Formalization and analysis of BPMN using graph transformation systems

Tim Kräuter[*], Harald König[†*], Adrian Rutle[*], Yngve Lamo[*]
[*]Western Norway University of Applied Sciences, Bergen, Norway
[†]University of Applied Sciences, FHDW, Hannover, Germany
tkra@hvl.no, harald.koenig@fhdw.de, aru@hvl.no, yla@hvl.no

The BPMN is a widely used standard notation for defining intra- and inter-organizational workflows. However, the informal description of the BPMN execution semantics leads to different interpretations of BPMN constructs and difficulties in checking behavioral properties. Other approaches to formalizing BPMN's execution semantics only partially cover BPMN. To this end, we propose a formalization that, compared to other approaches, covers most of the BPMN constructs. Our approach is based on a model transformation from BPMN models to graph transformation systems. As a proof of concept, we have implemented our approach in an open-source web-based tool.

1 Introduction

Business Process Modeling Notation (BPMN) is a widely used standard notation to define intra- and inter-organizational workflows. However, the informal description of the BPMN execution semantics leads to different interpretations of BPMN constructs and difficulties in checking behavioral properties [1, 6]. To this end, we propose a formalization that, compared to other approaches, covers most of the BPMN constructs.

Our approach is depicted as a BPMN process model in figure 1. It is based on a model transformation from BPMN process models to graph transformation systems. Thus, our approach constructs a new graph transformation system, i.e., graph transformation rules and a start graph for each BPMN process model. This is a significant difference compared to other approaches such as [1, 10], where only the BPMN process model is parsed, but the rewrite rules are fixed. Generating specific rules for each model leads to possibly more but simpler transformation rules that can be matched faster. Essentially, complexity is partly shifted from the transformation rules to their generation. The generated rules are tailored to a given process model and thus simpler than the general rules in [10].

The remainder of this paper is structured as follows. First, we describe the semantics formalization using graph transformation systems (section 3) before explaining how this can be utilized for model checking BPMN-specific and custom properties (section 4). Then, we shortly present the web-based tool implementing our approach. Finally, we discuss related work regarding BPMN construct coverage in section 6 and conclude in section 7.

2 Preliminaries

In this paper, we apply graph rewriting theory to formalize the execution semantics of BPMN. Thus, in this section, we will briefly introduce BPMN and its execution semantics. Please refer to [4] or the BPMN specification [6] for further information about BPMN. Furthermore, we describe the theoretical background behind our application of graph rewriting.

Submitted to:
TERMGRAPH 2022

© Kräuter *et al.*
This work is licensed under the
Creative Commons Attribution License.

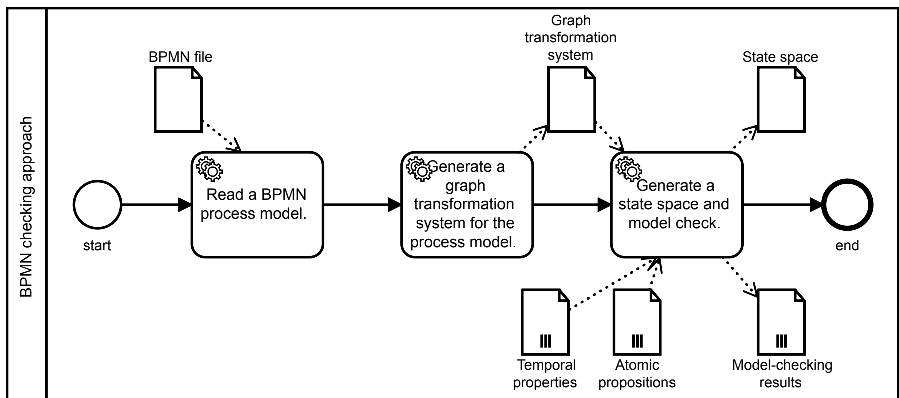

Figure 1: Overview of the proposed approach

2.1 BPMN

BPMN is a widely used standard notation to define intra- and inter-organizational workflows. Figure 2 depicts the structure of BPMN process models and the corresponding BPMN symbols contained in clouds.

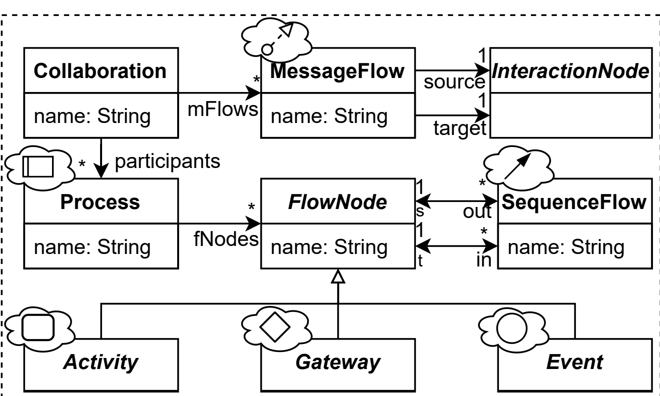

Figure 2: Simplified excerpt of the BPMN metamodel [6]

A BPMN process model is represented by a Collaboration that has a set of participants and MessageFlows between InteractionNodes contained by them (see figure 2). Each participant is a Process containing FlowNodes connected by SequenceFlows. A FlowNode is either an Activity, Gateway or Event. There exist many types of Activities, Gateways and Events such as start and end events, for example, used in figure 1. Activities represent certain tasks to be carried out during a process, while things may happen (events). Furthermore, gateways represent conditions or split and merge sequence flows [4].

The BPMN execution semantics are described using the concept of *tokens* [6]. BPMN process models are executed by triggering one or more of their start events, leading to the creation of a token at each triggered start event. Tokens are transported between flow nodes by sequence flows (arrows) [1]. Activities can start when there is at least one token located on an incoming sequence flow. The start of an activity

[1]The visual bpmm-js token simulation available at `https://bpmn-io.github.io/bpmn-js-token-simulation/modeler.html` greatly helps to understand BPMN execution semantics.

will move the incoming token to the activity. When an activity finishes it will create one token for each outgoing sequence flow. Different gateway types exist, for example, for split/merge or XOR/OR distribution of tokens. Events consume and create tokens similarly to activities but also have additional semantics depending on their type. For example, message events will create or consume messages.

2.2 Theoretical background

We use typed attributed graphs for the formalization of the BPMN execution semantics. Each state, i.e., token distribution during the execution of a BPMN model is represented as an attributed graph typed in the BPMN execution type graph introduced later in section 3.

Regarding graph transformation, we utilize the single-pushout (SPO) approach with negative application conditions (NAC) [3]. In addition, we utilize *nested rules* with quantification to make parts of a rule applied repeatedly or optionally [7, 8]. SPO is sufficient to formalize the BPMN execution semantics and the automatic removal of dangling edges is not a problem. Moreover, we utilize NACs to implement some more intricate parts in the BPMN execution semantics such as the termination of processes. In SPO rewriting, a graph transformation rule is defined as a partial graph morphism $L \to R$, while in our case L and R are typed attributed graphs.

3 BPMN semantics formalization

Since our approach is based on a model transformation from BPMN to graph transformation systems, we generate a *start graph* and *graph transformation rules* for a given BPMN process model. The approach supports the BPMN constructs depicted in figure 3.

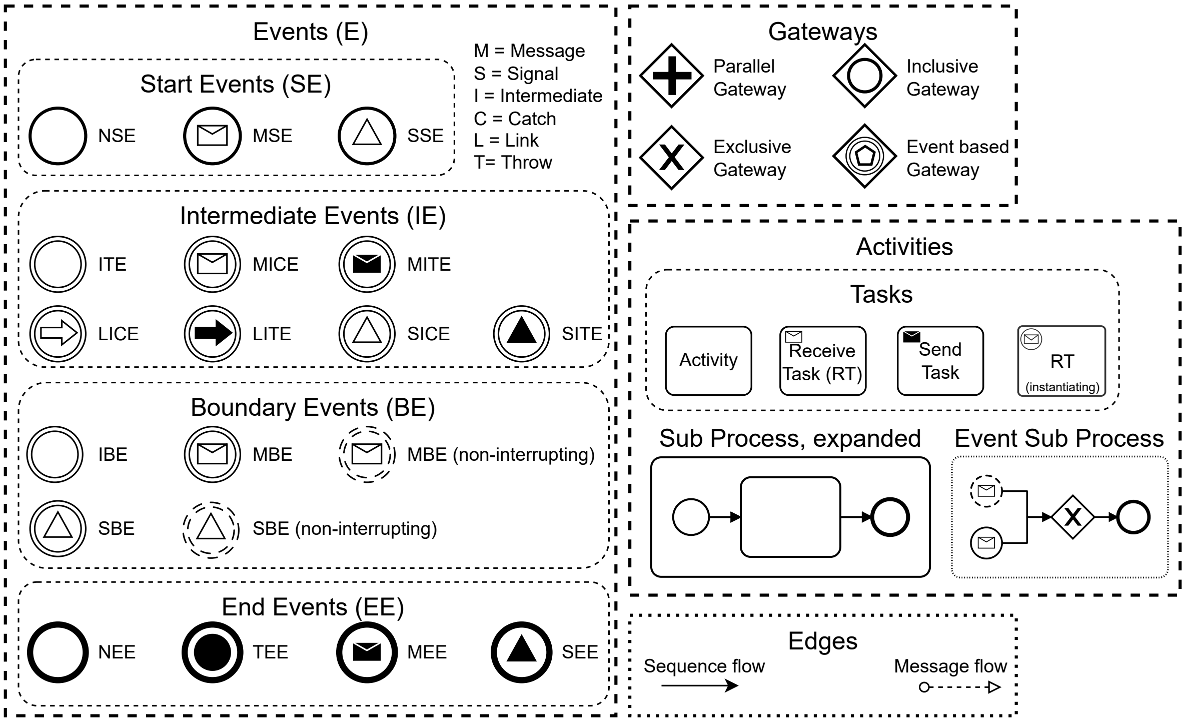

Figure 3: Overview of the supported BPMN constructs (structure adapted from [5])

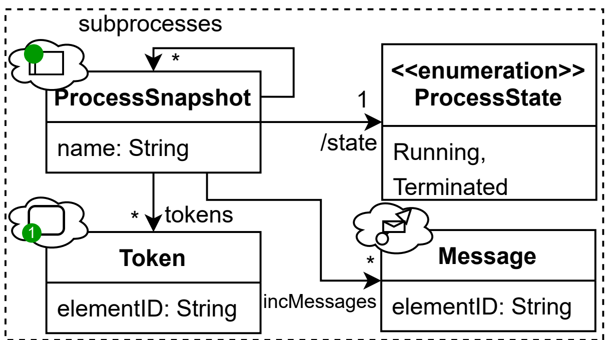

Figure 4: BPMN execution type graph

Our formalization of BPMN is token-based, as in the informal description of the BPMN specification [6]. Thus, to describe processes holding tokens during execution, we use the type graph shown in figure 4. The type graph is depicted using a UML class diagram-like syntax.

We use ProcessSnapshot to denote a running BPMN process with a specific token distribution which describes one state in the history of process states during the execution. Every ProcessSnapshot has a set of tokens, incoming messages, and subprocesses. A ProcessSnapshot has the state Terminated if it has no tokens or subprocesses. Otherwise, it has the state Running. A Token has an elementID, the BPMN activity or sequence flow at which it is located. A Message has an elementID, pointing to a MessageFlow. To depict graphs conforming to the type graph concisely, we introduce a concrete syntax in the clouds attached to the elements. It extends the BPMN syntax by adding tokens. Tokens are represented as colored circles and are drawn at their specified position in a model. Their color will match the color of the circle representing the process snapshot holding the token, which is located at the top left of the corresponding BPMN process. The concrete syntax was significantly inspired by the excellent bpmn-js-token-simulation[2]. Using this type graph, we can now define how the start graph and graph-transformation rules for the different BPMN constructs are created.

The generation of the start graph for a BPMN model is straightforward. For each process in the BPMN model, we generate a process snapshot if the process contains a none start event (NSE). Then, for each NSE, we add a token to the respective process snapshot. An example of a start graph is shown in figure 5 using abstract and concrete syntax. Furthermore, we consider allowing the user to define a start graph similar to how he can define atomic propositions for custom properties (see section 4.2).

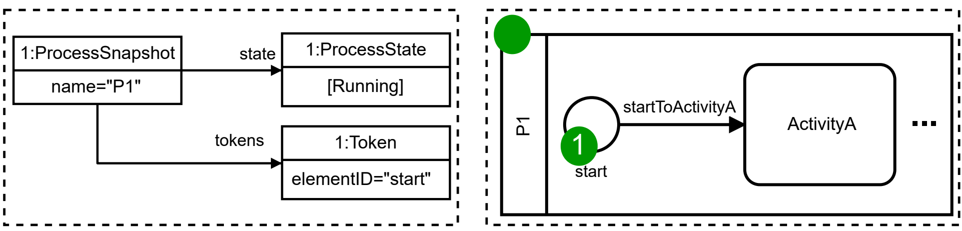

Figure 5: Example start graph in abstract (left) and concrete syntax (right)

The model transformation generates one or more graph transformation rules for each FlowNode in a BPMN model. We will now describe the rule generation for NSE's, tasks, and message events to give an

[2]https://github.com/bpmn-io/bpmn-js-token-simulation

overview of how our model transformation works. A table summarizing how the model transformation works for the main FlowNodes is contained in the artifacts of this paper [9]. Figure 6 depicts an example graph transformation rule ($L \rightarrow R$) for a NSE in abstract syntax. The rule is straightforward and moves a token from the start event to its outgoing sequence flow. For the rest of the paper, we will depict all rules in the concrete syntax introduced earlier. The rule from figure 6 depicted in concrete syntax is shown in figure 7.

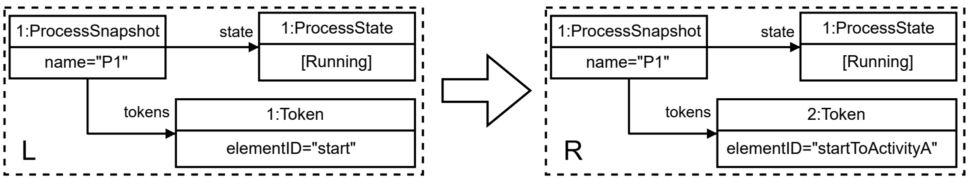

Figure 6: Example graph transformation rule for a NSE (abstract syntax)

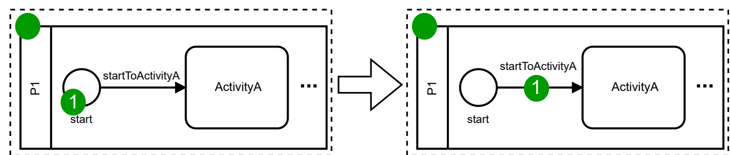

Figure 7: Example graph transformation rule for a NSE (concrete syntax)

The rule in figure 8 represents the start of the task, which will move one token from the incoming sequence flow to the task itself.

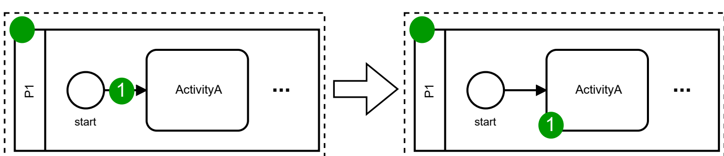

Figure 8: Example graph transformation rule to start a task

The left rule in figure 9 realizes a message throw event, and the right rule implements a message catch event. The message catch event rule consumes a token and a message and creates an outgoing token. The message throw event rule moves the token through the event and sends a message to a waiting process snapshot, which must have a token waiting at the corresponding message receive event. However, sending this message is optional, which is implemented using a nested rule with quantification. Concretely, we use an optional existential quantifier to send a message only if the receiving process is ready to receive it [7].

End events consume but do not produce tokens. Thus, process termination can be implemented using a general rule applicable to all process snapshots. The rule is automatically generated once during the model transformation to graph transformation systems and is used to terminate processes, sub processes, and event sub processes. It uses negative-application conditions to forbid tokens and sub processes for a process snapshot and then changes its state from running to terminated[3].

[3]The terminate rule implemented in Groove is contained in the artifacts of this paper [9].

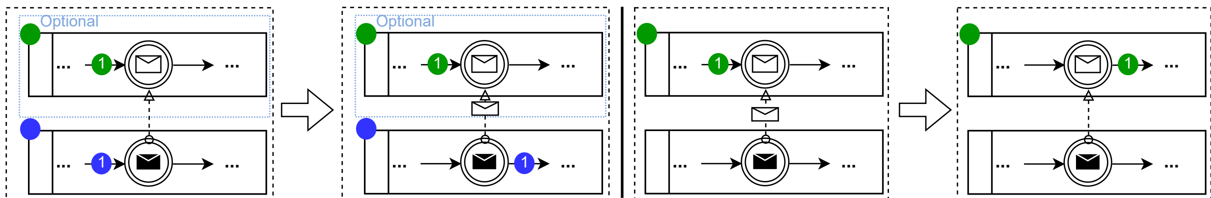

Figure 9: Rules for message intermediate throw events (left) and catch events (right)

4 Model checking BPMN

Model checking a BPMN process model is possible using the generated graph transformation system. Besides a graph transformation system, a set of temporal properties to be checked and the atomic propositions used in the properties must be supplied (see figure 1). An atomic proposition can be formalized as a graph and holds in a given state if it is a subgraph of the graph representing the state. This enables model checking of temporal properties, for example, LTL properties, using the defined atomic propositions.

Like other work, we differentiate between *BPMN-specific properties* defined generally for all BPMN process models and *custom properties* tailored towards a particular BPMN process model. We do not consider structural properties since they can be checked using a standard process modeling tool without implementing execution semantics. We will now give an example of two predefined BPMN-specific properties and how they can be checked using our approach. Then, we describe how custom properties can be constructed and checked.

4.1 BPMN-specific properties

Two BPMN-specific behavioral properties, namely, Safeness and Soundness, are defined in [2]. A BPMN process model is *safe* if, during its execution, at most one token occurs along the same sequence flow [2]. Soundness is further decomposed into (i) *Option to complete*: any running process instance must eventually complete, (ii) *Proper completion*: at the moment of completion, each token of the process instance must be in a different end event, (iii) *No dead activities*: any activity can be executed in at least one process instance [2]. As an example, we will now describe how to implement the *Safeness* and *Option to complete* properties.

Safeness is checked using the LTL property defined in (1). The atomic property Unsafe is true if two tokens of one process snapshot have the same position[4]. *Option to complete* is checked using the LTL property defined in (2). The atomic property AllTerminated is true if there exists no process snapshot in the state Running, i.e., all process snapshots are Terminated[4].

$$\Box(\neg \text{Unsafe}) \quad (1) \qquad\qquad \Diamond(\Box(\text{AllTerminated})) \quad (2)$$

Both properties can be checked using our implementation [9]. To fully check Soundness, we need to check *Proper Completion* and *No Dead Activities*. The information needed to check these properties is present in the generated state space.

4.2 Custom properties

To make model checking user-friendly, we envision users defining atomic propositions in the extended BPMN syntax, i.e., the concrete syntax introduced earlier. Thus, to define an atomic proposition, we

[4]Groove rules for the atomic properties Unsafe and AllTerminated are contained in the artifacts of this paper [9].

let the user attach tokens to a BPMN process model, which we can automatically convert to a graph representing an atomic proposition.

For example, the token distribution shown in figure 10 defines two running process snapshots with a token in task A. Differently colored tokens define different process snapshots. A user could use this property, for example, to check if, eventually, two processes are executing task A simultaneously. Thus, a user need not be aware of the graph transformation semantics used for execution, which is a significant advantage compared to other approaches.

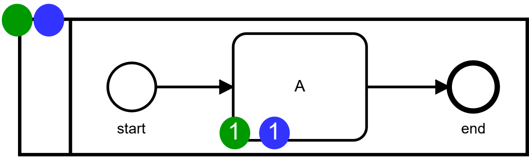

Figure 10: Token distribution defining an atomic proposition.

5 Implementation

Figure 11 depicts a screenshot of the implemented tool. The tool is open-source, publicly available, and does not require any installation [9].

Figure 11: Screenshot of the tool

The first two steps of our approach, i.e., reading and transforming BPMN models to graph transformation systems, are implemented and usable through the web-based tool. Then one can use the graph-transformation tool Groove[5] for state space generation and model checking locally. We are currently

[5] `https://groove.ewi.utwente.nl/about`

working on extending our tool such that model-checking can be done without installing Groove locally. Groove implements SPO with NACs and thus has all the features we need. To evaluate the correctness of our implementation, we created a comprehensive test suite, which demonstrates correct rule generation for the implemented BPMN constructs [9].

6 Related work

A BPMN formalization based on in-place graph transformation rules is given in [10]. The formalization covers a substantial part of the BPMN specification, including complex concepts such as inclusive gateway merge and compensation. In addition, graph transformation rules are visual and thus can easily be matched to the informal description of the execution semantics in the specification [6]. The graph transformation rules were implemented in a prototype using GrGen.NET. Unfortunately, the implementation is not publicly accessible anymore. Moreover, they do not support model checking since their goal is only formalization.

The tool BProVe[6] is based on formal BPMN semantics given in rewriting logic and implemented in the Maude system. Using these semantics, BProVe enables the verification of custom LTL properties and BPMN-specific properties, such as Safeness and Soundness. Furthermore, the tool is accessible online, not requiring any installation [1].

The verification framework fbpmn uses first-order logic to formalize and check BPMN process models [5]. This formalization is then realized in the TLA$^+$ formal language and can be model-checked using TLC. Their framework and related information is open source and freely available online[7]. Similar to BProVe, fbpmn allows checking BPMN-specific properties, such as Safeness and Soundness. However, they do not allow a user to define custom temporal properties.

We looked in detail at these three approaches since they support a significant subset of the BPMN constructs and have accessible and well-documented tools. Nevertheless, each approach supports a different subset of the BPMN constructs. The coverage of BPMN constructs greatly impacts how useful each approach is in practice. Table 1 depicts which BPMN constructs are supported by the different approaches compared to our approach.

Van Gorp et al. [10] cover a large part of the BPMN semantics. However, they do not support Event-based gateways and event subprocesses, while their support for boundary events is minimal. Especially, Event-based gateways are often used in practice. Corradini et al. [1] support message and terminate events. In addition to [1], Houhou et al. [5] support timer and the use of message and timer events as both interrupting and non-interrupting boundary events. However, many other event types exist and are used in practice.

Referring to table 1, we conclude that our formalization is comprehensive but still lacks support for some of the more advanced event types. An implementation of the missing event types is feasible, as shown in [10].

7 Conclusion & future work

The approach presented in this paper utilizes a model transformation from BPMN models to graph transformation systems. For each BPMN model, we automatically generate a start graph and a set of graph

[6] http://pros.unicam.it/bprove/

[7] https://github.com/pascalpoizat/fbpmn

Table 1: Constructs supported by different BPMN formalizations (overview based on [10]).

Feature	Van Gorp et al. [10]	Corradini et al. [1]	Houhou et al. [5]	This paper
Instantiation and termination				
Start event instantiation	X	X	X	X
Exclusive event-based gateway instantiation	X			X
Parallel event-based gateway instantiation				
Receive task instantiation				X
Normal process completion	X	X	X	X
Activities				
Activity	X	X	X	X
Subprocess	X	X	X	X
Ad-hoc subprocesses				
Loop activity	X			
Multiple instance activity				
Gateways				
Parallel gateway	X	X	X	X
Exclusive gateway	X	X	X	X
Inclusive gateway (split)	X	X	X	X
Inclusive gateway (merge)	X		X	X
Event-based gateway		X[1]	X	X
Complex gateway				
Events				
None Events	X	X	X	X
Message events	X	X	X	X
Timer Events			X	
Escalation Events				
Error Events (catch)	X			
Error Events (throw)	X			
Cancel Events	X			
Compensation Events	X			
Conditional Events				
Link Events	X			X
Signal Events	X			X
Multiple Events				
Terminate Events	X	X	X	X
Boundary Events	X[2]		X[3]	X
Event subprocess				X

[1] Does not support receive tasks after event-based gateways.
[2] Only supports interrupting boundary events on tasks.
[3] Only supports message and timer events.

transformation rules. In this way, the graph transformation rules get simpler while the complexity is shifted to the model transformation from BPMN to graph transformation systems. Our resulting BPMN formalization is comprehensive and supports model checking. In addition, we provide a prototype implementation in a web-based tool to make our ideas easily accessible to other researchers and potentially practitioners in the future.

We aim to improve our formalization and resulting tool in multiple ways in the future. First, we intend to extend our formalization to support even more BPMN constructs, for example, error, cancel, and compensation events. Second, we plan to evaluate our approach on models from open repositories such as the "BPM Academic Initiative Model Collection" [11] and "Camunda BPMN for Research"[8]. Third, we want to extend the features of our tool; one should be able to define atomic propositions for model checking in the tool directly, as described in section 4. Furthermore, counterexamples found during model checking should be visualized directly in the tool, like the implementation in [5], such that users need not understand the underlying implementation in Groove.

References

[1] Flavio Corradini, Fabrizio Fornari, Andrea Polini, Barbara Re, Francesco Tiezzi & Andrea Vandin (2021): *A Formal Approach for the Analysis of BPMN Collaboration Models*. Journal of Systems and Software 180, p. 111007, doi:10.1016/j.jss.2021.111007.

[2] Flavio Corradini, Chiara Muzi, Barbara Re & Francesco Tiezzi (2018): *A Classification of BPMN Collaborations Based on Safeness and Soundness Notions*. Electronic Proceedings in Theoretical Computer Science 276, pp. 37–52, doi:10.4204/EPTCS.276.5.

[3] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner & A. Corradini (1997): *ALGEBRAIC APPROACHES TO GRAPH TRANSFORMATION – PART II: SINGLE PUSHOUT APPROACH AND COMPARISON WITH DOUBLE PUSHOUT APPROACH*, pp. 247–312. WORLD SCIENTIFIC, doi:10.1142/9789812384720_0004.

[4] Jakob Freund & Bernd Rücker (2019): *Real-Life BPMN: Using BPMN and DMN to Analyze, Improve, and Automate Processes in Your Company*, 4th edition edition. Camunda, Berlin.

[5] Sara Houhou, Souheib Baarir, Pascal Poizat, Philippe Quéinnec & Laïd Kahloul (2022): *A First-Order Logic Verification Framework for Communication-Parametric and Time-Aware BPMN Collaborations*. Information Systems 104, p. 101765, doi:10.1016/j.is.2021.101765.

[6] Object Management Group (2013): *Business Process Model and Notation (BPMN), Version 2.0.2*. https://www.omg.org/spec/BPMN/.

[7] Arend Rensink (2006): *Nested Quantification in Graph Transformation Rules*. In Andrea Corradini, Hartmut Ehrig, Ugo Montanari, Leila Ribeiro & Grzegorz Rozenberg, editors: Graph Transformations, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 1–13, doi:10.1007/11841883_1.

[8] Arend Rensink (2017): *How Much Are Your Geraniums? Taking Graph Conditions Beyond First Order*. In Joost-Pieter Katoen, Rom Langerak & Arend Rensink, editors: ModelEd, TestEd, TrustEd, 10500, Springer International Publishing, Cham, pp. 191–213, doi:10.1007/978-3-319-68270-9_10.

[9] Tim Kräuter: *Artifacts - TERMGRAPH-2022*. https://github.com/timKraeuter/TERMGRAPH-2022.

[10] Pieter Van Gorp & Remco Dijkman (2013): *A Visual Token-Based Formalization of BPMN 2.0 Based on in-Place Transformations*. Information and Software Technology 55(2), pp. 365–394, doi:10.1016/j.infsof.2012.08.014.

[11] Mathias Weske, Gero Decker, Marlon Dumas, Marcello La Rosa, Jan Mendling & Hajo A. Reijers (2020): *Model Collection of the Business Process Management Academic Initiative*, doi:10.5281/zenodo.3758705.

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