARIS- Modellierungsmethoden, Metamodelle, Anwendungen*. Springer-Verlag. Vierte Auflage.

[Schütte, 1997] Schütte, R. (1997). *Grundsätze ordnungsgemäßer Referenzmodellierung: Konstruktion konfigurations- und anpassungsorientierter Modelle*. PhD thesis, Uni Münster.

[Weske, 2007] Weske, M. (2007). *Business Process Management - Concepts, Languages, Architectures*. Springer.