Modelling Business Process Variants using Graph Transformation Rules

Christine Natschläger[1], Verena Geist[1], Christa Illibauer[1] and Robert Hutter[2]
[1]*Software Competence Center Hagenberg, Hagenberg, Austria**
[2]*Prologics IT GmbH, Linz, Austria*

Keywords: Business Process Management, Variability Modelling, Variant Management, Graph Transformation.

Abstract: Business process variability is an active research area in the field of business process management and deals with variations and commonalities among processes of a given process family. Many theoretical approaches have been suggested in the last years; however, practical implementations are rare and limited in their functionality. In this paper, we propose a new approach for business process variability based on well-known graph transformation techniques and with focus on practical aspects like definition of variation points, linking and propagation of changes, as well as visual highlighting of differences in process variants. The suggested concepts are discussed within a case study comprising two graph transformation systems for generating process variants; one supports variability by restriction, the other supports variability by restriction and by extension. Both graph transformation systems are proven to be globally deterministic, but differ regarding their complexity. The overall approach is being implemented in the BPM suite of our partner company.

1 INTRODUCTION

Process variability is related to flexible business process management (BPM), one of the most active research areas (Reichert and Weber, 2012). In detail, process variability is concerned with design- and customization-time decisions and deals with differences and commonalities among processes of a given process family (Rosa et al., 2013). Typical fields of application are, e.g., production processes, invoice processes, and delivery processes.

There is a large number of theoretical approaches to process variability, focusing on variability by restriction and/or by extension. However, many experienced researchers assess the tool support for handling process variants as limited and not adequate (Reichert and Weber, 2012; Rosa et al., 2013). In addition, exponential growth makes modelling and maintaining process variants a complex endeavour in practice. Thus, there is need for establishing a sound method for explicitly supporting variability in business processes in a way that fosters industrial applicability.

In this paper, we propose a new concept for variant management based on graph transformation techniques. Important aspects of this concept are the individual definition of adaptable and blocked elements as well as linking and propagation of changes. A further advantage is the application of graph transformation rules to deal with the exponentially increasing number of variants. Instead of manually defining all variants, a few rules specify concrete variations and the corresponding variants are automatically generated.

The paper is structured as follows. In Section 2, we provide an overview of related work on graph transformation, variability modelling, and tool support. We present the general approach for modelling business process variants based on key requirements in Section 3 and discuss the proposed concepts within a case study using graph transformation techniques in Section 4. In Section 5, we summarise our findings.

2 STATE-OF-THE-ART

This section provides an overview of the state-of-the-art concerning graph transformation, variability modelling, and the current support of business process variability in BPM suites.

*The research reported in this paper has been partly supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH. This publication has been written within the project *AdaBPM* (number 842437), which is funded by the *Austrian Research Promotion Agency* (FFG).

Natschläger, C., Geist, V., Illibauer, C. and Hutter, R.
Modelling Business Process Variants using Graph Transformation Rules.
DOI: 10.5220/0005665800650074
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 65-74
ISBN: 978-989-758-168-7

2.1 Graph Transformation

Research on graph transformations started around the 1970s; the main idea is the rule-based modification of a graph, where each application of a rule leads to a graph transformation step. So the transformation of a graph L (left-hand side) to a graph R (right-hand side) is based on a rule r (also called production). Applying the rule $r = (L,R)$ means finding a match of L in the source graph and replacing L by R, leading to the target graph of the graph transformation. The application of rules is restricted by application conditions (AC), which can either be positive or negative. A negative application condition (NAC) is satisfied if it is not part of a match of L in graph G. A positive application condition (PAC) is the counterpart to a NAC and is satisfied if it is part of the match of L in G (Ehrig et al., 2006, p. 5f, 67ff).

Different approaches for graph transformation have been proposed like node label replacement, hyperedge replacement, or the algebraic approach (Ehrig et al., 2006, p. 10). The algebraic approach, which was initiated by Ehrig, Pfender, and Schneider in 1973, is applied for the suggested variability concept. Algebraic graph transformation is supported by a tool for attributed graph grammar systems (called AGG). AGG provides a graphical editor and can be used for specifying graph grammars with a start graph or for typed attributed graph transformations. In addition, AGG offers analysis techniques as for consistency checking, critical pair analysis, and termination evaluation (Ehrig et al., 2006, chapter 15). In previous work, AGG was applied to implement an algebraic graph transformation from standard BPMN to Deontic BPMN diagrams and to prove that this is a trusted model transformation (Natschläger et al., 2015). In Section 4, AGG is used to implement and evaluate the suggested variability concept.

The algebraic approach to graph transformation has been generalised to further graph types and high-level structures such as typed graphs, labelled graphs, hypergraphs, attributed graphs, and algebraic specifications. This extension to high-level structures is called high-level replacement (HLR) (Ehrig et al., 2006, p. 14ff). The graph transformation for the proposed variability concept is defined as typed attributed graph transformation system with inheritance that includes labels.

2.2 Variability Modelling

A survey on business process variability modelling is provided in (Rosa et al., 2013). The authors distinguish three phases in the lifecycle of customizable process models: design-time, customization-time, and runtime. Variability is concerned with design- and customization-time decisions and subdivided into variability *by restriction* (i.e., customizable model is the union or least common multiple of all process variants) and variability *by extension* (i.e., customizable model is the intersection or greatest common denominator of all process variants). The authors further argue that there is no explicit support for representing and maintaining multiple variants of a defined business process in conventional business process modelling languages. Available approaches typically extend a modelling language with additional constructs to capture customizable process models.

Variability by restriction is applied by so-called *configurable approaches*. Configurable approaches require a comprehensive reference model that comprises all possible process flows and which is configured at design-time to the required variants (see (Reichert and Weber, 2012) for details). In (Aalst, 2013), twenty BPM use cases are suggested, three of them relate to configurable process models: design configurable model, merge models into configurable model, and configure configurable model. Business process modelling languages supporting the configurable approach are, for example, *Configurable YAWL* (C-YAWL) and *Configurable EPCs* (C-EPCs). In C-YAWL, hiding and blocking operations are applied to the input and output ports of activities, so activities can either be hidden (skipped) or the triggering of the activity (and the subsequent path) is prevented (blocked) (Gottschalk, 2009). C-EPCs, on the other hand, provide configurable functions (may be included, excluded, or conditionally skipped), configurable connectors (can be mapped to more restrictive connectors), and configurable requirements and guidelines (*If-Then*-statements to define dependencies between configurable nodes) (Recker et al., 2006). Configurable approaches are useful to adapt, e.g., a domain-specific reference model to a concrete organisation. However, the underlying modelling language must be extended with configurable elements and the reference model is extensive since it must comprise all process flows. Furthermore, local process variants are restricted in their adaptability.

Adaptable Approaches, in contrast, only require a base process model, which comprises the standard process flow and can be adapted through structural model adaptations based on change patterns (variability by restriction *and* by extension). In (Weber et al., 2008), eighteen change patterns are suggested and divided into adaptation patterns (e.g., add/delete/move/replace fragment) and patterns for changes in predefined regions to deal with uncer-

tainty. In addition, (Doehring et al., 2011) propose a pattern catalogue with patterns for basic adaptations, time adaptation and exception handling. The adaptable approach is implemented, e.g., by Provop. Provop is an approach for modelling and managing process variants, covering the whole process lifecycle. In the modelling phase, a base process is defined, comprising adjustment points to restrict the regions to which adaptations may be applied. In addition, a set of reusable change operations is specified, which can be grouped into so-called options. Provop supports the following change patterns: insert/delete/move fragment and modify attribute. In the configuration phase, the user selects a sequence of options and applies them to the process to configure the desired process variant. The configuration may be context-aware, meaning that the selection and application of options is based on the context. The context model then comprises a set of context variables as well as restricting context constraints. Each context variable specifies one particular dimension, being visualised in a context cube in which every cell represents a variant. In the execution phase, a configured variant is transformed into an executable workflow model and deployed to the runtime environment. Runtime flexibility is provided by the possibility to dynamically switch execution between different process variants. Finally, in the optimization phase, the base process may be changed according to previous adaptations, so that the process family evolves over time (Hallerbach et al., 2009; Hallerbach et al., 2010).

The main similarity between the Provop approach and our variability concept is that both approaches apply variability by restriction and by extension. Differences concern the implementation of adaptations (concept of change operations and options vs. graph transformation techniques with formal foundation) and the restriction of adaptations (adjustment points vs. individual blocked/adaptable elements).

Summing up, none of the proposed configurable or adaptable approaches fits for our purpose. They either introduce additional constructs to a modelling language in order to support process variability, require extensive reference models, restrict adaptations, or propose concepts without formal foundation.

2.3 Variability Modelling in BPM Suites

We studied the practical application by comparing in detail six different BPM suites regarding their support for variability modelling and variant management. The first BPM suite is called Aeneis and, according to its documentation, claims to support variant management (Aeneis, 2014). Variant management is, however, implemented by a simple copy & paste mechanism. The variant (i.e., copy of a process) can be modified without restrictions, since elements are by default not linked to corresponding elements of the parent process. Links can be inserted manually, but then all linked elements are identical and neither side can be changed without propagation of the change to the other element. Processes (and variants) can be compared by a basic comparison tool that lists all elements of each process without highlighting any differences. So, in our opinion, Aeneis does not provide actual variability modelling, but it offers some functionalities required by a rudimentary variant management system.

The BPM suite Signavio (Signavio, 2015) does not mention the support of variant management in its documentation but provides appropriate functionality. First of all, Signavio supports rudimentary variant management by copying processes and permitting to adapt the copy (no linking). A graphical comparison tool identifies all changes between two process models except swap operations. In addition, Signavio supports stakeholder-specific views that resemble a variant management with a reference model. After having specified the overall process model (i.e., reference model), views are defined by selecting and omitting elements (variability by restriction). Only the settings of a view are saved and the view is generated whenever required. Graphical model comparison is provided between the reference model and the view highlighting all differences in the reference model.

Scheer Business Process as a Service (BPaaS) (Scheer, 2015) offers a simple variant management limited to variability by restriction with the Scheer Process Tailor. A reference process is defined and variants are created by copying the reference process and removing unnecessary activities. Thus, adaptation of local variants is restricted to delete operations.

Three further BPM suites only support copy & paste of a process model, which is insufficient for variant management. Bizagi with its products Modeler, Studio, and Engine is a comprehensive BPM suite, also including a simulation component but no variant management (Bizagi, 2014). Axon IVY, based on the eclipse platform, provides six modules for modelling, publication, analysis, design, execution, and monitoring but again variant management is missing (Axon IVY, 2015). Lastly, BPM suite FireStart of our industrial partner (Prologics, 2014) currently lacks support for variant management but is being extended with the suggested variability concept.

In addition, to the best of our knowledge, also other well-known tools such as the MS Workflow Manager, IBM Business Process Manager, jBPM,

and the BPM suites by Bonitasoft, Camunda, Inubit, TIBCO, K2, and Appian do not fulfill our requirements for variability modelling.

Summing up, although variability modelling and variant management have been discussed in detail from a theoretical perspective, practical applications are very limited. Only two of the analysed BPM suites (Signavio and Scheer BPaaS) support variability by restriction and thus part of variant management. Current implementations especially lack functionality regarding connections between parent and child processes (i.e., links), propagation of changes, and visual highlighting of differences. These issues are addressed by the following variability concept.

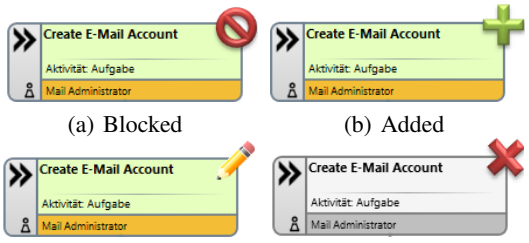
3 A CONCEPT FOR VARIABILITY MODELLING

In this section, we present our concept for variability modelling, which is called *Adaptive Variant Modelling* (AdaVM) and part of the overall AdaBPM framework for advanced adaptivity and exception handling in formal business process models. The concept of AdaVM is based on the following key requirements for business process variability of our partner company Prologics, producer of the BPM suite *FireStart*. The requirements stem from a comprehensive BPM vendor survey (2014):

1. definition of a main process and corresponding variants,
2. local adaptation/extension of variants,
3. visual highlighting of differences between main process and variants, and
4. propagation of changes in main process to variants with notification of responsible person.

So, at first, the *main process model* (also called *base* or *default* process model) is defined like any other process model, with two exceptions: The first exception is that AdaVM restricts variation points by providing Boolean attributes to globally block *insert* operations (to support *variability by restriction*) and to locally block *modify* and *delete* operations of individual elements or attributes. Blocked elements or attributes indicate parts common to all variants and can neither be changed nor deleted. The second exception is that the main process model may not be a semantically valid process variant. Five different policies have been suggested in (Reichert and Weber, 2012, p. 104f) for specifying the main process model: (1) reference process, (2) most frequently used process, (3) minimum adaptation efforts, (4) superset of all process variants, and (5) intersection of all process variants. Only the first and second policies represent a particular variant and are directly executable. The main process model of the fourth policy can only be adapted by delete operations (configurable approach). AdaVM does not prescribe a policy but it supports and recommends to use both techniques, *variability by restriction and by extension* (adaptable approach) (see discussion in Section 4).

In the following, we describe the general approach of AdaVM for modelling business process variants. A *derived variant* is at the beginning a duplication of the main process model. Every element in the variant is linked to the corresponding element in the main process model, e.g., by the same ID or an additional attribute specifying the ID of the parent element. Blocked elements are marked with a block symbol and disabled (see Figure 1(a)), so that they can neither be changed nor deleted. All unblocked elements of the variant represent variation points and can be adapted/extended by applying the following change operations: *insert*, *delete*, and *modify*. Change operation *move* suggested by Provop in (Hallerbach et al., 2010) is substituted by deletion and insertion of edges. Three symbols are introduced and correspond to the three change operations. An inserted element (node or edge) is marked with a plus symbol (see Figure 1(b)) and a modified element is marked with a pencil symbol (see Figure 1(c)). A deleted element, however, is usually removed from a process model, thereby losing any graphical information about the change. Thus, we decided to disable and grey-out deleted elements and mark them with a cancel symbol (see Figure 1(d)). Advantages of this approach are the visibility of the change and the possibility to restore a deleted element with its link to the corresponding parent element. So the link to the corresponding element is maintained even if one or more attributes are changed or the whole element is deleted, only new elements have no connection. An important aspect of the suggested concept is that also edges are linked to their parent edges. This provides the possibility to identify moved (or swapped) elements, a change that remains unrecognised by version comparison of many well-known BPM suites (e.g., Signavio or Aeneis).



(a) Blocked (b) Added

(c) Modified (d) Deleted

Figure 1: Marked Elements.

The number of variants derived from the main process model is not restricted. In some cases, however, a desired process variant has more similarities (and dependencies) with an existing variant than with the main process model. AdaVM addresses this issue by supporting single inheritance between variants, i.e., a variant is derived from another variant. The result is a tree-structured hierarchy of variants with the main process model as the root (see Figure 2). Inherited elements of a derived variant are always linked to the corresponding elements of its parent process model (e.g., elements of variant V2.1 are linked to elements of variant V2 which are in turn linked to elements of main process model MPM). In addition, every process model passes on all inherited blocking constraints without modification to its derived process models and may add further restrictions. For example, variant V2 inherits blocking constraints from the main process model MPM and may define additional blocking constraints. All blocking constraints are passed on to the derived variants, i.e. V2.1, V2.2 (and subsequently to V2.2.1).

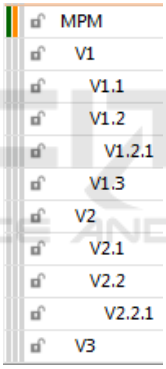


Figure 2: Hierarchy of Main Process Model and Variants.

The last requirement addressed by AdaVM is the propagation of changes in the main (or parent) process to derived variants with notification of the responsible person. Since variability is concerned with design- and customization-time decisions, no runtime adaptations are expected. The responsible person of a derived variant is the process owner. The process owner has the permission to create/modify the owned process model and to read all parent and derived processes. After every change in a process model (e.g., in the main process model MPM), all directly derived variants (V1, V2, and V3) are searched for an element that is linked to the concerned element. If such an element exists, the process owner of the variant is notified about the change and can either accept or decline it. If the change is accepted, then the next hierarchy level is investigated and further sub-variants (V1.1, V1.2, V1.3 as well as V2.1, V2.2) are considered. The change is forwarded until either a process owner declines the change, no corresponding element is found, or the leaves of the hierarchy are reached.

Finally, AdaVM implements rule-based graph transformation techniques to automatically generate process variants. The main advantages of graph transformation techniques are that context-specific variants are dynamically generated and that modifications are not only defined for one variant but used to generate several process variants (i.e., only a few graph transformation rules are required to produce a large number of process variants). In AdaVM, the algebraic graph transformation approach described in Section 2.1 is applied. The algebraic approach supports the above defined change patterns: an element is inserted if it is only part of R (right-hand side), an element is deleted if it is only part of L (left-hand side), and an element is modified if it is part of L and R but the values of one or more attributes differ. A graph transformation system can then either support variability by restriction (configurable approach) or variability by restriction and by extension (adaptable approach). A comparison of two graph transformation systems implementing these techniques is provided in the following section.

4 GRAPH TRANSFORMATION

In this section, the concepts of AdaVM are discussed within a case study. The case study comprises a business process for car services with various variants for different engine types (i.e., fuel, diesel, electro) and a distinction between basic (i.e., small) and extended (i.e., large) services. All variants are generated by applying algebraic graph transformation. An advantage of graph transformation is the possibility to deal with the exponentially increasing number of variants with a linearly increasing number of graph transformation rules, e.g., four activities each with four variations may lead to 256 variants, which can either be defined manually or automatically generated by 16 graph transformation rules (in ideal case). To compare the configurable with the adaptable approach, two graph transformation systems have been implemented in the tool AGG. The first graph transformation system is called RM_GTS, since the source graph defines a comprehensive reference model (RM) that comprises all possible process flows. The second graph transformation system is called VM_GTS, since the source graph comprises the main (or default) variant model (VM). For demonstration and evaluation purposes, the two graph transformation systems implement a subset of the AdaVM concepts (i.e., they

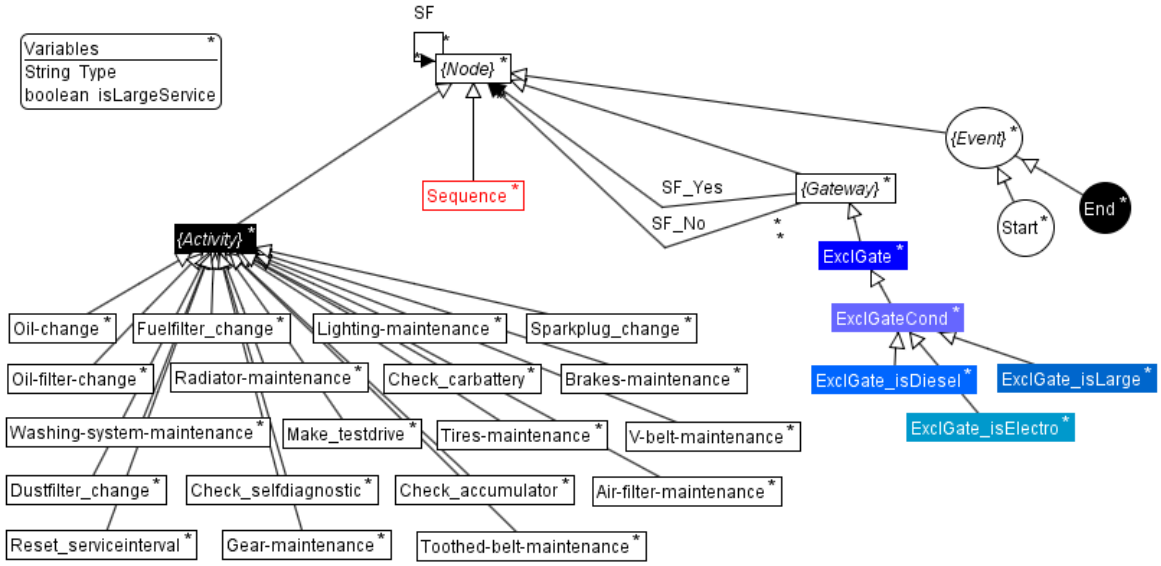


Figure 3: Type Graph of *RM_GTS*.

address the first and second requirement without links and blocking). The full approach is currently implemented in the BPM suite *FireStart*. Both graph transformation systems prove the feasibility of the suggested concepts but show differences regarding the complexity.

The type graph of *RM_GTS* is shown in Figure 3. *RM_GTS* requires abstract nodes and inheritance relations within the type graph, so an attributed type graph with inheritance is defined. The type graph provides four abstract node types (*Node*, *Activity*, *Gateway*, and *Event*) as highlighted by curly brackets and italic font. Derived from node type *Activity* are all possible tasks of a car service (e.g., *Oil-change*, *Oil-filter-change*, etc.). Abstract node type *Event* is the parent of two concrete node types for *Start* and *End* events, and node type *Sequence* is used as 'dummy' element to remove several subsequent activities. Derived from exclusive gateways without condition (*ExclGate*) and with condition (*ExclGateCond*) are variant-specific exclusive gateways checking whether diesel, electro, or large service is desired (i.e., configurable elements). One further node type is called *Variables* and used to store information regarding the desired variant including attributes for engine type and whether it should be a small or large service. Finally, the type graph provides an edge label called *SF* for general sequence flows and two further edge labels *SF_Yes* and *SF_No* applied only after splitting exclusive gateways. The type graph of *VM_GTS* is similar, but node types *Sequence*, *ExclGate_isDiesel*, *ExclGate_isElectro*, and *ExclGate_isLarge* are not required and omitted.

An example source graph of *RM_GTS* is shown in Figure 4. The process flows of all possible variants are provided within one reference model and variant-specific gateways are used to specify differences. The desired variant is defined by element *Variables* (variant: large service for car with electro engine).

For transformation of a source to a target graph, 30 graph transformation rules with corresponding application conditions have been defined in *RM_GTS*. Six of these rules are general and consider conditional exclusive gateways. In addition, eight concrete rules have been defined for each variant-specific gateway (diesel, electro, large). The aim of these transformation rules is to generate the desired variant and to remove variant-specific gateways. The rule in Figure 5, e.g., realises a large service by taking over the activity from the *Yes*-Path and by removing the *No*-Path and the surrounding gateways. A positive application condition ensures that the rule is only applied if a variant with a large service should be generated.

The resulting target graph after having applied all possible transformation rules is shown in Figure 6. The transformation has replaced all fuel- and diesel-specific activities with the activity *Check_accumulator* for electro engines. In addition, activity *Gear-maintenance* for a large service was included in the variant. Note that the other exclusive gateway for large service was removed, since *Toothed-belt-maintenance* and *Air-filter-maintenance* are only relevant for fuel and diesel engines. In sum, six variants can be automatically generated by changing the attribute values in the *Variables* element.

The second graph transformation starts with a default variant model and uses variability by restriction and by extension to automatically generate fur-

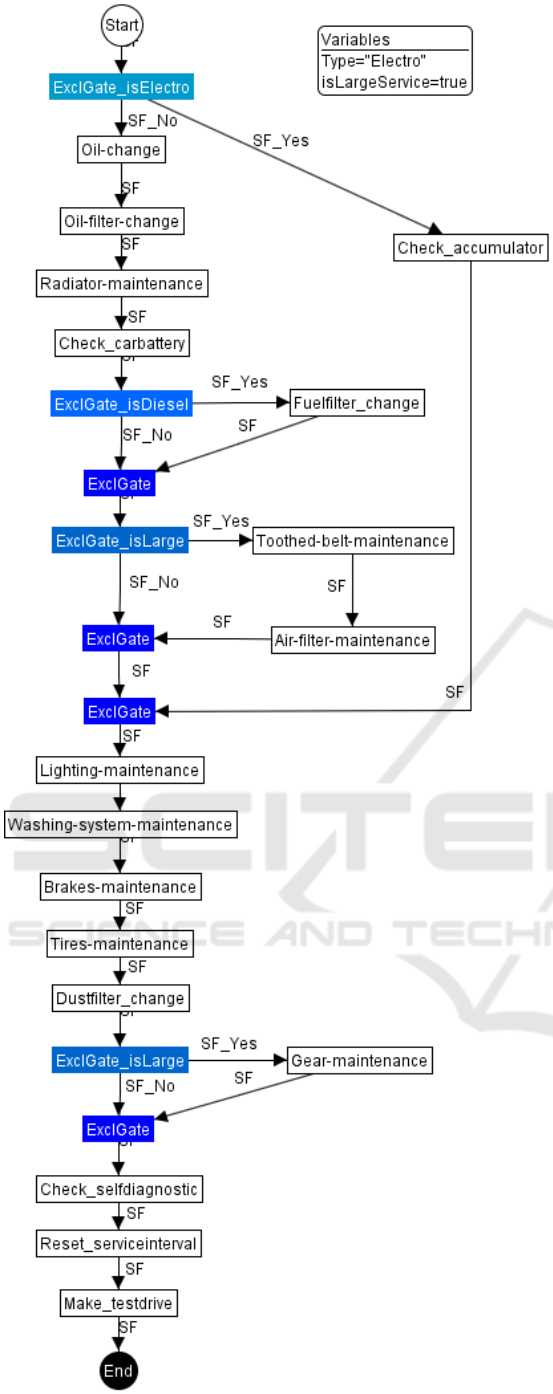


Figure 4: Source Graph of *RM_GTS*.

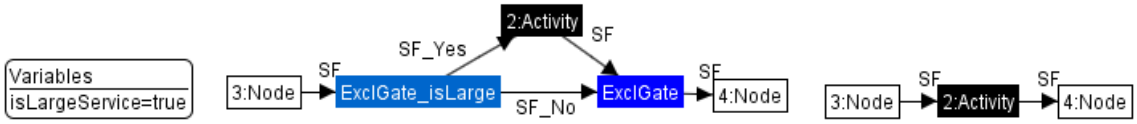
ther variants. An example source graph of *VM_GTS* is shown in Figure 7 and describes the default variant for fuel engine and small service. An advantage of the adaptable approach is that the source graph may already provide an executable variant. The *Variables* element again defines the desired variant (variant: large service for car with electro engine).

For transformation of a source graph to all possible variants, only four graph transformation rules with application conditions have been defined in *VM_GTS*. Each transformation rule actually corresponds to one variant-specific gateway in the reference model of *RM_GTS*. The graph transformation rule shown in Figure 8, for example, inserts activity *Gear-maintenance* in case of a large service (PAC) if it has not yet been inserted (NAC). The application of all possible transformation rules results in the same target graph as shown in Figure 6.

The presented graph transformations are restricted to structured diagrams (i.e., gateways are defined in a bracket structure) and a basic set of process modelling elements. However, further modelling elements like sub-processes, additional events, or gateways are only relevant for the business process and may appear in process variants, but their influence on variant generation is limited. For a more detailed discussion on configurable nodes and their influence see (Reichert and Weber, 2012, p. 96ff).

According to Varró et al., the most important correctness properties of a trusted model transformation are termination, uniqueness (confluence), and behaviour preservation (Varró et al., 2006). Behaviour preservation is not the aim of *RM_GTS* and *VM_GTS*, since different variants with different behaviour are generated. Thus, validation of the proposed graph transformation systems is accomplished by proving confluence and termination. These two properties ensure that the resulting graph is always the same, independent of the order of applied rules, and that rules are not cyclic.

In order to prove confluence, it is either necessary to show that each rule pair is parallel independent for all possible matches or, in case of parallel dependent rules, that the GTS is terminating and locally confluent. A GTS is locally confluent if all its critical pairs (parallel dependent and minimal) are strictly confluent (Ehrig et al., 2006, p. 59ff, 144). *RM_GTS* is terminating, since all transformation rules delete elements like gateways or activities. Only node type *Sequence* is inserted by some rules but subsequently removed by general rules. The full termination proof comprising calculated rule, creation, and deletion layers is not presented due to space limitations. Furthermore, all critical pairs of *RM_GTS* are of the same rule and same match and, thus, isomorphic. Hence, *RM_GTS* is locally confluent and terminating, so the whole graph transformation is confluent. The critical pairs of *VM_GTS* comprise different rules, but strict AC-confluence could be proven for each pair. A more challenging task was to prove termination of *VM_GTS*, since all transformation rules insert new



(a) PAC (b) LHS (c) RHS

Figure 5: Transformation Rule of *RM_GTS*.

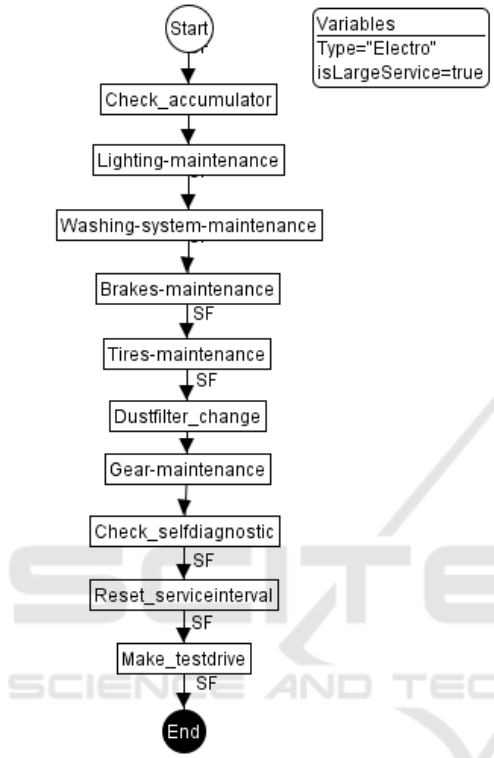


Figure 6: Target Graph [Variant: Electro, Large Service].

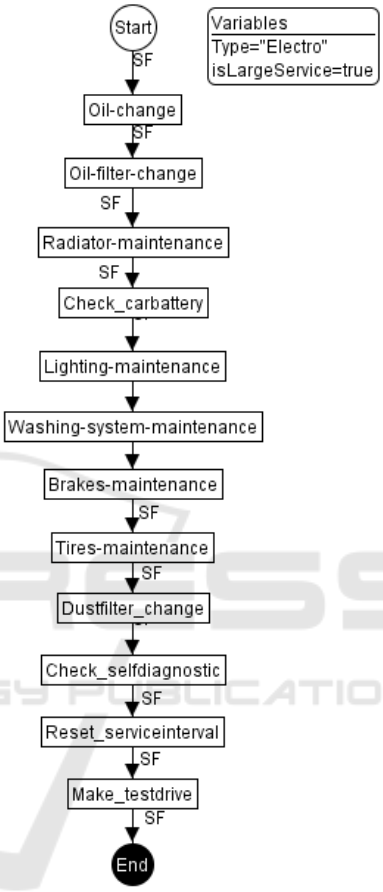
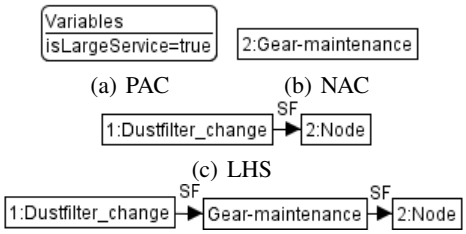


Figure 7: Source Graph of *VM_GTS*.



(a) PAC (b) NAC

(c) LHS

(d) RHS

Figure 8: Transformation Rule of *VM_GTS*.

node and edge types. Termination is ensured by negative application conditions (NAC: *NotYetInserted*). However, since sequence flows are deleted in every rule, all rules are automatically classified as deletion layers, thereby ignoring the NACs. Thus, we propose to extend the deletion layer conditions presented in (Ehrig et al., 2006, p. 31) to also consider NACs. We further flattened the hierarchy of *VM_GTS* with the flattening algorithm presented in (Natschläger and Schewe, 2012) to 598 rules without inheritance relationships and proved termination for all node types. Summing up, both graph transformation systems are confluent and, thus, globally deterministic.

We then compared the graph transformation systems *RM_GTS* and *VM_GTS* regarding their complexity. Complexity is assessed by the number of nodes in the type and source graphs as well as by the amount of transformation rules and application conditions. The complexity of *RM_GTS* and *VM_GTS* is shown in Figure 9. Interestingly, *VM_GTS* is less extensive in all measured aspects and, thus, also less complex. The reason is that the rules of *VM_GTS* are more specific and explicitly define concrete activities, e.g. *Gear-maintenance*, whereas the rules of *RM_GTS* use abstract node type *Activity* to represent

all possible concrete activities. Thus, an extension of the process with a new variation would require an adaptation of (every) reference model in *RM_GTS*. In contrast, *VM_GTS* would either require an extension of the default variant or (more likely) a new/extended transformation rule. So, due to the differences in required adaptations, assessing the complexity of future extensions is difficult. However, it can be assumed that further extensions mainly increase the complexity of source graphs of *RM_GTS* and the amount of rules and application conditions of *VM_GTS*.

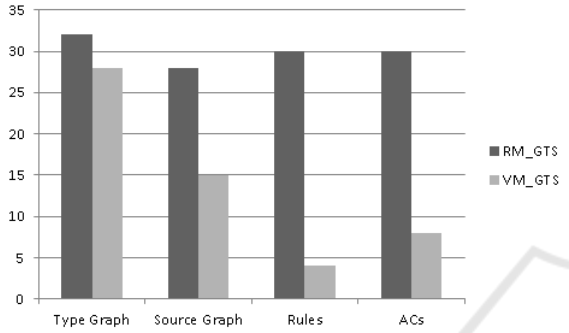


Figure 9: Complexity of *RM_GTS* and *VM_GTS*.

Summing up, we showed that graph transformation techniques are well-suited for variant management. We, therefore, implemented two graph transformation systems for generating process variants; one supporting only variability by restriction, the other supporting variability by restriction and by extension. Both graph transformation systems were proven to be globally deterministic. Interestingly, the adaptable approach was easier to implement (as it is more specific) and it is less complex regarding the type graph, source graphs, and the amount of rules and application conditions. Thus, we recommend to apply both techniques, variability by restriction *and* by extension, for generating process variants. Other concepts of AdaVM (e.g., linking, propagation of changes, and visual highlighting of differences) do not require verification and will be directly implemented in BPM suite *FireStart*.

5 CONCLUSION

In this paper, we presented our concept for *Adaptive Variant Modelling* (AdaVM) as part of the AdaBPM framework for advanced adaptivity and exception handling in formal business process models. The suggested approach requires fewer modifications of the underlying notation, i.e., no configurable elements or adjustment points are required, only an individual blocking of elements is proposed. In addition, AdaVM considers linking of elements, propagation of changes, and visual highlighting of differences in process variants.

We further demonstrated that graph transformation techniques are well-suited for process variant management and that variants can be automatically created by a few graph transformation rules specifying concrete variations. For that purpose, we provided a case study implementing two graph transformation systems; one supports variability by restriction, the other supports variability by restriction *and* by extension. The first system follows the *configurable* approach by defining the source graph as an extensive basic model, which comprises all possible process flows and can be configured to create variants. The second system follows the *adaptable* approach and the source graph comprises the main (default) variant model, which can be adapted by structural model adaptations to obtain new variants. We proved both systems to be globally deterministic but encountered differences regarding the complexity of the graph transformation systems. Interestingly, the case study reveals that the adaptable approach is less complex regarding the type graph, source graphs, and the number of rules and application conditions. Thus, we intend to apply both techniques, variability by restriction *and* by extension, in AdaVM. Nevertheless, our next step is to extend the case study to elaborately evaluate the complexity of configurable and adaptable approaches.

Summing up, the key differences of the proposed concept for variability modelling, compared to other state-of-the-art approaches, are (i) the support of variability by restriction *and* by extension with graph transformation techniques, (ii) linking and propagation of changes, (iii) individual blocking of elements/attributes, and (iv) visual highlighting of differences in process variants. We, thus, expect that the variant management implemented by BPM suite *FireStart* outperforms other applications with respect to functionality and usability.

REFERENCES

Aalst, W. v. d. (2013). Business process management: A comprehensive survey. *ISRN Software Engineering*, 2013:1–37.

Aeneis (2014). *aeneis Handbuch*. intellior AG, aeneis 5.7 edition.

Axon IVY (2015). *Axon.ivy 5.1 - Designer Guide*. ivyTeam AG.

Bizagi (2014). *Bizagi BPM Suite - User Guide*. Bizagi.

Doehring, M., Zimmermann, B., and Karg, L. (2011). Flexible workflows at design- and runtime using BPMN2

adaptation patterns. In Abramowicz, W., editor, *Business Information Systems*, volume 87 of *LNBIP*, pages 25–36. Springer Berlin Heidelberg.

Ehrig, H., Ehrig, K., Prange, U., and Taentzer, G. (2006). *Fundamentals of Algebraic Graph Transformation*. Springer.

Gottschalk, F. (2009). *Configurable Process Models*. PhD thesis, Technische Universiteit Eindhoven.

Hallerbach, A., Bauer, T., and Reichert, M. (2009). Guaranteeing soundness of configurable process variants in provop. In *IEEE Conference on Commerce and Enterprise Computing (CEC)*, pages 98–105.

Hallerbach, A., Bauer, T., and Reichert, M. (2010). Capturing variability in business process models: The provop approach. *Journal of Software Maintenance and Evolution: Research and Practice*, 22:519–546.

Natschläger, C., Kossak, F., and Schewe, K.-D. (2015). Deontic BPMN: a powerful extension of BPMN with a trusted model transformation. *Software & Systems Modeling*, 14(2):765–793.

Natschläger, C. and Schewe, K.-D. (2012). A flattening approach for attributed type graphs with inheritance in algebraic graph transformation. *Electronic Communications of the EASST*, 47.

Prologics (2014). *FireStart Tutorial - Modellierung & Ausführung*. Prologics.

Recker, J., Rosemann, M., van der Aalst, W., Jansen-Vullers, M., and Dreiling, A. (2006). Configurable reference modeling languages. In Fettke, P. and Loos, P., editors, *Reference Modeling for Business Systems Analysis*, pages 22–46. IGI Global, Pennsylvania.

Reichert, M. and Weber, B. (2012). *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer-Verlag Berlin Heidelberg.

Rosa, M. L., van der Aalst, W., Dumas, M., and Milani, F. (2013). Business process variability modeling: A survey. Technical report, Queensland University of Technology.

Scheer (2015). *Scheer BPaaS*. Scheer.

Signavio (2015). *Benutzerhandbuch*. Signavio.

Varró, D., Varró-Gyapay, S., Ehrig, H., Prange, U., and Taentzer, G. (2006). Termination analysis of model transformations by Petri nets. In Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., and Rozenberg, G., editors, *Graph Transformations*, volume 4178 of *Lecture Notes in Computer Science*, pages 260–274. Springer Berlin / Heidelberg.

Weber, B., Reichert, M., and Rinderle-Ma, S. (2008). Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering*, 66(3):438–466.