GIVUP: Automated Generation and Verification of Textual Process Descriptions

Quentin Nivon
Univ. Grenoble Alpes, CNRS, Grenoble INP, Inria, LIG
38000 Grenoble, France
quentin.nivon@inria.fr

Gwen Salaün
Univ. Grenoble Alpes, CNRS, Grenoble INP, Inria, LIG
38000 Grenoble, France
gwen.salaun@inria.fr

Frédéric Lang
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
38000 Grenoble, France
frederic.lang@inria.fr

Abstract

A business process is defined as a set of tasks executed in a certain order to achieve a specific goal. Business Process Model and Notation (BPMN) has become the standard modelling language for describing and developing business processes. One of the main challenges in the business process management area is to provide techniques and tools for analysing and optimising processes, which are necessary for example to avoid bugs or unexpected process executions. However, modelling and debugging processes is a difficult task. This paper presents GIVUP, a tool that takes as input descriptions of a process and of a functional property written in natural language. GIVUP transforms the textual process into BPMN and the textual property into the corresponding Linear Temporal Logic (LTL) formula. It then verifies whether the BPMN process satisfies the property or not, in which case it returns a diagnostic. These steps are achieved by using several internal transformations and model checking techniques. This approach is helpful for any kind of users, either novices or experts, and can be used during several stages of the lifecycle of a process, such as the design phase or any refinement phase. GIVUP is freely accessible online, and a demo video can be found at: https://youtu.be/MdM4NaPQXMk.

CCS Concepts

• **Computing methodologies** → **Information extraction**; • **Theory of computation** → Grammars and context-free languages; **Verification by model checking**; • **Software and its engineering** → **Visual languages**.

Keywords

BPMN, LLM, Generation, Model Checking, Verification, Counterexample Analysis

ACM Reference Format:
Quentin Nivon, Gwen Salaün, and Frédéric Lang. 2025. GIVUP: Automated Generation and Verification of Textual Process Descriptions. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3696630.3728593

FSE Companion '25, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06
https://doi.org/10.1145/3696630.3728593

1 Introduction

Business Process Management (BPM) is the activity of modelling, improving, and optimising processes of a company, so that these processes can be better understood while achieving correctly what is expected from them. Business Process Model and Notation (BPMN) is a workflow-based graphical notation for specifying business processes. One existing challenge is to provide techniques for supporting the formal development of processes in order to avoid erroneous descriptions and ensure correct process executions. Classic verification techniques require a certain level of expertise. The main goal of the tool presented in this paper is therefore to be accessible to any kind of users, either novices or experts, as one can describe both processes and their functional requirements in natural language. The tool is called GIVUP, which stands for *GeneratIon and Verification of Underspecified Processes*.

GIVUP takes as input a textual description of a process in natural language and automatically generates a corresponding BPMN process. To do so, it first uses GPT-4o [12] to extract task dependencies and additional information from the description, and then performs internal operations to manipulate these dependencies in order to generate a single BPMN process. GIVUP can also take as input a textual description of a functional property that must be preserved by the process being designed. In that case, beyond generating the BPMN process, GIVUP checks whether the process satisfies the given property. This is achieved by transforming both the BPMN process into a language understandable by model checkers, and the textual property into Linear Temporal Logic (LTL) [26], and finally by calling model checking techniques to verify if the process model satisfies the property. In order to refine the process description, the user can analyse as many properties as desired. If a property is violated, several options are proposed for visualising the diagnostic (classic counterexample, set of all counterexamples, or coloured BPMN process). GIVUP was applied to a large set of examples for evaluation purposes.

The rest of the paper is organised as follows. Section 2 introduces BPMN and the basics of model checking. Section 3 gives an overview of the steps applied by GIVUP to perform the generation and the verification of BPMN processes. Section 4 describes how GIVUP was evaluated. Section 5 surveys related work and Section 6 concludes.

2 Models

2.1 BPMN

BPMN 2.0 (BPMN, as a shorthand, in the rest of this paper) was published as an ISO/IEC standard in 2013 [17] and is nowadays extensively used for modelling and developing business processes.

This paper focuses on activity diagrams including the BPMN constructs related to control-flow modelling and behavioural aspects.

Specifically, the node types *event*, *task*, and *gateway*, and the edge type *sequence flow* are considered. Start and end events are used, respectively, to initialise and terminate processes. A task represents an atomic activity that has exactly one incoming and one outgoing flow. A sequence flow connects two nodes executed one after the other in a specific execution order. In this work, the two main kinds of gateways used in activity diagrams are considered, namely, exclusive and parallel gateways. Gateways with one incoming flow and multiple outgoing flows are called *splits*, while gateways with one outgoing flow and multiple incoming flows are called *merges*. A parallel gateway creates concurrent executions for all its outgoing flows or synchronises concurrent executions of all its incoming flows. An exclusive gateway chooses one out of a set of mutually exclusive alternative incoming or outgoing flows. Such gateways can also be used to represent repetitive behaviours (i.e., loops). For verification purposes, BPMN processes are often mapped to their equivalent Labelled Transition Systems (LTSs), which describe precisely the semantics of these processes.

2.2 Model Checking

Model checking consists in verifying that a model (an LTS in this work) satisfies a given temporal logic property, which specifies some expected requirements of the system. Temporal logic properties are usually divided into two distinct families: safety and liveness properties [1]. Safety properties state that something bad must never happen while liveness properties state that something good eventually happens. Several temporal logics exist and can be used for modelling such properties. In this work, we consider LTL because it is rather simple to use and expressive enough to represent properties on execution paths. The LTL syntax consists of propositions, Boolean operators and temporal operators (particularly next X and until U). Among the many existing model checkers, GIVUP makes use of the CADP toolbox [16] to perform the verification of the process.

3 Overview

GIVUP aims at automatically generating a BPMN process and a temporal logic property from a textual description, and verifying the validity of the property on the generated process. If the property is not satisfied, GIVUP returns a diagnostic that can be displayed in different formats: a trace of the LTS violating the property, an LTS containing all the traces violating the property, or a coloured version of the BPMN process where the parts violating the property are coloured in red, whilst the parts satisfying it are coloured in green. Figure 1 presents the different steps required to generate the process and the property, and verify whether this property holds or not.

3.1 Generation of BPMN and Temporal Logic Property from Textual Descriptions

The first part of this approach consists in generating the process that should be model checked and the property to verify from textual descriptions. The goal of such an approach is to allow any kind of user (i.e., familiar to BPMN/temporal logic or not) to build its own model and to be able to verify that this model fulfils its expected requirements (i.e., that it behaves as it is supposed to). Generating a syntactically correct BPMN process or the right temporal logic property from a textual description is a complex task. This step was made possible thanks to the use of Large Language Models (LLMs), and more precisely GPT-4o (GPT, as a shorthand, in the rest of this paper), for its capacities to analyse an informal text, extract the essential information contained in it, and return an output compliant with a required format.

Generating BPMN. Based on the experiments carried out on various well-known LLMs, it was concluded that LLMs are not yet able to produce correct BPMN processes directly. Thus, this step requires several intermediate steps. The first intermediate step consists in extracting the tasks of the process-to-be, along with their ordering constraints (i.e., the way they are related to each other). To be able to do so, GPT was fine-tuned on roughly 400 examples in order to produce expressions compliant with an internal grammar similar to the regular expressions' one available in [25]. Once such expressions are generated by GPT, they are mapped to their equivalent Abstract Syntax Trees (ASTs). This hierarchical structure was chosen because it provides by construction priority between nodes (the higher the node, the higher its priority) and consequently between the operators of the grammar. The information split in the original ASTs is then gathered using internal algorithms to produce a single AST, called *merged AST*, which contains all the information of the original ASTs. Finally, the merged AST is converted into its equivalent BPMN process according to some AST to BPMN transformation patterns.

Generating Temporal Logic Property. Similarly to the generation of BPMN processes, based on the experiments carried out in the context of this work, it was concluded that LLMs for now have a very basic knowledge of temporal logics. Thus, generating an LTL property matching exactly the expectations that the user writes in natural language is a challenging task. For this reason, GIVUP is for now recognising nine precise patterns, corresponding to the nine LTL formulas on which GPT was trained. These patterns, such as "Tasks T_1, T_2, ..., T_n must precede tasks T_{n+1}, ..., T_z", were used to fine-tune GPT in order to make it able to generate the correct property in these precise cases. This fine-tuning task was performed on approximately 260 descriptions of properties, with roughly the same number of descriptions for each available pattern.

3.2 Verification of the Property

Once the BPMN process is generated, it has to be model checked with regards to the generated LTL property in order to assess its validity. However, model checkers do not natively support the BPMN format as input. Thus, the process first has to be transformed into a format compliant with classical model checkers. VBPMN [21] is a tool allowing one to translate a BPMN process into LNT [9], a modern formal specification language that combines traits from process calculi, functional languages, and imperative languages. An LNT specification can then be model checked using the CADP model checker.

As CADP does not natively support the LTL language, the LTL property must be converted into a format that it can understand. This format is a Büchi automaton, whose generation is ensured by the SPOT toolbox [8]. Finally, CADP is called on the LNT specification and the Büchi automaton to verify the property.

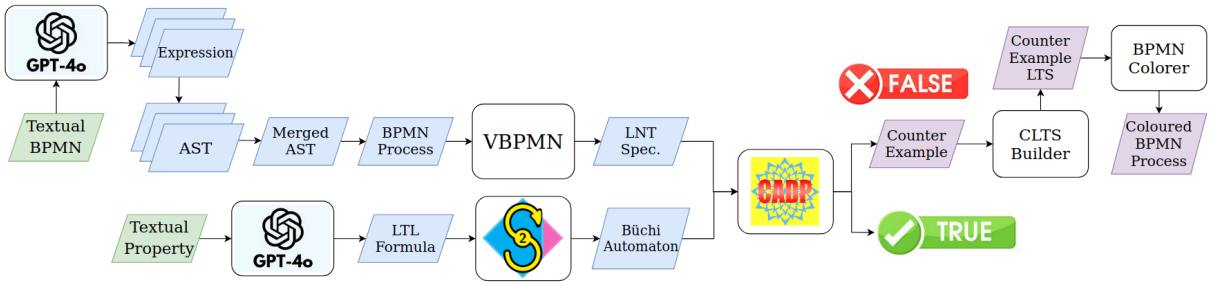

Figure 1: Overview of the Toolchain

3.3 Diagnostics

The verification of the property by CADP leads to a verdict, which is either *True* if the property holds on the given specification, or *False* if the specification violates the property. In the latter case, CADP returns a diagnostic in the form of a counterexample. This counterexample is a trace of the LTS that violates the given property.

Counter-example LTS. A single counterexample is often not showing precisely the source of the violation of a property. For this reason, GIVUP provides an additional debugging option, the counterexample LTS (CLTS [2]). A CLTS is basically an LTS containing all the traces violating the property, instead of the single one returned by the model checker. With this representation, the user is more likely to find the source of the error, or to understand the global impact that this error has on the model.

Coloured BPMN. The CLTS is very helpful to localise the source of the error, but can sometimes be difficult to interpret or understand due to its lack of similarity with the original BPMN process. For this reason, the last debugging option proposed by GIVUP is a coloured BPMN process highlighting in red (resp. green) the incorrect (resp. correct) parts of the model. The technique used to obtain this coloured process is detailed in [24]. This process is as syntactically close as possible to the original process, and aims at helping users not familiar with LTSs to understand the reason behind the violation of the property.

3.4 Implementation

GIVUP consists of approximately 20k lines of Java code embedded in the backend of a web server for distribution purposes. The web server UI is built with HTML, JavaScript and Bootstrap 5, while the backend relies on NodeJS. The training dataset, the validation dataset and the source code of GIVUP are available online.[1]

4 Evaluation

To evaluate the approach, several experiments were conducted. These experiments are divided into three parts: quality of the generated BPMN process, quality of the generated LTL property, and performance of GIVUP.

Quality of the Generated BPMN Process. The generation of BPMN processes was tested and validated on 200 descriptions coming from various sources. 25% come from the literature (PET dataset [5], proceedings, ...), while the remaining 75% were handcrafted by 9 users (5 experts and 4 novices) who experimented GIVUP. To the best of our knowledge, there are only two tools aiming at generating BPMN processes from natural language descriptions available online: ProMoAI [20] and NaLa2BPMN [7]. GIVUP was compared to them, and also to Gemini [10] and GPT-4-turbo [12] prompted directly. Table 1 shows the results obtained by the 5 tools. Column 1 presents the used tool, while Column 2, 3 and 4 respectively contain the percentage of valid processes (i.e., correct with regards to the description and corresponding to the processes expected by the experts), ambiguous processes (i.e., correct with regards to the description but not corresponding to the processes expected by the experts), and incorrect processes (i.e., incorrect with regards to the description). Finally, Column 5 gives the time taken by each tool to generate the BPMN process on average.

Table 1: Results of the BPMN Generation Experiments

Tool/Model	✓	?	✗	Avg. Exec. Time (s)
GIVUP	**83%**	9.8%	**7.2%**	**4.43**
NaLa2BPMN	32.8%	8.9%	58.3%	68.7
ProMoAI	50%	8.7%	41.2%	24.7
Gemini	32.2%	**8.1%**	59.7%	8.32
GPT-4-turbo	66.6%	21.1%	12.2%	19.2

GIVUP obtained the highest accuracy with 83% of exactly matching processes, along with the shortest execution time on average (4.43s). It also obtained the lowest invalid processes percentage with only 7.2%. It is interesting to note that GPT-4-turbo performed quite well, with a low invalid processes percentage of 12.2%.

Quality of the Generated LTL Property. In this approach, the generated LTL property must correspond exactly to the textual description written by the user. Otherwise, the verification step becomes useless, as the behaviour assessed by the model checker is not the expected one. To obtain the desired property, GIVUP currently restricts the user by forcing her/him to choose between nine well-defined patterns. On these nine patterns, the properties generated from one hundred examples were all correct.

Performance of GIVUP. These experiments aimed at assessing the scalability of GIVUP. They were conducted on both real-world examples coming from the literature, and handcrafted examples. Table 2 summarises the results. Column 1 provides the origin of the process, Columns 2 and 3 give details about the LTS corresponding to the specification (i.e., number of states and transitions of the LTS corresponding to the BPMN process), and Columns 4, 5, 6, 7 and 8 present the time taken by each major step of the approach (i.e., generation of the BPMN process, generation of the LTL property, model checking of the property, construction of the CLTS, and colouration of the BPMN process).

[1] https://github.com/QuentinNivon/Text_to_BPMN

Table 2: Results of the Performance Experiments Conducted on GIVUP

BPMN Process	States	Trans.	BPMN Gen. Time	Prop. Gen. Time	Model Check. Time	CLTS Cons. Time	BPMN Colo. Time
Evisa App. [27]	30	31	1.67s	1.43s	4.35s	3.12s	613ms
Patient Diag. [3]	38	40	2.71s	1.14s	4.57s	3.29s	661ms
Employee Rec. [13]	39	40	1.89s	1.33s	4.57s	3.22s	830ms
Employee Hiring [6]	78	105	2.17s	2.86s	4.34s	3.22s	758ms
Perish. Goods Trans. [28]	108	150	2.22s	1.1s	4.78s	3.31s	978ms
Acc. Open. Proc. [24]	304	657	2.31s	1.23s	4.56s	4.23s	1.42s
Hard. Ret. Ship. Proc. [14]	373	819	2.56s	1.1s	4.53s	3.13s	764ms
Online Shipping [22]	375	765	2.78s	1.07s	5.12s	3.27s	1.7s
Handcrafted 1	279k	1.63m	3.25s	1.07s	8s	14.6s	17.2s
Handcrafted 2	1.67m	11m	1.79s	1.27s	23.9s	5.42m	24m
Handcrafted 3	10m	75m	1.67s	1.18s	3.05m	>1h	>1h
Handcrafted 4	60m	503m	2.08s	867ms	27.7m	>1h	>1h
Handcrafted 5	362m	3.32b	2.31s	1.85s	>1h	>1h	>1h

These experiments show that the total execution time including the non-mandatory steps (CLTS construction and BPMN colouration) never exceeds 15s for real-world examples[2]. The mandatory steps are themselves executed in at most 10s for real-world examples. For the model checking step, the first limitations appear for processes containing approximately 10 million states, as CADP takes a few minutes to compute the result. This limit corresponds to the well-known state explosion issue that CADP (and all model checkers) suffers from. For the CLTS generation, the limit is reached earlier. Indeed, the construction of the CLTS requires the computation of all the counterexamples of the property, which implies repeating the model checking step multiple times, thus summing its execution time. Finally, to be able to colour the BPMN process, an intermediate step called *unrolling* is required. Unrolling consists in duplicating each node of the BPMN process having more than one incoming flow until no such node exists in the process. Consequently, this step increases exponentially the number of nodes of the process, making the colouring process longer than the CLTS generation. However, it is worth noting that, in practice, the LTS model of a BPMN process rarely exceeds a few thousand states, and therefore almost never suffer from any of the problems described above, thus ensuring a short execution time.

5 Related Work

The Sketch Miner [18] tool combines notes taken in constrained natural language with process mining techniques to automatically produce BPMN diagrams. This approach relies on a textual DSL, which defines how the user can describe its process textually. In [11], a series of question/answer exchanges with a chatbot enhanced with natural language processing capabilities are performed. The goal of these iterations is to create or improve process models. However, the authors restricted their focus on the extraction of tasks. These approaches either impose specific constraints on the input textual format or end up with fragments of processes that have to be connected manually, while our approach automatically provides an entire BPMN process without imposing any restriction on the input textual requirements.

[23] presents a software-based tool to support generating business process models in BPMN by using Natural Language Processing (NLP) methods. This approach starts with oral explanations, and thus integrates the Whisper automatic speech recognition system. This approach only works for German language. [20] introduces a novel framework that leverages the capabilities of LLMs to generate and refine process models starting from textual descriptions. This system uses the Partially Ordered Workflow Language (POWL) as intermediate representation. These two tool-based approaches are quite preliminary with limited evaluation and ongoing improvements. [7] and [19] present an approach to automate process model generation from textual descriptions using LLMs. These solutions apply successively different steps: the LLM analyzes, clarifies, and completes the textual description, and extracts process entities and relationships. The evaluation section showed that our approach outperforms these works in terms of performance and accuracy.

There have been several attempts to transform text to temporal logics. We introduce two recent libraries trying to do so for LTL. [15] presents a Python package called NL2LTL, which combines Natural Language Understanding (NLU) and LLMs to translate natural language instructions to LTL formulas. This approach works as ours for specific patterns of properties. [5] describes a framework for translating unstructured natural language to LTL. The methodology is to decompose the natural language input into subtranslations, which are mappings of formula fragments to relevant parts of the natural language input. The user can thus interactively correct some erroneous sub-translations. We preferred to rely on patterns to guarantee high accuracy of the generated formulas, and avoid any required expertise in temporal logics.

Several tools have attempted to provide verification techniques for BPMN. VBPMN [21] allows one to check the validity of functional properties on a given business process. To do so, it relies on model checking. BPROVE [4] enables the verification of properties such as soundness and safeness both at the process and collaboration levels. Additionally, more specific properties are defined in order to ensure the conformance of the model to specific behavioural patterns. Both VBPMN and BPROVE require a BPMN process and a temporal property as input, whereas in this work we start from textual descriptions for these two inputs. We also provide more expressive diagnostics, useful for debugging purposes.

6 Concluding Remarks

In this paper, we presented GIVUP, a tool taking as input descriptions of a business process and of expected requirements written in

[2]The reader testing GIVUP may experience longer execution times due to the distance from the server hosting it.

natural language. A LLM is first used to extract structured information from this text, which allows us to obtain both a BPMN process and an LTL property. Several transformations are then applied to be able to reuse existing model checkers for verifying whether the BPMN process satisfies the given property. If this is not the case, a diagnostic is returned by the model checker and can be visualised using different formats. One option shows the semantic model with colouration techniques to emphasise the traces that do not respect the property. Another option colours the BPMN process for showing which (syntactic) parts of the process violate the property. This work offers several perspectives, such as improving the quality of the generated processes, or continuing training the LLM model to cover a larger set of LTL patterns.

References

[1] C. Baier and J.-P. Katoen. 2008. *Principles of Model Checking*. MIT Press.

[2] G. Barbon, V. Leroy, and G. Salaün. 2017. Debugging of Concurrent Systems Using Counterexample Analysis. In *Proc. of FSEN'17 (LNCS)*. Springer, 20–34.

[3] E. Bazhenova, F. Zerbato, B. Oliboni, and M. Weske. 2019. From BPMN process models to DMN decision models. *Information Systems* 83 (2019), 69–88.

[4] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, and A. Vandin. 2017. BProVe: tool support for business process verification. In *Proc. of ASE'17*. 937–942.

[5] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. 2023. nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models. In *Comp. of CAV'23 (LNCS, Vol. 13965)*. Springer, 383–396.

[6] F. Durán, C. Rocha, and G. Salaün. 2018. Computing the Parallelism Degree of Timed BPMN Processes. In *Proc. of FOCLASA'18*. 1–16.

[7] A. Nour Eldin, N. Assy, O. Anesini, B. Dalmas, and W. Gaaloul. 2024. A Decomposed Hybrid Approach to Business Process Modeling with LLMs. In *Proc. of CoopIS'24 (LNCS, Vol. 15506)*. Springer, 239–256.

[8] A. Duret-Lutz et al. 2022. From Spot 2.0 to Spot 2.10: What's New?. In *Computer Aided Verification*. Springer International Publishing, 174–187.

[9] D. Champelovier et al. 2018. Reference Manual of the LNT to LOTOS Translator (Version 6.7). (2018). INRIA/VASY and INRIA/CONVECS, 153 pages.

[10] Gemini Team et al. 2024. Gemini: A Family of Highly Capable Multimodal Models. arXiv:2312.11805 [cs.CL] https://arxiv.org/abs/2312.11805

[11] N. Klievtsova et al. 2023. Conversational Process Modelling: State of the Art, Applications, and Implications in Practice. In *Proc. BPM'23*. Springer, 319–336.

[12] OpenAI et al. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[13] Y. Falcone, G. Salaün, and A. Zuo. 2021. Semi-automated Modelling of Optimized BPMN Processes. In *Proc. of SCC'21*. IEEE, 425–430.

[14] Y. Falcone, G. Salaün, and A. Zuo. 2022. Probabilistic Model Checking of BPMN Processes at Runtime. In *Proc. of IFM'22*. Springer, 1–17.

[15] F. Fuggitti and T. Chakraborti. 2023. NL2LTL - a Python Package for Converting Natural Language (NL) Instructions to Linear Temporal Logic (LTL) Formulas. In *Proc. of AAAI'23*. AAAI Press, 16428–16430.

[16] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. 2013. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. *STTT* (2013), 89–107.

[17] ISO/IEC. 2013. International Standard 19510, Information technology – Business Process Model and Notation.

[18] A. Ivanchikj, S. Serbout, and C. Pautasso. 2020. From text to visual BPMN process models: design and evaluation. In *Proc. of MoDELS'20*. ACM, 229–239.

[19] H. Kourani, A. Berti, D. Schuster, and W.M.P. van der Aalst. 2024. ProMoAI: Process Modeling with Generative AI. In *Proc. of IJCAI'24 (IJCAI-2024)*. International Joint Conferences on Artificial Intelligence Organization. https://doi.org/10.24963/ijcai.2024/1014

[20] H. Kourani, A. Berti, D. Schuster, and W. M. P. van der Aalst. 2024. Process Modeling with Large Language Models. In *Proc. of BPMDS'24 (LNBIP, Vol. 511)*. Springer, 229–244.

[21] A. Krishna, P. Poizat, and G. Salaün. 2017. VBPMN: Automated Verification of BPMN Processes. In *Proc. of IFM'17 (LNCS)*. Springer, 323–331.

[22] A. Krishna, P. Poizat, and G. Salaün. 2019. Checking business process evolution. *Science of Computer Programming* 170 (2019), 1–26.

[23] M. Mößlang, R. Bernsteiner, C. Ploder, and S. Schlögl. 2024. Automatic Generation of a Business Process Model Diagram Based on Natural Language Processing. In *Proc. of KMO'24 (LNBIP)*. Springer.

[24] Q. Nivon and G. Salaün. 2022. Debugging of BPMN Processes Using Coloring Techniques. In *Proc. of FACS'22*. Springer, 90–109.

[25] Q. Nivon and G. Salaün. 2024. Automated Generation of BPMN Processes from Textual Requirements. In *Proc. of ICSOC'24*. Tunis, Tunisia, 1–16.

[26] A. Pnueli. 1977. The Temporal Logic of Programs. In *Proc. of FOCS'77*. 46–67.

[27] G. Salaün. 2022. Quantifying the Similarity of BPMN Processes. In *Proc. of APSEC'22*. 1–10.

[28] P. Valderas, V. Torres, and E. Serral. 2022. Modelling and executing IoT-enhanced business processes through BPMN and microservices. *J. Syst. Softw.* 184 (2022).