BPMN Analyzer 2.0: Instantaneous, Comprehensible, and Fixable Control Flow Analysis for Realistic BPMN Models

Tim Kräuter[1], Patrick Stünkel[1], Adrian Rutle[1], Yngve Lamo[1] and Harald König[2,1]

[1] *Western Norway University of Applied Sciences, Bergen, Norway*
[2] *FHDW Hannover, Germany*

Abstract

Many business process models contain control flow errors, such as deadlocks or livelocks, which hinder proper execution. In this paper, we introduce a new tool that can instantaneously identify control flow errors in BPMN models, make them understandable for modelers, and suggest corrections to resolve them. We demonstrate that detection is instantaneous by benchmarking our tool against synthetic BPMN models with increasing size and state space complexity, as well as realistic models. Moreover, the tool directly displays detected errors in the model, including an interactive visualization, and suggests fixes to resolve them. The tool is open source, extensible, and integrated into a popular BPMN modeling tool.

Keywords

BPM, Verification, Control flow analysis, BPMN model checking, Soundness, Safeness

1. Introduction

Business Process Modeling Notation (BPMN) is becoming increasingly popular for automating processes and orchestrating people and systems. However, many process models suffer from control flow errors, such as deadlocks, livelocks, and starvation [1]. These errors hinder the correct execution of BPMN models and may be detected late in the development process, resulting in elevated costs.

In this paper, we describe a new tool, the *BPMN Analyzer 2.0*[1], for analyzing BPMN process models to detect control flow errors *already* during modeling. Figure 1 shows an overview of the tool. The tool front-end is based on the popular *bpmn.io* ecosystem, while the analysis is implemented in Rust for performance and memory efficiency reasons. We implemented a breadth-first state space exploration. While generating the state space, we check BPMN soundness and safeness [3] *on the fly* to uncover control flow

Proceedings of the Best BPM Dissertation Award, Doctoral Consortium, and Demonstrations & Resources Forum co-located with 22nd International Conference on Business Process Management (BPM 2024), Krakow, Poland, September 1st to 6th, 2024.

tkra@hvl.no (T. Kräuter); past@hvl.no (P. Stünkel); aru@hvl.no (A. Rutle); yla@hvl.no (Y. Lamo); harald.koenig@fhdw.de (H. König)

[1] In the following, we will use BPMN Analyzer to refer to the BPMN Analyzer 2.0, not our previous work in [2].

errors. Consequently, the tool can detect deadlocks, livelocks, starvation, dead activities, and lack of synchronization in BPMN models. The BPMN Analyzer is open source and accessible online[2] alongside a video demonstration[3] [4].

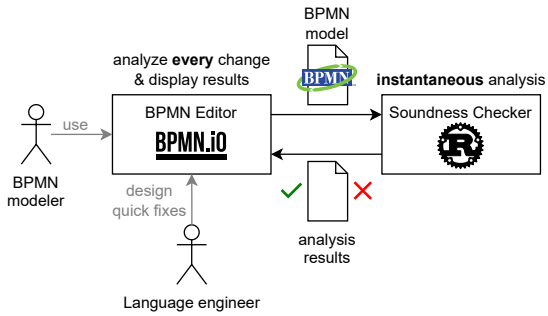

Figure 1: Overview of the BPMN Analyzer 2.0

The tool can check models after each change since analysis is *instantaneous* according to [1], i.e., it takes 500ms or less. Furthermore, we ensure the results are *comprehensible* by highlighting possible violations directly in the model and displaying an interactive counterexample visualization. Finally, the tool suggests *fixes* for the most common control flow errors and can be extended to suggest more fixes in the future.

Fahland et al. [1] describe *coverage*, *immediacy*, and *consumability* as the main challenges for users unaccustomed to formal analysis. The BPMN Analyzer addresses all these challenges since it supports the most common BPMN elements used in practice (coverage), provides *instantaneous* results (immediacy), and a *comprehensible* user interface (consumability), even including suggestions for fixes. Developers of industrial BPMN software also like our tool, especially the End-to-end user journey [4]. Thus, this supports our claim that the UI is easy for users unfamiliar with formal analysis to understand.

In the remainder of the paper, we describe how instantaneous, comprehensible, and fixable control flow error detection is achieved in section 2. Then, we discuss tool maturity in section 3 before concluding in section 4.

2. Innovations

The BPMN Analyzer has three main innovations: **instantaneous**, **comprehensible**, and **fixable** control flow error detection. In this section, we will present the innovations, and more details can be found in our extended paper [4].

2.1. Instantaneous Analysis

We demonstrate instantaneous control flow analysis by benchmarking our tool in *three* scenarios. For all our benchmarks, we use the hyperfine benchmarking tool (version

[2]https://timkraeuter.com/bpmn-analyzer-js/
[3]https://www.youtube.com/watch?v=Nv2W-hXNZYA

1.18.0), which calculates the average runtime when executing each control flow analysis ten or more times. We ran the benchmarks on Ubuntu 22.04.4 with an AMD Ryzen 7700X processor (4.5GHz) and 32 GB of RAM (5600 MHz). All used BPMN models, our tools to generate them, and benchmarking scripts to run them are available in [4].

First, we benchmarked how our tool handles **BPMN models of growing size**. We generated 500 synthetic BPMN models starting with five elements up to 4000. The models repeatedly contain three activities and an exclusive/parallel block with two branches containing one activity per branch (see [4]). The BPMN Analyzer spends from 1 ms up to 9 ms for the BPMN models [4] compared to 0.7 s up to 14 s in our previous tool [2]. In summary, the linear growth of the models leads to a linear growth in state space and, consequently, a linear growth in runtime.

Second, we benchmarked the tool against a synthetic data set of models that led to a state space explosion. This represents a *worst-case* scenario for formal analysis. We generated a data set of models [4] with a growing number of parallel branches with increasing length, as shown in Figure 2, like [5].

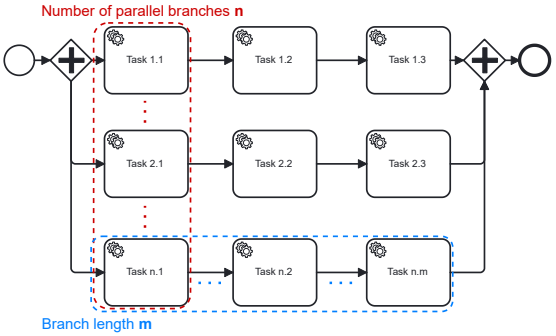

Figure 2: Models with a growing number of parallel branches and branch length

Table 1 shows the average runtime of our tool when analyzing these models. The BPMN Analyzer explores the entire state space while simultaneously analyzing the control flow, i.e., verifying soundness properties. The models' state space grows exponentially, leading to the same order of growth in runtime. Our analysis is not instantaneous anymore when approaching 17 parallel branches of length 1 (see Table 1). However, analysis is still instantaneous for more reasonable models with five parallel branches of length 5 or 3 branches of length 20. Other tools report 2-3s of runtime for most soundness properties and 30s for a model with five parallel branches [5], which took milliseconds in our tool.

Third, we applied our tool to eight **realistic models**, where three models (e001, e002, e020) are taken from [6], and the remaining five models are part of the Camunda BPMN for research repository[4]. Table 2 shows each model's average runtime and number of states. The BPMN Analyzer takes 1-10ms for e001, e002, and e020 [4], while [6] and [2] report 3.66-10.26s and 1-1.75s respectively. The benchmarks in [2] were run on the same hardware, while the machine used in [6] was slightly less powerful. To summarize, our

[4]https://github.com/camunda/bpmn-for-research

Table 1
Benchmark results of the parallel branches models

Branches	Branch Length	Runtime	States
5	1	1 ms	35
10	1	3 ms	1.027
15	1	161 ms	32.771
16	1	360 ms	65.539
17	1	790 ms	131.075
20	1	8.803 ms	1.048.579
5	5	14 ms	7.779
3	20	11 ms	9.264

Table 2
Benchmark results of the realistic BPMN models

Model name	Runtime	States
e001 [6]	1 ms	39
e002 [6]	1 ms	39
e020 [6]	10 ms	5356
credit-scoring-async[4]	1 ms	60
credit-scoring-sync[4]	1 ms	140
dispatch-of-goods[4]	1 ms	103
recourse[4]	1 ms	77
self-service-restaurant[4]	1 ms	190

analysis is instantaneous for nearly all models since most models have relatively small state spaces (less than 1000 states, see [1]).

2.2. Comprehensible Analysis

We implemented two features to make control flow analysis understandable for everyone. **First**, we highlight the problematic elements that cause control flow errors by directly attaching red overlays to them in the BPMN model. In addition, there is a summary panel in the top-right stating if any errors are found.

Second, we use *tokens* to visualize errors *interactively*, i.e., show an execution leading to the error. Our analysis provides sample executions leading to the possible control-flow errors, which we visualize directly in the BPMN editor by showing how tokens move from the process start to an erroneous state. We are unaware of other tools that visualize errors directly in the editor and allow for interactions, such as stopping/resuming, restarting, and speeding up the execution.

In Figure 3, the visualization has been *paused* just before an *unsafe* state was reached. One token is already located at the marked sequence flow, while a second token is currently waiting at the exclusive gateway `e1`. The visualization can be resumed or restarted using the play and restart button on the left side. The gateway `e1` will execute when resumed, resulting in two tokens at the subsequent sequence flow, i.e., an unsafe execution state. In addition, one can control the visualization speed using the bottom buttons next to the speedometer.

2.3. Fixable Analysis

In addition to detecting, highlighting, and visualizing control flow errors, the BPMN Analyzer suggests fixes like *quick fixes* in Integrated Development Environments. Quick fixes cannot be provided for all errors, but we currently cover many patterns leading to deadlocks, lack of synchronization, message starvation, and reused end events. The quick fixes we support are described in detail in [4] and can be extended independently of the

formal analysis. We are unaware of other tools that offer fixes for identified control-flow errors.

For example, Figure 3 shows a screenshot of our tool, where quick fixes are depicted as green overlays containing a light bulb icon. A user can apply a quick fix by clicking on a green overlay and instantly see the changes regarding control flow errors. If unhappy with the result, a user can undo all changes since each quick fix is entirely revertible due to the command pattern. A user might not like a quick fix if it not only fixes an error but also has unintended side effects, such as introducing a different control flow error.

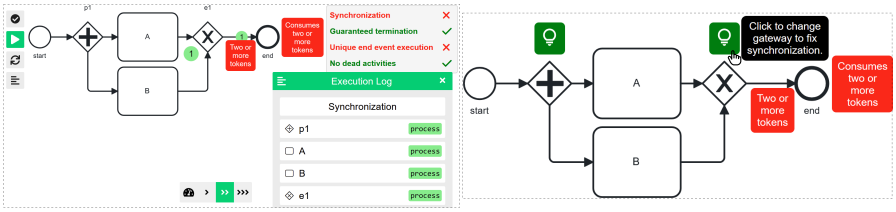

Figure 3: Execution visualization (left) and suggested *quick fixes* (right) in the BPMN Analyzer

3. Maturity of the Tool

The BPMN Analyzer is our newest tool, incorporating many findings from our previous work [2] while focusing on instantaneous and understandable error detectionn, as described in the previous section. The tool is open source [4], and we aimed to achieve high code quality by employing industry best practices such as rigorous static analysis, testing, and consistent formatting. Furthermore, we demonstrated the tool to companies in the BPMN process orchestration space and received positive feedback [4].

4. Conclusion & Future Work

In this paper, we describe the novel *BPMN Analyzer* that provides instantaneous, comprehensible, and fixable BPMN control flow error detection and is integrated into a popular BPMN modeling tool. We benchmarked our tool against synthetic and realistic BPMN models to demonstrate instantaneous soundness checking. We address the three main challenges, *coverage*, *immediacy*, and *consumability*, to provide formal analysis to non-expert users as identified in [1]. In addition, our tool offers quick fixes for common patterns that lead to control flow errors. One can understand the BPMN Analyzer as a BPMN-specific model checker, implemented in Rust paired with an intuitive user interface based on the popular *bpmn.io* ecosystem that is open for extension by design.

In future work, we aim to improve our tool by providing more quick fixes, considering advanced BPMN elements such as different events, and ranking quick fixes based on usefulness and previous user behavior. Finally, we aspire to test our tool in a real-world scenario to gather feedback and measure its impact on productivity.

References

[1] D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, K. Wolf, Analysis on demand: Instantaneous soundness checking of industrial business process models, Data & Knowledge Engineering 70 (2011) 448–466. doi:10.1016/j.datak.2011.01.004.

[2] T. Kräuter, A. Rutle, H. König, Y. Lamo, A higher-order transformation approach to the formalization and analysis of BPMN using graph transformation systems, 2024. arXiv:2311.05243.

[3] F. Corradini, C. Muzi, B. Re, F. Tiezzi, A Classification of BPMN Collaborations based on Safeness and Soundness Notions, Electronic Proceedings in Theoretical Computer Science 276 (2018) 37–52. doi:10.4204/EPTCS.276.5.

[4] T. Kräuter, P. Stünkel, A. Rutle, H. König, Y. Lamo, Instantaneous, Comprehensible, and Fixable Soundness Checking of Realistic BPMN Models, 2024. arXiv:2407.03965.

[5] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, A. Vandin, A formal approach for the analysis of BPMN collaboration models, Journal of Systems and Software 180 (2021) 111007. doi:10.1016/j.jss.2021.111007.

[6] S. Houhou, S. Baarir, P. Poizat, P. Quéinnec, L. Kahloul, A First-Order Logic verification framework for communication-parametric and time-aware BPMN collaborations, Information Systems 104 (2022) 101765. doi:10.1016/j.is.2021.101765.

[7] F. Corradini, C. Muzi, B. Re, L. Rossi, F. Tiezzi, Formalising and animating multiple instances in BPMN collaborations, Information Systems 103 (2022) 101459. doi:10.1016/j.is.2019.101459.