Guaranteeing Soundness of Configurable Process Variants in Provop

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

Manfred Reichert
Institute of Databases and Information Systems
Ulm University, Germany
manfred.reichert@uni-ulm.de

Abstract

Usually, for a particular business process a multitude of variants exist. Each of them constitutes an adjustment of a reference process model to specific requirements building the process context. While some progress has been achieved regarding the configuration of process variants, there exists only little work on how to accomplish this in a sound and efficient manner, especially when considering the large number of process variants that exist in practice as well as the many syntactical and semantical constraints they have to obey. In this paper we discuss advanced concepts for the context- and constraint-based configuration of process variants, and show how they can be utilized to ensure soundness of the configured process variants. Enhancing process-aware information systems with the capability to easily configure sound process models, belonging to the same process family and fitting to the given application context, will enable a new quality in engineering process-aware information systems.

1. Introduction

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

1.1. Problem Statement

When engineering process-aware information systems (PAIS) one of the fundamental challenges is to cope with business process variability and the large number of variants that may exist for a particular process [2], [3], [4], [5]. Usually, each of these variants is valid in a particular context [6]. Regarding vehicle repair in a garage, for example, we have identified hundreds of process variants which smoothly differ from each other depending on country-specific, garage-specific, and vehicle-type-specific characteristics. Similar observations can be made for release management processes, for which we identified more than 20 different variants in an automotive company depending on the respective car series, involved suppliers, and development phases [7]. Or when studying the product creation process in the automotive domain, we can identify dozens of variants. Each of them 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. We denote such a collection of related variants as *process family*.

While some progress has been achieved regarding the modeling and management of process variants, there are only few approaches dealing with the fundamental question of how to configure variants out of a master process such that a sound (i.e. correct) execution behavior can be guaranteed for them [3], [4], [5], [8].[1] Though there exists considerable work on how to ensure structural and behavioral soundness of single process models [9], issues related to the correct configuration of a whole process family have been neglected so far. Here, the challenge is to guarantee soundness of a collection of related process variant models taking into account syntactical as well as semantical constraints. Thereby, our goal is not to develop just another approach for checking soundness of single process models, but to provide a framework for configuring semantically valid and sound process variants. In particular, soundness checks should be limited to those process variants, which are relevant in practice, instead of considering all configurable variants. This is particularly important for scenarios with large numbers of variants.

1.2. Contribution

As motivated, variants exist for many business processes and should therefore be adequately managed. In previous work, we introduced the Provop (Process Variants by Options) approach for configuring and managing process variants [6], [10]. Provop considers the whole process life cycle and supports variants in all phases following an operational approach [2], [11], [12]. More precisely, a concrete variant can be configured out of a master process model (denoted as *base process* in Provop) by applying a set of high-level change operations to it. Thereby, information about the process context can be utilized for enabling automated configuration of process variants [2], [10]. Provop provides a generic approach for

1. For a definition of soundness we refer to [9]. Behavioral soundness, for example, implies the absence of deadlocks and livelocks.

variant management, which is independent of a particular process meta model (e.g. BPMN or EPC). The illustrating examples used in this paper are based on process patterns, which are common to most existing process meta models.

So far, we have neglected the aspect of guaranteeing soundness for a family of configurable process variants. This paper extends the Provop framework to deal with this challenge. In particular, we present advanced concepts for the context- and constraint-based configuration of process variants, while guaranteeing their soundness in an effective and efficient way. In particular, we deal with the following research questions:

- How to adequately pre-define the adjustments of a given base process and use them later to configure process variant models?
- How to enable context-based configuration of process variant models?
- How to cope with the structural and semantical constraints to be met when configuring process variants?
- How to guarantee soundness for a process family; i.e., for a collection of process variant models?

Section 2 gives background information on Provop. Section 3 extends our Provop framework by enabling context- and constraint-based configuration of process variants. Picking up this extension, Section 4 presents a procedure that ensures soundness of all configurable process variants. Section 5 discusses related work and Section 6 concludes with a summary and outlook.

2. Background - The Provop Approach

Generally, a process model variant (*process variant* for short) can be created by "cloning" a given process model and adjusting it according to the specific requirements of its application context [13]. Provop adopts this metaphor for variant creation. A particular process variant can be configured by applying a set of predefined adaptations to a common master process (denoted as *base process* in Provop). For describing respective adaptations, Provop supports well-defined change patterns [14]: `INSERT` *fragment*, `DELETE` *fragment*, `MOVE` *fragment*, and `MODIFY` *attribute*. While the first three patterns may be applied to a model fragment (i.e., a connected subgraph), the latter pattern can be used to modify the value of process element attributes. (We provide a formal semantics of change patterns in [15].)

In Provop, a base process can be associated with *adjustment points* that correspond to entries or exits of activities and connector nodes (i.e., split and join nodes) respectively (cf. Fig. 1). This, in turn, enables designers of process adaptations to refer to specific process fragments. By the use of explicit adjustment points, we can restrict the regions of the base process to which adaptations (e.g., insertion or deletion of a fragment) may be applied when configuring a variant. Finally, to enable more complex process adaptations as well as their reuse in different context, Provop allows to group change operations into reusable operation sets, which we denote as

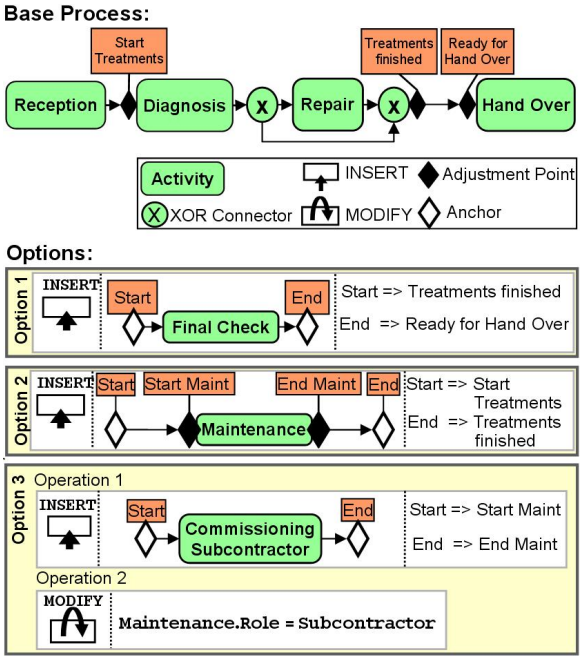


Fig. 1: Examples of options in Provop

options. In summary, a particular variant can be configured by applying one or more options to the given base process. The set of configurable process variant models is denoted as *process family*.

Fig. 1 presents basic Provop elements along a simple example. The depicted vehicle repair process starts with the reception of a vehicle in the garage. After a diagnosis is made, the vehicle is repaired (if necessary). The process finishes when handing over the repaired and maintained vehicle back to the customer. Depending on the process context, different variants are required. In our simplified example, three predefined options exist, out of which a subset can be chosen to configure a particular variant. Option 2, for example, suggests to insert activity *Maintenance* between adjustment points *Start Treatments* and *Treatments finished*. Option 3 comprises two operations which allow to insert activity *Commissioning Subcontractor* and to update attribute *Role* of activity *Maintenance*. Fig. 4 shows different variants that can be configured out of the base process from Fig. 1 by applying a subset of the defined options. Note that for more complex scenarios the number of variants becomes by far larger (e.g., dozens up to hundreds of variants for a vehicle repair process in automotive companies), and thus more options have to be defined to cover all cases.

3. Process Variant Configuration

Provop provides support for the automatic configuration of process variants making use of the process context and considering semantic constraints regarding possible adaptations of the given base process. To better understand those factors which are relevant for configuring process variants,

we conducted several case studies in domains like automotive engineering and healthcare. From this case study research we have learned that there is a strong linkage between the adaptations becoming necessary to configure a specific variant and the current process context; i.e., the concrete adaptations of the given base process depend on the process context and application context respectively. Furthermore, there exist different kinds of relations between the potential adaptations of a base process. While certain adaptations are mutually exclusive, for example, others are always applied conjointly. If we explicitly express such (option) constraints we are able to reduce the number of valid option combinations and thus to decrease efforts for guaranteeing soundness of the configurable variants. This section summarizes how Provop enables context- and constraint-based variant configuration. We pick up these concepts in Section 4 when presenting the Provop approach for guaranteeing soundness for the configurable variants of a process family.

3.1. Context-aware Selection of Options

As particular process variants are often required in a specific context, Provop enables *context-based configuration* of business process variants. For this purpose, a *context model* capturing the *process context* has to be provided. In Provop, such context model comprises a set of *context variables* with a discrete and finite value range (cf. Fig. 2a). Thereby, each context variable specifies one particular dimension of the process context. Regarding our vehicle repair process, for example, this can be visualized by a *context cube* as depicted in Fig. 2c.[2] Each sub-cube is enumerated for the sake of readability. It represents one possible combination of values assigned to the different context variables. We denote corresponding value assignments as *context descriptions* in the following. As not all possible context descriptions may be semantically meaningful, Provop allows to restrict them by *context constraints* (cf. Fig. 2b). Regarding the given example, for instance, activity *maintenance* will have to be performed if the required *security level* is high. Consequently, the corresponding context constraint (cf. Fig. 2b) invalidates sub-cubes 16, 17, and 18 (cf. Fig. 2c).

We define such a *context model* for each process family. Based on it, *context rules* can be created and assigned to one or more *options* (i.e., pre-defined adjustments of the given base process). This, in turn, enables context-aware option selection; i.e., if the context rule of a particular option evaluates to true for a given *context* this option will be (automatically) applied to the base process when configuring a corresponding variant. Generally, for a particular *context description*, the context rules of multiple options may evaluate to true. In such a case, all selected options are considered when configuring the corresponding process variant out of the base process.

Fig. 3 visualizes the options from Fig. 1 together with their associated context rules and constraints (see Section 3.2).

2. Note that the context cube is just a visualization used in this paper to illustrate process context.

a) Context table:

Variable Name	Value Range
Security level	low, medium, high
Maintenance	yes, no
Workload	low, medium, high

b) Context constraints:

IF security-level = "high"
THEN maintenance = "yes"

c) Context cube:

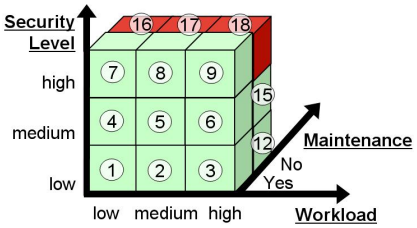


Fig. 2: Context model (a), constraints (b) and context cube (c)

From the context rule of Option 2, for example, we can conclude that Option 2 will be applied to the base process if context variable `Maintenance` has value "Yes" for a given context description.

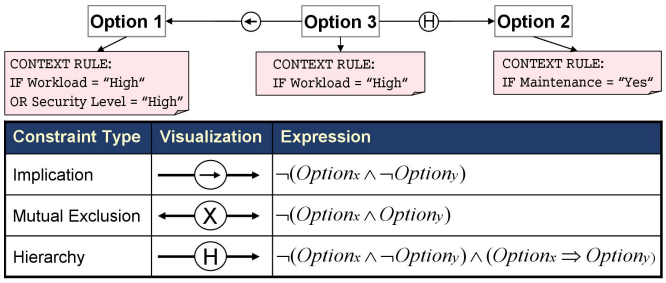
3.2. Constraint-based Use of Options

The adjustments becoming necessary to configure a particular process variant are often structurally or semantically correlated. Regarding our example from Fig. 1, for instance, the application of Option 3 to the depicted base process requires the prior introduction of Option 2 (since Operation 2 of Option 3 refers to the activity inserted by Option 2). Besides such structural dependencies semantical constraints have to be considered as well. For instance, Option 3 semantically implies Option 1 since activity `Maintenance` will always require subsequent execution of activity `Final Check`, if maintenance is not done by the garage itself, but quality of service has to be ensured. (Option 3 updates the role attribute of activity `Maintenance` to `Subcontractor`.) Amongst others, Provop supports the following kinds of option relations in order to constrain the use of options.

- **Implication:** If two options shall be always applied together to the base process (e.g. due to a semantical dependency) the option designer can define a directed implication relation between them. Generally, from relation *"Option 1 implies Option 2"* we must not conclude the reverse one (i.e., *"Option 2 implies Option 1"*).
- **Mutual exclusion:** This constraint is useful to specify that two options must never be applied together when configuring a particular process variant.
- **Option hierarchy:** The explicit definition of an *option hierarchy* enables inheritance of change operations. If an option with ancestors is selected to configure a particular process variant, its ancestor options will be applied as well. This structures the total set of options, and also

reduces the average number of change operations needed to define a particular option.

Fig. 3 shows the application of these constraints to our example from Fig. 1.



Constraint Type	Visualization	Expression
Implication	→	$\neg(Option_x \wedge \neg Option_y)$
Mutual Exclusion	X	$\neg(Option_x \wedge Option_y)$
Hierarchy	H	$\neg(Option_x \wedge \neg Option_y) \wedge (Option_x \Rightarrow Option_y)$

Fig. 3: Option constraints

Generally, options and their corresponding change operations are not commutative. Consequently, for both options and operations we need to define the order in which they shall be applied at configuration time. Based on this information, Provop allows to configure process variants through the sequential application of a set of options (and their change operations) to the given base process. In particular, the chosen option set needs to match the current process context and comply with the defined option constraints. Consider Fig. 4, for example, which shows the process family that can be derived from the base process and the options depicted in Fig. 1. Note that only those option sets are considered that match the given context and that are compliant with the defined option constraints (cf. Fig. 3).

How the latter issue is achieved and how Provop guarantees soundness for the configurable variants of a process family is discussed in the next section.

4. Guaranteeing Soundness of Variants

Provop targets at guaranteeing soundness of all configurable process variants. As mentioned before, our focus is not on checking soundness of single process models, based on a specific process metamodel (see [9] for details), but on establishing a framework for ensuring soundness of a potentially large collection of process variant models. In the following we show how Provop enables the context- and constraint-based configuration of a process family consisting of *sound* variant models.

4.1. Basic Issues and Motivation

One possibility to ensure soundness of configurable process variants is to start the configuration procedure with a sound model of the base process and to enforce soundness after each applied change operation. Consequently, the application of a set of change operations and a set of options respectively, would result in a sound variant model.

Unlike existing configuration techniques [3], Provop does not necessarily require a sound process model as starting point for variant configuration. Instead we want to provide high flexibility to users and therefore support different policies when defining the base process of a process family. For example, a base process may be designed in a way such that it covers all configurable variants or only constitutes a minimal process model (i.e., an *intersection* of its variant models) [2], [10].

As example consider Fig. 5a where Variant 1 describes a car-specific and Variant 2 a bus-specific process model. If we define the base process as "intersection" of these two variant models, we obtain the process model depicted at the bottom of Fig. 5a. This base process comprises activities `CA1`, `CA2`, and `CA3`, but does not contain the car- or bus-specific activity. Interestingly, this model is not sound when considering data flow, since the data object read by activity `CA3` is neither written by `CA1` nor `CA2`. However, this will be not a problem if we ensure that any variant model resulting through configuration is sound.

Enforcing a sound base process, however, is not appropriate in the given scenario. When choosing the model of Variant 1 as base process (cf. Fig. 5b), for example, visibility constraints may become violated. Note that Variant 1 would then be visible to the process owner of Variant 2, who additionally must be able to track the adjustments of Variant 1 in order to correctly evolve Variant 2 over time. Another inadequate approach would be to restore soundness of the base process model by adding an abstract activity to it, which writes the data element. This would increase modeling efforts unnecessarily when configuring the two variants depicted in Fig. 5a.

Another possibility, followed by Provop, is to first apply the desired options to the base process and then to check soundness of the resulting process model afterwards. A naive approach for guaranteeing soundness of a whole process family would be to apply all possible combinations of options to the base process and then to check soundness for each of the resulting process models. However, this would be very expensive. As example consider the simple scenario from Fig. 1 for which three options exist. Assuming that options are not commutative in general, we would have to check for 16 different option combinations whether or not their application to the base process results in a sound process model. Obviously, for more complex scenarios with dozens of options this is not a feasible approach. Thus, the challenge is to reduce the number of configurable models (i.e., the process family) that need to be checked. Therefore, we only consider those process variant candidates for which the applied options satisfy the corresponding context rules and also meet the defined option constraints. Thereby, we do not only reduce the number of process variant models for which soundness has to be checked, but ensure structural and semantical soundness of the whole process family as well.

4.2. Overview of the Provop Soundness Check

Guaranteeing soundness of configurable process variants is accomplished in a number of steps (cf. Fig. 6). In Steps 1

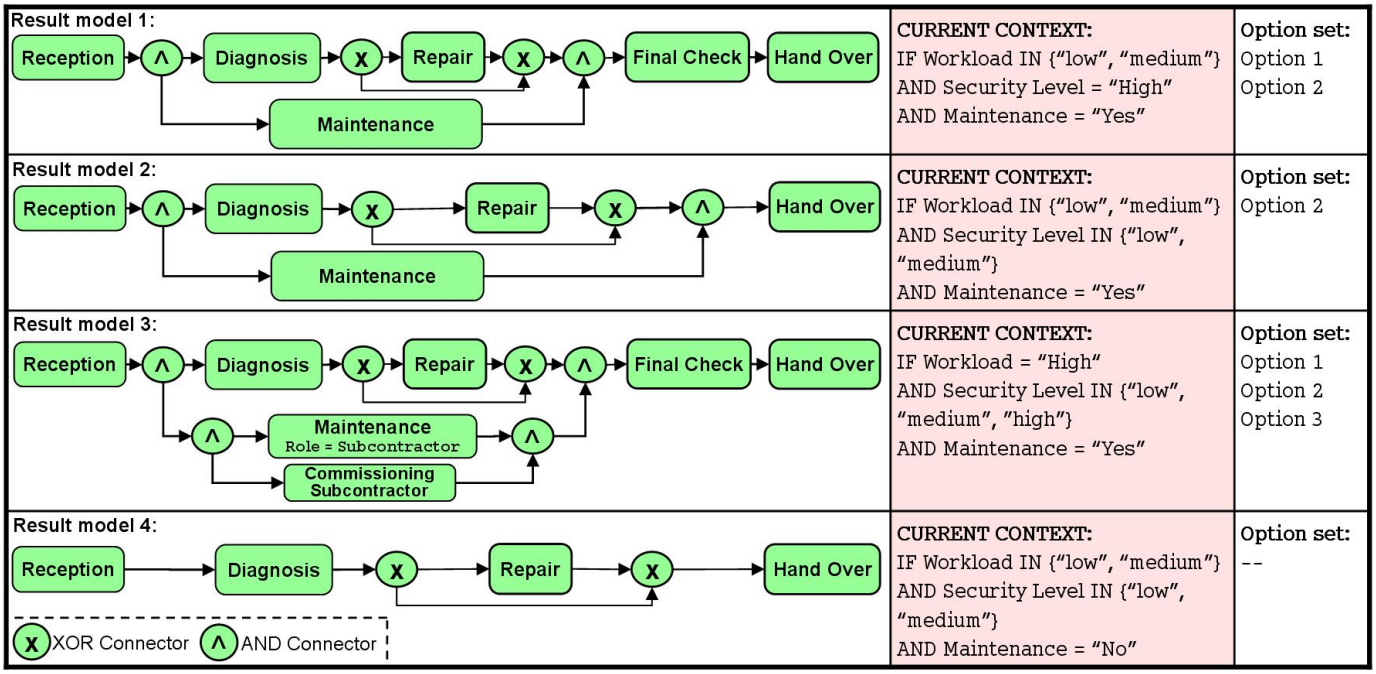


Fig. 4: Resulting process family

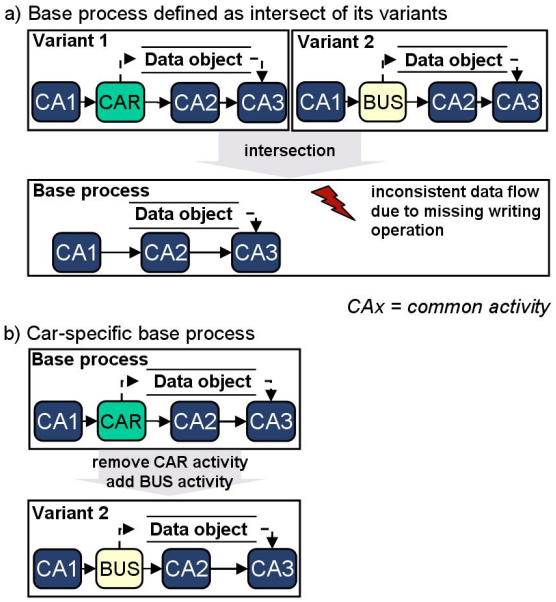


Fig. 5: Inconsistent base process (a) with exemplary soundness scenario (b)

and 2, valid context descriptions are identified, and for each of them the corresponding option set (i.e., adjustments of the base process) is determined. Step 3 then checks whether or not the calculated option sets comply with the defined option constraints (cf. Section 3.2). If an option set is not valid an error will be reported to the designer. Otherwise, Steps 4 and 5 apply the options from this set to the base process and check whether or not the resulting process variant model is sound. Results of Steps 4 and 5 are logged in a report. In the following we describe these five steps in more detail (see [16] for a report with more technical details).

4.3. Basic Steps

We now describe the sketched procedure for checking soundness of a process family in detail. It starts with identifying all valid context descriptions (cf. Section 3.1) for which process variants shall be configured. Consequently, for corresponding cases we need to guarantee soundness of the configurable variants. As invalid context descriptions are implicitly specified by the given set of context constraints, Provop first evaluates all possible allocations of values for context variables. In order to check whether or not a given context description is valid, function `ctxtDescrValid` is provided. For a given context description this function will return true if there is no context constraint in the given context model that invalidates this context description. As result we obtain the set of valid context descriptions (i.e., CD_{valid}).

// Step 1: Identify valid context descriptions
$CD_{valid} = \emptyset$ *// Initializing the set of valid context descriptions*
// Create context descriptions by simulating all possible values in the value range of each context variable $CtxtVar_i$, *i=1,..,n defined in the context model. We assume that corresponding value ranges* $ValueRange(CtxtVar_i)$ *are discrete and finite.*
for all CtxtDescr $\in$ (ValueRange($CtxtVar_1$) $\times$ ... $\times$

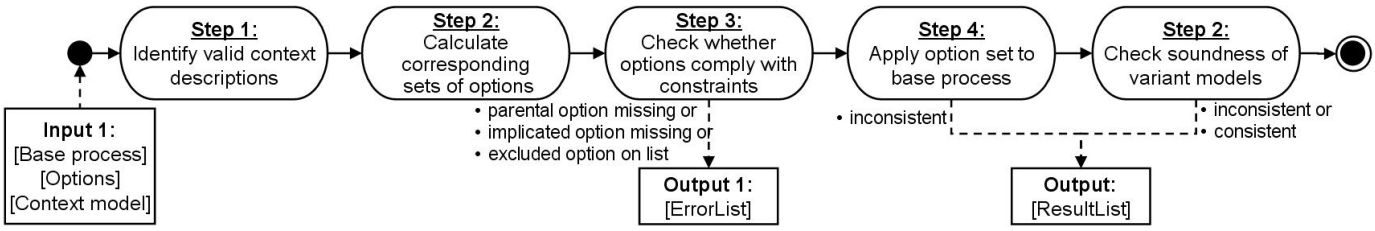


Fig. 6: Overview of the Provop procedure for guaranteeing soundness

```
ValueRange(CtxtVar_n)) do
  // check whether or not the context description is valid
  if ctxtDescrValid(CtxtDescr,CtxtModel) = true then
   CD_valid := CD_valid ∪ {CtxtDescr}
```

Example 1: In our example from Fig. 2, sub-cubes 16, 17, and 18 become invalid due to the constraint "`IF security-level = high THEN maintenance = yes`". However, all other context descriptions are valid. Therefore, we add them to the set of valid context descriptions and obtain CD_{valid} = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}.

For each valid context description, Step 2 calculates the option set to be applied when configuring the corresponding variant. (An option set can be empty as the base process itself can be a variant.) For this purpose, Step 2 utilizes function `contextRuleValid`, which can be used to check whether or not a particular option shall be applied in the given context. This function returns true if the context rule of an option (cf. Sect. 3.2) is valid regarding currently chosen values of the context variables (i.e., regarding the given context description). Thus `contextRuleValid` is applied to each option. If it returns true for a selected one, this option is added to the option set of the currently considered context description. Otherwise, it is not considered for the given context.

As depicted in Fig. 7 different context descriptions may have the same option set. To check only once whether or not a particular option set is correctly applicable to the base process, context descriptions with same option set are grouped into *context blocks*. This is accomplished by functions `insCtxtBlock` and `extCtxtBlock`. As a result of Step 2, we obtain a set of <CtxtBlock,OptionSet> pairs; i.e., for each context block (i.e., a set of context descriptions), we obtain the option set to be applied when configuring the process variants for the respective context. We denote this object as *variant candidates*.

```
// Step 2: Calculate corresponding sets of options
// consider CD_valid as determined in Step 1
for each CtxtDescr ∈ CD_valid do
  CalcOptions := ∅
  // check validity of context rules for all defined options O
  for each Option ∈ O do
```

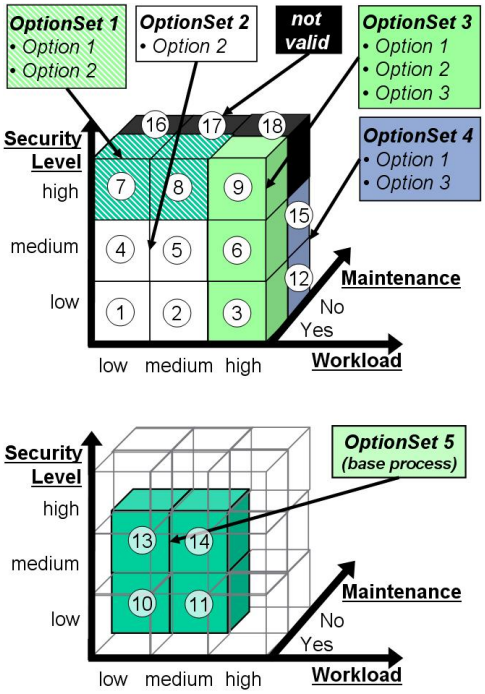


Fig. 7: Blocks of context descriptions and respective options

```
  if contextRuleValid(Option,CtxtDescr) = true then
    CalcOptions := CalcOptions ∪ {Option}
  // check if set of options has already been created
  if hasOptSet(VariantCandidates,CalcOptions) = true then
   // insert new block of context descriptions together with
     one common option set
   insCtxtBlock(VariantCandidates,CtxtDescr,CalcOptions)
  else
   // extend existing block with current context description
   extCtxtBlock(VariantCandidates,CtxtDescr,CalcOptions)
```

Example 2: For the context model from Fig. 2, we obtain the following variant candidates (i.e., <CtxtBlock,OptionSet>): <{12, 15}, {Option 1}>, <{7, 8},{Option 1, Option 2}>, <{1, 2, 4, 5},{Option 2}>, <{3, 6, 9},{Option 1, Option 2, Option 3}>, <{10, 11, 13, 14},{}>. The option set of the latter candidate is empty; i.e., its model corresponds to the base process.

For each variant candidate, Step 3 checks semantic compatibility of its option set with the defined option constraints. This could be based, for example, on Linear Temporal Logic (LTL) or some other model checking technique. Here, we simply assume that there is a function `checkOptionConstraints`, which returns true if the corresponding option set complies with all defined option constraints (cf. Section 2). Otherwise, the respective <CtxtBlock,OptionSet> pair is deleted from the set of variant candidates, and corresponding information is added to an error report (i.e., `ErrorList`).

```
// Step 3: Check whether options comply with constraints
for each <CtxtBlock,OptionSet> ∈ VariantCandidates do
  if checkOptionConstraints(OptionSet) = false then
   // remove candidates that are not compliant
   removeCandidate(VariantCandidates,<CtxtBlock,OptionsSet>)
   writeErrorList(...)
```

Example 3: The hierarchy constraint (cf. Section 3.2) requires that all ancestors of an option are also applied to the base process. As example consider the constraints defined for the options from Fig. 3. OptionSet 4, which contains Options 1 and 3 (cf. Fig. 7), does not comply with the hierarchy constraint. Reason is that ancestor of Option 3 (i.e., Option 2) is not contained in the option set. Consequently, the context block associated with OptionSet 4 is removed from the list of variant candidates. Generally, in addition to context dependencies, option constraints ensure semantical soundness of option sets and further reduce efforts for checking soundness.

After completing Step 3, we have obtained relevant variant candidates. Following this, in Step 4, for each candidate the elements from its option set have to be ordered, i.e. the sequence in which the options shall be applied to the base process has to be fixed. Provop provides different concepts for ordering options (e.g., based on timestamps or user-defined orders). Due to lack of space we omit details here, but assume that there is a function `sortOptionSet` that defines a partial order for options (see [16] for details). If an error occurs (e.g., cyclic ordering constraints), the procedure will stop. It further adds an entry to the report list, which specifies that the current variant candidate is invalid. Otherwise, the checking procedure continues with Step 5.

```
// Steps 4+5: Apply option set to base process and check
soundness of variant models
for each <CtxtBlock,OptionSet> ∈ VariantCandidates do
  // create partial order of OptionSet
    in sortedOptionSet
  if sortOptionSet(OptionSet,sortedOptionSet) = true then
   // calculate resulting model by applying an option set
   to the base process
   if calculateVariant(BaseProcess,sortedOptionSet,ResultModel)
   = true then
    if checkSoundness(ResultModel) = true then
     // result model is sound
     storeResult(OptionSet,true,..)
    else
     // result model is not sound
     storeResult(OptionSet,false,..)
```

At the end of Step 4, for each variant candidate we have obtained its option set and the order in which the options shall be applied to the base process when configuring the corresponding variant. Step 5 then calculates the candidate models, if possible, by using function `calculateVariant`. If an option and its associated change operations are not applicable (e.g., due to missing object references) Provop does not calculate a candidate model for the corresponding option set, but adds an entry to the error report. Otherwise, structural and behavioral soundness of the resulting variant model are checked, considering the specifics and verification techniques of the underlying meta model (function `checkSoundness`). Finally, the variant candidate, together with the respective consistency check result (i.e., either "consistent" or "inconsistent") are stored in the report list.

After completing Steps 1-5 of the Provop soundness checking procedure, the error and result list created during Steps 3 to 5 will be evaluated. Obviously, the process family is sound, if for all valid context descriptions a structurally and semantically sound model can be created. Otherwise, users are supported by precise suggestions when correcting modeling errors.

5. Related Work

Process variants are relevant for reference process modeling. Such a reference process has recommending character, covers a family of process models, and can be customized to meet specific needs. Configurable event process chains (C-EPCs), for example, provide support for both the specification and customization of reference process models [4], [5]. When modeling a reference process, EPC functions (and decision nodes) can be annotated to indicate whether or not they are mandatory or optional. This information is considered when configuring C-EPCs.

A similar approach is presented in [8]. Here the concepts for configuring a reference process model (i.e., to enable, hide or block a configurable process element) are transferred to workflow models. Similar to Provop, constraints regarding adjustments of the reference process can be defined (e.g., two activities either may have to be deleted together or none of them). As opposed to Provop, it neither is allowed to move or add model elements nor to adapt element attributes when configuring a variant. Finally, [3] shows how to configure reference process models incrementally and in a way that ensures the soundness of single process variants, both with respect to syntax and (behavioral) semantics. As opposed to

Provop, this approach presumes that the original process model is sound.

Different work exists on how specialization can be applied to deal with process model variability taking advantage of the generative nature of a specialization hierarchy [17], [18]. [17] has shown how specialization can be realized for state and dataflow diagrams, respectively. For both diagram types a set of transformation rules is provided resulting in process specializations when applying them to a particular model. Similarly, [18] discusses transformation rules to define specialization for models based on Petri Nets.

Fundamental characteristics of software variability in software engineering are described in [19]. In particular, software variants exist in software architectures and software product lines [20], [21]. Often feature diagrams are used for modeling software systems with varying features; soundness issues are not considered. Another contribution stems from PESOA [22] which provides basic concepts for variant modeling based on UML. Different variability techniques like inheritance, parameterization, and extension points are provided. As opposed to PESOA, Provop provides a more powerful approach for describing variance in a uniform and easy manner. Finally, [23] goes beyond control flow and extends business process configuration to roles and objects.

6. Summary and Outlook

We have described the Provop approach which enables context- and constraint-based configuration of process variants. This paper has put emphasis on how to ensure soundness of the configurable variants of a whole process family, taking into account semantical as well as structural constraints. We prototypically implemented Provop on top of the ARIS tool utilizing its programming interface [24]. In future research, we will apply Provop in industrial context.

References

[1] Mutschler, B., Reichert, M., Bumiller, J.: Unleashing the effectiveness of process-oriented information systems: Problem analysis, critical success factors and implications. IEEE Transactions on Systems, Man, and Cybernetics (Part C) **38** (2008) 280–291

[2] Hallerbach, A., Bauer, T., Reichert, M.: Managing Process Variants in the Process Life Cycle. In: Proc. 10th Int. Conf. on Enterprise Information Systems. (2008) 154–161

[3] van der Aalst, W.M.P., Dumas, M., Gottschalk, F., ter Hofstede, A.H.M., la Rosa, M., Mendling, J.: Correctness-preserving configuration of business process models. Fundamental Approaches to Software Engineering (2008) 46–61

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

[5] Rosa, M.L., Lux, J., Seidel, S., Dumas, M., ter Hofstede, A.H.M.: Questionnaire-driven Configuration of Reference Process Models. In: Proc. CAiSE'07. (2007)

[6] Hallerbach, A., Bauer, T., Reichert, M.: Context-based configuration of process variants. In: Proc. TCoB 2008 Workshop. (2008) 31–40

[7] Müller, D., Herbst, J., Hammori, M., Reichert, M.: IT Support for Release Management Processes in the Automotive Industry. In: Proc. 4th Int. Conf. on Business Process Management. LNCS 4102 (2006) 368–377

[8] Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H., la Rosa, M.: Configurable Workflow Models. In: Int. Journal of Cooperative Information Systems. (2007)

[9] van der Aalst, W.M.P.: Workflow verification: Finding control-flow errors using petrinet-based techniques. In: Proc. BPM'00. (2000) 161–183

[10] Hallerbach, A., Bauer, T., Reichert, M.: Issues in modeling process variants with provop. In: Proc. BPM'08 Workshops. LNBIP 17 (2009) 54–65

[11] Hallerbach, A., Bauer, T., Reichert, M.: Capturing Variability in Business Process Models: The Provop Approach. Software Process: Improvement and Practice (2009) (to appear).

[12] Hallerbach, A., Bauer, T., Reichert, M.: Configuration and management of process variants. In Rosemann, M., Brocke, J.V., eds.: Handbook on Business Process Management, Springer-Verlag (2009) (to appear).

[13] Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. Data and Knowledge Engineering **50** (2004) 9–34

[14] Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data and Knowledge Engineering **66** (2008) 438–466

[15] Rinderle-Ma, S., Reichert, M., Weber, B.: On the formal semantics of change patterns in process-aware information systems. In: Proc. ER'08. LNCS 5231 (2008) 279–293

[16] Hallerbach, A., Bauer, T., Reichert, M.: Correct Configuration of Process Variants in Provop. Technical Report UIB-2009-03, Uni Ulm (2009)

[17] Malone, T., Crowston, K., Herman, G.: Organizing Business Knowledge - The MIT Process Handbook. MIT Press (2007)

[18] van der Aalst, W.M.P., Basten, T.: Inheritance of Workflows: An Approach to Tackling Problems Related to Change. Technical report, TU Eindhoven (2002)

[19] Bachmann, F., Bass, L.: Managing Variability in Software Architectures. In: Proc. of 2001 Symp. on Software Reusability, New York, ACM Press (2001) 126–132

[20] Becker, M., Geyer, L., Gilbert, A., Becker, K.: Comprehensive Variability Modeling to Facilitate Efficient Variability Treatment. In: 4th Workshop on Product Family Eng. (2001)

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

[22] Puhlmann, F., Schnieders, A., Weiland, J., Weske, M.: PESOA - Variability Mechanisms for Process Models. Technical Report 17/2005, Hasso-Plattner-Institut, Potsdam (2005)

[23] La Rosa, M., Dumas, M., ter Hofstede, A., Mendling, J., Gottschalk, F.: Beyond control-flow: Extending business process configuration to roles and objects. In: Proc. ER'08. (2008)

[24] IDS Scheer AG: ARIS Platform Method 7.1. (2008) www.ids-scheer.com.