Towards Flexibility in Business Processes by Mining Process Patterns and Process Instances*

Andreas Bögl[1], Christine Natschläger[2] and Verena Geist[2]
[1]*pascom Kommunikationssysteme GmbH, Arbing, Austria*
[2]*Software Competence Center Hagenberg, Hagenberg, Austria*

Keywords: Business Process Flexibility, Dynamic Adaptation, Process Mining, Process Pattern Library.

Abstract: The possibility to react to unexpected situations in business process execution is restricted since all possible process flows must be specified at design-time. Thus, there is need for a flexible approach that reflects the way in which human actors would handle discrepancies between real-life activities and their representation in business process definitions. In this paper, we propose a novel approach that supports dynamic business processes and is based on a framework comprising a process pattern library with domain-specific patterns and execution logs for mining related process instances. Given a running business process and an unexpected situation, the proposed approach provides a largely automatic adaptation of the business process by replacing failed activities with fitting process alternatives identified by exploring existing process knowledge. The feasibility of the approach is demonstrated by applying the main steps to a business scenario taken from the industry domain.

1 INTRODUCTION

Flexible business processes are a key issue for modern enterprises to operate efficiently in highly competitive markets. While many *Business Process Management* (BPM) methods and techniques have reached a relatively high level of maturity, the ability to flexibly react to changing circumstances and to support process changes at runtime are still matters of concern.

Thus, flexibility and dynamic adaptation of business processes are among the most active research areas with great potential (Reichert and Weber, 2012; Dixon and Jones, 2011). Also the *Industry 4.0* project of the German government especially emphasizes the demand for flexible processes in traditional industries. In this paper, we discuss in detail a novel approach and framework for flexible business processes that adapt to changing environments by applying predefined process patterns and evaluating former process executions. The overall idea was previously presented in (Bögl et al., 2014). We now extend the four basic steps to *retrieve alternatives*, *retrieve instances*, *select & rank alternatives*, and *integrate alternative* with a further step for *manual process adaptation* and provide detailed specifications of these steps. In addition, we put special emphasis on the process pattern library and dynamic adaptation of business processes, following previous research conducted in similar domains (Bögl et al., 2009; Natschläger et al., 2014).

This paper is structured as follows: Section 2 introduces preliminary definitions and presents a running example that relates to an actual ordering process of a sand and fertilizer producer. Section 3 describes how the process pattern library, a key component of the envisaged framework, can be built from existing knowledge. The main steps of the proposed approach are presented in Section 4. Related work is studied in Section 5 and Section 6 concludes the paper with an overview of the main results and future work.

2 PRELIMINARIES AND BUSINESS SCENARIO

The chosen business scenario relates to an ordering process (see Figure 1) of a sand and fertilizer producer in Upper Austria, called `S&F` company, and is used as a running example throughout this paper. The company's products are mainly used for cultivating sport

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

Bögl, A., Natschläger, C. and Geist, V.
Towards Flexibility in Business Processes by Mining Process Patterns and Process Instances.
DOI: 10.5220/0005652704690476
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 469-476
ISBN: 978-989-758-168-7

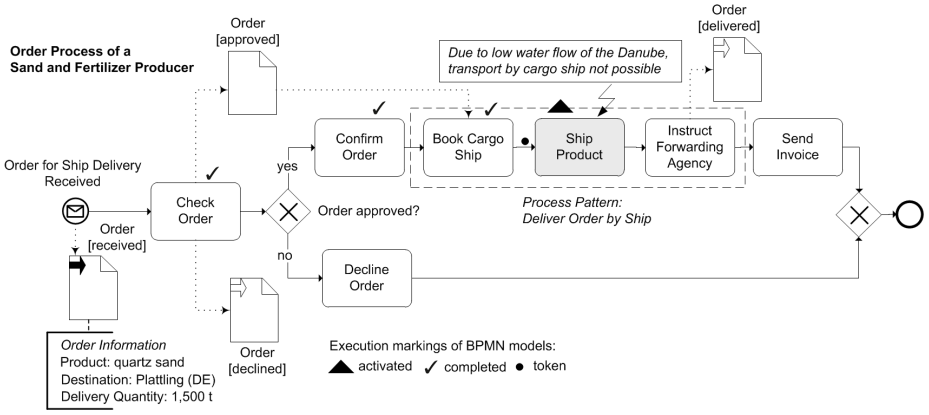

Figure 1: Ordering Process of a Sand and Fertilizer Producer in Upper Austria.

fields and golf courses, and for producing other final products in the cement industry. In most cases, these products are delivered to various destinations in Europe by ship due to low transportation costs and the fact that the S&F company is located at the Danube river and has an own harbour. Other alternatives are, e.g., delivery by train, by truck, or by aircraft.

Figure 1 illustrates a running process for delivering 1,500 t quartz sand to Plattling (Germany) by ship. It starts with checking the corresponding order after its reception. Depending on this decision, the order is either confirmed and executed or declined. In the running example, three activities were executed so far and process activity `Ship Product` is activated. Thereby, properties and corresponding data objects are assigned concrete values. For example, process activity `BookCargoShip` assigns data object `ShippingDestination` the value `Plattling(DE)`. The cargo ship transports the sand to Deggendorf, where a local forwarding agency carries out the transport to Plattling, and finally an invoice is sent.

The key to the description of processes are process activities and sequential orderings over these activities. Given a finite set of data objects $\mathbf{D}$ and process activity labels $\mathbf{L}$, then a process activity a is a structure (I,l,O), where $I \subseteq \mathbf{D}$ denotes the set of input data objects, $O \subseteq \mathbf{D}$ denotes the set of output data objects, and $l \in \mathbf{L}$ denotes the activity's label.Thereby, $input(a)$ (or $in(a)$ for short) refers to the input objects of a, $output(a)$ (or $out(a)$) refers to the output data objects of a, and $label(a)$ refers to an activity's label.

In addition, a hierarchical arrangement of activity labels is assumed. This hierarchy is a complete lattice structure $(\mathbf{L},\mathbf{R})$, where $\mathbf{R} \subseteq \mathbf{L} \times \mathbf{L}$ is the set of all relationships between the activity labels in $\mathbf{L}$. Let $(l_s,l_a) \in \mathbf{R}$, then $l_s \prec l_a$ denotes l_s as *label specialization* of l_a and conversely l_a as *label generalization* (Bögl et al., 2015). To give an idea behind label specialization and generalization, suppose $l_a =$ "Deliver Order" and $l_s =$ "Deliver Order by Ship". Intuitively, l_s represents a label specialization of l_a because l_s states a more expressive meaning than l_a.

Input and output data objects that are associated with process activity $a = (I,l,O)$ may have assigned state values, represented by the functions $state^I_a(d_1)$ and $state^O_a(d_2)$ with $d_1 \in I$ and $d_2 \in O$. For example, input of process activity `BookCargoShip` (a) in Figure 1 is given by $in(a) = \{\texttt{Order}\}$ and $state^I_a(\texttt{Order}) = \{[\texttt{approved}]\}$, output of process activity `InstructForwardingAgency` (b) is given by $out(b) = \{\texttt{Order}\}$ and $state^O_b(\texttt{Order}) = \{[\texttt{delivered}]\}$. Given a process activity a, its assigned input/output data objects and state values are also referred to as *initial/result context*.

A sequential ordering over a given set of process activities is referred to as *execution trace*. An execution trace is a pair (A,R), where $A = \{a_1,\ldots,a_n\}$ is a finite set of process activities taken from the universe $\mathfrak{A}$, i.e. $A \subset \mathfrak{A}$, and $R \subseteq A \times A$ is a total order over A that represents the sequence in which activities A are executed, denoted by $t = \langle a_1,\ldots,a_n\rangle$. The set of process activities composing an execution trace $t = (A,R)$ is also denoted by $activities(t)$ (or $act(t)$ for short), i.e. $act(t) = A$. The function $sub(t)$ returns for a given execution trace t the set of all sub-execution traces. For instance, given $t = \langle a_1,a_2,a_3\rangle$, then $sub(t) = \{\langle\rangle,\langle a_1\rangle,\langle a_1,a_2\rangle,\ldots,\langle a_1,a_2,a_3\rangle\}$. A process P is given by a finite set of execution traces, i.e. $P = \{t_1,\ldots,t_n\}$. The set of process activities composing the execution traces of a process P, denoted by $act(P)$, is given by $act(P) = \bigcup_{t \in P} act(t)$.

A process instance either relates to a running process or an already executed process. Process instances are assumed to be associated with a *process execution log* $\mathcal{L}$, typically produced by some process aware information system (van der Aalst and Weijters, 2005).

3 THE PROCESS PATTERN LIBRARY

In response to changing environments running processes need to be adaptable. To give an idea, reconsider the running example in Figure 1 and suppose that transportation by ship is currently not possible due to low water flow of the Danube river. This unexpected situation X prevents the execution of process activity `Ship Product`. In light of this unexpected situation, a process analyst or a computer system is engaged in defining an alternative process solution to successfully deliver the product to Plattling. Thereby, this alternative process solution substitutes the process activities affected by unexpected situation X. In the following, four substitution scenarios are outlined:

- Scenario `A` reflects a $1:1$ substitution. This means, unexpected situation X affects exactly process activity A in running process instance R and X can only be resolved by substituting A with a process solution consisting of process activity A'.
- Scenario `B` reflects a $n:1$ substitution. In this scenario unexpected situation X affects multiple process activities $\{A_1,\ldots,A_n\}$ in running process instance R. The situation is resolved by substituting $\{A_1,\ldots,A_n\}$ with a process solution that consists of process activity A', only.
- Scenario `C` reflects a $1:m$ substitution. As opposed to scenario `A`, an affected process activity A is substituted with a process solution that consists of multiple process activities $\{A'_1,\ldots,A'_m\}$.
- Scenario `D` reflects a $n:m$ substitution. In this case, unexpected situation X affects multiple process activities $\{A_1,\ldots,A_n\}$ and X is resolved by substituting $\{A_1,\ldots,A_n\}$ with another process solution which also consists of multiple process activities $\{A'_1,\ldots,A'_m\}$.

According to these scenarios, the idea is to mine existing process knowledge for process alternatives and to capture the mining results in a *process pattern library* $\mathcal{P}$. The key purpose of a pattern library is to serve as a container for potential (alternative) process solutions for dealing with unexpected situations that probably may arise during running process instances.

From a practical point of view, existing process knowledge may be available in terms of a process instance log and/or a repository of process models. In particular, a finite set of processes $\mathbf{P} = \{P_1,\ldots,P_n\}$ is assumed, where each process is represented by a set of execution traces, and further, $\mathbf{P}$ results from extracting the respective trace sets either from underlying process models or from an underlying process execution log $\mathcal{L}$. So, input for the proposed mining approach is a set of processes $\mathbf{P}$ and output is a pattern library $\mathcal{P}$.

A pattern library $\mathcal{P}$ that results from mining a given set of processes $\mathbf{P}$ is considered as a conceptual hierarchical structure consisting of nodes and edges similarly to the approaches presented in (Thom et al., 2008; Malone et al., 1999). The nodes represent process solutions or process patterns on the one side and process activities on the other side. The edges represent relationships between the nodes. More precisely, $\mathcal{P}$ is a structure $(\mathcal{A},\mathcal{S},ico,iso)$ where $\mathcal{A} \subset \mathfrak{A}$ is a set of process activities, $iso \subseteq \mathcal{S}\times\mathcal{S} \cup \mathcal{A}\times\mathcal{A}$ is the set of all *is specialization of* relations between process solutions $\mathcal{S}$ and process activities in $\mathcal{A}$, and $ico \subseteq \mathcal{A}\times\mathcal{S}$ is the set of all *is concretization of* relations between process activities $\mathcal{A}$ and process solutions $\mathcal{S}$.

Thereby, $isConcretizationOf(x,\{a_1,\ldots,a_k\})$ expresses a relationship between a complex process activity x and a set of activities $\{a_1,\ldots,a_k\}$. The relationship means that $\{a_1,\ldots,a_k\}$ represents a substitution for x. It is required that x is not a member of $\{a_1,\ldots,a_k\}$ and $|\{a_1,\ldots,a_k\}| > 1$. The purpose of an *isSpecializationOf* relationship is twofold: (i) It expresses a relationship between a pair of activities (x,y), where y is said to be a specialization of x. An activity y is a specialization of x if the label of y is a specialization of the label of x; i.e., x can be substituted with y. (ii) It expresses a relationship between a pair of process solutions (P_1,P_2), where P_2 is a specialization of P_1; i.e., P_1 can be substituted with P_2. Notably, P_1 and P_2 represent execution traces. Then, P_2 is a specialization of P_1 if P_1 is a sub-trace of P_2.

In light of the running example, complex process activity `Deliver Order` (A) in Figure 2 is the root with four derived concrete process solutions specifying delivery by ship (`S1`), by train (`S2`), by truck (`S3`), and by aircraft (`S4`). These process solutions are provided by corresponding process schema variants (called *VShip*, *VTrain*, *VTruck*, and *VAir*). Two further concrete process solutions for transportation outside the EU (`S2.1`) and of hazardous goods (`S3.1`) have been defined manually and refine standard delivery by train and truck.

4 MINING PROCESS INSTANCES

The overall approach to support flexible business processes comprises the five steps (1) *Retrieve Alternatives*, (2) *Retrieve Instances*, (3) *Select and Rank Alternatives*, (4) *Adapt Alternative* (optional), and finally (5) *Integrate Alternative* as shown in Figure 3.

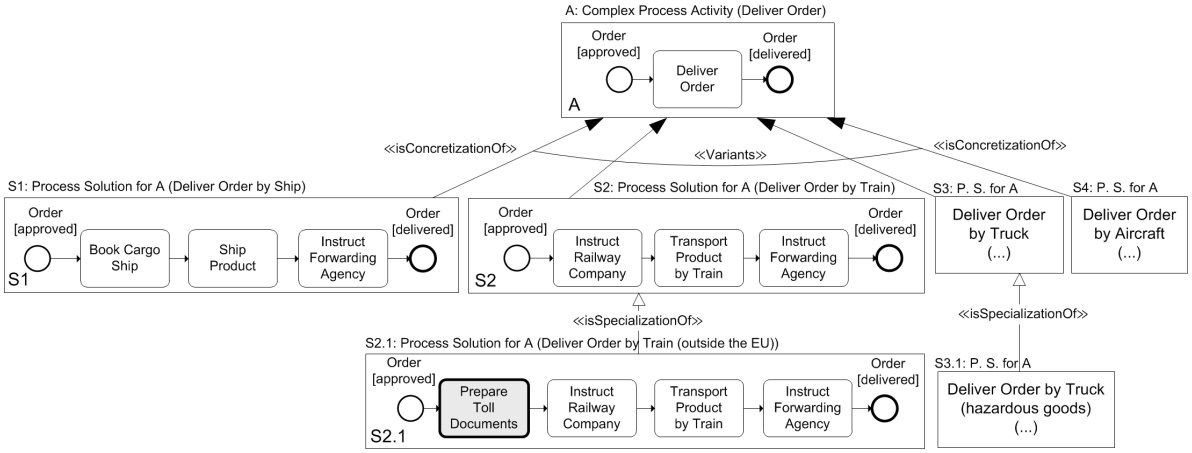

Figure 2: Process Pattern Library $\mathcal{P}$.

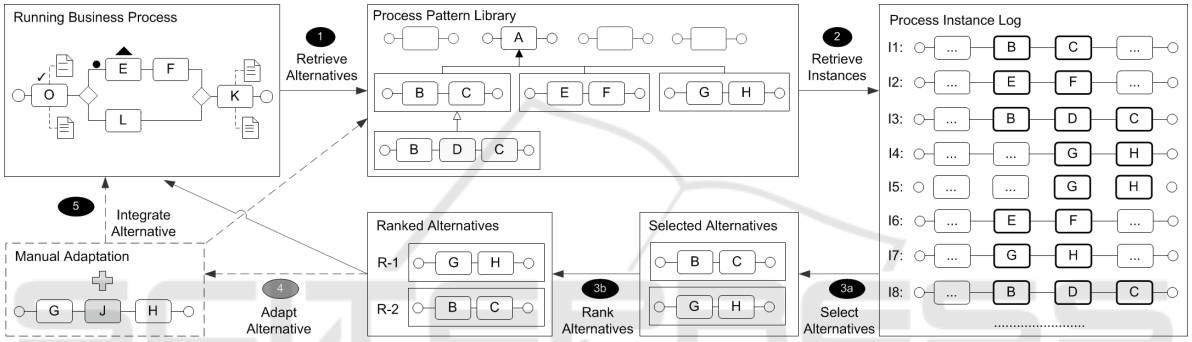

Figure 3: Overall Approach.

4.1 Retrieve Alternatives and Instances

The *retrieve alternatives* step addresses the identification of process solutions $S_1, \ldots, S_n \subseteq \mathcal{S}$ in the process pattern library $\mathcal{P}$ to successfully handle an unexpected situation.Thus, a complex process activity A in $\mathcal{P}$, which covers the context of process activities affected by X, needs to be identified. Then all derived process solutions and associated specializations indicate potential candidates to deal with X, apart from the child nodes which include one or multiple failed activities and further specializations of these child nodes.

The subsequent *retrieve instances* determines all process instances for the identified potential candidates. The main idea is to exploit process knowledge implicitly captured by process execution log $\mathcal{L}$. The execution log provides process instance specific knowledge given by the assignment of process execution data to input and output data objects. This knowledge is then used by the subsequent steps *select and rank alternatives*. To make this knowledge available in realm of decision making, function $\gamma(S)$ is assumed, which returns for a given pattern solution S a set of associated process instances I in $\mathcal{L}$.

In the running example, the initial context of the process activities affected by X is given by input data object `Order[approved]` and the result context is given by data output object `Order[delivered]`. These input and output data objects correspond to the input and output data objects of the complex process activity `Deliver Order` in the pattern library (see Figure 2). Then, process solutions `S2`, `S2.1`, `S3`, `S3.1`, and `S4` represent potential candidates to deal with X.

An extract of the process instance log comprising 30 process executions is presented in Table 1. For every executed process instance, the instantiated process schema variant, the applied concrete process solution of complex process activity `Deliver Order`, and the instance-specific data is provided. In sum, 20 process instances were supposed to deliver by ship (instantiated process schema *VShip*) of which 15 were in fact delivered by ship. In the other five cases, the process instance was adapted at runtime; three times a train and two times trucks were taken instead. In addition, process schema *VTrain* was instantiated three times and *VTruck* four times (no instance was adapted). Finally, process schema *VAir* was executed three times (with small product samples of 1-3 kg).

Table 1: Extract of Process Instance Log.

Nr.	Schema	S	Instance Data
1	VShip	S1	{Quartz S., Plattling, 1,500 t}
2	VShip	S1	{Coarse S., Wien, 2,000 t}
3	VTrain	S2	{Quartz S., Berlin, 1,700 t}
4	VShip	S1	{Fine S., Plattling, 2,000 t}
5	VShip	S1	{Quartz S., Passau, 1,000 t}
6	VAir	S4	{Quartz S., Moscow, 1 kg}
7	VShip	S3	{Quartz S., Passau, 800 t}
8	VShip	S2	{Quartz S., Plattling, 1,500 t}
9	VShip	S1	{Fine S., Regensburg, 1,500 t}
10	VAir	S4	{Fine S., Berlin, 3 kg}
11	VShip	S1	{Fine S., München, 1,500 t}
12	VShip	S1	{Quartz S., München, 1,500 t}
13	VShip	S1	{Fine S., Passau, 1,500 t}
14	VTruck	S3	{Quartz S., Salzburg, 700 t}
15	VShip	S1	{Fine S., Wiesbaden, 1,500 t}
16	VAir	S4	{Fine S., Paris, 2 kg}
17	VShip	S2	{Quartz S., München, 2,000 t}
18	VTruck	S3.1	{Fertilizer, Passau, 200 t}
19	VShip	S1	{Fine S., Regensburg, 900 t}
20	VShip	S3	{Quartz S., Plattling, 1,500 t}
21	VShip	S1	{Fine S., Plattling, 1,500 t}
22	VTrain	S2.1	{Quartz S., Moscow, 1,000 t}
23	VShip	S1	{Fine S., Wiesbaden, 1,500 t}
24	VTrain	S2	{Quartz S., Plattling, 1,500 t}
25	VShip	S1	{Fine S., Plattling, 1,500 t}
26	VTruck	S3.1	{Fertilizer, Graz, 500 t}
27	VShip	S1	{Quartz S., München, 1,400 t}
28	VShip	S2	{Fine S., Plattling, 1,500 t}
29	VTruck	S3	{Fine S., Passau, 400 t}
30	VShip	S1	{Quartz S., Passau, 1,500 t}

4.2 Select and Rank Alternatives

The *select alternatives* step reduces the number of possible alternatives to *consistent* alternatives, satisfying the constraints imposed by the context of *R*. For a particular process, semantic constraints may be related to various dependencies, including time constraints that may affect the choice of transport mode for a particular delivery and resource constraints that may need to ensure that the actual charge quantity must not exceed a permitted amount (e.g. using an aircraft). Further semantic constraints may be given by location (e.g. actual start and destination points), environment (e.g. existence of an airport at the respective locations), and the overall delivery costs.

Constraints can either be defined manually in the process schema or identified by mining existing process instances. The proposed approach distinguishes between *hard* and *soft* semantic constraints (Sadiq et al., 2005; Pesic et al., 2007):

- *hard constraints* are constraints that must always hold, i.e. they exhibit the same behaviour in all process instances,
- *soft constraints* eventually hold, i.e. they represent guidelines, only mapping to some instances.

In the running example, hard constraints are on the one hand given by all order-related data like product type, destination, and delivery quantity and on the other hand by the transportation means in combination with the environment. To illustrate the identification of constraints consider Figure 4. For all alternative process solutions, applied instance data are summarized within soft constraints. For example, it is a soft constraint that an aircraft only transports quartz and fine sand (coarse sand might be transported as well). Similarly, it might be possible to send a product sample with 4 kg (violation of soft constraint). However, for such implications it is important that the process instance log is of sufficient size and that the ranges for soft constraints are based on a normal curve of distribution. Evidently, the delivery of 1,500 t quartz sand exceeds the capacity of a standard aircraft (violation of hard constraint), so this process solution can be removed from the list of remaining potential alternatives: `S2`, `S2.1`, `S3`, and `S3.1`.

The *rank alternatives* step computes a ranking over consistent alternatives. There are different strategies to provide such a ranking by assessing alternatives on the basis of relevant criteria (Zimmermann and Gutsche, 1991; Tzeng and Huang, 2011; Behzadian et al., 2012). In addition, also a log analysis can contribute to evaluate the suitability of a process alternative, e.g., by relating outcome and soft constraint adherence (Ly et al., 2012). A convenient strategy may also relate to taking the frequency of process instances associated with a consistent alternative into account. Another strategy computes some similarity measures (e.g. based on weights or preference) between the instances and running instance *R*. The higher the frequency or similarity respectively, the more relevant becomes a process alternative *S*.

Referring to the running example, all soft constraint violations are considered first. According to the process instance log, `S3.1` was only used to transport fertilizers. Although it might be possible to also transport sand with a special truck for hazardous goods, it is not recommended. Similarly, process solution `S2.1` was only used for destinations outside the EU, but never for Plattling. Thus, both process solutions are ranked ex aequo behind other alternatives (3^{rd}rank). The remaining two process solutions `S2` and `S3` fulfil all soft constraints and must be ranked based on frequency and similarity regarding instance data. In the extract of the process instance log, the products were more often delivered by train than by truck and the train was more frequently delivering to Plattling, thus, `S2` is ranked before `S3`.

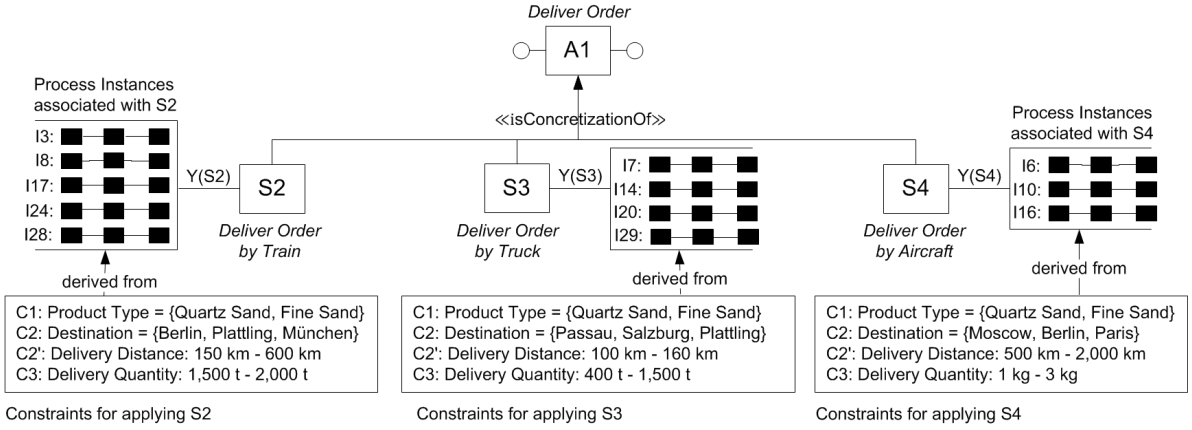

Figure 4: Constraints of Alternative Process Solutions.

4.3 Adapt and Integrate Alternative

The proposed framework also provides the (optional) *adapt alternative* step if manual adaptation is explicitly permitted. An executing actor with sufficient permission can either select an alternative process solution S from the ranked list and manually adapt it resulting in S', or define a completely new process solution S^* if, e.g., no other alternative fits or if the previous steps result in an empty set of possible alternatives. In both cases, possible adaptations are given by the context of the process activities affected by X, which is covered by complex process activity A. Available change operations are then to insert a new activity, and to move, delete, or modify existing activities (in accordance with the four change patterns of the Provop approach (Hallerbach et al., 2010)). In addition, corresponding constraints must be defined that specify the limitations of the new process solution. Manual adaptation is further restricted by the input and output data objects defined in A, which every process solution S' and S^* must consider. In addition, manual adaptation leads to the creation of a new entry in the process execution log $\mathcal{L}$ that comprises S' or S^* on the one hand and inclusion of the adapted process solution in the process pattern library $\mathcal{P}$ on the other hand, thereby generating new knowledge.

For the process instance in our running example, no further process solutions are required. However, for previous orders two process solutions `S2.1` and `S3.1` were defined manually. The *integrate alternative* step automatically integrates either a new process solution or the top-ranked alternative in running process R by replacing the process activities affected by X. (Another possibility is to let domain experts validate the ranked alternatives, so they still have the opportunity to integrate an alternative other than a top-ranked alternative.) This may require a semantic rollback or compensation of already executed activities of the failed process solution, e.g., an executed activity E must be rolled back if the execution of E led to a state change that is not needed by the selected alternative (Reichert and Weber, 2012).

Let us assume that for every possibly failing activity a compensation handler was defined in the process schema. Before replacing the failed process solution `Deliver Order by Ship` with the top-ranked alternative `S2`, the already performed activity `Book Cargo Ship` must be rolled back, i.e. the ship must be cancelled. Then, process solution `S2` is executed, resulting in a successful termination of R and a new entry in the execution log.

5 RELATED WORK

Variability and flexibility both support business process adaptations; flexibility is concerned with runtime decisions, while variability is concerned with design- and customization-time decisions (la Rosa et al., 2013). Regarding dynamic business process modelling languages, it is distinguished between configurable and adaptable approaches. Configurable approaches, such as *Configurable YAWL* and *EPCs*, are useful to adapt, e.g., a domain-specific reference model to a concrete organization but cannot be used for unexpected situations since the process flow must be configured at design-time. Adaptable approaches, in contrast, only require a base process model, which comprises the standard process flow and is adapted at runtime through structural model adaptations based on change patterns. This approach is implemented, e.g., by Provop (Hallerbach et al., 2010) and vBPMN (Doehring and Zimmermann, 2011). Both solutions provide rather basic change operations and patterns respectively, and

they do not support the selection of alternative activities that fulfil given preconditions and result in the desired effect. Another advantage of our approach is that it does not affect the underlying notation, i.e. no adjustment points or adaptive segments are required.

In (Adams et al., 2007), *exlets* are presented as an extensible repertoire of self-contained exception-handling processes for enabling dynamic flexibility in workflows using ripple down rules. The authors further introduce generic handling primitives in the form of patterns characterized by the exception type (Russell et al., 2006). In contrast, the framework proposed in this work captures and maintains domain-specific process patterns by a pattern library, which provides for a hierarchical arrangement of complex process activities and associated process solutions. The hierarchical arrangement reflects specialization and concretisation relations between the elements in the pattern library, which improves intelligibility.

Relevant work regarding integrated compliance of semantic constraints in flexible process management systems is given in (Ly et al., 2012; Sadiq et al., 2005; Pesic et al., 2007). In (Meseguer et al., 2006), the authors give a comprehensive literature review on different formalisms of soft constraints and discuss how they can be dealt with in general solving methods.

Related research on optimal decision making is addressed, e.g., in the mathematical domain, where a variety of problems involving planning, resource allocation, and selecting the optimal solution have been studied. Of particular interest are the resource-constrained project scheduling, dynamic optimization, and constrained optimization problems (Cruz et al., 2011; Hartmann and Briskorn, 2010), as well as multi-objective optimization (Deb, 2014).

In the business process domain, related research is available concerning the recording and analysis of process data to improve business process efficiency and flexibility, typically based on goals or constraints (e.g. in the sub-field of business process mining (van der Aalst, 2011) or business process intelligence (Grigori et al., 2004)). Process mining has become a major topic in recent years. In particular, existing work focuses on reconstructing meaningful process models from process instances given by process execution logs (van der Aalst et al., 2003; Wen et al., 2006; van der Aalst, 2011). In contrast to this application, the proposed approach exploits process instances to identify constraints in the context of a running process instance.

The proposed approach further relates to *case-based reasoning* (CBR) systems (Schulze, 2001; Krampe and Lusti, 1997). CBR-systems are typically used to support the design of complex business processes by finding a similar case in a case library and by subsequent adaptation of a retrieved case to context-specific process needs. The retrieval of relevant cases is realized by sophisticated graph matching algorithms on a process schema level which do not take process knowledge in process execution logs into account. We, thus, argue that the combination of process schema information in terms of process patterns joined with knowledge implicitly captured in process execution logs represents a major contribution in improving selection and ranking of process alternatives for successfully dealing with unexpected situations.

6 CONCLUSION

A key issue in realizing our framework for dynamic business processes represents the population of a process pattern library such that reuse of alternative process solutions becomes feasible. To address this issue, the presented mining approach exploits existing process knowledge available in process models and process execution logs. Advantages of the proposed approach are (i) the possibility to flexibly react to unexpected situations by specifying alternatives at runtime, (ii) a hierarchical structure of process patterns, and (iii) a more intelligible business process, since defining a variety of possible alternatives for every activity (also for very unlikely exceptions) makes a process schema unreadable.

Its feasibility is demonstrated by applying the main steps to a business scenario taken from the industry domain. An ordering process instance was started but could not successfully terminate due to an unexpected situation. Thus, an alternative process solution was identified at runtime, based on domain-specific process patterns and an analysis of related process instances, and automatically integrated in the running process instance.

The work in this paper further motivated research in the domain of resource utilization. Initial ideas for an approach to optimize resources by combining activities across running process instances and by splitting activities to use different transportation means have already been suggested in (Natschläger et al., 2014; Natschläger et al., 2015). Next research goals are to also retrieve and consider external knowledge for the selection and ranking of alternatives, e.g. considering current weather information provided by an appropriate service can further restrict the number of possible delivery options, and to prototypically implement parts of our framework to facilitate an automated analysis of steel production processes.

REFERENCES

Adams, M., ter Hofstede, A., van der Aalst, W., and Edmond, D. (2007). Dynamic, extensible and context-aware exception handling for workflows. In *OTM'07*, pages 95–112. Springer.

Behzadian, M., Khanmohammadi Otaghsara, S., Yazdani, M., and Ignatius, J. (2012). A state-of the-art survey of TOPSIS applications. *Expert Systems with Applications*, 39(17):13051–13069.

Bögl, A., Karlinger, M., Schütz, C., Schrefl, M., and Pomberger, G. (2015). Exploiting semantic activity labels to facilitate consistent specialization of abstract process activities. In *SOFSEM 2015: Theory and Practice of Computer Science*, pages 475–485.

Bögl, A., Natschläger, C., Karlinger, M., and Schrefl, M. (2014). Exploiting process patterns and process instances to support adaptability of dynamic business processes. In *DEXA 2014*, pages 173–177. CPS.

Bögl, A., Schrefl, M., Pomberger, G., and Weber, N. (2009). Automated construction of process goal trees from EPC-models to facilitate extraction of process patterns. In *ICEIS 2009*, pages 427–442.

Cruz, C., González, J., and Pelta, D. (2011). Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*, 15(7):1427–1448.

Deb, K. (2014). Multi-objective optimization. In *Search Methodologies*, pages 403–449. Springer US.

Dixon, J. and Jones, T. (2011). Hype cycle for business process management. Technical Report G00214214, Gartner.

Doehring, M. and Zimmermann, B. (2011). vBPMN: Event-aware workflow variants by weaving BPMN2 and business rules. In *Enterprise, Business-Process and Information Systems Modeling*, volume 81 of *LNBIP*, pages 332–341. Springer.

Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., and Shan, M.-C. (2004). Business process intelligence. *Computers in Industry*, 53(3):321–343.

Hallerbach, A., Bauer, T., and Reichert, M. (2010). Capturing variability in business process models: The Provop approach. *Journal of Software Maintenance and Evolution: Research and Practice*, 22(6 & 7):519–546.

Hartmann, S. and Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *EJOR*, 207(1):1–14.

Krampe, D. and Lusti, M. (1997). Case based reasoning for information system design. In *ICCBR*, pages 63–73.

la Rosa, M., van der Aalst, W., Dumas, M., and Milani, F. (2013). Business process variability modeling: A survey. Technical report, QUT.

Ly, L. T., Rinderle-Ma, S., Göser, K., and Dadam, P. (2012). On enabling integrated process compliance with semantic constraints in process management systems. *Information Systems Frontiers*, 14(2):195–219.

Malone, T., Crowston, K., Lee, J., Pentland, B., Dellarocas, C., Wyner, G., Quimby, J., Osborn, C., Bernstein, A., Klein, M., and O'Donnell, E. (1999). Towards a handbook of organisational processes. *Magement Science*, 45(3):425–443.

Meseguer, P., Rossi, F., and Schiex, T. (2006). Soft constraints. In Rossi, F., Van Beek, P., and Walsh, T., editors, *Handbook of constraint programming*, pages 281–328. Elsevier.

Natschläger, C., Bögl, A., and Geist, V. (2014). Optimizing resource utilization by combining running business process instances. In *ICSOC 2014 workshops and satellite events*, LNCS, pages 120–126. Springer.

Natschläger, C., Bögl, A., Geist, V., and Biro, M. (2015). Optimizing resource utilization by combining activities across process instances. In *European & Asian System, Software & Service Process Improvement & Innovation*, LNCS, pages 155–167. Springer.

Pesic, M., Schonenberg, M. H., Sidorova, N., and Van Der Aalst, W. M. P. (2007). Constraint-based workflow models: Change made easy. In *OTM'07*, pages 77–94. Springer.

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

Russell, N., van der Aalst, W., and ter Hofstede, A. (2006). Workflow exception patterns. In *CAiSE'06*, pages 288–302. Springer.

Sadiq, S. W., Orlowska, M. E., and Sadiq, W. (2005). Specification and validation of process constraints for flexible workflows. *Inf. Syst.*, 30(5):349–378.

Schulze, D. (2001). *Grundlagen der wissensbasierten Konstruktion von Modellen betrieblicher Systeme*. Shaker Verlag.

Thom, L. H., Reichert, M., Chiao, C. M., Iochpe, C., and Hess, G. N. (2008). Inventing less, reusing more, and adding intelligence to business process modeling. In *DEXA*, pages 837–850.

Tzeng, G.-H. and Huang, J.-J. (2011). *Multiple attribute decision making: methods and applications*. CRC Press.

van der Aalst, W. M. and Weijters, A. J. M. M. (2005). *Process Aware Information Systems: Bridging People and Software Through Process Technology*, chapter Process Mining. Wiley-Interscience.

van der Aalst, W. M. P. (2011). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company.

van der Aalst, W. M. P., van Dongena, B. F., Herbst, J., Marustera, L., Schimm, G., and Weijters, A. J. M. M. (2003). Workflow mining: A survey of issues and approaches. *Data & Knowledge Eng.*, 47:237–267.

Wen, L., Wang, J., and Sun, J.-G. (2006). Detecting implicit dependencies between tasks from event logs. In *APWeb*, volume 3841 of *LNCS*, pages 591–603. Springer.

Zimmermann, H.-J. and Gutsche, L. (1991). *Multi-Criteria Analyse*. Springer.