Configuration and Management of Process Variants

Alena Hallerbach, Thomas Bauer

Group Research and Advanced Engineering, Daimler AG, Ulm, Germany,
{alena.hallerbach,thomas.tb.bauer}@daimler.com

Manfred Reichert

Database and Information Systems Group, University of Ulm, Germany,
manfred.reichert@uni-ulm.de

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

Introduction

Process support is required in almost all business domains (Mutschler et al. 2008). As examples consider healthcare (Lenz and Reichert 2007), au-

tomotive engineering (Müller et al. 2006), and public administration (Becker et al. 2007). Characteristic process examples from the automotive industry, for instance, include product change management (VDA 2005), release management (Müller et al. 2006), and product creation (see below).

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

In this paper we refer to the service process handling vehicle repair in a garage (cf. Figure 1 a). Basically this process works as follows: It starts with the reception of a vehicle. After a diagnosis is made, the vehicle is repaired (if necessary). During diagnosis and repair, the vehicle is maintained; e.g., oil and wiping water may be checked and refilled. The process completes when handing the repaired and maintained vehicle back to the customer. Depending on the process context different variants of this process are required, whereas the context is described by country-specific, garage-specific, and vehicle-type-specific variables. In our case studies we have identified hundreds of such variants and we have learned that existing process modeling tools do not provide sophisticated support for modeling and maintaining such large number of process variants.

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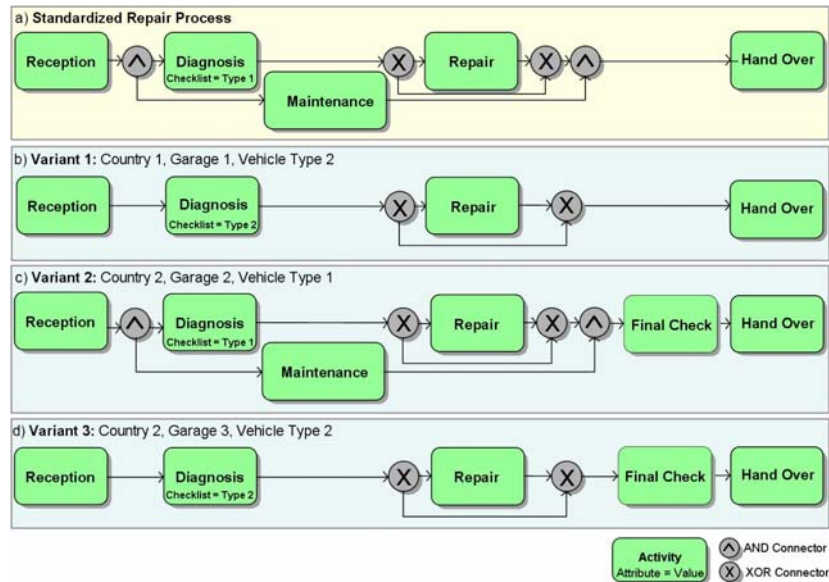
Figure 1: Variants of a standardized vehicle repair process (simplified view).

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

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

Dealing with Process Variants in Existing BPM Tools

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

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

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

Single-Model Approach: Another approach, frequently applied in practice, is to capture multiple variants in one single model using conditional branchings (i.e., XOR-/OR-Splits). As example consider Figure 2, which shows the repair process together with different variants (cf. Figure 1 a-d). Each execution path in the model represents a particular variant. Thereby, branching conditions indicate which path belongs to which variant.
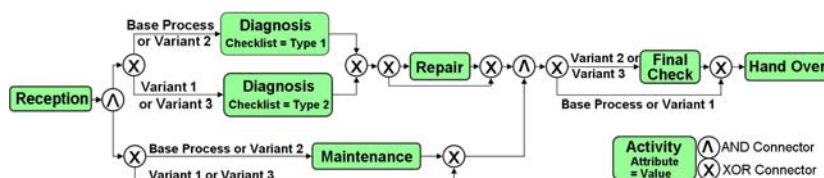

Figure 2: Process Variants realized by Conditional Branches.

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

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

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

Requirements

We conducted several case studies in the automotive industry (Müller et al. 2006, VDA 2005), but also in other domains like healthcare (Lenz and Reichert 2007), to elaborate key requirements for the configuration, adaptation, and management of process variants. This strong linkage to practice was needed in order to realize a complete and solid approach for process variant management. The requirements we identified are related to different aspects including the modeling of process variants, their linkage to process context and context-driven configuration, their execution in workflow management systems (WfMS), and their continuous optimization to deal with evolving needs; i.e., we have to deal with requirements related to the whole process life cycle (Hallerbach et al. 2008c, Weber et al. 2006, Weber et al. 2009). The standard process life cycle is depicted in Figure 3. It consists of three phases, namely the design and modeling of the process, the creation of a particular process variant, and the deployment of this variant in a runtime environment. The process life cycle can be described as a (feedback) loop of these phases during which a process is continuously optimized and adapted (Weber et al. 2006, Weber et al. 2009). The major requirements to be met are described in the following.
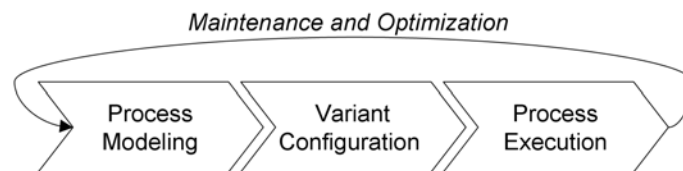

Figure 3: Process Life Cycle.

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

Variant Configuration. The configuration of a process variant (i.e., its derivation from a given master or base process) should be done automatically if possible. Therefore, the specific circumstances (i.e., the *process context*) under which this configuration takes place have to be considered. In particular, an elaborated procedure for context-aware, automated variant con-

figuration is required. At the same time consistency and correctness of the configured process variants have to be ensured throughout the entire process life cycle.

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

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

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

The Provop Approach

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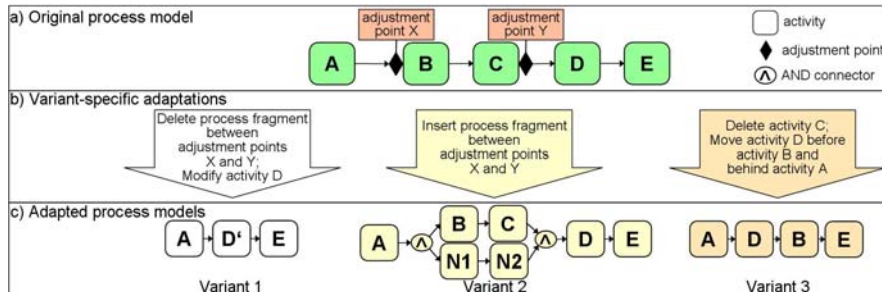
Figure 4: Variant configuration by process model adaptation

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

Modeling

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

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

– *Policy 1 (Standard Process):* Here, the base process represents a domain-specific standard or reference process. In the automotive domain, for example, such reference processes exist for Engineering Change Management. Usually, a standard process has to be adjusted to meet specific requirements; i.e., it must be possible to derive variants from it. Provop assists designers in correctly defining the necessary adjustments when configuring a process variant out of the reference process.
– *Policy 2 (Most Frequently Used Process):* If one process variant is used more frequently than others, it can be chosen as base process. This reduces configuration efforts in terms of the number of processes for which adjustments become necessary. Provop maintains statistics on the use of process variants to enable Policy 2. Generally, Policy 2

does not ensure that the average number of change operations needed to configure the variants out of the base process becomes minimal.

– *Policy 3 (Minimal Average Distance):* When applying change mining to a collection of variants we can derive a base model such that average distance between this model and its variants (i.e., the number of high-level operations needed to transform the base process into the process variant) becomes minimal (Li et al. 2008). Thus, configuration efforts can be reduced accordingly. For mining process variants, we utilize algorithms we developed in the MinAdept project (Li et al. 2008a).

– *Policy 4 (Superset of all Process Variants):* The base process is created by merging all variants into one process model using conditional branchings; i.e., the base process realizes a "superset" of all relevant variants. Consequently, every element that is part of at least one variant belongs to the base process as well. When deriving process variants, therefore, only DELETE operations have to be applied.

– *Policy 5 (Intersection of all Process Variants):* The base process comprises only those elements that are part of all variants; i.e., the base process realizes a kind of "intersection" of relevant variants. Therefore, the base process covers the identical elements of the process variants. When deriving process variants no DELETE operations have to be performed, but elements may have to be moved, modified or inserted.

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

Change Operations: A base process can be adjusted in different ways to configure a specific variant. Provop supports the following adaptation patterns: INSERT, DELETE, and MOVE process fragments, and MODIFY process element attributes. Thereby, fragments constitute connected process subgraphs (including single activity nodes and edges respectively), which not necessarily have a single entry and single exit. To refer to fragments and elements of the base process within such change operations we use *adjustment points*, which correspond to the entry or exit of an activity or connector node (e.g., split and join nodes) of the base pro-

cess.[1] Adjustment points are labeled with unique names. As example consider "adjustment point X" in Figure 4, which corresponds to the entry of activity B.

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

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

1. INSERT-Operation	
Symbol	
Purpose	Addition of *process fragments* (A process fragment consists of at least one process element, e.g., activity nodes or control edges).
Parameters	• Process fragment to be added with entries and exits marked by adjustment points • Target position of the process fragment within the base process, marked by adjustment points for entries and exits • 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)
2. DELETE-Operation	
Symbol	
Purpose	Removal of process elements
Parameters	• Process fragment to be deleted with entries and exits marked by adjustment points • Alternatively: deleting single elements by referring to their ID
3. MOVE-Operation	
Symbol	
Purpose	Change execution order of activities
Parameters	• Process fragment to be moved with entries and exits marked by adjustment points • Target position of the process fragment marked by adjustment points
4. MODIFY-Operation	
Symbol	

[1] If only single elements are affected by a particular change operation their process element IDs may be used alternatively.

Purpose	Change attributes of process elements
Parameters	• Element ID • Attribute name • Value to be assigned

Grouping Change Operations into Options: 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*. This is useful, for example, if the same change operations are always applied in conjunction with each other when configuring certain variants. Think of, for example, the handling of a medical examination in the radiology unit of a hospital. While for ambulant patients no transport between ward and radiology room is required, abasic patients first have to be transferred from the ward to the radiology unit and later back to the ward. To capture the latter variant we need to add two activities at different positions of the respective base process. This can be achieved by defining the two insert operations and grouping them in one option.

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

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

When defining relations between options, generally, the designer does not only use one relation type, but may apply them in combination with each other as well. Provop allows for the combined use of multiple relations and ensures consistency of a set of relations applied in a given context. For ex-

ample, contradictory relations (e.g., a mutual exclusion between an option and its parental option) must not be applied. Due to lack of space, we omit further details on how such contradicting constraints can be identified.

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

Context Model: Provop allows for context-aware process configurations; i.e., it allows for the configuration of a process variant by applying only those options relevant in the given *process context* (Hallerbach et al 2008b). This, in turn, necessitates a model capturing the process context. In Provop, such context model comprises a set of *context variables*. Each context variable represents one specific dimension of the process context, and is defined by a name and value range. Table 2 shows an example of the context model defined for the vehicle repair process from Figure 1. The depicted context variables do not only differ in their names and range of values, but also in another important aspect. While some context variables are defined as *static,* others are classified as *dynamic*. For example, the value of the context variable *Workload* is raised or lowered from time to time according to the current workload of the garage (e.g., switching from "medium" to "high" if many new repair orders emerge at the same time). Thus, this variable is of dynamic nature, as its value may change during process execution. The context variable *Vehicle Type*, in turn, is static as the vehicle type is set once and does not change during the repair process.

Table 2: Context model of a vehicle repair process

Variable name	Range of Values	Behaviour
Vehicle Type	Type 1, Type 2, Type 3, Type 4	static
Maintenance	Yes, No	static
Security level	low, medium, high	static
Workload	low, medium, high	dynamic

Variant Configuration

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

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

A more sophisticated approach is to select relevant options based on contextual knowledge. Rather than mapping already configured process variants to a context description, *context-aware process configuration* allows for the combination of the concepts provided by options and context models. In Provop, this linkage is realized by the use of *context rules*. Such rules, can be assigned to the options and make use of the defined *context model*. Regarding a given context, all options whose context rules evaluate to `true`, are applied to the base process and therefore determine the respective variant. As special case, the base process itself may serve as variant (i.e., no option is applied). In Step 3 we describe in which order the selected options are applied to the base process.

Figure 5 illustrates how the three variants of the repair process (cf. Figure 1) are captured in Provop: The standardized process of Figure 1a is defined as the base process out of which the variants are configured. This base process contains several adjustment points (e.g. "Start Maintenance" at the entry of activity *Maintenance*). As mentioned, adjustment points may be referred to by options and their change operations. Furthermore, Figure 5b depicts three options: Option 1 performs a modification of activity *Diagnosis*. It will be applied if the type of the vehicle is of value *Type 2*. Option 2, in turn, will delete the maintenance activity if no maintenance of the vehicle is requested. Finally, Option 3 inserts a final security check
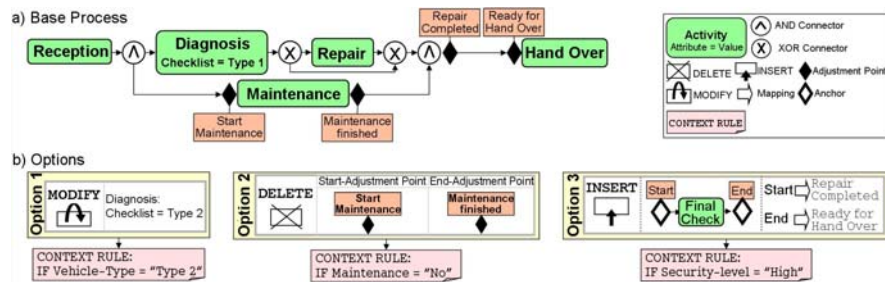

Figure 5: Example of context dependent options.

activity in case of high security levels. The variants of Figure 1b-d can now be configured by applying a subset of these options to the base process. For example, if the context of a process variant is defined by the expression "*Vehicle-Type = Type* 2 AND *Maintenance = No* AND *Security-Level = Low*" Options 1 and 2 will be applied resulting in Variant 1 (cf. Figure 1b).

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

Step 3: Determine the order in which options shall be applied. Generally, selected options have to be applied in sequence; i.e., their order has to be specified when configuring a variant. A naïve approach would be to sort these options in the order they were created; e.g., by making use of their creation time stamps. Obviously, this approach will only make sense if the options and their change operations are commutative. Otherwise, unintended and inconsistent variant models can result, particularly when applying options in the wrong order. Figure 6 shows an example: After applying Option 1 to the base process an intermediate model is derived with activity D and adjustment point Y being deleted.[2] This model is now used as "reference model" for applying Option 2. In the present case, Option 2 cannot be applied, as the adjustment point Y it refers to was deleted when applying Option 1. In order to avoid such inconsistencies, Provop allows defining the order in which selected options shall be applied. Furthermore, wrong option sequences, resulting in erroneous variant models afterwards, are excluded based on well-defined correctness criteria (see Step 5). Final-

---

[2] Note that this example indicates that we need more advanced change support considering the special semantics of adjustment points. Generally, the user should be able to define whether adjustment points may be deleted when applying certain change operations or shall be kept in the intermediate model. In the latter case the deleted activities and nodes respectively are replaced by silent activities without associated actions. Generally, silent activities and adjustment points are removed after application of all selected options.
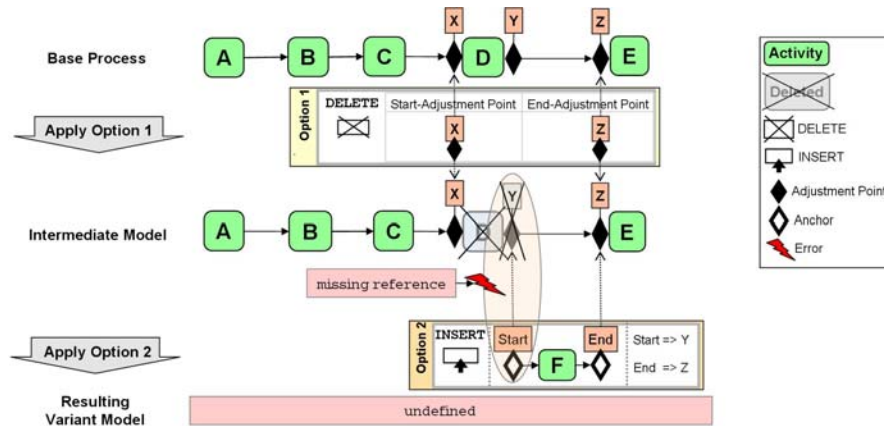
Figure 6: Syntactical error after applying options in wrong order.

ly, by evaluating predefined sequencing constraints a correct application order can be determined.

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

Step 5: Checking consistency. The variant models resulting from the sketched configuration procedure are supposed to be executed in the process enactment phase. Therefore, consistency and correctness of the models have to be guaranteed. In addition to the already described constraint-based selection approach (cf. Step 2), Provop validates the resulting models by checking the consistency and correctness of data and control flow. Unlike other variant configuration approaches (Aalst et al 2008), Provop does not necessarily require a consistent and correct base process as starting point when configuring variants. This follows from the above described policies for defining the base process. Assume, for example, a base process being defined as intersection of its variants. If two variants

---

[3] For a formal semantics of respective change patterns we refer to (Rinderle-Ma et al. 2008).

have different activities to write a data object, read by a common activity, the base process would only contain the reading activity and thus be inconsistent in terms of data flow. Of course, Provop excludes such flaws for the configured variant models.

Deployment and Execution

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

One major aspect concerns the *context-aware configuration* of the different variants. To also capture *context changes* during process instance execution, Provop supports *dynamic context variables*; i.e., variables whose values may change during process execution. When using dynamic context variables for defining a context rule of an option, the decision whether to apply the corresponding change operations or not has to be made at runtime. As a consequence, the respective process variant either cannot be completely configured when creating the process instance or it has to be reconfigured during runtime. To allow for the dynamic reconfiguration of a process instance of a variant model, Provop supports *variant branches*. Basic idea is to encapsulate the adjustments of single options within these variant branches. The split condition at a variant branching corresponds to the context rule of the option. Whenever process execution reaches a variant branch, the current context is evaluated. If the split condition evaluates to `true` the variant branch will be executed, i.e., the change operations will be applied to the base process. Otherwise, the variant branch is skipped and therefore all adjustments of the option are ignored. Provop ensures the constraints regarding the use of options in the context of such dynamic re-configurations as well. However, the handling of respective correctness issues is outside the scope of this chapter.
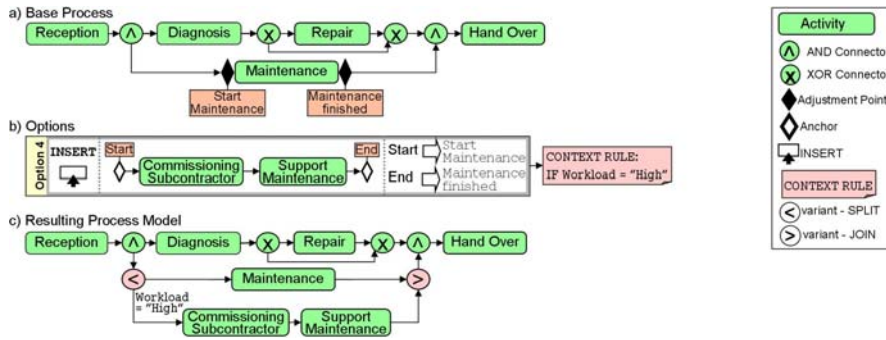
Figure 7: Dynamic configuration of process variants.

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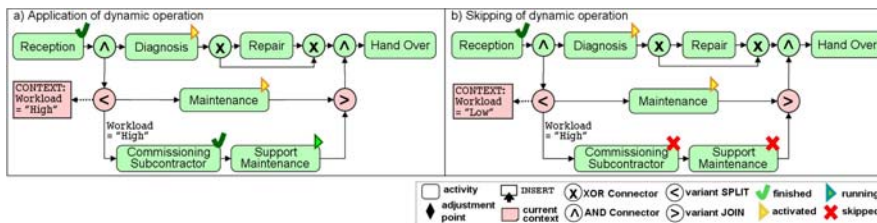

Figure 8: Determine variant at runtime.

Maintenance and Optimization

When evolving base processes in Provop (e.g. due to organizational optimization efforts) all related process variants (i.e. their models) are recon-

---

[4] Note, that every change operation supported by Provop requires specific considerations here.

figured automatically. Thus, maintenance efforts can be significantly reduced. However, evolving and optimizing the base process may affect existing options, for example, when referred adjustment points are moved to a new position or are even deleted. Such problems are detected in Provop; e.g., by checking whether the definitions of existing options are affected by the adaptations of the base process model. Furthermore, solving those conflicts is largely automated.

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 a large number of variants based on a common master process model.

There exist approaches which provide support for the management and retrieval of separately modeled process variants (i.e., optimizations of the multi-model approach). As example, (Lu and Sadiq 2006) 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 the structure of 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 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 (Rosemann and Aalst 2007, Rosa et al. 2007). When modeling a reference process, EPC functions (and decision nodes) can be annotated to indicate whether they are mandatory or optional. Respective information is considered when configuring the C-EPCs. A similar approach is presented in (Gottschalk et al. 2007). 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).

In principle, respective approaches constitute optimizations of the *single model approach* introduced at the beginning of this chapter. 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 process model. Basically, the provided configuration support corresponds to the one of Policy 4 where the chosen base process (i.e., reference process) constitutes the superset of all process variants. Obviously, in this specific scenario only delete or optional delete operations (i.e., dynamic delete operations in Provop) become necessary in order to configure a particular process variant out of a reference process model. However, Policy 4 is only one out of several configuration policies supported by Provop; i.e., a base process can be defined in a more flexible way.

Different work exits on how specialization can be applied to deal with process model variability taking advantage of the generative power of a specialization hierarchy (Wyner et al. 2003; van der Aalst 2002). In the context of the MIT Process Handbook, for example, (Wyner et al. 2003) shows 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 result in process specializations when being applied to a particular model. Similarly, (van der Aalst 2002) discusses transformation rules to define specialization for process models based on Petri Nets. Finally, (Wyner et al. 2003) 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, Provop provides comprehensive support for the 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 (Bachmann and Bass 2001). In particular, software variants exist in software architectures and software product lines (Becker et al. 2001, Halmans and Pohl 2003). In many cases, feature diagrams are used for modeling software systems with varying features. A similar approach is offered by the so-called plus-minus-lists known from variant management in bill-of-materials. Correctness issues are not considered in both cases.

Another contribution stems from the PESOA project (Bayer et al. 2005, Puhlmann et al. 2005), which provides basic concepts for variant modeling based on UML. More precisely, different variability techniques like inheritance, parameterization, and extension points are provided and can be

used when describing UML models. As opposed to PESOA, the operational approach enabled by Provop provides a more powerful instrument for describing variance in a uniform and easy manner; i.e., no distinction between different variability mechanisms is required.

Finally, (La Rosa, 2008) presents an approach which goes beyond control flow and extends business process configuration to roles and objects.

Summary and Outlook

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

In future research we will apply Provop in industrial context. One of the challenges we have to tackle concerns flexible execution of variants; i.e., to allow for dynamic switches between variants during runtime. Finally, a detailed case study based on a prototype implementing the Provop approach will be conducted. This prototype is based on the ARIS tool utilizing the programming interface provided by ARIS (IDS Scheer 2007).

References

van der Aalst, W., Dumas, M., Gottschalk, F., ter Hofstede, A.H.M., La Rosa, M. and Mendling, J. (2008): Correctness-Preserving Configuration of Business Process Models. In: Proc. Fundamental Approaches to Software Engineering, Budapest, LNCS 4961, pp. 46-61.

van der Aalst, W., and Basten, T. (2002): Inheritance of Workflows: An Approach to Tackling Problems Related to Change. Theoretical Computer Science, 270(1-2):125-203.

Bachmann, F. and Bass, L. (2001): Managing Variability in Software Architectures. In: Proc. Symposium on Software Reusability, New York, ACM Press, pp. 126–132.

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. TR 18/2005, Hasso-Plattner-Institute, Potsdam, Germany.

Becker, M., Geyer, L., Gilbert, A. and Becker, K. (2001): Comprehensive Variability Modeling to Facilitate Efficient Variability Treatment. In: Proc. 4th Int. Workshop of Product Family Engineering (PFE'01), Bilbao, Spain, LNCS 2290, pp.294-303.

Becker, J., Lis, L., Pfeiffer and D., Räckers, M. (2007): A Process Modeling Language for the Public Sector – the PICTURE Approach. In: Wybrane Problemy Elektronicznej Gospodarki, pp. 271-281.

Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H. and la Rosa, M. (2007): Configurable Workflow Models. In: Int'l J of Coop Inf. Systems (IJCIS), 17(2):177-221.

Hallerbach, A., Bauer, T. and Reichert, M. (2008 a): Modellierung und Darstellung von Prozessvarianten in Provop. In: Proc. Modellierung'08, Berlin, Germany, LNI P-127, pp. 41-56. (in German)

Hallerbach, A., Bauer, T. and Reichert, M. (2008 b): Context-based Configuration of Process Variants. In: Proc. 3rd Int. Workshop on Context-Aware Business Process Management (TCoB'08), Barcelona, Spain, pp. 31-40.

Hallerbach, A., Bauer, T. and Reichert, M. (2008 c): Managing Process Variants in the Process Life Cycle. In: Proc. 10th Int. Conf. on Enterprise Information Systems (ICEIS'08), Barcelona, Spain, pp. 154-161.

Hallerbach, A., Bauer, T. and Reichert, M. (2008 d): Issues in Modeling Process Variants with Provop. In: Proc. BPM'08 workshops, Milan, Italy.

Hallerbach, A., Bauer, T. and Reichert, M. (2008 e) Anforderungen an die Modellierung und Ausführung von Prozessvarianten. Datenbank Spektrum, 24:48-58.

Halmans, G. and Pohl, K. (2003): Communicating the Variability of a Software-Product Family to Customers. Software and System Modeling, 2(1):15–36.

IDS Scheer (2008): ARIS Platform Method 7.1.

La Rosa, M., Lux, J., Seidel, S., Dumas, M. and ter Hofstede, A.H.M. (2007): Questionnaire-driven Configuration of Reference Process Models. In: Proc. of the 19th Int. Conf. on Advanced Information Sys. Engineering, Trondheim, Norway, LNCS 4495, pp. 424-438.

La Rosa, M., Dumas, M., ter Hofstede, A.H.M., Mendling, J. and Gottschalk, F. (2008): Beyond Control-Flow: Extending Business Process Configuration to Roles and Objects. Proc. ER'08, pp. 199-215.

Lenz, R. and Reichert, M. (2007): IT Support for Healthcare Processes - Premises, Challenges, Perspectives. Data and Knowledge Engineering, 61(1): 39–58.

Li, C., Reichert, M. and Wombacher, A. (2008): Mining Process Variants: Goals and Issues. In: IEEE 5th Int'l Conf. on Services Computing (SCC'08), pp. 573-576.

Li, C., Reichert, M. and Wombacher, A. (2008 a): Discovering Reference Process Models by Mining Process Variants. In: Proc. 6th Int'l Conf. on Web Services (ICWS'08), Beijing, China, IEEE Computer Society Press, pp. 45-53.

Lu, R. and Sadiq, S. (2006): On Managing Process Variants as an Information Resource. Technical Report No. 464, University of Queensland, Australia.

Müller, D., Herbst, J., Hammori, M. and Reichert, M. (2006): IT Support for Release Management Processes in the Automotive Industry. In: Proc. 4th Int. Conf. on Business Process Management (BPM'06), Vienna, Austria, LNCS 4102, pp. 368–377.

Mutschler, B., Reichert, M. and Bumiller, J (2008): Unleashing the Effectiveness of Process-oriented Information Systems: Problem Analysis, Critical Success Factors and Implications. IEEE Transactions on Systems, Man, and Cybermetics (Part C), 38(3):280-291.

Puhlmann, F., Schnieders, A., Weiland, J. and Weske, M. (2005): PESOA - Variability Mechanisms for Process Models. TR 17/2005, Hasso-Plattner-Institute, Potsdam, Germany.

Rinderle-Ma, S., Reichert, M. and Weber, B. (2008): On the Formal Semantics of Change Patterns in Process-aware Information Systems. In: Proc. 27th Int'l Conf. on Conceptual Modeling (ER'08), Barcelona, Spain, LNCS 5231, pp.279-293.

Rosemann, M. and van der Aalst, W. (2007): A Configurable Reference Modeling Language. Information Systems, 32:1–23.

VDA (2005): Engineering Change Management (ECM) - Part 1: Engineering Change Request (ECR) Version 1.1, Recommendation 4965 T1.

Weber, B., Reichert, M., Rinderle, S. and Wild, W. (2006): Towards a Framework for the Agile Mining of Business Processes. In: BPM'05 Workshop Proceedings, LNCS 3812, pp.191-202.

Weber, B., Rinderle, S. and Reichert, M. (2007): Change Patterns and Change Support Features in Process-Aware Information Systems. In: Proc. 11th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'07), Trondheim, Norway, LNCS 4495, pp. 574-588.

Weber, B., Rinderle, S. and Reichert, M. (2008 a): Change Patterns and Change Support Features – Enhancing Flexibility in Process-aware Information Systems. Data and Knowledge Engineering, 66(3):438-466.

Weber, B., and Reichert, M. (2008 b): Refactoring Process Models in Large Process Repositories. In: Proc. of the 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'08), Montpellier, France, LNCS 5074, pp.124-139.

Weber, B., Reichert, M., Wild, W. and Rinderle-Ma, S. (2009): Providing Integrated Lifecycle Support in Process-Aware Information Systems. Int'l Journal of Cooperative Information Systems (IJCIS), 18(1)

Wyner, G.M., and Lee, J. (2003): Defining Specialization for Process Models. In: Malone, T.W, Crowston, K., and Herman, G.A. (eds.) Organizing Business Knowledge – The MIT Process Handbook. MIT Press, pp. 131-174.