Detection of Process Antipatterns: A BPEL Perspective

Francis Palma*, Naouel Moha†, Yann-Gaël Guéhéneuc*
*Ptidej Team, DGIIL, École Polytechnique de Montréal, Canada
†Département d’informatique, Université du Québec à Montréal, Canada

Abstract—With the increasing significance of the service-oriented paradigm for implementing business solutions, assessing and analyzing such solutions also becomes an essential task to ensure and improve their quality of design. One way to develop such solutions, a.k.a., Service-Based systems (SBSs) is to generate BPEL (Business Process Execution Language) processes via orchestrating Web services. Development of large business processes (BPs) involves design decisions. Improper and wrong design decisions in software engineering are commonly known as antipatterns, i.e., poor solutions that might affect the quality of design. The detection of antipatterns is thus important to ensure and improve the quality of BPs. However, although BP antipatterns have been defined in the literature, no effort was given to detect such antipatterns within BPEL processes. With the aim of improving the design and quality of BPEL processes, we propose the first rule-based approach to specify and detect BP antipatterns. We specify 7 BP antipatterns from the literature and perform the detection for 4 of them in an initial experiment with 3 BPEL processes.

Keywords—Business processes; Antipatterns; Service-based systems; Specification; Detection; Design;

I. INTRODUCTION

Service Oriented Architecture (SOA) [1], as an architectural trend, is increasingly growing and widely adopted by practitioners because it allows low-cost and flexible development by composing services, i.e., software units that are autonomous, reusable, platform-independent, and are easily accessible over the Internet.

Any systems as well as Service-Based Systems (SBSs) may involve some antipatterns. SBSs are often evolved, i.e., are modified or added new functionalities. This evolution may hinder the design and quality of service (QoS) of SBSs, and thus may introduce ‘antipatterns’. An antipattern in a process generally captures common design errors [2], which causes a poor design resulting bad QoS.

SOA business solutions might also contain antipatterns while realizing the processes. The Business Process Modeling Notation (BPNN, [3]) is widely used for process modeling and provides the base for modeling control-flow, data-flow, and resource allocation. As for the Business Process Execution Language (BPEL, [4]), it provides an executable transformation of BPMN to the developers and is currently a de facto standard for Web services orchestration.

The automatic detection of business process (BP) antipatterns is an important task to assess the design and QoS of SBSs, i.e., BPEL processes. However, no efforts have been given to detect BP antipatterns in BPEL processes. Our goal is to assess the design and QoS of BPEL processes. To achieve this goal, we propose an approach to specify BP antipatterns and detect them automatically in BPEL processes.

In the past years, several catalogs of BP antipatterns [2], [5], [6], [7] and analysis techniques [8], [9], [10] to discover those antipatterns have been proposed. Indeed, most of these catalogs and techniques focus on BPMN models.

There are some conceptual differences between BPEL and BPMN. BPEL processes define all its semantic details whereas the BPMN models do not. Moreover, the transition from notational (i.e., graphical) to scripted (i.e., executional) is very often prone to be broken and certain semantic ambiguities. Very often this may cause the loss of various design decisions [11]. There are approaches focusing on detecting antipatterns in BPMN models at the early design phase. But there is no approach for the similar detection in BPEL processes which are at the later phase in the development cycle, and involves higher risk of introducing brokenly implemented design decisions due to semantic gaps between BPMN and BPEL. Therefore, in this paper we focus on BPEL processes rather than BPMN models due to several rationales:

- Firstly, antipatterns in BPMN artifacts already got much attention in the literature;
- Secondly, BPEL processes are more off-the-shelf executable entities with more detailed operational semantics, and thus may facilitate both the early design-time and run-time investigation of structural and behavioral properties of processes; and,
- Finally, BPEL is designed for the execution of the models. While the business analysts create the models, and the developers implement the technology, there may arise some translation, adaptation, and—or implementation fault-occurrences. Even, analysts may take some wrong design decisions that may eventually be transferred to the executable processes.

Designs defects, i.e., antipatterns must be detected and corrected to improve the design and QoS of SBSs. Therefore, with the goal of detecting BP antipatterns, we propose to:

(i) specify BP antipatterns using classical Rules of Inference
for their detection, (ii) define a concrete approach for the detection of BP antipatterns within BPEL processes, and (iii) perform an experiment with four BP antipatterns using our proposed approach on three example BPEL processes provided by Oracle, FraSCAti1, and OASIS [4].

This paper is structured as follows. Section II surveys related work on the catalog of BP antipatterns and their analysis and detection in BPMN models. Section III presents our proposed approach, while we show some detection results in Section IV. Finally, Section V concludes the paper and sketches future work.

II. RELATED WORK

Current literature is quite rich with a number of structural and behavioral antipatterns [2], [5], [6], [7] within models, along with some analysis and detection techniques [8], [9], [10], [12], in particular within BPMN models.

For example, Onoda et al. [5] first provided a catalog of five deadlock patterns using the concept of reachability and transferability based on the structure of a business process (BP) model. These antipatterns have been detected using the deadlock detection algorithm proposed by Maruta et al. [8]. Persson et al. [6] and Stirna et al. [7] provided a list of six patterns and 13 antipatterns related to enterprise modeling mainly focusing on the quality aspects of models. Based on their own practical experiences, the authors identified what mistakes the modelers need to avoid.

Gruhn and Laue [9] also proposed a heuristic-based approach for discovering problems in BP models and suggesting improvements. The authors first translated the models into a set of Prolog facts using simple XSLT transformations. Then, they defined some basic terminologies of BPMN modeling in the form of rules and identified some errors related to soundness and correctness (e.g., deadlock) of the models. Koehler and Vanhatalo [2] described 14 structural antipatterns in IBM WebSphere Business Modeler2 process models. However, if the antipatterns could be read visually then the localization and correction would become much easier for the modelers. Laue and Awad [12] were able to first visually represent the BP antipatterns. After the detection, the authors visually presented four antipatterns proposed by Onoda et al. [5] in BPMN models. Trčka et al. [10] formalised nine BP antipatterns using temporal logic that are caused by various data dependencies within the workflows and improper data handling.

From the above discussion, we can highlight the drawbacks of the current literature as follows: (i) antipatterns and approaches to detect them were considered only for BPMN models, whereas the de facto language BPEL was not considered at all; (ii) there are no other specifications for BP antipatterns except the one in [10]; (iii) BPMN models are not always executable. Therefore, various runtime quality aspects (e.g., availability or response time of Web services) were not considered, which can be obtained for the executable BPEL processes; finally, (iv) there is no detection approach for BP antipatterns in BPEL processes until now. We focus on these issues with a viable solution to propose a concrete approach for specifying and detecting antipatterns within executable BPs.

III. APPROACH

With the aim of detecting antipatterns in BPEL processes, we propose an approach as shown in Figure 1 involving three major steps:

**Step 1. Rule Specification:** This step concerns specifying rules for the detection of BP antipatterns that, later on, will be applied on BPEL processes.

**Step 2. Process Transformation to Analyse:** In this step, we transform BPEL processes into a more abstract and simplified representation, by filtering some process elements those are not required to apply a certain rule, to ease: (i) the implementation of the rules defined in the previous step and (ii) the further analysis of the processes.

**Step 3. Detection of BP Antipatterns:** The third step consists in applying the rules defined in Step 1 on the transformed processes from previous step. This step reports a list of existing antipatterns with the involved process fragments.

The following sections detail each of the previous steps.

A. Defining Rules for the Detection of BP Antipatterns

As the prerequisite to defining the rules, we carry out a thorough domain analysis of process antipatterns by studying their definitions in the literature, namely [2], [5], [6], [7]. This domain analysis allows us to identify clues, i.e., bad design criteria relevant to each process antipatterns. These identified clues have direct link to different design elements used for specifying BPs, i.e., gateways, decision points, and-or loops. Therefore, the identification of these relevant controllers within antipattern specifications is also an important

---

task while defining rules. To define the rules, we use the classical Rules of Inference that is simple to understand and implement for the developers. In this paper, we mostly focus on BP antipatterns defined in [2] because that paper explains the antipatterns that are mostly found in practice, and are collected from a wide range of process modeling tools including IBM WebSphere Business Modeler, Aris\(^3\), and Adonis\(^4\).

Figure 2 shows the rules for the two common BP antipatterns, i.e., Lack of Synchronization and Dangling Inputs and Outputs that we define. All process antipatterns in the literature are defined for BPMN models. However, we do a simple mapping between BPMN models and BPEL processes using a well-known and straightforward approach proposed by Weidlich et al. [13]. To define rules, we use different logical operators including OR and AND, and different relational operators, including PRECEDE, NOT_PRECEDE, BACKCONNECT, etc., that define relations between process fragments and nodes. For example, A PRECEDE B means B appears after A, or A BACKCONNECT B implies B has a backward connection to A in the process.

Lack of Synchronization [2] is an antipattern with the presence of fork-merge pair. The fork, i.e., parallel gateway triggers output on all of its outgoing branches, while the merge, i.e., inclusive gateway waits for input on only one of its incoming connections. Further later in the process, another final merge may cause synchronization problem because the latter merge requires all the input which may not available (see Figure 2(a)), thus a lack of synchronization occurs. Dangling Inputs and Outputs [2] is a form of antipattern where inputs and outputs of an activity or gateway remain unconnected in the process. Dangling data outputs are produced by a task or subprocess, but never used. In contrast, dangling inputs might cause deadlocks if the input is a data input of a gateway or an activity (see Figure 2(b)). We also specify five other BP antipatterns as shown in Figure 3, and graphically present them in Figure 4.

B. Transforming Business Processes

BPs are very complex entities and their complexity increases with their sizes. BPs define a collection of tasks, i.e., Web services and the communication details among them including data handling. For our analysis, we do not require all those details because in this paper we consider only the static analysis of BPEL processes. To filter optional details, while maintaining the process integrity, we generate a simplified model of the original BPEL. We make sure that we retain all the required information to apply our predefined rules. The transformation is done in the following way:

- From the original BPEL process to a simplified BPEL process: Figure 5 shows an example how we perform this transformation. We parse and filter all the required details, i.e., all the tasks, input data, output data, and control-flow information, etc., and generate a simplified process (see Figure 5(a)). Furthermore, we generate another process skeleton that can be mapped easily to the rules defined previously (see Figure 5(b)). Developers can use this latter version of the process for further investigation and analysis, i.e., implementing the executable versions of the rules.

C. Detection of BP Antipatterns

The detection phase follows the specification of the rules and the transformation of BPs. We implement the rules

---

\(^3\)http://www.ariscommunity.com/aris-express

\(^4\)http://www.adonis-community.com/
shown in Figure 3 programmatically using a language like JAVA. We implement the rules in a modularized way, i.e., we implement each side of different logical operators (e.g., AND, OR) in a rule, and join them afterwards to check the conformance with the defined conditions. Then we apply those code segments on transformed processes to detect BP antipatterns. This process is currently not automatic, however, one of our future goals is to automate this code generation phase.

### IV. Experiments and Results

We show with a small scale experiment the effectiveness of our proposed approach. We performed the experiment with three small BPEL processes: (1) TravelProcess, a reference example provided by Oracle, (2) sales-bpel, developed by FraSCAti and available in FraSCAti repository, and (3) auctionProcess, a reference example in BPEL 2.0 specification [4]. TravelProcess is a composite Web service containing three other Web services and seven I/O variables, whereas sales-bpel includes two other Web services and four I/O variables. Finally, auctionProcess involves three Web services and six I/O variables.

We analyse and specify seven process antipatterns from the literature and we perform detection for four BP antipatterns, namely Dangling Input and Output, Lack of Synchronization, Deadlock Through Decision-Join, and Stop-Node In Parallel Branches. Detection for other three BP antipatterns also currently is in progress.

#### A. Results

After the initial experiment, with our defined rules, we could not detect any antipatterns on TravelProcess and sales-bpel processes because they are small in size, and indeed, there were no such antipatterns. We also perform the detection for those antipatterns on auctionProcess without injecting or changing any variables or nodes. Indeed, we detect Lack of Synchronization in auctionProcess, i.e., it has two Forks and merge, and one Fork precedes merge, then this precedes another merge. However, we do not detect any such antipattern in sales-bpel as it does not possess any Forks, i.e., the parallel gateway (see Table II). We then ask a student who is not involved in the experiment and has knowledge on BPEL, to add randomly some I/O data, or to change the original control flow while maintaining the integrity of TravelProcess and sales-bpel processes. Obviously, we inform the student our goal, i.e., the antipatterns we intend to detect. The main goal of these changes is to inject antipatterns intentionally without biasing the results. The changes made by the student are summarized in Table I.

#### B. Discussion

With the modified BPs after injecting antipatterns, we again perform the detection and this time we detect the Dangling Input and Lack of Synchronization antipatterns in two processes as shown in Table II for TravelProcess and sales-bpel. Namely, in TravelProcess, we detect TempAmericanAir and TempDeltaAir as Dangling Input.
C. Threats to Validity

The main threat to the validity of our results involves external validity, i.e., the possibility to generalize our approach to other large BPs. Indeed, the availability of large and existing BPs is a real limit for this research. As the future work, we plan to run the experiment on other BPEL processes. Also, the subjective nature of defining rules is a threat to construct validity. We try to minimize this threat by defining rules based on a thorough literature review.

V. Conclusion and Future Work

Business processes, in particular, BPEL processes are the key standard to orchestrate Web services for building composite services. While designing, BPEL processes may possess antipatterns. Thus, the detection of process antipatterns is important to ensure and improve the quality of design of business processes (BPs). In this paper, we presented an approach, for the detection of BP antipatterns. We also defined rules for seven BP antipatterns from the literature to ease their detection. We applied our approach with four BP antipatterns on three example BPEL processes.

As the future work, we intend to fully automate the approach with more detected process antipatterns. Furthermore, we intend to perform experiments on other large and complex BPs. In this paper, we analyzed the BPs statically, i.e., without executing them. One of our future goals is to analyze the processes dynamically, i.e., executing them, to acquire run-time properties.

VI. Acknowledgment

This work is supported by the NSERC Discovery Grant.

REFERENCES