Why do process variants matter for process monitoring?

Matthias Schrepfer[1], Juliane Siegeris[2], Gunnar Obst[1], and Matthias Kunze[1]

[1] Zalando SE, Mollstr. 1, 10178 Berlin, Germany
firstname.lastname@zalando.de
[2] HTW Berlin, University of Applied Sciences, Germany
juliane.siegeris@htw-berlin.de

Abstract Business processes models typically serve as a specification for a future system or as a documentation of an already existing one; it can also serve both purposes. As precise documentation of an implemented business process, process models provide an input to configure a process monitoring system, enabling the specification of monitoring points and metrics. However, complex business processes show an unexpected quantity of potential variants, which impede the activation of process monitoring.
In this paper, we elaborate on the impact of variants on the configuration of a process monitoring system, and show how the number of model variants can be significantly reduced by analyzing the syntactic and semantic information related with decisions in a business process. Applied to an existing business process, we identified almost 60,000 variants, which we were able to reduce by over 65%. At the same time, we improved the quality of the process model.

1 Introduction

Business processes management is a well-established discipline and widely used in industry. Many companies focus on well-established methods to design, analyze, control, and optimize their business processes to ensure high customer satisfaction and close alignment with IT systems. Especially in the context of rapidly growing multinational companies in the e-commerce sector, organizations must overcome challenges in business process management in order to scale up their business and reach ambitious business goals. Consequently, business processes in the e-commerce sector are automated to a large extent. Setting up a consistent and scalable process monitoring and process controlling enables the fast detection of problems and thus allows companies to derive remediating actions to address these problems immediately after they were detected.

1.1 Business Process Management at Zalando

At Zalando, business process management found its entrance in 2012, when we set out to document our core processes in a structured way. Due to the

rapid growth of the company, we decided to develop our own ERP system ZEOS (Zalando E-Commerce Operating System) tailored to our needs. For the requirements specification of this system and to ensure proper business-IT-alignment, all involved departments contributed to the precise documentation of the relevant business processes using BPMN. Over time, more and more processes of Zalando's value chain were documented and integrated into the company's process landscape.

One year later, we began to use the documented business processes also for operational tasks. On the one hand, we experimented with a self-developed process engine to automate our core order processes, which led eventually to the integration of an open source BPM engine and the first fully automated business process going live early in 2014. Since then, we are continuously increasing the automation of our processes.

On the other hand, we found a significant value in detecting anomalies in the execution of our processes – including non-automated or hard-coded behavior. We devised an approach that enables the monitoring of business processes using realtime event data that are provided from all involved IT systems. Using a very scalable architecture, we are able to monitor hundreds of thousands of orders per day and provide early warnings and near realtime anomaly detection for our end-to-end core processes. The created data remains available for ex-post analysis as a basis for continuous improvement.

In Zalando's endeavor to become a widely used platform that connects people with fashion beyond our core business, BPM has become one of the driving forces and key factors of success.

1.2 The Role of Process Monitoring

Enabling process monitoring requires that process models contain all business logic required by underlying business scenarios and consider processes across the entire IT landscape and organizational boundaries. This typically results in a large number of detailed and thus complex process models capturing all possible cases. While the creation of models of a high syntactic and semantic quality is a very challenging task in practice, it is required not only for process monitoring, but also bridges the gap between business and IT and therefore builds the basis for process execution, compliance checking, and continuous improvement.

Effective process monitoring ensures that business goals are met by checking the state and performance of business processes continuously. This includes detecting process problems and consequently raising warnings and alarms in case of problems or deviations. While this may sound straightforward given detailed process models, it is subject to several constraints in practice. But what makes process monitoring complicated?

To rapidly detect and resolve problems of a business process, all process instances must be monitored. In an e-commerce setting, this number quickly rises beyond 100,000 process instances per 24 hours, which is already a technical challenge toward the scalability of the monitoring system. Furthermore, the more complex the process is, the more complex the activation of process monitoring

will be, because more process variants need to be treated separately. Here, the term process variant refers to all possible paths in a process model that need to be monitored. Different process paths are triggered by parameters such as the chosen shipping or payment method. Each parameter yields an individual process flow in a way such that individual values, e.g., payment methods like credit card and invoice, are handled properly. For business and IT users it is important to know whether all these flows are executed properly to ensure process conformance. However, each variant that shall be monitored needs to be treated separately, which results in an enormous effort to set up the monitoring system.

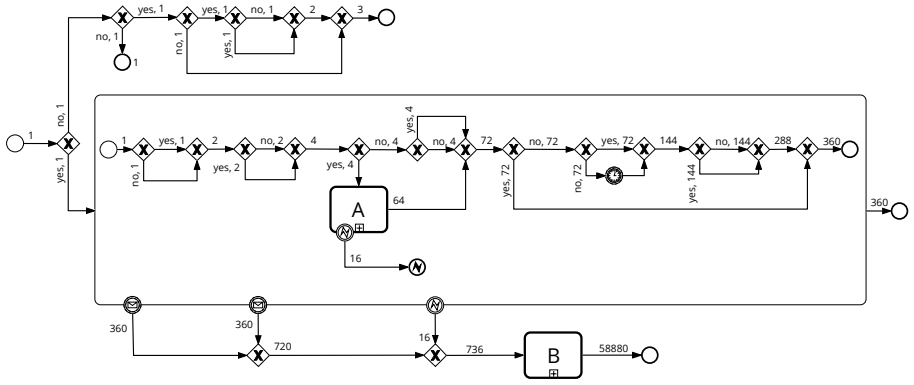
With regard to the enabling of process monitoring, the lower the number of process variants in a process is, the easier is its activation. In this paper we present approaches to analyze parameters that trigger process variants, aiming at the reduction of process variants. By analyzing process variants, we further show opportunities to increase the quality of process models from a semantic point of view. Our ultimate goal is to reduce the effort and increase the efficiency to activate process monitoring.

2 Motivating Example

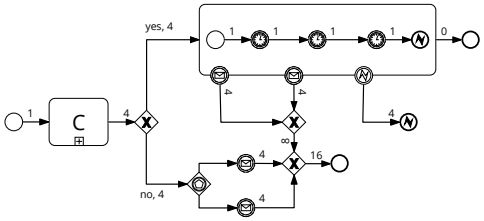
We illustrate our approach using a part of an order-to-cash process of a real-word example, depicted in Figure 1. The investigated part starts with the placement of a customer order and ends with the decision to which warehouse the shipment of the ordered goods is assigned. The business process is modeled using BPMN; it consists of one parent process and three subprocesses. The process model shows only the branching structure for our order-to-cash process, as we removed activities and labels. The original models consist of 20 to 100 elements and comprise basic as well as advanced process modeling structures, such as error-handling, process hierarchy, and attached boundary events. In our case, all process steps are executed sequentially, which becomes apparent by the absence of concurrency in the process models.

We have annotated control flow edges with the number of different variants that can pass through these points in the process model. In subsection 4.1, we discuss in greater detail, how these numbers were computed. All in all, we inferred 59,244 variants. In the main process, see Figure 1a, subprocess B, cf. Figure 1d can be executed in 736 different variants. The subprocess itself creates 80 variants, which can be the continuation of each of the incoming variants. Hence, the number of variants multiply when subprocesses are called. This leads to 58,880 different variants after the subprocess completes.

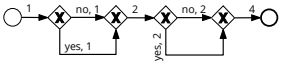
Along the process, we established several measurement points for which our monitoring system records the time and process data, when an instance passes such a point. Our monitoring solution allows us to continuously compute the time period between two measuring points, a so-called metric, and to compare these with threshold values. This allows for the visualization of current and historic performance figures of business processes. If, for a given metric, the threshold



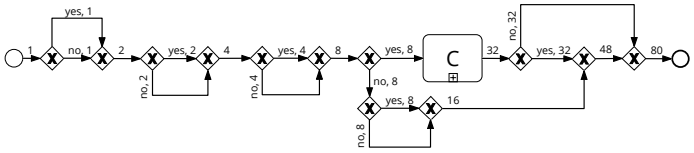
(a) Main process



(b) Subprocess A



(c) Subprocess C



(d) Subprocess B

Figure 1: Order to cash main process and subprocesses

value is exceeded for a minimum number of instances, the system notifies affected personnel.

Although we may not monitor all variants, we cannot exclude any part of the process model from monitoring a priori. Even domain experts typically do not know, which variants are less frequent without a proper throughput analysis. However, to do this already requires monitoring of all variants. In case variants are excluded from monitoring, practice has proven that process problems are detected late. This results in a drop in customer satisfaction and revenue.

As we show in the remainder of this paper, the above number of variants is an upper bound and can be significantly reduced.

3 Related Work

The methods presented in this paper refer to the discovery of process variants in business process models. In the literature, two different approaches are advocated. The multi-model approach uses a number of related process models to capture different variants, typically as a result of manipulating one central reference model [3,12]. In contrast, the single-model approach consolidates all possible variants in one process model that offers different configurations for a particular variant [1,10]. Here, some gateways are specifically marked as configuration points, where different variants follow different branches. Still, in both cases, a process variant is a complete business process model, in particular including control-flow branching structures.

In this paper, variants are understood as distinct sequences of activities and events similar to the notion of traces in process mining [14]. Process mining analyzes logs of business process executions and strives to discover process models by reverse-engineering ordering relations between activities and detecting points where a path in a process might diverge. However, in contrast to process mining, we do not take process logs as the basis to generate a process model, but start with the model itself to reveal all possible variants. The number of variants is related to the cyclomatic number of programs [4,8]. However, in our case also different numbers of iterations of the same process model fragment are considered individual variants.

Based on the discovered variants, this work attempts to improve the quality of the process model by reducing the number of variants and increasing consistency within models. Model quality has been in the focus of a wide range of research work, an overview of factors affecting process model quality is presented by Mendling et al. [6]. With regard to this work, our primary focus is towards the semantic and pragmatic quality of process models [9].

One particular aspect we have not found being addressed in the literature is the consistency of the configuration of points in the business process model, where process execution diverges. We refer to these as trigger parameters for variants of a process model. For instance, if two distinct exclusive choice gateways model the very same decision, they should be labeled identically. The following sections present our approach to discovering, characterizing, and reducing process variants, as well as normalizing choices within a process model.

Other proposals towards increasing process model quality include, for example, Mendling's Seven Process Modeling Guidelines [5]. This work introduces rules, based on empirical research, to keep process models simple, consistent, and easily comprehensible. While these guidelines improve a single model and reduce its cognitive complexity, refactoring of process models [16] strives to increase the consistency between several models in a collection, such as, consistent labeling of activities across all models. Please be aware that the latter work uses the term variants in a different meaning than we use it here.

Many of these approaches towards increasing process model quality have already been applied during the modeling of our business processes, for instance the labeling of objects and the extraction and linking of common subprocesses.

The latter can also be perceived in our example in Figure 1, where subprocess C is linked in processes A and B. However, with regards to the number of variants in a business process, these approaches do not change semantics of a business process model, but rather their organization. They have, therefore, no impact on the number of variants of process models.

4 Variants in Business Processes

Contrary to related work discussed in the previous section, where a process variant refers to different versions of a complete business process model, we define a process variant as a class of process instances.

> *A process variant is a complete and unique sequence of activities, events, and decisions carried out in compliance with a business process model. Every process instance of this model belongs to exactly one process variant.*

The order-to-cash process above is executed among a number of independent and distributed software systems, each of which adds fragments and execution alternatives to the process itself. Our definition of a business process variant embraces this aspect and captures one variant of the overall process as a particular ordering scenario. Variants need to be complete with regard to the start and end of a business process.

4.1 Identification of Process Variants

Having defined the term process variant, the question arises, how variants can be identified. Business process mining offers a straightforward solution to the identification of unique variants by examining a process log. However, in our scenario, such a log is not available, as we aim at setting up a monitoring solution prior to the rollout of a business process.

It is, nevertheless, possible to derive process variants from a process model, if it is normative and sufficiently detailed, i.e., on an executable level. Essentially, all model constructs that yield alternative outcomes lead to a set of process variants – each alternative adds another process variant. In the case of BPMN, such constructs are, for instance, exclusive gateways and interrupting boundary events.

Our approach to computing the number of variants in a process model is based on Sadiq and Orlowska [11]. The authors present an approach to identify behavioral anomalies in sequential process models, by iteratively eliminating paths in the model that are correct. For instance a set of n alternative paths that are split and joined in a well-structured fashion are reduced to a single path. If the remaining model is trivial, then the original model was correct. Models that show deadlocks or lack of synchronization cannot be reduced completely.

In our case, the process models underwent a verification process a priori and hence are considered to be correct. However, we reused the iterative reduction

technique to identify variants in process models. For every reduction, we counted the number of variants that were created by the reduced fragment. In case of the aforementioned set of alternatives, we would infer n variants. The number of variants is then annotated to the outgoing control flow sequence and fuels into the next fragment to be reduced. In Figure 1c, we see two fragments, each with two alternatives, which results in an overall count of 4 variants. Similarly, hierarchical decomposition in process models, i.e., the use of subprocesses, adds significantly to the number of variants of the business process.

Note that the above method to derive process variants is applicable only for well-structured, sequential process models as is pointed out in [15]. Furthermore, the interleaved execution of parallel paths quickly leads to an explosion of the number of process variants [13]. In our case these prerequisites apply, because all parts of the end-to-end business process are carried out in a sequential fashion by different IT systems.

4.2 Characterizing Process Variants

A small number of process variants – in our experience around 500 – is not problematic for process monitoring, as not every activity, event, or decision is tracked by a monitoring system. In our example, we have computed the number of variants according to the above method for the first part of our end-to-end business process, which resulted in 59,244 process variants; a number that becomes unmanageable provided that our monitoring system is configured manually. The high number of variants was not expected but confirmed our initial concerns of process complexity.

As shown in Figure 1, only 360 variants are successful, i.e., lead to the continuation of the order-to-cash process towards the next subprocess – often referred to as the happy path. Comparing this number with the overall number of variants demonstrates that most variants address deviations from the happy path. A semantic analysis shows that indeed almost all other variants cover process parts for error-handling and customer-interaction, e.g., order cancellations triggered by customers. Yet, for a complete monitoring of the business process, it is not sufficient to just focus on the happy path. As an early requirement we stated that 100% of all process variants must be monitored.

One important observation we made during the identification of process variants is that one cannot distinguish between important and less-important variants from a business point of view. One must treat all variants equally important, because the process is executed fully automated. There are no knowledge workers performing any activities. Even the ratio of instances per variant would not help to judge on the importance of process variants.

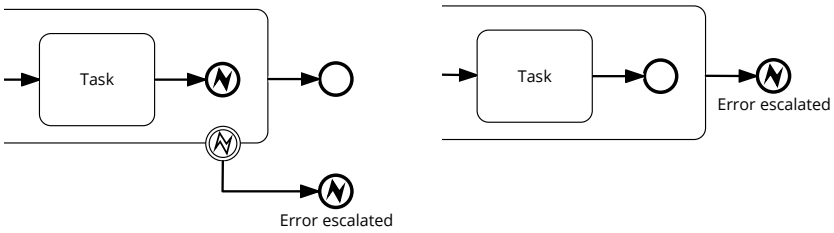
5 Reducing Process Variants

Having experienced a vast number of 59,244 process variants for only a part of an end-to-end business process, we aimed at analyzing why different variants are

triggered. To this end, one goal is to remove variants whenever possible as this is the most effective way to ease the activation of process monitoring. Moreover, such a large number of variants may also be a sign of the potential to increase the quality of our process model. In this section, we report on various approaches to reduce variants that we identified by carefully studying the process model and related information.

5.1 Zero Variants

One of the first reasons triggering process variants are paths in the process model that can never occur, which we therefore call zero variants. Although all of our process models underwent careful reviews prior to the variant analysis, this behavior turned out to be a flaw from a semantic point of view. An example from subprocess (d) in Figure 1 is shown in detail in Figure 2 that internally handles an error before escalating that error to the parent scope.



(a) Original model (b) Refactored model

Figure 2: Zero variants

The process path identified by the outgoing blank end event of the subprocess is unreachable because the subprocess always terminates with an error event. Analyzing this path, we concluded that it increases the effort to understand the model and may lead to misinterpretations by model readers. Hence, all paths with zero variants must be refactored to increase model quality. In the above example refactoring was performed without changing the semantics from a business point of view. The quality of the process model in Figure 2b was increased while the number of variants did not change. In other scenarios of the same category, the number of variants did change and has, in some situations, even been increased, e.g., in case of boundary events. Hence, the number of variants has to be computed again after model refactoring.

5.2 Duplicate Variants

The semantic analysis of process models, i.e., the matching of model elements such as activities, events, and decisions, to their counterpart in our actual business

revealed a second opportunity to improve process model quality and reduce the number of variants. Frequently, choices from a set of alternatives in the process models are not made completely independent. That is, the choice made at one point may depend on a choice made earlier in the course of executing the process. Figure 3 illustrates this with a fictitious example.

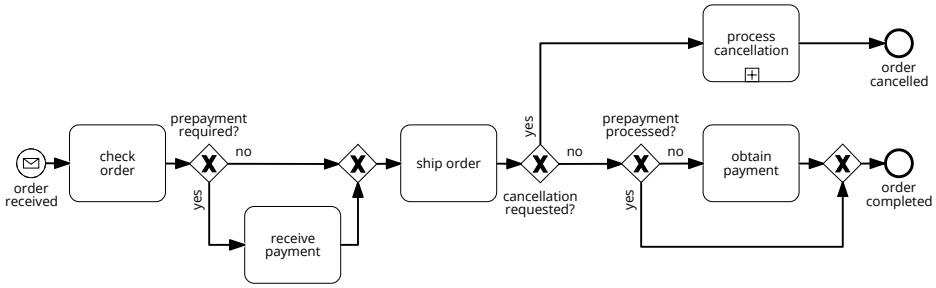


Figure 3: Non-normalized decisions

The business process contains a number of decisions. Two of them, namely *prepayment required* and *prepayment processed* refer to the point in time, when the payment for an order is carried out. If the customer chose a form of prepayment, it will be carried out before shipment of the order. If, on the other hand, prepayment has not been chosen, the money needs to be obtained after the shipment.

Looking only at the model, the process produces six variants, one for each combination of alternative paths. Taking into account the actual implementation of these decisions, we discovered that both decisions regarding payment are based on the chosen payment method of the customer. From a set of payment methods, one part qualifies for prepayment, whereas the remaining part does not. Hence, these two decisions are based on the same semantic context and there exist, in fact, only four variants in the process model.

We introduce trigger parameters and configuration parameters, and methods to identify such dependencies and resolve them.

> *A configuration parameter, short CP, is a variation dimension, i.e., a set of values that denote different alternatives.*
>
> *A trigger parameter, short TP, denotes a variation point in the process model that uses configuration parameters to specify the logic to choose between alternatives.*

Trigger parameters characterize variants based on either conditions, e.g., at an XOR-gateway, or based on events, e.g. at attached intermediate boundary message events, and hence correlate process variants with elements of the process model. To identify duplicates, the complete process is analyzed and all trigger parameters are listed separately with a unique id, the condition of a gateway or

the name of the event, and the process in which it is contained. Subsequently, a number of checks are carried out.

Duplicate Trigger Parameters. First, duplicate labels of trigger parameters are identified and marked. Corresponding points in the model are not yet refactored, as there is the chance to find further replicas of the trigger parameter. Also, duplicate labels do not necessarily imply duplicates, as also the configuration parameters for these TPs must coincide. Currently, the duplicate detection uses only simple string comparison; language processing is done by a domain expert to identify duplicates. In future, natural language processing could assist, cf. [2]. A second quality check focuses on labels assigned to TPs, i.e., their corresponding conditions and event names. The labels of TPs should comply with a style that ensures that readers can quickly comprehend the semantic information. As labeling style we focus on a best practice approach, see, for instance, [5,7]:

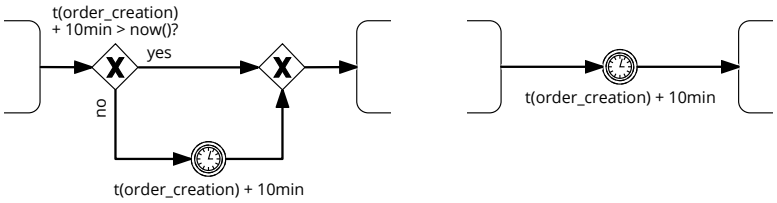
for events object + verb past perfect
for gateways a question attached to the gateway, condition expressions must be an answer to the question stated at the corresponding gateway; both question and answers are brief and precise.

The result of these checks is stored along with the specification of TPs. Labels that violate the above standards are marked for refactoring. However, refactoring labels is still postponed due to further checks. Moreover, not all labels may be refactored, as some of them are used in a close business-IT-alignment. That means, that some labels, in particular event names, are also used in the implementation of IT systems and used for monitoring. Hence, to keep models and implementation in sync, best practices may be neglected.

In our example, we identified 23 trigger parameters in the first part of our order-to-cash process. Only four of them complied with our best practice naming standards. Five out of the remaining 19 TPs could not be renamed for better style due to their reuse in IT systems; the rest has been improved.

Redundant Trigger Parameters. The next check focuses on TPs that can be eliminated, which would decrease the number of variants. The check verifies whether a process model can be refactored in a way such that the TP is eliminated without changing process semantics. It is important to keep semantics equivalent as otherwise the process logic would change. This task is performed by process experts together with domain experts to ensure consistency. A reduction of TPs improves comprehension of the model and increases its quality additionally.

Figure 4, which shows an excerpt of our example process, illustrates this check. The fragment on the left figure (a) shows a split XOR-gateway corresponding to a TP. Assuming that only a single variant is provided as input, there will be two variants – one that includes the timer event, and another that does not. Upon a careful review of the left model, one recognizes that the question addressed at the split gateway and the condition at the intermediate timer event are similar.



(a) Original model (b) Refactored model

Figure 4: Redundant trigger parameters

From a semantic point of view, the condition is checked twice: If the time has not progressed far enough, the process will wait for it using a timer event. An equivalent logic is shown on the right hand side (b) of Figure 4 where only the intermediate timer event is used. The TP is avoided, reducing the complexity of the model and eliminating another variant locally. Recall that the number of variants can multiply throughout the process, e.g., in case of subprocesses. Hence, even saving one variant locally, can reduce the overall number of variants significantly.

In fact, using our approach to eliminating redundant trigger parameters, the total number of variants decreased to approx. 36,000 – a reduction of 39.3% in total. Putting the large reduction into practice, one would not have guessed such tremendous reduction. The effect has such an impact, because the removal of a single TP may effect the complete process hierarchy. In cases where processes or parts of them are scoped by boundary events, the decrease of local variants might also turn down variants on a global scale as shown in our example.

Duplicate Trigger Configurations Identification and documentation of duplicate and redundant trigger parameters is the first step towards understanding why variants occur. Information on TPs already support the analysis of variants and actions can be derived from the analysis results. These actions can either lead to a removal of TPs and, consequently, a decrease in process variants and an increase in model quality. The next step towards reducing process variants is the analysis of configuration parameters. Configuration parameters (CP) are used to split up a business context, for instance payment methods in the order handling process. Configuration parameters are bound to trigger parameters, assigning information about the business context to it. A trigger parameter, linked to a specific process model element, determines the process' behavior based on the information of a configuration parameter.

Recall the above example shown in Figure 3, where the decision, which path is chosen, is based on the customer's choice of a payment method for two out of three gateways. Yet, the information in the process model alone is insufficient to decide, whether these decisions are taken on identical conditions. In practice "yes" and "no" do not determine, which path to chose; the actual selection of the payment

method is required. Further on, it could very well be possible that payment methods exist that require both, a prepayment before and a final payment after the shipment. Here, CPs come into play as they bind actual conditions of trigger parameters to values from a business context. For each CP, we store a unique id, its name, and all unique values.

Furthermore, we distinguish three types of configuration parameters, which indicate how the value of the parameters is determined, respectively.

design-time the parameter is static, e.g., to configure an IT system
a priori run-time the parameter is determined at the creation of a process instance and does not change anymore for this instance, e.g., based on the received order that triggered a process instance
live run-time the parameter is determined during the run of the process instance, e.g., the outcome of a human task or a value computed by an IT system

These types are closely related to process variant definitions, see section 3, where one variant is comprised of a complex model including decisions. Here, the CP types *design-time* and *a priori run-time* can be used to exclude certain variants (in our definition) from the process model, when a process is instantiated. *Design-time* CPs are comparable to configuration points in complex variant definitions.

In our example, the configuration parameter "payment method" consists of values credit card, paypal, invoice, and cash-on-delivery; the former two options being prepayments, and the latter payments after shipping. The CP type is *a priori run-time*, because it is based on the customer's choice of a payment method recorded in the incoming order. As we have discussed above, the CP is used for the TP of two gateways in the process model of Figure 3.

The binding of the values of a CP to the conditions of the outgoing paths of the gateway bridges the gap between domain knowledge, i.e., the actual process execution, and trigger parameters in process models. Using this connection, we can now identify trigger parameters that use the same configuration parameters. In combination with the search for duplicate trigger parameters, see above, we can now normalize the process model, by making identical and similar decisions consistent in their labelling.

First, we verify that all duplicate TPs that have been identified, see above, are actually duplicates. That is, they must use the same CP values for the decision. If this is not the case, we detected ambiguous labels in our model, which should be resolved by renaming one or both of the TPs in the model. Duplicate TPs with identical CP values are recorded as actual duplicates into the list of variants. These duplicates are later refactored manually.

In our example process, we found two TPs "gift voucher" and "gift voucher bought", which – looking at the labels – suggest a potential duplicate. However, the values of the corresponding CPs revealed that the first addressed the payment of an order using a gift voucher, whereas the second incorporated the purchase of a gift voucher by the customer. This highlights the importance to verify duplicates using configuration parameters.

Second, we analyze trigger parameters that have the same configuration parameter values. If a number of choices with semantically identical TPs and CPs occur in one process instance, i.e., they lie on a common path in a process model without concurrency, then a decision for one choice determines also the decision in other choices, which leads to a reduction of variants.

To refactor the process model, we need to examine, why the same business context, i.e., set of configuration parameters, is applied to trigger parameters with different labels. In our case, a gap or mismatch between modeling and interpreting business information, and errors in process models were the main reasons. Next, process and domain experts must clarify how to remediate this discrepancy. Typically, they decide for one TP and update the label of the other to properly match the context if necessary.

An example for the second case are trigger parameters "articles exist" and "article exists". The first conveys the impression that *all* articles must be available, whereas the second suggests only *one* article was sufficient. However, consulting domain experts and configuration parameters led to the decision that both trigger parameters are identical, and one has been renamed accordingly.

Of the initially 23 identified trigger parameters four could be removed due to duplicate TPs or duplicate CPs, which led to an increase in the consistency and unambiguity of the process model. The labels now better fit the domain knowledge. Even more important is the fact that readers of the process model can analyze, which domain information is used in trigger parameters and how variants are triggered. The semantic binding helped to increase process model quality significantly.

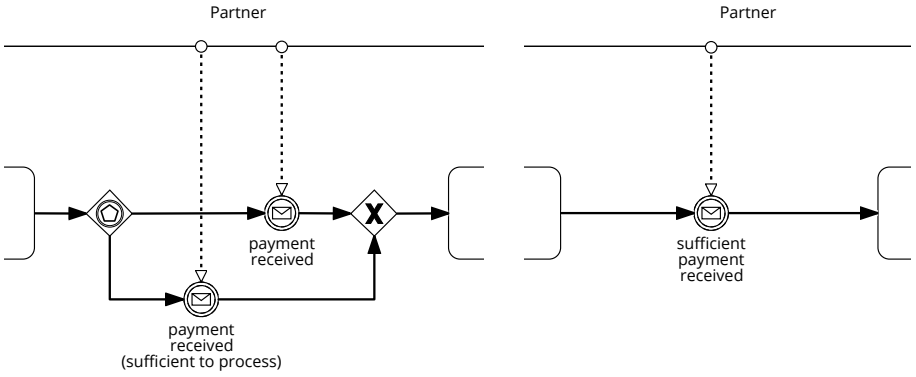
5.3 Merging of Events

During our analysis of the process model, we disclosed another opportunity to reduce the number of process variants. In some cases, variants were triggered by two or more message-receive events that indicated the same business trigger, but differed in the data payload. Such variants can be treated as a single instance under the condition that some constraints are met.

- All events must have the same scope, e.g., boundary events are attached to the same scope.
- The control flow of all events must be merged directly succeeding the message events.

These constraints ensure that different events, e.g., different messages, do not have different effects on the state of the business process. The decision whether events can be merged must be taken by domain experts. They must agree that some events cannot be distinguished later on in the monitoring system. If they deny, the monitoring system must be configured in a way that all events are monitored.

In the example in Figure 5, we identified two variants that were triggered by two message events, respectively, of which only one is processed according to the



(a) Original model (b) Refactored model

Figure 5: Merging of events

event-based gateway preceding these events. Both events indicate an incoming payment, however subtle differences led to their distinction in the model. After the issue has been disclosed following the conditions above, domain experts confirmed that merging the events for the purpose of monitoring was allowed. The original model is not changed in this case. The refactored model is only used to configure the monitoring solution. This has the disadvantage that two models need to be in synchronization. Still, in many cases the benefit of reducing variants outweighs the cost of maintaining two models.

6 Conclusion and Future Work

In this paper, we introduced an approach to characterize and reduce variants in business process models. It is based on the notion of trigger parameters and configuration parameters that provide insight into the data and logic that is applied when control flow diverges within the process model.

Introducing the optimizations presented above reduced the number of variants in a single process model from 59,244 to 11,000 variants, respectively 69.5 percent. Due to the process hierarchy, the number of variants on the happy path dropped from 360 down to 120. The reduction of variants eases the activation of monitoring to a large extent. Nevertheless, bear in mind that this immense reduction was triggered by optimizing only two local areas. Moreover, the changes applied to the process model also reduced overhead, normalized decision labels, and thereby increased the quality of our process model significantly.

For future work, we are looking into extending our approach to discover process variants in process models. This addresses the automatic discovery of variants, where also concurrent activities are taken into account. This means that the different ordering of interleaved activities is not considered as different variants, if they follow along the same paths in the process model.

References

1. Alena Hallerbach, Thomas Bauer, and Manfred Reichert. Capturing variability in business process models: the provop approach. *Journal of Software Maintenance*, 22(6-7):519–546, 2010.
2. Henrik Leopold. *Natural Language in Business Process Models - Theoretical Foundations, Techniques, and Applications*, volume 168 of *Lecture Notes in Business Information Processing*. Springer, 2013.
3. Chen Li, Manfred Reichert, and Andreas Wombacher. Mining business process variants: Challenges, scenarios, algorithms. *Data Knowl. Eng.*, 70(5):409–434, 2011.
4. Thomas J. McCabe. A complexity measure. *IEEE Trans. Softw. Eng.*, 2(4):308–320, July 1976.
5. J. Mendling, H. A. Reijers, and W. M. P. van der Aalst. Seven process modeling guidelines (7pmg). *Inf. Softw. Technol.*, 52(2):127–136, February 2010.
6. Jan Mendling, Jan Recker, and Hajo A. Reijers. Process modeling quality: A framework and research agenda. *BPM Center Report*, BPM-09-02, 2009.
7. Jan Mendling, Hajo A. Reijers, and Jan Recker. Activity Labeling in Process Modeling: Empirical Insights and Recommendations. *Inf. Syst.*, 35(4):467–482, 2010.
8. Glenford J. Myers. An extension to the cyclomatic measure of program complexity. *SIGPLAN Not.*, 12(10):61–64, October 1977.
9. Hajo A. Reijers, Jan Mendling, and Jan C. Recker. Business process quality management. In Jan vom Brocke and Michael Rosemann, editors, *Handbook on Business Process Management 1 : Introduction, Methods, and Information Systems*, International Handbooks on Information Systems, pages 167–185. Springer Berlin / Heidelberg, 2010.
10. Michael Rosemann and Wil M. P. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 32(1):1–23, 2007.
11. Wasim Sadiq and Maria E. Orlowska. Analyzing Process Models Using Graph Reduction Techniques. *Inf. Syst.*, 25(2):117–134, April 2000.
12. Sherif Sakr, Emilian Pascalau, Ahmed Awad, and Mathias Weske. Partial Process Models to Manage Business Process Variants. *International Journal of Business Process Integration and Management (IJBPIM)*, 6(2):20, sep 2011.
13. Antti Valmari. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528, London, UK, UK, 1998. Springer.
14. Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
15. W.M.P. van der Aalst, A. Hirnschall, and H.M.W. Verbeek. An alternative way to analyze workflow graphs. In AnneBanks Pidduck, M.Tamer Ozsu, John Mylopoulos, and CarsonC. Woo, editors, *Advanced Information Systems Engineering*, volume 2348 of *Lecture Notes in Computer Science*, pages 535–552. Springer Berlin Heidelberg, 2002.
16. Barbara Weber and Manfred Reichert. Refactoring Process Models in Large Process Repositories. In *Advanced Information Systems Engineering*, volume 5074 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2008.