BPMN2Constraints: Breaking Down BPMN Diagrams into Declarative Process Query Constraints

Arvid Bergman[1,2], Adrian Rebmann[1,3] and Timotheus Kampik[1,2,*]

[1]SAP Signavio, Berlin, Germany
[2]Umeå University, Umeå, Sweden
[3]University of Mannheim, Mannheim, Germany

Abstract
This paper presents BPMN2Constraints, a tool that compiles BPMN diagrams into sets of declarative constraints that can then, for example, be used for conformance checking. Notably, BPMN2Constraints does not rely on Petri net replay for generating the constraints; by generating constraints directly from a control flow graph extracted from the BPMN model, the tool avoids indirection. BPMN2Constraints can generate constraints in several languages: DECLARE, finite-trace linear temporal logic, and SIGNAL, a proprietary process querying language.

Keywords
Business process management, Declarative constraints, Conformance checking, Process querying

1. Introduction

As process mining is increasingly adopted in the context of enterprise information systems, facilitating the generation of meaningful queries on process data in a business user-friendly manner becomes an important practice challenge. One way to facilitate querying is the extraction of conformance constraints from business process models such as Business Process Model and Notation (BPMN) diagrams, which often exist in organizational contexts in which process mining can be applied, for example in the form of reference models. However, these models have typically not been created for the explicit purpose of mining and their complete, *imperative* specification may be too rigid for meaningful process querying, e.g., in the case of conformance checking. Traditional imperative process models typically characterize only the most common process execution variants. This means that traces that are – from a business domain perspective – unproblematic, but diverge from a model's rigid control flow, may be incorrectly assessed as non-compliant. Hence, in many scenarios, it makes more sense to view models as sets of declarative constraints, only some of which should be activated. Here, traditional approaches to checking process conformance against BPMN models by characterizing the models as Petri nets and re-playing the traces that have been extracted from an enterprise system [1] fall short.

Proceedings of the Demonstration & Resources Track, Best BPM Dissertation Award, and Doctoral Consortium at BPM 2023 co-located with the 21st International Conference on Business Process Management, BPM 2023, Utrecht, The Netherlands, September 11-15, 2023
*Corresponding author.
arvid.bergman@sap.com (A. Bergman); adrian.rebmann@sap.com (A. Rebmann); timotheus.kampik@sap.com (T. Kampik)

Instead, it is preferable to extract declarative constraints from the models. These constraints can then be filtered to exclude the ones that do not carry any relevant business meaning. Also, constraints of several models can be aggregated, e.g., to identify the most prevalent constraints across several business process landscapes or collections of best practice models. An additional advantage of compiling declarative constraint sets without an additional Petri net replay step, but directly from imperative process models, is that at least some industry-scale process querying languages use regular expression-based matching as the central abstraction for reasoning about control flow, which is closer to declarative temporal reasoning. Considering the prevalence of declarative approaches in large-scale industry applications outside of the process domain, e.g., in distributed systems [2], this allows for the application of well-established reasoning approaches that engineers are familiar with.

To achieve our goal of moving from business-level imperative process models directly to declarative conformance checking constraints, this demonstration paper presents BPMN2Constraints, a software library that compiles the control flow of BPMN models to constraints in several declarative languages: finite-trace linear temporal logic (LTL_f), DECLARE [3] (which provides somewhat user-friendly abstractions on top of LTL_f), and SIGNAL [4], which is a proprietary process querying language used in industry. The tool is available as a Python module. The source code alongside usage instructions and a (video) tutorial are available at https://github.com/signavio/bpmn2constraints. To highlight BPMN2Constraints's robustness (as well as robustness limitations), the tool is applied to thousands of openly available process models, as well as to a collection of thousands of proprietary best practice process models.

2. Mapping BPMN to Declarative Process Query Constraints

In order to illustrate how BPMN2Constraints compiles BPMN-like process models to declarative constraints, let us introduce a simple (*happy path*) example BPMN model (Figure 1). The process starts when a credit is requested (cr). Subsequently, the request is reviewed (rr). If standard terms are applicable to the request, the terms are calculated (ct); else, special terms are prepared (pst) and then, the contract is prepared (pc). In parallel to either ct or pst and pst, the risk of fulfilling the request is assessed (ar). Finally the quote is sent (sq), triggering the end event qs.

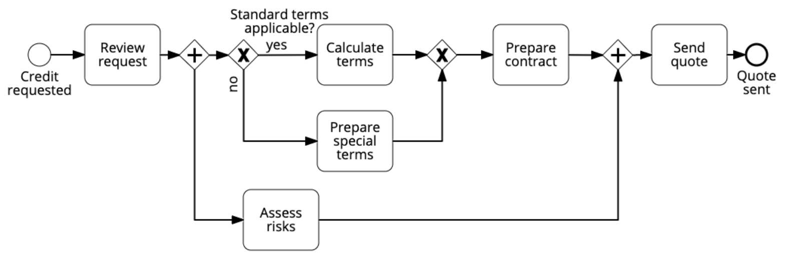


Figure 1: A toy example process: credit quote creation.

Given the BPMN model, we can observe the following constraints, for example: i) rr precedes ar (DECLARE: $Succession(rr, ar)$); ii) Assuming transitivity, rr precedes sq as well ($Succession(rr, sq)$); iii) Either ct and pst occur, but not both ($ExclusiveChoice(ct, pst)$);

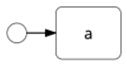
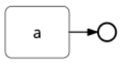
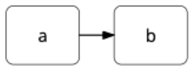
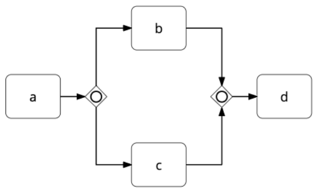
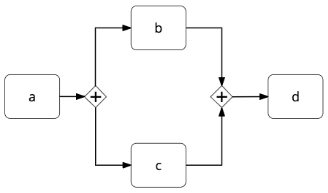
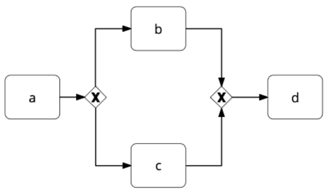
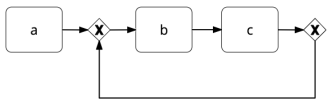
BPMN	Description	DECLARE	SIGNAL
	Starts with a	$Init(a)$	`^a`
	Ends with a	$End(a)$	`a$`
	a leads to b	$Succession(a, b)$	`(^NOT(a|b)*(a~>b) *NOT(a|b)*$)`
	b OR c	$Choice(b, c)$	`(b | c )`
	b AND c	$CoExistence(b, c)$	`(^NOT('b'|'c') (('b'ANY'c'ANY*)| ('c'ANY*'b'ANY*)) NOT('b'|'c')$)`
	b XOR c	$ExclusiveChoice(b, c)$	`(^(((NOT('c')*) ('b'NOT('c')*)*) | ((NOT('b')*) ('c'NOT('b')*)*))$)`
	Loop	$AlternateSuccession(b, c)$	`(^NOT('b'|'c')* ('b'NOT('b'|'c')* 'c'NOT('b'|'c')*)* NOT('b'|'c')* $)`

Table 1
Basic BPMN constructs & exemplary corresponding DECLARE and SIGNAL statements.

iv) ar and pc occur together, but are not constrained regarding their temporal order ($CoExistence(ar, pc)$).

To generate the constraints, we parse the BPMN diagram, extract activity relations based on the control flow and instantiate the constraints based on templates that follow DECLARE semantics. Note that we exclude negative constraint templates as well as existence templates to improve readability of the output. Implicitly, there is an $Absence(a, 1))$ constraint for each $a \notin A$, where A is the set of the task labels that occur in the model at hand, which means that we exclude any tasks that are not part of the model[1]. We treat events and tasks/collapsed sub-processes equally as propositional variables and handle sequence flows, OR, XOR, AND, and event-based gateways for operator generation (interpreting event-based gateways as XORs). All other element types are ignored.

[1]The full list of DECLARE templates we use as a basis can be found in our repository.

Table 1 provides an overview of some common "patterns" that occur in BPMN diagrams and how the compiler maps them to DECLARE statements and SIGNAL queries. It is important to note that this is not a one-to-one mapping from these BPMN constructs to constraints, because when constructs are nested within other constructs, the behavior allowed by the overall model naturally changes, which in turn needs to be reflected by the declarative constraints.

3. Evaluation on Large Process Model Collections

In order to assess the robustness of BPMN2Constraints and demonstrate its maturity, we compiled thousands of BPMN models from two large process model collections into DECLARE and SIGNAL constraints and analyzed the results with respect to the following properties.
Ability to parse and compile. We check whether the parsing/compilation procedure fails and whether all control flow-relevant elements (events, activities, sequence flows) can be parsed and appear in the output.
Output similarity relative to a Petri net replay-based approach. We compare the DECLARE constraints BPMN2Constraints generates with the output of a Petri net replay-based constraints, measuring: i) precision: how many of the constraints generated by BPMN2Constraints are also generated by the Petri net replay-based tool? ii) recall: how many of the constraints generated by a Petri net replay-based tool are also generated by BPMN2Constraints? Here, it is crucial to note that the constraints generated by the Petri net replay-based tool are not a proper ground truth. In particular, the imperfect definition of BPMN execution semantics does not allow for an unambiguous transformation into Petri nets; the complexity of the standard causes challenges even for mundane tasks such as model interoperability, which is continuously improved by a dedicated working group[2]. Still, the results can shed light on the extent to which the two approaches are aligned, thus indicating potential directions for future work.

For our evaluation, we used the following two process model collections.
A collection of research and educational BPMN models (SAP-SAM). The SAP Signavio Academic Models (SAP-SAM) [5] contains around 550,000[3] models that have been created by researchers, students, and teachers for research and educational purposes. The dataset is very large and diverse; however, most of the models it contains have not been created for real-world process management purposes, i.e. it cannot be considered as representative of practice-oriented process model collections. From the dataset we sampled 10,000 BPMN models.
A vendor-provided proprietary collection of reference BPMN models (VBPMN). To mitigate the limitations of SAP-SAM, we extended the evaluation of BPMN2Constraints to the SAP Signavio Process Explorer reference BPMN models[4], a collection of around 7,000 BPMN models that provide vendor-specific best practices.

From both collections, we excluded models with any of the following properties (heuristics to select diagrams that are either of too low quality or out-of-scope for BPMN2Constraints): i) the model contains five elements or less; ii) the model does not have any task, any start event, or

[2]See: https://www.omgwiki.org/bpmn-miwg/doku.php, accessed at 10-06-2023.
[3]This number excludes vendor-provided example processes, duplications of which we excluded.
[4]See: https://www.signavio.com/products/process-explorer/, accessed at 02-05-2023.

any end event; iii) the model fails a syntax check (executed by a commercial process modeling tool); iv) the model has more than one BPMN *pool*, i.e. it models a cross-organizational process. After filtering, 3,960 SAP-SAM models and 2,196 VBPMN models remained.

Our constraint generator managed to parse and compile 94.13% of the SAP-SAM models and 99.9% of the VBPMN models. The total amount of constraints generated by the SAP-SAM models where $149,098$ for the Petri net replay-based approach, and $142,982$ constraints were generated by BPMN2Constraints. The VBPMN models generated $81,043$ constraints from the Petri net replay-based approach and $81,956$ by our tool. The compiler achieves 64.46% precision and 66.7% recall for the SAP-SAM models and 93.32% precision and 93.57% recall for the VBPMN models. While the results indicate that there is still room for improvement and deeper evaluation, let us highlight that this preliminary study is, to our knowledge unique in its large scope. Also, we expect that many issues can be traced back to the ambiguity of the BPMN standard and the quality of the models in the SAP-SAM dataset, which presumably contains – despite the filtering – a substantial amount of incomplete or intuitively "wrong" models.

4. Conclusion

We have presented BPMN2Constraints, a tool for extracting control flow from BPMN-based process models as declarative constraints. We hope that the work can make the relation between imperative and declarative constraints more tangible. In the future, the tool can be extended, e.g., to consider constraints beyond process control flow such as roles and responsibilities or to support the output of additional process query languages such as the Process Query Language (PQL) [6]. Also, a future extension of the tool could support the translation of declarative constraints from one language to another (e.g., from SIGNAL to DECLARE and back). Beyond tool functionality, future work could study formal aspects and limitations of breaking down "imperative" process models into declarative constraints.

References

[1] W. M. P. van der Aalst, A. Adriansyah, B. F. van Dongen, Replaying history on process models for conformance checking and performance analysis, WIREs Data Mining Knowl. Discov. 2 (2012) 182–192.

[2] L. Lamport, The temporal logic of actions, ACM Trans. Program. Lang. Syst. 16 (1994) 872–923.

[3] C. Di Ciccio, M. Montali, Declarative Process Specifications: Reasoning, Discovery, Monitoring, Springer International Publishing, Cham, 2022, pp. 108–152. doi:10.1007/978-3-031-08848-3_4.

[4] T. Kampik, A. Lücke, J. Horstmann, M. Wheeler, D. Eickhoff, Signal – the sap signavio analytics query language, 2023. arXiv:2304.06811.

[5] T. Kampik, C. Warmuth, D. Sola, B. Schäfer, L. Axworthy, E. Ivarsson, K. Ouda, D. Eickhoff, Sap signavio academic models, 2022. doi:10.5281/zenodo.6964944.

[6] A. Polyvyanyy, Process query language, in: A. Polyvyanyy (Ed.), Process Querying Methods, Springer, 2022, pp. 313–341.