Anti-pattern Detection in Process-Driven Decision Support Systems

Jonas Kirchhoff(✉) and Gregor Engels

Department of Computer Science, Paderborn University, Paderborn, Germany
{jonas.kirchhoff,engels}@upb.de

Abstract. Decision makers increasingly rely on decision support systems for optimal decision making. Recently, special attention has been paid to process-driven decision support systems (PD-DSS) in which a process model prescribes the invocation sequence of software-based decision support services and the data exchange between them. Thus, it is possible to quickly combine available decision support services as needed for optimally supporting the decision making process of an individual decision maker. However, process modelers may accidentally create a process model which is technically well-formed and executable, but contains functional and behavioral flaws such as redundant or missing services. These flaws may result in inefficient computations or invalid decision recommendations when the corresponding PD-DSS is utilized by a decision maker. In this paper, we therefore propose an approach to validate functionality and behavior of a process model representing a PD-DSS. Our approach is based on expressing flaws as anti-patterns for which the process model can be automatically checked via graph matching. We prototypically implemented our approach and demonstrate its applicability in the context of decision making for energy network planning.

Keywords: Decision support systems · Process-driven applications · Process model validation · Citizen development · Digital ecosystem

1 Introduction

Current business environments require corporate decision makers to consider many frequently changing influencing factors with uncertain future developments and unknown cause-effect relationships during decision making [6,21]. For this reason, many decision makers rely on a digital decision support system (DSS) to assist them in the identification of optimal decisions utilizing a combination of (model-based) simulation and optimization [29]. DSS developers face two related challenges [15]: First, the requirements and preferences for decision support of decision makers are often very individual (cf. [25, p. 4]). For instance, decision

Partially supported by the North Rhine Westphalian Ministry of Economic Affairs, Innovation, Digitalisation and Energy (MWIDE) through grant 005-2011-0022 and the European Regional Development Fund (ERDF) through grant EFRE-0801186.

N. Carroll et al. (Eds.): ICSOB 2022, LNBIP 463, pp. 227–243, 2022.
https://doi.org/10.1007/978-3-031-20706-8_16

makers' requirements are influenced by decision goals, constraints imposed by their business or the law, and availability of data and time to identify an optimal decision. This individuality implies that decision makers should ideally be provided with a DSS which is tailored to their individual requirements for decision support to ensure the recommendation of optimal decisions. Second, the volatility of business environments requires a quick availability of such a tailored DSS [23]. Otherwise, decisions are delayed or suboptimal decisions are accepted due to a lack of computation time. Both delayed and suboptimal decisions – regardless whether they originate from a misalignment with decision support requirements or an untimely supply of the DSS – may result in a decreased competitiveness of the business and therefore must be avoided.

Businesses recently show an increased interest in low-code or no-code development platforms which enable domain experts with no programming skills (sometimes also referred to as *citizen developers*) to quickly create individual software applications by utilizing concepts from model-driven engineering [28]. This can also be applied to DSS development: Non-programmers, in particular domain experts or decision makers themselves, could quickly create a tailored DSS for their individual decision support requirements by combining multiple automated, interoperable decision support services encapsulating reusable decision support functionality provided by DSS developers [15]. The combination of decision support services can be formally described by a process model which specifies the invocation sequence of services and the exchange of data between them. This approach of describing a DSS as a process-based composition of services is referred to as a *process-driven decision support system (PD-DSS)* [14].

A challenge of this process-driven approach to DSS creation is ensuring the correctness of the process model representing a PD-DSS (cf. [2]). During *process execution*, i.e., PD-DSS usage, any flaws introduced during *process design* could potentially result in invalid decision recommendations or the inability to compute any decision recommendations at all. In previous work, we have therefore focused on validating a correct dataflow between decision support services [14] and checking the alignment of selected services and a decision maker's requirements for decision support [16]. In this paper, we focus on detecting additional *functional and behavioral flaws* in the process model which may represent an actual (semantic) error or at least a deviation from the widely accepted norm. For instance, the process model may specify repetitive invocations of the same service although a subsequent invocation will not yield any new insights and therefore only introduce a delay during DSS execution; or a post-processing is missing which is required to reverse a previous pre-processing to ensure that the computed recommendations actually apply to the original data. These flaws usually occur accidentally, either due to carelessness, or due to an unfamiliarity of domain experts and decision makers with the decision support services provided by DSS developers. Nevertheless, they negatively impact effectiveness or efficiency of the corresponding PD-DSS when used by the decision maker.

The remainder of this paper is structured as follows: Sect. 2 provides additional background on PD-DSS creation to support the explanations throughout

the paper. We discuss related work on process validation in Sect. 3 and derive requirements for an approach to detect functional and behavioral flaws in PD-DSS throughout Sect. 4. We explain our proposed PD-DSS validation approach in Sect. 5 which is based on describing flaws as visual process anti-patterns that can be automatically identified in the PD-DSS process model via graph matching. Section 6 discusses our approach with respect to the initially formulated requirements using insights of a prototypical implementation which was applied the example domain of electricity distribution network planning. Section 7 summarizes our findings and presents future work.

2 Background: Process-Driven Decision Support Systems

In this section, we explain the environment for creating PD-DSS (Sect. 2.1), provide an example process model representing a PD-DSS (Sect. 2.2), and discuss the need for assistance in the form of process model validation during PD-DSS creation (Sect. 2.3).

2.1 Decision Support Ecosystems for PD-DSS Creation

Section 1 presented the challenge of DSS developers to quickly provide each (corporate) decision maker with a DSS that is tailored to the decision maker's individual requirements for decision support. Otherwise, there is a risk of decreased competitiveness due to delayed or suboptimal decisions. As a solution to this challenge, we previously proposed the concept of a *decision support ecosystem* (DSE) [15]. Inspired by the citizen development concept of modern low-code platforms [28], the idea of a DSE is to provide a shared platform where multiple stakeholders can collaborate to assist decision makers in the creation of their own, tailored DSS using concepts from model-driven software engineering.

A simplified architecture for the platform of a DSE is shown in Fig. 1: Initially, a *decision maker* formulates his or her *requirements for decision support*. This information is used by a *PD-DSS engineer* to describe a tailored PD-DSS by composing multiple interoperable (software-based) *decision support services* provided by *DSS developers* via a public service repository. These services may encapsulate a single simulation or optimization model, but can also provide pre- or post-processing functionality such as data visualization. During composition, the PD-DSS engineer is supported by an *assistance* which ensures the implementation of best practices documented by domain experts. The composition of services for a PD-DSS is specified using an executable *process model*. The model-based composition combined with the composition assistance enables domain experts or decision makers without programming skills to assume the role of the *PD-DSS engineer*. After the process model representing the PD-DSS has been created, it can be executed by the *PD-DSS enactment application* such that a decision maker is under the impression of interacting with a regular DSS, i.e., has to provide input data and is provided with output data in the form of decision recommendations.

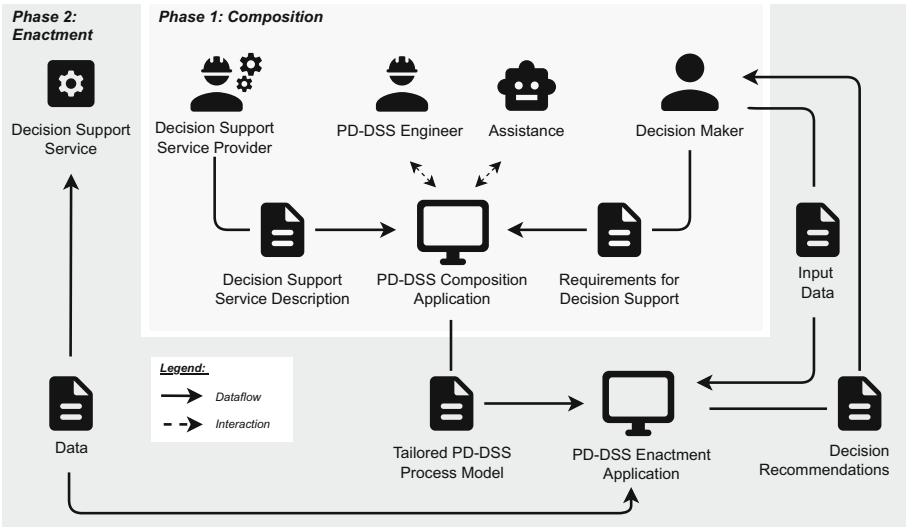

Fig. 1. Overview of the DSE platform (based on [15]).

2.2 PD-DSS: DSS Specification Using Process Models

The usage of a process model to represent a tailored DSS is motivated by two advantages [14]: First, process models have already been successfully applied for service composition in (generic) service-oriented architectures in the form of *process-driven applications* [34]. This can be explained by the fact that process models are able to capture multiple perspectives, in particular, the functional and behavioral perspective describing the conditional execution sequence of activities, the informational perspective describing the data exchange between activities, and the operational perspective documenting the software services executing an activity [35]. Second, by using BPMN [24] – the "de-facto standard" for process modeling popular with domain experts [2,20] – the creation of a process model representing a tailored, process-driven DSS can even be done by stakeholders with no programming skills and without professional or extensive upfront training [14,27].

An example for a process model representing a PD-DSS in the application domain of regional electricity distribution network planning [13] is given in Fig. 2. Regional electricity distribution networks are responsible for transporting generated electricity to end consumers. During network planning, distribution network operators must decide which parts of the network infrastructure such as cables and transformers should be replaced within the next decade. In this context, operators try to minimize investment costs while maintaining a high degree of network reliability given the expected electricity demand of consumers. The PD-DSS shown in Fig. 2 supports such an investment-minimizing planning for an electricity network provided by the decision maker during PD-DSS execution (shown as the "original network" input data object). Each activity is associated

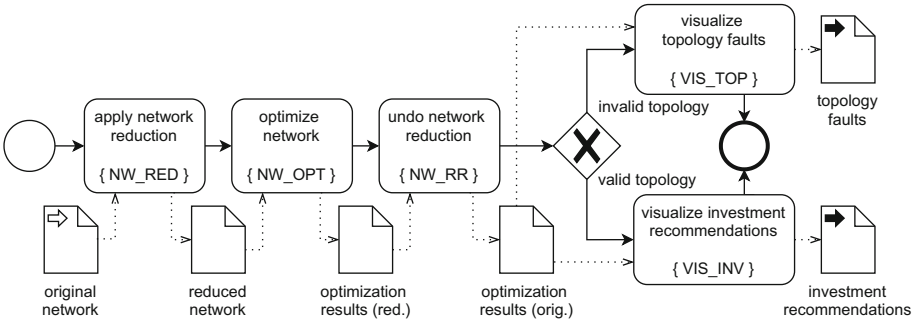

Fig. 2. Exemplary BPMN process for a PD-DSS.

with an automated software service used to execute the activity. The service name is written in curly braces. Initially, a network reduction aggregates multiple consumers and cables using the `NW_RED` service to comply with network size restrictions of the subsequent network optimization service `NW_OPT`. The network reduction must be reversed after network optimization using the `NW_RR` service to ensure that the optimization results computed for the aggregated network are actually applicable to the original network. Next, the output of the optimization output is checked using an exclusive gateway. In case the optimization encountered an error in the topology which made it infeasible to compute any investment recommendations, the topology errors are visualized using the `VIS_TOP` service. Otherwise, if optimization completed successfully, the investment recommendations are visualized using the `VIS_INV` service. Any visualization will be returned to the decision maker (as indicated by the BPMN output data object).

2.3 Modeling Assistance to Reduce PD-DSS Flaws

Flaws accidentally introduced into the process model will either result in invalid or no decision recommendations, which in turn result in suboptimal or delayed decisions decreasing competitiveness. Flaws can affect all perspectives of the process model. In previous work, we focused on detecting flaws in the informational perspective, i.e., incompatible data being provided to a service [14], and flaws in the operational perspective, i.e., the selection of a service incompatible with a decision maker's requirements for decision support [16].

This paper focuses on the detection of flaws in the functional and/or behavioral perspective of a PD-DSS process model. A process model with such kind of flaws is shown in Fig. 3 based on the same use case presented in Fig. 2. The process contains three flaws: First (and still fairly easy to spot), the network optimization is executed twice although the optimization of an already optimized network does not yield any additional improvements. While this is not an error which would prevent execution of the PD-DSS at runtime, it adds an unnecessary delay when using the DSS due to the time consumption when (re-)creating the

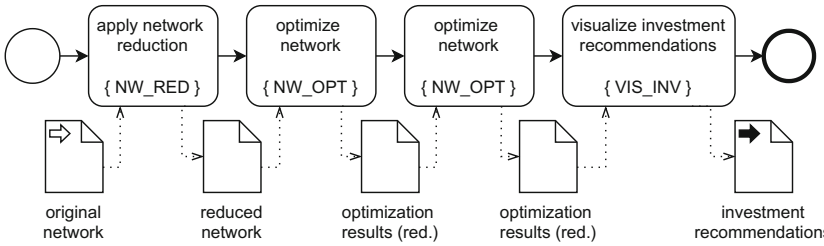

Fig. 3. Exemplary BPMN process for a PD-DSS with flaws.

underlying optimization model. Second, the initially performed network reduction is not reversed after optimization. As a result, the investment recommendations which are later presented to the decision maker are likely not applicable to the original network, as for instance the recommendation to replace cable #7 would actually correspond to the replacement of cables #2, #3 and #5 in the original network which – unknown to the decision maker – were aggregated into cable #7 during network reduction. Third, it was not considered that the result of network optimization does not always describe investment recommendations. Instead, the result can also document faults in the network topology which prevented the optimization from computing optimal investment recommendations (cf. gateway in Fig. 2).

The example demonstrates that functional and behavioral flaws in a process model representing a PD-DSS are often specific to a concrete application domain. They may affect (non-)consecutive activities as well as the conditional execution and non-existence of activities. Flaws contained in the process model during design time can manifest as both errors or "bad smell" during process execution when using the PD-DSS, i.e., invalid or delayed decision recommendations, and consequently should be avoided.

3 Related Work

We already discussed the relation between our previously summarized concept of a decision support ecosystem and variants of decision support systems, digital ecosystems and end-user programming in previous work (cf. [15]). In alignment with the focus of this paper, we subsequently discuss approaches for the functional and behavioral validation and verification of executable process models describing a software application in the context of a service-oriented architecture.

A software application characterized by an executable process model which describes the composition of multiple software and data services has previously been referred to as *process-driven application* (PDA) [34]. Schneid et al. present approaches for the detection of errors in PDAs with respect to the data exchange between activities/services, either based on static analysis [31,33] or test generation [32]. We also describe approaches for static validation of dataflow correctness [14] and service selection [16] with a specific focus on PD-DSS. In the app-

roach by Schiffner et al. [30], an automatic validation verifies message exchange between activities, while other aspects of the process model are validated manually among stakeholders. The time consumption of such manual validation is obstructive for the quick creation of a tailored PD-DSS. In summary, the discussed validation approaches can only automatically detect informational or operational flaws, but no functional or behavioral flaws.

Since process model validation for PDAs is limited, we also briefly assess the reusability of validation and verification approaches for non-executable business process models. Many approaches are based on formal (mathematical) verification methods such as automata, petri-nets and process algebras [22]. The complexity of these approaches requires experts to define validation rules, which in turn target generic, domain-agnostic flaws such as deadlock detection to maximize reusability. Consequently, they cannot be immediately reused for the detection of domain-specific functional and behavioral flaws. Process anti-patterns or weakness patterns are often domain-specific and have emerged as a (graphical) abstraction which can sometimes be mapped to formal approaches for automated validation [11]. These types of patterns "represent typical problems a process may have together with ideas of how to address them" [4]. The taxonomy of anti-patterns created by Koschmider et al. [17] shows that anti-patterns can detect flaws regarding the process model syntax, control-flow, understandability, cooperation, data-flow, business rules and the modeled business process itself. However, when reviewing the publications of the (sub-)categories "rule-related defects", "need for process improvements" and "compliance" which seem closest to the goal of our approach, we quickly found limitations that prevent reuse for our use case: Unlike our paper, many approaches do not focus on flaws in the functional and/or behavioral perspective, but instead consider flaws in dataflow [10], security requirements [26], natural language labels [19], process inefficiencies with respect to complexity or resource consumption [5], process adaptation [9], or exchange across processes [18]. Some papers require the definition of anti-patterns in a query language such as SQL [4] or GMQL [7,8] which is unsuitable in the citizen development context of our DSE. Other approaches either explicitly require a manual detection [12] or do not describe the automated detection approach [3]. Four other papers which are listed by Koschmider et al. [17] for the selected categories but are not discussed here present collections of anti-patterns but no (automated) detection approaches.

4 Solution Requirements

The subsequently presented requirements for an approach to detect functional and behavioral flaws in PD-DSS were identified based on experience in a research project for energy distribution network planning [13] and the review of related work (cf. Sect. 3). The requirements are grouped into two categories: *Detection* requirements (Sect. 4.1) focus on expressiveness required to detect flaws in PD-DSS process models, while *integration* requirements (Sect. 4.2) focus on the integration of our approach with our DSE concept (cf. Sect. 2.1).

4.1 Detection Requirements

We define requirements *D1–D4* for the functional capabilities of the detection approach with illustrating examples referring to Sect. 2.2.

D1 – (Non-)Consecutiveness. Errors involving multiple activities must consider the sequential flow between activities. In the simplest case, an activity immediately follows another activity, e.g., the consecutive network optimization in Fig. 3. However, other activities may happen in between activities involved in the flaw. In example shown in Fig. 2, the network reduction just must be reversed, but other activities such as network optimization will happen in between.

D2 – Conditionality. A BPMN process model supports the conditional execution of activities, in particular using gateways and/or events. Consequently, it must also be possible to consider conditional activity execution during the definition of flaws, e.g., to ensure that different activities are performed based on the optimization results (cf. Fig. 2). The conditional execution of activities also implies multiple "execution paths" throughout the process model. Although flaws may only be present in one of those paths, a single flawed path can already limit the usefulness of the corresponding PD-DSS at runtime.

D3 – Missing Elements. A flaw may not only be characterized by the existence of elements in the model, but also by the non-existence of elements. In this case, "elements" may refer to activities, gateways, events and flows. For instance with respect to Fig. 3, the reversal of a previously performed network reduction or the conditional check of optimization results are missing.

D4 – Generalization. Some flaws can be generalized to multiple services/activities. For instance, the double optimization in Fig. 3 should also be avoided for network reduction and visualization of investment recommendations. For this purpose, it should be possible to match arbitrary activities if desired. As the "redundant invocation" example suggests, it might be necessary to identify the matched activity/service across the remainder of the process model, e.g., to ensure that two consecutive activities are implemented using the same service.

4.2 Integration Requirements

We define requirements *I1–I4* for the integration of the error detection approach in our proposed decision support ecosystem (DSE) concept (cf. Sect. 2.1):

I1 – Adaptability. Since our proposed DSE concept is agnostic of a concrete application domain, the detection approach should also work for arbitrary application domains. Within a concrete application domain, the approach should be extensible to detect newly discovered flaws, acknowledging the continuous knowledge gain in PD-DSS composition. Lastly, the detection approach should also consider concrete preferences of a PD-DSS modeler, i.e., a modeler may disagree with certain best practices and therefore prefer to ignore certain flaws.

I2 – Expert Definition. Our proposed DSE concept strives for citizen development, i.e., the development of PD-DSS by decision makers and domain experts without programming skills. Consequently, these stakeholders should also be able to define flaws to be detected in a process model representing a

PD-DSS. As an additional benefit, stakeholders should be able to use a familiar notation for the definition of these errors to minimize upfront training.

I3 – Traceability. The overall goal of the detection approach is the improvement of a PD-DSS process model created by a DSE participant. Consequently, instead of simply reporting that a flaw was discovered, it is instead necessary to report which flaw was found and where in the process model it is located in order to fix it quickly. For additional comprehensiveness, it is beneficial to explain why the error is considered detrimental and how it can be fixed.

I4 – Continuous Feedback. The approach should enable continuous feedback – ideally whenever the PD-DSS modeler edits the process model. This ensures that a flaw is detected early and subsequent "follow-up" flaws are avoided. The ability to provide continuous feedback requires automatic detection of flaws with a sufficient speed. However, it is hard to further quantify the speed requirement as the detection runtime is influenced by the size of the process model and the collection documenting potential flaws, as well as the execution hardware.

5 Anti-pattern Detection in PD-DSS

Our solution approach to detect functional and behavioral flaws in PD-DSS is based on visually describing flaws as process anti-patterns which can be identified in the process model representing a PD-DSS via graph matching. Considering our discussion of related work in Sect. 3, the motivation to use anti-patterns is twofold: First, anti-patterns align with our use case of PD-DSS improvement as they not only document flaws of a process model, but are usually also accompanied with a description of how to address the flaw. Second, (visual) anti-patterns provide an abstraction which enables any domain expert to describe composition flaws which benefits the collaboration aspect of our proposed DSE (cf. Sect. 2.1).

The following subsections explain our solution architecture (Sect. 5.1), the visual definition of anti-patterns (Sect. 5.2), and the translation of (anti-pattern) process models into graph database queries to identify flaws (Sect. 5.3).

5.1 Solution Architecture

Figure 4 presents an architecture for the integration of our anti-pattern based detection approach into our DSE concept. *Domain experts* use the *anti-pattern definition application* to define an *anti-pattern* which is stored in the *anti-pattern repository*. *PD-DSS engineers* use the *PD-DSS composition application* (cf. Sect. 2.1) to define a composition of decision support services as a *process model*. PD-DSS engineers also use the *anti-pattern selection application* to select the anti-patterns they want to check their process model for, including a severity level for each anti-pattern of error, warning, etc. The resulting model and selected anti-patterns are forwarded to the *PD-DSS validation service* which checks whether the anti-patterns are present in the process model. For this purpose, the validation service's *converter module* initially converts the process

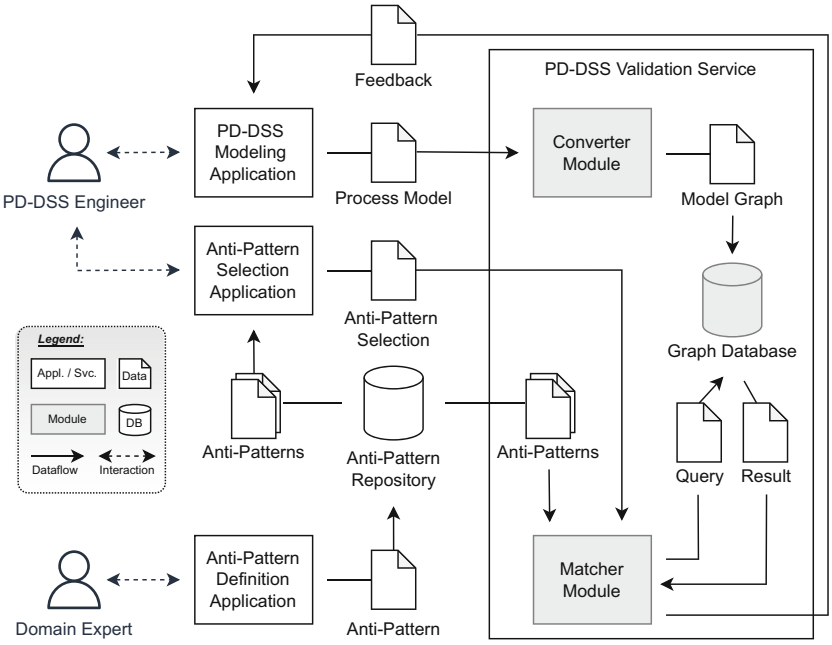

Fig. 4. Proposed architecture for flaw detection.

model into a graph representation which is stored in a *graph database*. Subsequently, each anti-pattern – which was translated into a database query during its initial definition with the *anti-pattern definition application* – can be queried against the graph database to check if the anti-pattern is contained in the process model. This is done iteratively by the *matcher module* which also aggregates and transforms the database results in a *feedback* format which is shown to the PD-DSS engineer via the PD-DSS modeling application to highlight discovered anti-patterns, i.e., flaws in the PD-DSS composition.

The described flow through the architecture can be viewed as iterative, i.e., whenever the PD-DSS engineer makes changes to the process model, the *PD-DSS validation service* can check the (intermediate) model for the existence of anti-patterns and provide feedback to the PD-DSS engineer. The *anti-pattern selection application* can be omitted if all anti-patterns should always apply with the same severity.

5.2 Anti-pattern Definition

Following the recommendations extracted by Koschmider et al. [17], our anti-pattern definition consists of (1) a human-readable name for easier recognition, (2) a visual specification of the anti-pattern as a partial process model, (3) the reason for defining the anti-pattern to support acceptance and comprehensiveness, and (4) a natural language description of how to resolve the anti-pattern

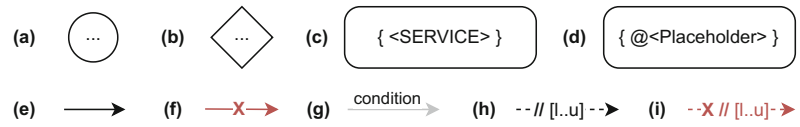

Fig. 5. (Extended) BPMN elements to model anti-patterns.

Fig. 6. Exemplary anti-patterns related to PD-DSS process models of Sect. 2.

to quickly improve the process model. Additionally, we consider (5) a default severity level, e.g., error or warning. This information is specified by the anti-pattern author during definition, although the severity level can be overwritten by the DSS modeler during anti-pattern selection (cf. Sect. 5.1).

We use a visual yet formalized notation to support both anti-pattern definition by non-developers as well as a subsequent automatic anti-pattern detection. Since we already use BPMN for the definition of PD-DSS process models, we base our visual anti-pattern specification on *BPMN-Q*, an already established visual query language for BPMN process models [1]. We nevertheless explain all supported elements for anti-pattern specification as we slightly deviate from BPMN-Q to better support our use case of detecting flaws in PD-DSS.

All elements available for anti-pattern definition are shown in Fig. 5. They include (a) arbitrary events, (b) arbitrary gateways, (c) activities matching a specific service such as `NW_OPT`, (d) a named or unnamed placeholder (always) matching a concrete but arbitrary activity, (e) a sequential flow, (f) a non-existing sequential flow, (g) a condition to augment *(e)* or *(f)*, (h) a non-sequential flow ("sometime later"), optionally with a minimum of l and a maximum number of u nodes in between, and (i) a non-existing non-sequential flow.

Figure 6 shows the application of the elements to specify anti-patterns based on the examples discussed in Sect. 2. In natural language, they can be described as (a) *"No consideration of different optimization outputs"*, (b) *"Consecutive redundant activities"*, (c) *"Missing reversal of a network reduction"*, and (d) *"Relying on the network optimization to detect topology faults"* (because a specialized service may be faster or more precise than the optimization heuristic).

5.3 Translation to Graph Database

We translate the original process model into a graph representation suitable for storage in a graph database, and anti-patterns into queries which can be run against the database. Using a graph database is motivated by the fact that

a process model (and consequently the anti-patterns) already correspond to a graph [8]. This reduces transformation effort and could potentially result in faster query execution compared to other database types. Additionally, the chosen graph database *Neo4j*[1] provides enterprise-grade robustness and is freely available while we were unable to quickly find a BPMN-Q compatible toolchain.

Translation "Process Model to Graph". Each activity, gateway and event is translated into a node in the graph. The *type* (`ACTIVITY`, `GATEWAY`, `EVENT`) and *variant* (e.g., `EXCLUSIVE` or `PARALLEL` for gateways and `START` or `EXCEPTION` for events) of the element is preserved using labels. For activities, the variant corresponds to the implementing service, e.g., `NW_OPT`. Additionally, the ID and name of the element in the original process model are recorded in the graph for traceability. An edge is inserted between two nodes if a sequential flow exists between the two elements in the process model. For conditional flows, a *condition* label documents the condition associated with the flow. Elements are inserted using `CREATE` queries expressed in *Cypher*, *Neo4j*'s query and graph manipulation language. We refer to Sect. 6 for exemplary *Cypher* queries for graph creation.

Translation "Anti-pattern to Query". Since anti-patterns are partial process models, translating an anti-pattern into a graph query fundamentally follows the same translation concept previously applied for the process model. Additionally, we must consider the elements introduced in Fig. 5. We describe their translation in Table 1 with respect to *Cypher*'s fundamental query structure `MATCH p=... WHERE ... RETURN p` which matches a path `p` containing the specified nodes and fulfilling the specified conditions. Elements *(a)* to *(d)* match nodes in the graph. Within parenthesis, a *variable* (e.g., `n`) identifies a matched node while labels (e.g., `ACTIVITY`) are separated by a colon. We use two expressions which must be replaced for a concrete element, i.e., `<SERVICE>` with the name of the service implementing the activity and `<VARIANT>` with a gateway/event variant (cf. process model transformation). For activity placeholders *(d)*, the service label is omitted and a condition is inserted which ensures the equality of implementing services in case of named placeholders. Elements *(e)* to *(i)* focus on edges between nodes which are fundamentally expressed as `-[]->`. Within the square parenthesis, a variable name or repeatability can be specified. Due to the augmentation aspect of conditions, *(g)* does not introduce a match clause. For missing edges *(f)* & *(i)*, a (non-consecutive) path to the end event should be included in the anti-pattern to return the path where the node is missing instead of the starting node only. Potentially missing activities are optionally matched.

Given a complete anti-pattern to translate into a query, each edge is traversed and the `MATCH` and `WHERE`-statements of the associated elements as per Table 1 are collected. During this traversal, duplicate node matches are avoided and unique variable names are ensured using global counters (instead of the generic `n,m,p,r` in Table 1). Furthermore, nodes using named placeholders are tracked.

[1] https://neo4j.com/.

Table 1. Translation of anti-pattern elements of Fig. 5 to partial *Cypher* queries.

Figure 5	MATCH	WHERE
(a)	`(n:<VARIANT>:EVENT)`	
(b)	`(n:<VARIANT>:GATEWAY)`	
(c)	`(n:<SERVICE>:ACTIVITY)`	
(d)	`(n:ACTIVITY) (m:ACTIVITY)`	`labels(n)=labels(m)`
(e)	`p=(n)-[r]->(m)`	
(f)		`NOT exists((n)-[r]->(m))`
(g)		`r.condition="..."`
(h)	`p=(n)-[*<l>..<u>]->(m)`	
(i)		`NOT exists((n)-[*<l>..<u>]->(m))`

After the anti-pattern has been traversed, the `MATCH`-statements are sorted to ensure that nodes are matched before the edges they are involved in. Additional `WHERE`-statements are added for ensuring placeholder equality which cannot be done during edge traversal because placeholder activities may not immediately follow each other. `MATCH` and `WHERE`-statements are concatenated using a logical `AND` for `WHERE`-statements, and all matched paths variables are `RETURN`ed. Again, we refer to Sect. 6 for exemplary *Cypher* queries.

6 Demonstration and Discussion

We show the technical feasibility of our anti-pattern detection approach with a publicly available demonstration[2]. We prototypically implemented the translation of anti-patterns to *Neo4j*'s *Cypher* queries and applied it in the example application domain of energy distribution network planning. Many examples from the demonstration have already been utilized to visualize concepts explained in the paper, e.g., Figs. 2, 3 and 6. Interested readers may consult the implementation for additional details such as the resulting *Cypher* queries. Based on the demonstration and the explanations throughout the paper, we conclude that our approach addresses the requirements presented in Sect. 4 as follows:

The detection requirements *D1–D4* are primarily addressed by the anti-pattern elements shown in Fig. 5. In particular, *D1 – (Non-)Consecutiveness* is addressed with elements *(e)* to *(i)*, *D2 – Conditionality* with elements *(b)* and *(g)*, *D3 – Missing Elements* with elements *(f)* and *(i)*, and *D4 – Generalization* with element *(d)*. A sufficient expressiveness of these elements in anti-pattern construction was confirmed throughout the demonstration using our prototypical implementation based on *Neo4j*. However, at least to some extent, these requirements were derived from our experience in a particular application domain and additional detection functionality may be needed in other domains.

[2] https://github.com/krchf/process-graph-antipattern-detection.

The integration requirements *I1–I4* are primarily addressed by the solution architecture discussed in Sect. 5.1. Extensibility required as part of *I1 – Adaptability* is provided by the *anti-pattern repository* filled by domain experts using the *anti-pattern definition application* (which also addresses *I2 – Expert Definition* by utilizing the graphical anti-pattern notation of Sect. 5.2). Preferences of PD-DSS engineers are considered via the *anti-pattern selection application*. Furthermore, the architecture can be instantiated in arbitrary application domains. For *I3 – Traceability*, paths matching an anti-pattern can be translated to parts of the original process model. *I4 – Continuous Feedback* is primarily addressed by the repeated, automatic matching of anti-patterns. In our prototypical implementation, the uncached queries representing anti-patterns seldomly exceeded 15ms using *Neo4j-Docker* on a 2018 laptop. However, we found anti-patterns including a missing flow between activities to take significantly longer, potentially exceeding 100ms. Nevertheless, all subsequent queries usually completed in less than 3ms, most likely due to caching. We believe the utilization of caching is reasonable as process model changes do not require the re-generation of the whole graph but only updates for affected elements. Furthermore, queries could be optimized by combining node and edge matches as discussed in the prototypical implementation. While we can provide measurements for *I4 – Continuous Feedback*, we acknowledge that other requirements call for additional studies, e.g., with domain experts to evaluate *I2 – Expert Definition* or case studies in other application domains to evaluate requirement *I1 – Adaptability*.

7 Summary and Future Work

We presented an approach for the detection of functional and behavioral flaws in process models describing a process-driven decision support system as composition of software-based decision support services. Our approach is based on the visual definition of flaws as process anti-patterns which can be automatically detected in the original process model via graph matching. We furthermore presented an architecture to integrate our approach into a digital ecosystem for the collaborative creation of PD-DSS. The technical feasibility of the approach was demonstrated in the context of energy distribution network planning. In addition to the already discussed extensions, future work could adapt the approach to support automatic anti-pattern fixes based on graph transformations (e.g., to automatically remove redundant activities), exceptions to anti-patterns under certain conditions, or full BPMN-Q compatibility.

References

1. Awad, A.: BPMN-Q: a language to query business processes. In: Enterprise Modelling and Information Systems Architectures - Concepts and Applications, pp. 115–128. Gesellschaft für Informatik e. V. (2007)

2. Awad, A., Decker, G., Lohmann, N.: Diagnosing and repairing data anomalies in process models. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 5–16. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12186-9_2
3. Becker, J., Bergener, P., Räckers, M., Weiß, B., Winkelmann, A.: Pattern-based semi-automatic analysis of weaknesses in semantic business process models in the banking sector. In: ECIS 2010 Proceedings (2010)
4. Becker, J., Bergener, P., Breuker, D., Raeckers, M.: An empirical assessment of the usefulness of weakness patterns in business process redesign. In: ECIS 2012 Proceedings (2012)
5. Becker, J., Weiß, B., Winkelmann, A.: Automatic identification of structural process weaknesses - experiences with semantic business process modeling in the financial sector. In: Wirtschaftsinformatik Proceedings 2011, pp. 787–807 (2011)
6. Bennett, N., Lemoine, G.J.: What a difference a word makes: understanding threats to performance in a VUCA world. Bus. Horiz. **57**(3), 311–317 (2014)
7. Bergener, P., Delfmann, P., Weiss, B., Winkelmann, A.: Detecting potential weaknesses in business processes. Bus. Process. Manag. J. **21**(1), 25–54 (2015)
8. Delfmann, P., Steinhorst, M., Dietrich, H.A., Becker, J.: The generic model query language GMQL - conceptual specification, implementation, and runtime evaluation. Inf. Syst. **47**, 129–177 (2015)
9. Döhring, M., Heublein, S.: Anomalies in rule-adapted workflows - a taxonomy and solutions for vBPMN. In: 2012 16th European Conference on Software Maintenance and Reengineering, pp. 117–126 (2012)
10. Eleftheriou, I., Embury, S.M., Brass, A.: Data journey modelling: predicting risk for IT developments. In: Horkoff, J., Jeusfeld, M.A., Persson, A. (eds.) PoEM 2016. LNBIP, vol. 267, pp. 72–86. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48393-1_6
11. Förster, A., Engels, G., Schattkowsky, T., Van Der Straeten, R.: Verification of business process quality constraints based on visual process patterns. In: First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE 2007), pp. 197–208 (2007)
12. Held, M., Blochinger, W.: Structured collaborative workflow design. Futur. Gener. Comput. Syst. **25**(6), 638–653 (2009)
13. Kirchhoff, J., Burmeister, S.C., Weskamp, C., Engels, G.: Towards a decision support system for cross-sectoral energy distribution network planning. In: Energy Informatics and Electro Mobility ICT (2021)
14. Kirchhoff, J., Gottschalk, S., Engels, G.: Detecting data incompatibilities in process-driven decision support systems. In: Shishkov, B. (ed.) BMSD 2022. LNBIP, vol. 453, pp. 89–103. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-11510-3_6
15. Kirchhoff, J., Weskamp, C., Engels, G.: Decision support ecosystems: definition and platform architecture. In: Cabral Seixas Costa, A.P., Papathanasiou, J., Jayawickrama, U., Kamissoko, D. (eds.) ICDSST 2022. LNBIP, vol. 447, pp. 97–110. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-06530-9_8
16. Kirchhoff, J., Weskamp, C., Engels, G.: Requirements-based composition of tailored decision support systems. In: Bernhaupt, R., Ardito, C., Sauer, S. (eds.) Human-Centered Software Engineering, pp. 150–162. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-14785-2_10
17. Koschmider, A., Laue, R., Fellmann, M.: Business process model anti-patterns: a bibliography and taxonomy of published work. In: Proceedings of the 27th European Conference on Information Systems (ECIS) (2019)

18. Kurniawan, T.A., Ghose, A.K., Lê, L.-S.: Resolving violations in inter-process relationships in business process ecosystems. In: Ghose, A., et al. (eds.) ICSOC 2012. LNCS, vol. 7759, pp. 332–343. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37804-1_34
19. Laue, R., Koop, W., Gruhn, V.: Indicators for open issues in business process models. In: Daneva, M., Pastor, O. (eds.) REFSQ 2016. LNCS, vol. 9619, pp. 102–116. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30282-9_7
20. Lohmann, N., Nyolt, M.: Artifact-centric modeling using BPMN. In: Pallis, G., et al. (eds.) ICSOC 2011. LNCS, vol. 7221, pp. 54–65. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31875-7_7
21. Mack, O., Khare, A.: Perspectives on a VUCA world. In: Mack, O., Khare, A., Krämer, A., Burgartz, T. (eds.) Managing in a VUCA World, pp. 3–19. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-16889-0_1
22. Morimoto, S.: A survey of formal verification for business process modeling. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008. LNCS, vol. 5102, pp. 514–522. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69387-1_58
23. Mustafin, N., Kopylov, P., Ponomarev, A.: Knowledge-based automated service composition for decision support systems configuration. In: Silhavy, R., Silhavy, P., Prokopova, Z. (eds.) CoMeSySo 2021. LNNS, vol. 231, pp. 780–788. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90321-3_63
24. Object Management Group: Business process model and notation (2014). https://www.omg.org/spec/BPMN/. Accessed 21 June 2022
25. Parnell, G.S., Bresnick, T.A., Tani, S.N., Johnson, E.R.: Handbook of Decision Analysis. Wiley, Hoboken (2013)
26. Ramadan, Q., Strüber, D., Salnitri, M., Riediger, V., Jürjens, J.: Detecting conflicts between data-minimization and security requirements in business process models. In: Pierantonio, A., Trujillo, S. (eds.) ECMFA 2018. LNCS, vol. 10890, pp. 179–198. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92997-2_12
27. Recker, J.: Opportunities and constraints: the current struggle with BPMN. Bus. Process. Manag. J. **16**(1), 181–201 (2010)
28. Sahay, A., Indamutsa, A., Di Ruscio, D., Pierantonio, A.: Supporting the understanding and comparison of low-code development platforms. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 171–178 (2020)
29. Savić, D.A., Bicik, J., Morley, M.S.: A DSS generator for multiobjective optimisation of spreadsheet-based models. Environ. Model. Softw. **26**(5), 551–561 (2011)
30. Schiffner, S., Rothschädl, T., Meyer, N.: Towards a subject-oriented evolutionary business information system. In: 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, pp. 381–388 (2014)
31. Schneid, K., Kuchen, H., Thöne, S., Di Bernardo, S.: Uncovering data-flow anomalies in BPMN-based process-driven applications. In: Proceedings of the 36th Annual ACM Symposium on Applied Computing, SAC 2021, pp. 1504–1512. Association for Computing Machinery (2021)
32. Schneid, K., Stapper, L., Thöne, S., Kuchen, H.: Automated regression tests: a no-code approach for BPMN-based process-driven applications. In: 2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC), pp. 31–40 (2021)
33. Schneid, K., Usener, C.A., Thöne, S., Kuchen, H., Tophinke, C.: Static analysis of BPMN-based process-driven applications. In: Proceedings of the 34th

ACM/SIGAPP Symposium on Applied Computing, SAC 2019, pp. 66–74. Association for Computing Machinery (2019)
34. Stiehl, V.: Definition of process-driven applications. In: Stiehl, V. (ed.) Process-Driven Applications with BPMN, pp. 13–41. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07218-0_2
35. Sun, S.X., Zhao, J.L., Nunamaker, J.F., Sheng, O.R.L.: Formulating the data-flow perspective for business process management. Inf. Syst. Res. **17**(4), 374–391 (2006)