Partial Process Models to Manage Business Process Variants

Emilian Pascalau
Hasso-Plattner-Institute,
University of Potsdam, Germany,
E-mail: emilian.pascalau@hpi.uni-potsdam.de

Ahmed Awad
Hasso-Plattner-Institute,
University of Potsdam, Germany,
E-mail: ahmed.awad@hpi.uni-potsdam.de

Sherif Sakr
National ICT Australia (NICTA),
University of New South Wales, Australia
E-mail: ssakr@cse.unsw.edu.au

Mathias Weske
Hasso-Plattner-Institute,
University of Potsdam, Germany,
E-mail: mathias.weske@hpi.uni-potsdam.de

Abstract
Today’s process aware information systems deal often with the problem of variants. Variants of process models have to be defined frequently due to several reasons such as: the need to target different customer types, rely on particular IT systems or comply with specific country regulations. Management of process models variants that addresses issues such as consistency between process variants, uncontrolled redundancy, huge modeling efforts, etc is not adequately supported by current Business Process Management (BPM) tool. Thus an automated maintenance of process variants able to tackle the mentioned issues is a well coveted goal. This paper presents an approach to address the issues of providing consistent mechanisms for managing processes variants consistency and reducing the redundancy between process variants in order to achieve a more efficient and effective business process modeling Task. Our approach, Partial Process Models (PPM), is a query-based approach which maintains the link between the variant process models by means of defining process model views. These views are defined using, BPMN-Q, a visual query language for business process models and employ a software engineering like inheritance methodology. Thus, dynamic evaluation for the defined queries of the process views guarantee that the process modeler is able to get up-to-date and consistent status of the process model. The PPM approach provides a flexible way to deal with process models variants both via a top-down as well as a bottom-up approach.

Keywords: Business process design, Reuse, Querying business processes, Process variants.


Biographical notes:
Emilian Pascalau is a PhD student in the "Service-oriented Systems Engineering” Research School at Hasso Plattner Institute in Potsdam, Germany. There he is a member of the Business Process Technology Group (BPT). He received a Diploma in Computer Science from the University of Craiova, Romania and a Master degree in "Distributive Systems in Internet and Intranet” from Babes-Bolyai University, Cluj Napoca, Romania.
1 Introduction

In the last years companies have developed a growing interest for better processes, better interactions with their customers and business partners. There has been an increased adoption of business process management (BPM) technologies and tools. Nevertheless new technologies bring also new challenges.

Business Process Management (BPM) \cite{38} aims at the automated support and coordination of business in an integrated manner by capturing, modeling, implementing and controlling all activities taking place in an environment that defines the enterprise. Business processes enable a better understanding of the business by facilitating communication between business analysts and IT experts. However, because of the very complex environments of today’s business, business processes, like many information systems, do not exist only under a single version which covers all the issues or the whole market. Instead, many variants of a process exist which are specialized for particular customer types, or for particular IT systems, or some country-specific regulations. Thus multinational companies have to keep variants of business processes in order to be compliant with local regulations or domain specific settings. We can establish an analogy between process variants on the one hand and object oriented inheritance on the other hand. A process variant is like a child class, where a process variant (child) extends or overrides the behavior of the parent process. In many cases, these variants are maintained manually, even in the case of big companies. The risk of inconsistency as well as the efforts required to improve and maintain over time these processes is huge. Inconsistency appears when a parent process’s behavior is updated without updating the child’s behavior accordingly.

Complex business environment, such as eBay business environment where country regulations, different IT systems and so forth could produce only for business process more than 8000 variants require special management methodologies that have to be simple, yet powerful enough to tackle a wide range of issues which concern process variants. Current approaches that deal with process variants such Provop \cite{11} or C-EPCs \cite{28} can provide only partial support for complex business environments such as the eBay environment.

In this article we discuss at large a query-based approach to tackle the problem of business process variants management. While our approach is mainly focusing on tackling the issue of automated consistency maintenance between process variants, several other issues concerning process variants are addressed as well. In particular, instead of the manual save-as style of processes to create variants, we keep the link between child and parent processes by means of defining views in child processes that reflect the behavior of parent processes. These views are created by means of queries. Thus, a process variant combines concrete activities that are meant to provide the behavior specific to the new variant and queries that inherit behavior from parent processes. To model process variants this way, we introduce so-called Partial Process Models (PPMs). For the concrete part of the variant, ordinary process modeling constructs are used, e.g., BPMN constructs. For the view part, we rely on BPMN-Q, a visual language for querying business processes models \cite{2,29}. In particular, each time a child process is invoked for editing or execution, the defined queries (views) of the child process is evaluated against the parent process and an up-to-date result is returned to the modeler. This view-based approach has a two fold meaning: (1) Maintaining the consistency between variants of process models and (2) Extracting views for different process variants from holistic process models. Moreover, using queries to link variants to their parent processes provides more flexibility in designing one process variant as a child for more than one parent (e.g. inheritance chain or multiple inheritance). In
Partial Process Models to Manage Business Process Variants

principle, this is a unique advantage for our approach in comparison to the state-of-the-art of existing approaches [15, 11].

This paper improves the approach published in [21]. The major contributions of this work are:

- A discussion of a set of requirements extracted from real life scenarios to manage business process variants
- An approach to manage variants and their consistency based on a view-based approach to define variants. We call this approach partial process modeling.
- Evaluation and comparison of our approach to existing approaches that manage process variability.

The remainder of this paper is organized as follows: We discuss some background knowledge about business process models and BPMN-Q in Section 2. To illustrate the problem, we discuss several real-world use case scenarios (online trading: eBay; business process compliance) in Section 3. The sections concludes with the set of requirements for our approach. Section 4 describes the details of our approach regarding the maintenance of variants consistency through the definition of process model views. An architectural framework that realizes the implementation of our approach is presented in Section 5. Section 6 evaluates our approach in connection with current approaches for variants management. The related work is discussed in Section 7 before we conclude the paper in Section 8.

2 Preliminaries

This section formally introduces process modeling and querying concepts which form the groundwork for our approach.

2.1 Business Process Modeling

Currently, there is a number of graph-based business process modeling languages, e.g. BPMN [19], EPC [12], YAWL [36], and UML Activity Diagram [20]. Despite their variance in expressiveness and modeling notation, they all share the common concepts of tasks, events, gateways (or routing nodes), artifacts, and resources, as well as relations between them, such as control flow. Without loss of generality, we can abstract from particular node types as their execution semantics are not vital to structural query matching, which is rather based on the concept of a process graph.

Definition 2.1: (Process Model) A process model $P$ is a connected graph $(N, E)$, where $N$ is a non-empty set of control flow nodes and $E \subseteq N \times N$ is a set of directed control flow edges where $n (n \bullet)$ stands for the set of immediate predecessor (successor) nodes of $n \in N$.

A process model has exactly one start event $n_{\text{start}} \in N$ with no incoming and at least one outgoing control flow edge, i.e., $\bullet n_{\text{start}} = 0 \land |n_{\text{start}} \bullet| \geq 1$, and at least one end event $n_{\text{end}} \subseteq N$ with at least one incoming and no outgoing control flow edge, i.e., $\forall n \in n_{\text{end}} : | \bullet n_{\text{end}} | \geq 1 \land |n_{\text{end}} \bullet| = 0$. Each other control flow node $n \in N \setminus \{n_{\text{start}} \cup n_{\text{end}}\}$ is on a path from $n_{\text{start}}$ to some end event node $n_{\text{end}} \in n_{\text{end}}$. $P$ is the set of all process models.

A connected sub-graph of a process model is a process model fragment. We refer to a specific type of process model fragments that have a single entry node and at least one exit node as process model components.

Definition 2.2: (Process Model Component) A connected subgraph $(N', E')$ of a process model $(N, E)$, where $N' \subseteq N$, $E' \subseteq E$, is a process model component $PC$ iff it has exactly one incoming boundary node $n_{\text{in}} \in N'$, i.e., $n_{\text{in}} \subseteq N \setminus N'$ and at least one outgoing boundary node $n_{\text{out}} \in N'$, such that, $n_{\text{out}} \subseteq N \setminus N'$.

2.2 Business Process Model Querying

Based on the definition of process models and process model components, we introduce the concept of process model queries, as a means to obtain business process components from a collection of business processes by structurally matching a query to each of them. BPMN-Q is a visual process model query language designed to help business process designers access repositories of business process models [2]. The language supports querying all the control and artifact concepts of business process models. Moreover, it introduces a set of new abstraction concepts that are useful for different querying scenarios.

Definition 2.3: (BPMN-Q Query) A BPMN-Q query is a tuple $Q = (QC, QCF, QP, exclude)$ where:

- $QC$ is a finite set of control flow nodes in a query,
- $QCF \subseteq QC \times QC$ is the control flow relation between control nodes in a query,
- $QP \subseteq QC \times QC$ is the path relation between control nodes in a query,
- $\text{exclude} : QP \rightarrow 2^{QC}$ is a function that determines nodes to be excluded from the paths.

A BPMN-Q model is called a query [2]. A query declaratively describes a structural connectivity that must be satisfied by a matching process model. In addition to the core business process modeling concepts, BPMN-Q introduces a new concept of Path edges. A path edge connecting two control flow nodes represents an abstraction over an arbitrary set of control flow nodes that could exist in-between in the matching process model. Moreover, a path edge has an exclude property. That is, one might be interested in paths from node $A$ to node $C$ without visiting node $B$.

2.3 Matching Queries to Processes.

A BPMN-Q query is matched to a candidate process model via a set of refinements to the query. With each refinement node (edge) in a query is replaced with the corresponding node (edge) of the matching process model. If one node can have more than one possible replacement within the process model, a new refined copy of the query is created for each possible replacement. We call the replacement a resolution of an element of the query. Figure 1(a) illustrates a sample process model definition using the BPMN notations, Figure 1(b) illustrates a sample definition of a process model view using
the BPMN-Q notations. The nodes and edges highlighted in gray in Figure 1(a) illustrate the matching part of the process model. To evaluate path edges, we rely on the transitive closure of nodes regarding the control flow relation.

**Definition 2.4:** (Reachability) Let \( P = (N, E) \) be a process model. Let \( E^* \) be the transitive closure over the control flow relation \( E \). For each node \( n \in N \), we define the set of reachable nodes from \( n \) as \( [n]_{E^*} = \{ m : (n, m) \in E^* \} \).

Relying on Definition 2.4, we can define the reachable nodes from some node \( x \) under different conditions. For instance \( [x]_{E \setminus \{(x, y)\)}^* \) defines the reachable node from node \( x \) where we removed the edge \((x, y)\) from the control flow relation \( E \). In light of nodes’ reachability, a path edge \((source, target, EXCLUDE)\) evaluates to a sub-graph in which all nodes are: 1) Reachable from source node, 2) target node is reachable from every node, 3) every node \( e \in EXCLUDE \) is not in the sub-graph and every node solely reachable by \( e \) is not in the sub-graph either 4) edges between nodes in the sub-graph are those between these nodes in the process model.

A process model component \((N', E')\) of a process model \((N, E)\) is a resolution to the path edge \((source, target, EXCLUDE)\), if
- \( N' = \{ n : n \in [source]_{E \setminus \bigcup_{e \in EXCLUDE} (e,-)}^* \land target \in [n]_{E \setminus \bigcup_{e \in EXCLUDE} (e,-)}^* \} \)
- \( E' = \{ (x, y) : x, y \in N' \land (x, y) \in E \} \)

A process model is said to match the query if it satisfies all control flow and path edges as described in Definition 2.5.

**Definition 2.5:** (Matching) A process model \( P = (N, E) \) matches a query \( Q = (QC, QCF, QP, \text{exclude}) \) if:
- \( QC \subseteq N \) control flow nodes in the query find resolvents in the process,
- \( QCF \subseteq E \) Control flow relations in the query are included in the process,
- \( \forall p \in QP \exists (N', E') : \text{sub-graph}(N', E') \text{is not empty} \),
- \( \exists p \in QP \exists (N', E') : \text{sub-graph}(N', E') \text{is not empty} \).

3 Motivating Scenarios & Requirements

In this section we discuss a set of motivating scenarios that encouraged the development of our approach to model and manage process variants.

3.1 Online Trading: eBay

With more than 90 million active users globally, eBay is the world’s largest online marketplace. eBay connects individual buyers and sellers, as well as small businesses in 38 markets using 16 languages (see [http://www.ebayinc.com/whd](http://www.ebayinc.com/whd) June 6th 2010). eBay has huge repositories of business processes. Though, many of these processes are variants of other processes. We can argue that variability is imposed on a vertical axis (represented by different departments within the organization) and on a horizontal axis (embodied by different business elements and/or business aspects, i.e., regulations, IT infrastructure, for example different Customer Relationship Management systems, customer types, countries, payment methods).

The number of possible process variations is determined by the degree of freedom the system has, i.e., the number of possible arrangements of different business contexts. A business process that is influenced by 6 business context elements \( \{b_1..b_6\} \), e.g., country, region, etc, that respectively have the following number of subtypes \( \{8, 2, 5, 5, 3, 7\} \), will end up having more than 8000 variants. Figure 2 which contains a small excerpt from an eBay taxonomy of business elements, exemplifies such a process influenced by six business contexts. eBay Customer Support provides individual services to buyers and sellers striving for the best possible individual customer service [22]. Such a goal requires mass customization.

Processes need to support Customer Support Representatives (CSR) with tailored variants. As an example a Spanish business seller having an issue with mass listings would need a completely different support than an American casual customer trying to sell his first item. Although both cases are dealing with the same question on how to list an item on eBay. Both cases would require different service levels, different entry points such as local phone numbers or web forms and of course different answers in different languages from different teams [22].

Currently two approaches are basically used at eBay: clean sheet approach and branching approach [22]. With the
clean sheet approach, the process variant gets redesigned from scratch. Therefore, this approach gets applied when current process variants are not considered to be reasonable, i.e. the business elements, the business context is quite different. With the branching (save as) approach, a process variant gets copied entirely and then is modified to fit the business needs of the new variant. It is important to underline that in this case a complete new process documentation is created and managed separately.

It should be noted that in both cases the issues of inconsistency is not addressed. Variants of the same process model are decoupled from each other, although they are semantically connected. Semantically in the sense that they address the same problem but the business context is slightly different.

To illustrate the inconsistency case, we are going to use the two processes from Figure 5. These processes are two real life variants of an eBay process model in the context of customer support. As the labels in the figures state one of the models is called a Parent process and the second one is called a Child process. A child process can reuse either parts or an entire parent process. The terminology of child and parent is related to the inheritance concept, as the child (sub) process reuses behavior from the parent (super) process, similarly to how subclasses reuse (inherit) functionality from the superclasses. Any arbitrary process can be used as a parent process. Currently, a child process is derived by making a copy of the parent process and editing that copy, e.g., adding new activities or arbitrary control flow elements. At this point, there is no connectivity between the parent and the child processes. That is, a parent process could be edited by another modeler who might add new functionality without it being reflected on the child process, thus causing inconsistency between the child and parent processes. Moreover, overriding behavior in the parent process is also not tracked.

3.2 Configuration of Reference Models

The concept of reference process models has at least two interpretations. According to one interpretation [11] the meaning of a reference process model corresponds to a set of policies: (1) the process model in cause is a domain specific standard; (2) is the most used process variant; (3) based on a structural mining of a collection of process variants the reference process model is the one for which the average distance to the variant becomes minimal; (4) is a superset of all process variants; (5) is the intersection of all process variants.

The second major interpretation for a reference process model is the one defined in [28]. The big picture as it is called comprises the entire list of variants into one big model. This big model is reduced later into relevant parts based on the needs. We rely on this interpretation for a reference process model. In the case of the second interpretation, modelers normally start with holistic models that contain the configurable nodes [28]. Variants correspond to specific configurations. Management of process variants by means of holistic models requires basically two major things: (i) the existence of the holistic model and (ii) the beforehand knowledge about which elements can be configured and what will be the outcome of the configuration.

For exemplification Figure 4 depicts two eBay variants out of which a holistic model needs to be created. There are at least two possibilities to create holistic models from the variants of processes. One is to purely merge the variants, by simply "overlapping processes" and keeping only one copy of the elements that are the same (type, label and position in the control flow) in all processes, and adding all other different elements. The holistic model obtained by such an approach would not guarantee the maintenance of the initial variants' behavior. Behavior maintenance is a requirement of the approach proposed in [28].

Figure 5 depicts the holistic model of the two variants from Figure 4 created by simple merge. The holistic model itself created using this approach does not maintain the initial behavior specified in the two separate variants. A second modality to create the holistic model would be to add control flow elements that would allow the holistic model to maintain also the individual variants' behavior. Such a holistic model maintaining the initial behavior is presented in Figure 6.

Figures 5 and 6 only sketched out how holistic models could be created. For an in depth approach that tackles an automatic way to create holistic process models one could refer to [28]. Holistic process models by themselves do not resolve the problem of variants as these have to be extracted from the reference models. Current approach [28] uses configurable nodes and special logical constructs to identify and configure (extract) a proper variant.

Due to the complexity of the eBay context such an approach would be almost an impossible endeavor. The modeler would be required each time to traverse the entire process model from start to end and to know the configuration points and all the variants in order to configure (extract) the required variant(s). Thus the question that arises is: "How to extract process variants from holistic models in a more comfortable and consistent way?".

3.3 Business Process Compliance

Business processes have to adhere to compliance requirements enforced by regulations such as SOX [1]. In response, process- and compliance-experts develop process templates that are compliant by design [32]. Each newly designed process model, related to the compliance requirements, must inherit behavior from these templates. Thus, such new process models represent variants over the compliant templates. Any change to the templates must be reflected in the variants. Moreover, a variant could be related to one or more compliant template. That is, multiple inheritance is needed to manage variants.

A newly designed process model contains a view on the compliant template [33]. So, a systematic approach is needed to manage the templates and complete processes that depend on them. Changes in the templates must be reflected to the detailed processes and inconsistencies must be located and reported. Otherwise, the company might be subject to penalties due to the violation of compliance requirements. One of the major requirements with managing compliance templates is that the inherited component of the compliance template must be distinguished within the child process. This is required in
order to give the process modeler and practitioner the awareness of the compliance requirements. Thus, it is not possible to allow editing such parts in the child process. Moreover, as the operational process model might be subject to more than one set of compliance requirement, i.e., needs to inherit from more than one process template, the process model becomes polluted [33] and in specific settings it is necessary to abstract from some process details.

Figure 7 shows a compliant template to handle loan applications within a bank. As illustrated, a component of that model, loan application handling, is required to be reused and respected by operational process models. Similarly, Figure 8 shows another template in which the check data completeness component is intended for reuse. Finally, Figure 9 shows an operational model that is intended to inherit and reuse the two different components from Figure 7 and 8.
3.4 Requirements for Managing Process Inheritance

Based on the three use cases explained above, in this section, we discuss briefly the major requirements that guided the contribution in this paper.

- **Defining variants as separate process models:** As indicated by the use case in Section 3.1 the number of variants, based on the context might be huge. Thus, it is unrealistic to keep all variants stored within one model, cf. [11]. Rather, each variant should be maintained as a separate artifact.

- **Maintaining Consistency:** This is the core requirement. Each variant must maintain a link with its parent process(es). Moreover, updates on the parent process that are relevant to the variant (child) must be reflected on the child process.

- **Allowing variants to override parent processes:** Variants can restrict, extend or override the behavior inherited from the parent processes.

- **Multiple level inheritance:** The inheritance lattice can be of arbitrary depth. That is, a variant process $c$ can inherit from another variant process $b$ that in turn inherits from a parent process $a$.

- **Multiple Inheritance:** Motivated by compliance templates discussed in Section 3.3, the modeling of process variants (child) must provide the chance for a variant to inherit behavior from more than one parent.

- **Ability to lock inherited parts against editing:** Based on the compliance templates scenario, there should be the possibility to mark the inherited part of a template as read only. That is, it is not possible for the designer of the child process to edit that part of the process.

- **Ability to hide details of parent processes:** As a variant can inherit from more than one parent. It should be possible to abstract from behavior inherited from a specific parent and just showing behavior inherited from others. For instance, there should be a possibility to present the process in Figure 9 while hiding the behavior inherited from the process in Figure 8, cf. [33].

- **Identification of process models breaking consistency:** In case that a (child) variant is inconsistent with its parent processes, the reason of inconsistency must be identified. In other words, the modeling environment must help the designer locate the process that caused inconsistency.

- **Support for bottom-up and top-down development of variants:** On one hand, the two use cases discussed in Section 3.1 and Section 3.3 identify the situation where firstly basic process models are designed and then they are refined and tuned for a specific context by means of creating variants. In both cases, a variant inherits behavior from a parent process. Yet, it might contain a new functionality to meet a specific context. We call these scenarios bottom-up development of variants. On the other hand, the customization of holistic process model, as shown in Section 3.2 is a top-down approach.
where the variants are restrictions on the behavior of the holistic model.

4 Partial Process Models to Manage Inheritance

In Section 3 we explained two problems that stem from process variants management and configurable processes which are: variants consistency and deriving variants from holistic processes. In this section, we describe our approach to address these two problems by means of defining queries on process models. In this sense, we see process queries as a means to support reuse. We use BPMN-Q to create queries (views) on other processes. However, variants may extend or limit the behavior inherited from parent processes. Thus, it is necessary as well to give the modeler the means to define new functionalities in variants. Thus, we see queries as means to reuse or restrict the behavior of a process in a variant. In the mean time, queries maintain consistency between a process and its variants. Ordinary modeling constructs are used to define new, overriding, functionality.

Section 4.1.1 introduces our approach to model variants. We call these models partial process models (PPM). Section 4.2 explains the evaluation of PPMs and how this maintains consistency among processes and their variants or indicates a problem. Finally, Section 4.3 describes how PPMs can be used to address bottom-up and top-down modeling of variants.

4.1 Defining Partial Process Models

The issue of consistency between variants cannot be managed in an automated way as long as the processes are decoupled from each other. Thus to be able to resolve the issue of consistency a proper way to relate processes with each other is required. We use queries to relate processes to each other.

To maintain the parent-child process consistency, we introduce the notion of partial process models, Definition 4.1 that describes a desired process model through a combination of process model components (fragments) and process model queries (views). Thus, to derive a child process, instead of copying the parent process and then editing that copy, the modeler starts with a partial process model. In this partial process model, ordinary process modeling constructs, e.g., activities, events, are used to model the new behavior that distinguishes the variant, we call these the concrete elements. On the other hand, to reuse behavior of the parent process, BPMN-Q queries are embedded within that partial process model.

Figure 9 A compliant operational process inheriting from templates in figures 7, 8

Figure 10 Partial process modeling approach provides coupling between process models

model. Each query declaratively describes the behavior to be inherited from a specific parent process. Next, queries and concrete parts of the process are connected via control flow edges.

Our approach is inspired by the inheritance relation from software engineering. Object-oriented design is based on the fundamental concept of reuse, reuse of software components. The Unified Modeling language (UML) [20] has been accepted as the de facto standard in the software industry for designing, modeling and documenting of object-oriented software systems. Inheritance is a key mechanism to achieve reuse. The designer through the inheritance mechanism can design a class, the subclass, that basically inherits features from a different class, its superclass. For our approach depicted in Figure 10 we take into account this basic feature reuse mechanism from Object-oriented design.

The graphical representation of our approach presented in Figure 10 emphasizes several constructs that we support through partial process models. First, the issue of decoupling is overtaken as the queries refer to processes, and thus child processes are related to their parent processes. We call a child process a view of the parent process. Process P1 is a parent process for process P2. P2 is both a child and a parent process: is a child process for P1 and a parent process for P4. Process P3 is a parent process for P4. For our queries based approach the basic inheritance feature from software engineering works differently. Here opposed to software engineering via queries several explicit relationships can exist between two processes. If, for example, in software engineering when defining a class we would be allowed to use only one of the keyword extend (i.e. class B extends class A), here we can define several queries that refer to the same process. Thus saying that class B extends class A, extends class A, ... The inheritance between processes P4 and P2 exemplifies this. P4 contains two queries that point to P2. Through this mechanism a child
process inherits exactly the behavior that is needed from the parent process, and not the entire behavior. Here, if we would have used the keyword extends instead of using queries it would have referred to a specific element or a fragment from the parent process and not to an entire process. The query contains both the link between processes as well as the mechanism to perform the inheritance, i.e. the query defines if we override behavior or just add new behavior, or restrict behavior. A finer granularity can be achieved through this mechanism. Figure 10 emphasizes also the possibility to use multiple inheritance. Process P4 inherits from both P2 and P3. We can say also that P4 is view of both P2 and P3. Our approach supports also inheritance chaining, thus P3 inherits from P2 and P2 inherits from P1.

**Definition 4.1:** *(Partial Process Model)* Let \( \mathcal{P} \) be the set of all process models. A partial process model \( \mathcal{P} = (\mathfrak{F}, \mathcal{Q}, \mathcal{E}) \) is a connected graph that consists of disjoint sets of process model fragments \( \mathfrak{F} \) and process model queries \( \mathcal{Q} \) connected through directed edges \( \mathcal{E} \subseteq (\mathfrak{F} \times \mathcal{Q}) \cup (\mathcal{Q} \times \mathfrak{F}) \), where each outgoing boundary control flow node \( n_{\text{out}} \in \mathcal{N} \) of a process model fragment \( F \in \mathfrak{F} \) is connected to at least one incoming boundary control flow node \( n_{\text{in}} \in \mathcal{Q} \) of a process model query \( Q \in \mathcal{Q} \) and vice versa. \( \mathcal{P}, \mathcal{M} \) is the set of all partial process models.

To establish the link between child processes and parent processes, each query \( Q \in \mathcal{Q} \) in a PPM \( \mathcal{P} = (\mathfrak{F}, \mathcal{Q}, \mathcal{E}) \) is assigned a reference to a specific process or another PPM. That is \( \text{ref}_{\mathcal{Q}} : \mathcal{Q} \rightarrow \mathcal{P} \cup \mathcal{P}, \mathcal{M} \). As shown in Figure 10 a partial process model may contain an arbitrary number of queries, each assigned a reference, we define a function \( \text{ref}_{\mathcal{Q}} : \mathcal{P}, \mathcal{M} \rightarrow 2^{\mathcal{P}, \mathcal{M}} \) that returns all (partial) process models directly referenced by queries within a specific partial process model. To obtain (partial) process models referenced within the inheritance lattice, we define \( \text{ref}^*_{\mathcal{Q}} : \mathcal{P}, \mathcal{M} \rightarrow 2^{\mathcal{P}, \mathcal{M}} \). In this way, the link between the child process and its parent process(es) is maintained.

Cyclic relations are not allowed. The partial process model P2 contains a query that refers to P3 and P3 contains a query that refers to P2 creating in this way a cyclic relation between the processes P2 and P3. This is not possible because in order to query a process for a particular flow the process must exist, this cyclic relationship would generate an infinite loop and hence is a forbidden construct, formally, \( \forall p \in \mathcal{P}, \mathcal{M}: p \notin \text{ref}^*_{\mathcal{Q}}(p) \).

As was stated in Section 3.4 it might be required that a specific group of users are shown only parts of the inherit behavior. Thus, in PPM, for each query it is possible to select whether to evaluate it, \( \text{eval} : \mathcal{Q} \rightarrow \{\text{true, false}\} \). Similarly, a query result can be defined as whether read-only or editable. That is \( \text{lock} : \mathcal{Q} \rightarrow \{\text{true, false}\} \). If for some query \( q \in \mathcal{Q} \) \( \text{lock}(q) = \text{true} \), at the evaluation, all resolved nodes and edges of the query will be marked as read only and the user cannot change them.

Figure 11 shows a partial process model that corresponds to the use case illustrated in Section 3. The partial process model is intended to show how the Child process of Figure 3 can be obtained and maintained from the Parent process, depicted in the same figure. In Figure 11 the parts with grey background represent new activities that are introduced on the child process. To keep the relationship with inherited behavior, queries are used. Q1 keeps the link with the parent process and activity "take call" from the parent process model is inherited. Q2 keeps the relationship with the same parent model and the behavior of the parent process between activity "ask for authentication data" on the one hand and a termination possibility and activity "ask for issue" on the other hand is inherited.

### 4.2 Evaluating Partial Process Models

Once a partial process model is defined, it can be stored in the repository as a separate artifact that can be invoked in future. Indeed, there are two ways to invoke partial models. The first invocation is to view it. In this case, queries in the partial model are matched to the respective parent processes, based on the configuration. Matching parts are merged with concrete parts and the modeler is given an up-to-date view on how the child process looks like. In the view mode, the modeler might make changes to the process. In this case, if the change concerns overriding the behavior from the parent process, the modeler is warned and switched to the editing mode. In this sense, we partially address the problem of overriding behavior. The other invocation is to edit the partial process model. In that case, the modeler is allowed to arbitrarily edit query components or concrete components of the child process. Here, we elaborate more on how PPM evaluation maintains consistency or indicates a problem.

Referring to Figure 11 to obtain an up-to-date version of the variant, both queries Q1 and Q2 have to be evaluated against their referenced process models, in this specific case they refer to the same parent process in Figure 3. The evaluation yields process components according to Definition 4.2. The resulting process components are composed with concrete elements in the PPM with respect to the control flow edges between concrete elements and query elements. Finally, the resulting process model reflects the up-to-date variant.

To maintain consistency between parent and child processes, a change on the parent (partial) process must be reflected on the child process by means of query evaluation. On the parent-process level, a change could be adding new functionality, e.g., adding new tasks, removing functionality, or restructuring the process. Now, we explain how queries reflect these changes, if the changes happen to be in the part referenced by queries.

Imagine that a modeler edits the parent process in Figure 3 by adding a new task "Record case" between the task "authenticate manually via CSI" and the succeeding XOR-split. In that case, when the partial process model of Figure 11 is evaluated, the newly added task will appear on the evaluation because the newly added task is on a path from "Ask for authentication data" to "Ask for issue". In the same way, if the modeler for some reason removes the task "authenticate manually via CSI" this will be reflected on the evaluation of the PPM. In both cases, the modeler of the PPM gets a consistent view on the parent process.
There might be editing operations on the parent process that break the consistency. For instance, imagine that a modeler removes the task "ask for authentication data" or simply edits its label in the parent process of Figure 3. In that case, matching query Q2 of the partial process model of Figure 11 will fail, cf. Definition 2.5. When evaluation of a query fails, it is replaced with a dummy node in the resulting variant indicating that the specific query failed to find a match. At this point, the modeler of the variant is informed about a change in the parent process and she has to either adapt the PPM definition to the new change or revoke that change on the parent level. Thus, consistency between the child and the parent process model is restored.

As Figure 11 allows multiple levels of inheritance, it could be the case that a parent process is by itself a partial process model. Thus, the evaluation of a partial process model might trigger recursive evaluations of intermediate partial process models. In case any of the intermediate PPMs' queries fails to find a match. The recursion is aborted and the user is informed about the partial process model and its query that failed to find a match, as discussed in the previous paragraph.

Three concrete process models that inherit behavior from each other are depicted in Figure 12. The most simple process is at the top of the figure. Second process model inherits entirely parent's behavior and so does the third process model which inherits behavior from the second process model. Thus we are dealing with an inheritance chain.

4.3 Deriving Variants from Holistic Models

The problem of deriving variants from holistic process models can also be addressed in the same way. We store the holistic process model and a set of partial process models that define how variants can be obtained. The step of obtaining the holistic model is out of the scope of this paper. Currently, we assume that we are able to obtain a holistic process model from a set of variants. Our approach addresses and supports the concerns of the steps after the process of obtaining the holistic model. That is, we keep only the holistic model and a set of partial process models that define the variants and get rid of the existing original variants.

In this case, each PPM consists solely of a set of queries that are connected with control flow edges. These queries extract the behavior inherited by the variant from the parent, holistic, model. No, concrete elements are allowed in the variants. Otherwise, the PPM is extending the behavior of the parent process. In this scenario, the inheritance is shallow. The holistic model is the parent process and all variants are PPM on the same level as direct children of the parent process.

5 Framework Architecture

In this section, we describe the architecture for the partial process modeling of business processes, illustrated in Figure 13, which consists of the following main components.

- **Process Modeling, Querying, and Composition Environment** provides the process designer with a graphical modeling interface [7]. Users express their intention by means of a partial process model (see Section 4). The query interface extracts the set of process model queries from the partial process model, and passes them on to the query processor. The matches returned by the set of queries will then be composed with the model fragments from the partial model through the model composer.

- **Process Model Repository** is a central storage of business process models that is accessed in a uniform way [27].

- **Query Processor** The query processor evaluates the queries received from the query interface [29] and
The architecture describes a prototype that already exists. The client, particularly the model designer, is the Oryx editor, an extensible process modeling platform for research that has been designed to model and manage process models online [7]. Query interface and query processor for BPMN-Q [2] have been implemented as client-side and server-side plugins respectively to the Oryx editor and are able to run process model queries against the Oryx process model repository. The model composer component that integrates the results of queries with the concrete parts of the partial process model is implemented also as a client-side plugin. The running prototype of our approach can be accessed at [http://oryx-project.org/try](http://oryx-project.org/try) (see Figures 15, 16).

### 6 Approach evaluation

We evaluate our approach in comparison to the state-of-the-art for similar approaches (with respect to requirements, methodology) that are targeting the same problem as follows.

Hallerbach et al. proposed the Provop approach to model process variants [10] [11]. Although the approach is meant to support process variants all over the process life cycle, we are concerned with the support at the modeling phase.

The authors of Provop emphasize also in [11] both that the multi-model approach (variants are defined and maintained in separate process models, with out being related to each other in a consistent way) as well as the single-model approach (one model that comprises all the variants) do not constitute viable solutions in many cases.

The set of requirements which stand at the base of the Provop approach [11] touch several directions modeling, configuration, execution and maintenance. In Provop different variants are represented by a set of change operations (e.g. insert, delete and move process fragments) that describes the differences between the basic process model and the associated variant. In consequence the methodology of Provop or the Provop lifecycle as depicted in Figure 4 in [11] consists of three major phases: the process modeling phase, the configuration of variants phase and the process execution phase. At the modeling phase there is only one process model (also called a base process or the reference process model) defined and a set of reusable change options which refer either explicitly to process elements or implicitly to a particular process fragment (by means of adjustment points). In the configuration phase the base process model is configured automatically into a variant, based on the set of change options, or a subset of those change options. The automatic process takes use of the context to select the proper change options.

Our methodology is based on inheritance from software engineering. Thus we also start with a basic process but in comparison with Provop we have a direct linkage between the process variant definitions (here called PPMs) and the base process models. For us a base process model can be a concrete process model or a PPM, thus allowing multiple inheritance. As stated before to obtain a variant with the Provop approach one will always start with the base process model and will have to apply a set of change operations. One could argue that with an enough complex set of change operations the desired variant could be obtained from the base process model. However for complex set ups such as the eBay environment and not only (Figure 14) it would be much easier if inheritance directly from variants would be allowed as the complexity will be lower. Similar to software engineering one does not have to maintain all the changes from the previous steps in the inheritance chain but just those for the current step. Nevertheless a set of basic change operations is also used in our approach. We do not take into account context.

Figure 14 has multiple purposes: (1) to show that our approach can be used successfully no matter the case (we employ an example used to present the Provop approach); (2)
show inheritance chaining in action; (3) to show the simplicity brought by an inheritance chain.

The example addresses the problem of a product change. In the upper part of the Figure [14] there is the very base process model annotated Parent process. The definition of the Process Variant 1 is depicted in PPM1. PPM1 is very simple and contains one BPMN-Q query Q1 and one activity named d) comments outside of Q1. The evaluation of Q1 basically returns the Parent process. In order to obtain the Process Variant 1 the Parent process requires a new activity d) comments. The two parallel gateways from Q1 mark the spot where the new activity has to be added to the Parent process. Q1 is directly linked with the Parent process via a direct link which in the tool is the id of the Parent process. In consequence the evaluation of PPM1 is Process Variant 1.

In a similar way Process Variant 2 is obtained based on the definition depicted in PPM2. Both queries Q1 and Q2 are linked to Parent process, however because of the layout only one link is depicted in Figure [14] PPM2 is a little bit more complex compared to PPM1.

As it can be easily seen Process Variant 3 is a variant of Process Variant 2. They differ only because of the activity d) comments. Query Q1 in PPM3 is linked directly to PPM2, thus we are dealing with an inheritance chain as PPM2 inherits from Parent process. We argue that PPM3 is much simpler than PPM2 as the modeler is required only to add a new activity to PPM2 and is not required to deal directly with Parent process anymore. Thus the design is incremental, and the effort required at each step is comparable and manageable.

Table 1 Feature Comparison: Provop, C-EPCs and PPMs

<table>
<thead>
<tr>
<th>Features</th>
<th>Provop</th>
<th>C-EPCs</th>
<th>PPMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variants as separate process models</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Consistency maintenance</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Variants breaking inconsistency</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Concerning inheritance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple Inheritance</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Multiple Level Inheritance</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Operations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add new elements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allow variants to override parent processes</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Ability to lock inherited parts against editing</td>
<td>-</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Process elements that can be configured</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control flow</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technique</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bottom up</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Top down</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

A second major approach for dealing with variants is the single-model approach, the holistic model approach. In this case the reference model comprises the "big picture" [28]. Such a reference model is later configured (restricted) into a more specific process model.

Configurable Event driven process Chains (C-EPCs) [28] are an example of such an approach that tackles the problem of process models variants starting from a holistic process model. Functions as well as connectors can be configured towards achieving a more refined process model. These elements can be configured based on set of configuration requirements and configuration guidelines. As in the Provop case there is a configuration phase when these configurations are performed.

The management as well the configuration of such a holistic process model depending on the number of variants involved could be a very difficult task, subject to errors. This fact is also underlined by a large number of publications (see for instance [13], [14]) towards creation of a questionnaire approach.

Configuration of holistic process models by means of PPMs is not an issue for the approach. PPMs do not require the existence of any specific configurable nodes or functions. With the help of BPMN-Q queries elements or process fragments can be skipped for instance.

Requirements such as: (1) support for the entire processes, functions, control flow, data; (2) configuration made at different levels are entirely or partially supported by our approach. We do not support data elements. We address the issue of configuration at different levels by means of inheritance chains, thus any process variant can be even more specialized. The requirement of context based configuration is not supported by our approach. Interrelationships between configuration is supported by our approach via inheritance chain, as child processes that are for example on the third, forth level depend on child processes closer to the root of the inheritance chain.

Table [1] depicts a side by side comparison between major approaches that tackle the problem of process model variants and our approach. This comparison is based on a mixture of requirements of the presented approaches. Available features are represented in the table with the "+" sign and those missing with "-".

7 Related Work

The management of variants has been addressed in various domains, such as software configuration management [8, 34, 35] and feature diagrams [5, 6, 31]. For process models, different approaches have been defined: configurable reference process models, inheritance based and annotations based.

Lu and Sadiq [16] presented a process modeling framework that is conducive to constrained variance by supporting user driven process adaptations. In [17] they described another approach for facilitating the discovery of preferred variants based on the notion of process similarity where multiple aspects of the process variants are compared according to specific query requirements. Compared to our approach, we address the issue of maintaining consistency among variants rather than deciding whether two or more processes are variants of each other.

C-EPCs [28] allows the configuration of process models by distinguishing between choices that can be made at runtime and those that have to be made before, i.e., configuration time. Configurable nodes are used as the means to introduce configurability to EPCs. On the other hand aEPC [24] works on the principle of projection [3] and only elements that have a particular label are included in the extracted model. Inheritance of behavior in workflows [37, 4] is a formal approach.
for tackling problems that are related to change. Four inheritance rules (protocol inheritance, projection inheritance, protocol/projection inheritance, life-cycle inheritance) are defined to tackle dynamic change.

Annotations based approaches, e.g., the PESOA (Process Family Engineering in Service-Oriented Applications) project [25] defines so-called variant-rich process models as process models that are extended with stereotype annotations to accommodate variability. Both UML Activity Diagrams as well as BPMN models can be tackled with this approach. The places of a process where variability can occur are marked with the stereotype VarPoint. Several other stereotypes, e.g., Variant, Default, Abstract, Alternative, Null, Optional) are used to specify different configuration options. Compared to our approach, we do not predetermine configuration points.

Variants at execution time are addressed in [18]. The notion of process constraints is used to tackle the need for flexibility and dynamic change at execution time. Here a variant of a process is considered as an instance of a process.

La Rosa et al. [26] defined a questionnaire based approach to extract variants from C-EPCs. However, to do so, one needs first the holistic model (the C-EPC), then it needs a questionnaire model and a mapping between the C-EPC and the questionnaire model. Our approach, however, does not require an additional mapping. Queries can be applied directly to the holistic models to derive variants. For us, a holistic or a specific model makes no difference. Because the queries are expressed with constructs similar to those in BPMN, the easiness introduced by the questionnaire approach to extract variants, we argue that it is also maintained here.

8 Conclusion

This article discussed at large a new approach to tackle the issue of managing process model variants in a consistent way. The approach is based on the idea of defining Partial Process Models by means of BPMN-Q process views. Thus, the process modeler can define a new process model by specifying concrete parts in addition to queryable process views. Each time the defined model is invoked for viewing, the process views are evaluated, which guarantees returning the up-to-date and consistent model status for the process modeler. In addition, our view-based approach can promise other benefits such as extracting different views for process variants from holistic process models.

The methodology behind the approach is influenced by the inheritance mechanism from software engineering. The particularities of this methodology are of great advantage in use cases such as those discussed in this article. Situations such as the eBay environment where only for process model more than 8000 variants could exist are almost impossible to address with the current technologies for process variants management.

A set of requirements and fundamentals of our approach are discussed in strong connection with concrete examples. We have compared our approach with current major approach for process variants management (Provop, C-EPCs) and underlined the fact that our approach can fulfill most of the requirements that make the current approaches, but in addition we address other issues that have not been touched by the current approaches. Thus we provide support for maintaining consistency of process models variants, we allow for multiple level inheritance, as well as multiple inheritance. Reusing existing business knowledge materialized in existing process models. The reuse is not only on the level of a whole process model, but rather on a fine grain level which is in the form of process model components. Configuration of holistic process models is also possible with our approach.

Context is an important factor when dealing with process variants as emphasized also both by the Provop and C-EPCs approach. Thus future work foresees the improvement of the approach discussed here with contextual information. We have already put the bases for towards using contextual information for process variants management using also a methodology inspired from software engineering in [22].

Acknowledgements

We would like to thank Clemens Rath from eBay’s European Center of Excellence for his support.

References


Figure 14  Product change use case. Variants based on Figure 1 from [10]
Figure 15  Running Prototype - Query

Figure 16  Running Prototype - Query Result