

BeePath: a Framework for Textual Descriptions of Business Processes with Guarantees

Andrea Burattin¹, Antonio Grama¹, Ana-Maria Sima¹, Andrey Rivkin¹, and
Barbara Weber²

¹ Technical University of Denmark, DTU Compute, Denmark
`{andbur,ariv}@dtu.dk`

² University of St.Gallen, St.Gallen, Switzerland
`barbara.weber@unisg.ch`

Abstract. Formal modeling is essential for understanding, analyzing, and optimizing processes, yet formal techniques often pose a barrier for domain experts unfamiliar with them. This paper introduces BeePath, a framework designed to bridge this gap by allowing textual descriptions of business processes while maintaining formal guarantees. Unlike existing approaches that either eliminate ambiguities or impose rigid structures, BeePath leverages a constrained natural language to capture processes in a structured yet accessible manner. The framework also employs large language models (LLMs) to assist in converting unstructured descriptions into this structured format, which is then parsed into formal representations such as Petri nets, BPMN, and DECLARE. Our evaluation demonstrates BeePath’s effectiveness in capturing key process patterns and ensuring model consistency. By integrating natural language with formal process modeling, BeePath aims at enhancing accessibility for domain experts, fostering a more intuitive and democratized approach to business process management.

1 Introduction

Business processes are often described using natural language. However, natural language introduces ambiguities, inconsistencies, and interpretation gaps that hinder automation and formal verification. Existing process modeling approaches attempt to address these challenges through formal modeling languages, structured annotation techniques, or automated ambiguity resolution. However, these methods either require expertise in formal methods or risk distorting domain experts’ intent through rigid rules. Inspired by structured natural language paradigms such as FRETISH [9] (a language adopted in NASA’s Formal Requirement Elicitation Tool) and principles from Behavior-Driven Development (BDD), we introduce BeePath, a framework that allows domain experts to describe business processes in a structured, yet natural, way.

BeePath follows a two-step approach depicted in Fig. 1: (1) **Guided Natural Language Structuring:** Users provide an initial free-text description of a process. This is then transformed into a constrained natural language format,

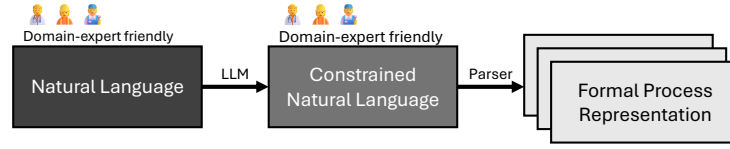


Fig. 1. The general pipeline of the BeePath framework.

ensuring key constraints and flow dependencies are made explicit, similar to how BDD ensures software behaviors are structured. *(2) Model Generation (with Guarantees):* The structured representation is parsed into multiple formal process models, including Petri nets and DECLARE, enabling formal reasoning and automation while maintaining alignment with the original textual intent.

Large Language Models (LLMs) are used to assist in the guided natural language structuring, ensuring that process descriptions remain understandable to end users. This allows for human-in-the-loop validation, enabling domain experts to correct any misinterpretations by the LLM. Unlike fully automated approaches, BeePath maintains expert control over the formalization process, ensuring that the structured language output follows a deterministic mapping to process models. This algorithmic mapping prevents hallucinations and guarantees consistency in model generation. A similar approach is seen, for example, in Celonis’ LLM for PQL queries³. This, however, requires the end-user to know the underlying formal language in order to validate the generated queries.

The rest of the paper is structured as follows: Sect. 2 presents related work; Sect. 3 and 4 describe the general pipeline and the language; Sect. 5 and 6 present the prototype and the evaluation and Sect. 7 concludes the paper.

2 Related Work

The challenges encountered in business process modeling have been addressed, at least in part, by defining process descriptions unambiguously, outlining possible interpretations of ambiguities, automatically resolving ambiguities, and generating or validating process models from structured or unstructured descriptions.

Papers [3, 22] employ user stories to generate structured representations but their effectiveness is limited by the requirement that process descriptions must follow the “user story” structure. User stories do not express the process directly but *how* the process can look like. Papers [19, 24] introduce annotation-based methods, where marked text is converted into temporal logic [23] or DCR models [10], offering flexibility but requiring careful text analysis. Other studies, such as [13, 14, 25], propose custom structured languages that can be mapped to BPMN. This language describes stories of possible runs of the process, which are transformed into traces of an event log and fed to a process mining. From this point of view, the proposed language is capturing traces rather than processes.

³ See <https://www.celonis.com/blog/celonis-brings-process-intelligence-to-every-business-user-with-llm-for-pql-generation/>.

In [11], authors use an intermediate model description based on a spreadsheet to capture the essence of a process. The intermediate language is limited to a total order over activities. More recently, generative AI models have been explored for the automatic generation of BPMN models from unstructured text [18], but these still require challenging manual validation on the target language as well as their understanding, which, oftentimes, can be more difficult than modeling in the first place [4]. Additionally, some research [16, 17] focuses on converting BPMN models into textual descriptions to aid validation, but this task falls outside the scope of this paper (which tackles the dual problem).

BeePath is partially inspired by the FRET framework [9]. The structured natural language adopted in FRET is template-based, which, above all, allows to remove ambiguity in requirement specifications. In BeePath, we use process fragments (which are similar to FRET’s templates) and allow compilation of the resulting specifications into multiple formalisms (namely, Petri nets [20] and DECLARE [7]).

3 A Pipeline for Constrained Natural Language Process Description

When tackling our problem, we decided to adopt an approach that does not assume that the original textual description is free from ambiguities, nor that the proposed system should solve these ambiguities automatically. Instead, our rationale was to provide a tool that would allow domain experts to remain in control of the formalization. Therefore, the involvement of domain experts during the entire pipeline remains a fundamental requirement for us.

We assume that modeling is carried out by domain experts who understand the process but lack formal modeling expertise. Natural language serves as a common medium for expressing process knowledge. By introducing a constrained natural language, we enable domain experts to create descriptions that can be systematically transformed into formal (process) models.

As shown in Fig. 1, our framework follows a two-step approach. First, a natural language process description is converted into a constrained natural language according to a predefined grammar. Second, a parser translates the resulting constrained descriptions into formal process representations. An LLM can support the first step by generating an initial draft, while the second step is handled by a parser that produces the formal model. Since the LLM is used only to assist in generating an intermediate textual representation, any inaccuracies or hallucinations that may arise (and which are common when LLMs are used as “functions” for deriving process models from natural language) have limited impact and can be subsequently validated by domain experts.

The current parser supports three modeling languages: Petri Nets, BPMN, and DECLARE. These were selected to cover both imperative and declarative paradigms as well as theoretical and industrial perspectives. Nevertheless, extending the framework to additional modeling languages only requires the development of a corresponding parser.

4 Constrained Natural Language for Process Specification

In this section, we are going to discuss the grammar of our process description language. We show how this grammar can be used to describe processes and how the obtained descriptions can be automatically translated into Petri nets and DECLARE. While the translation into BPMN is also supported in our implementation, it is not covered in this paper due to the page limit and similarity with the Petri net translation.

4.1 A Language for Describing Processes

The designed process description language, called BeePath, serves as the foundation of the solution, enabling the users to describe processes using a constrained natural language, close to unstructured natural language descriptions.

Conceptually, BeePath relies on having a catalog of textual process description *fragments* available. These fragments can be instantiated several times to achieve the expected final model. We decided to leverage the well-known *work-flow patterns* initiative [1] for our fragments. These are language-independent workflow structures that capture business characteristics in a generalized manner, making them applicable across various domains. Our primary goal with BeePath is not to be comprehensive of all possible fragments but rather to define a *framework* that can be easily extended to include more behavior. For this reason, in addition to the “Basic Control Flow Patterns”, we introduced some additional fragments. To identify these fragments, we collected process descriptions from papers explicitly tackling the problem of modeling from natural language, and we ended up with three descriptions (two from [8], and one from [24], see Fig. 2). Similar to pattern elicitation processes [15], we started with only the Basic Control Flow Patterns, and we carefully inspected the process descriptions to find empirical evidence of new fragments. Given the low number of processes analyzed, we required each candidate fragment to appear once in order to be included on the list. After each new fragment was identified, the process descriptions were analyzed again until no new fragments could be recognized (i.e., saturation). Please note that the extra fragments identified are not meant to be new *patterns* (i.e., “reusable solution to a commonly occurring problem” [2]). Instead, they are meant to show that BeePath can accommodate arbitrary behavioral concepts beyond the Basic Control Flow Patterns.

Fig. 2 illustrates the identified fragments within the corresponding textual description. In the figure, different text segments are highlighted in colors corresponding to the fragments they represent, as indicated in the legend. Additionally, arrows are used to connect fragments and activities visually, enhancing the reader’s understanding.

BeePath (more specifically, considered process description fragments) has been converted into a formal grammar that includes all the identified process fragments. This grammar is described using extended Backus–Naur form (EBNF), and is available in the accompanying technical report [5] with a detailed description of all its productions.

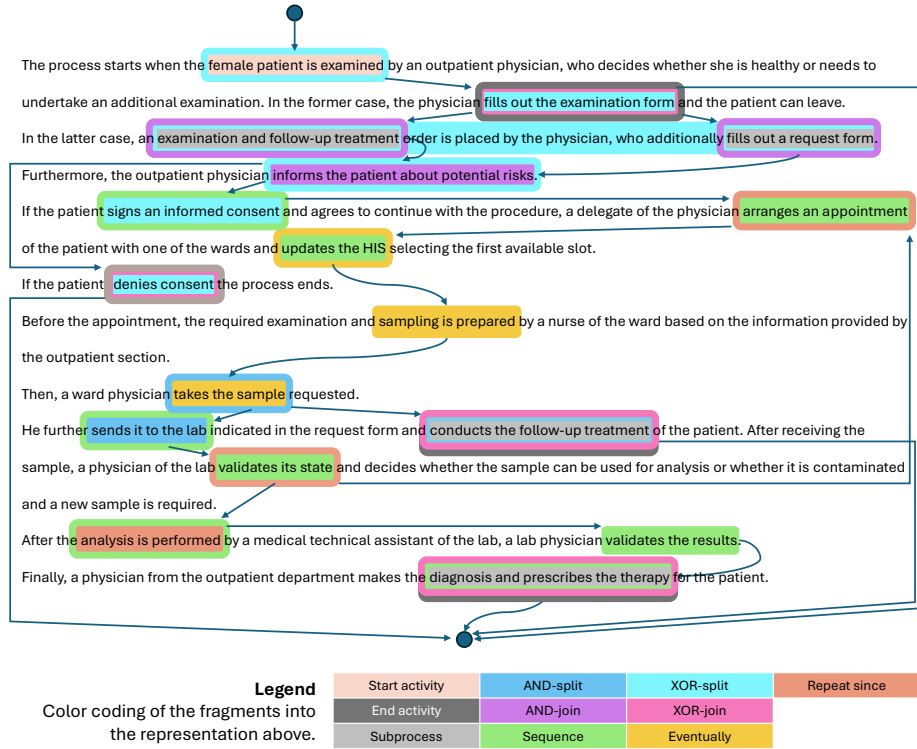


Fig. 2. Graphical representation of the Hospital treatment description with the constraints highlighted. Full resolution at <https://github.com/processintelligence/beepath/tree/main/paper-material/hospital-description.pdf>.

At the grammar’s core, a process is defined as a collection of fragments, that describe local patterns of behavior. These fragments, and examples of strings derived from such a list can be seen in Table 1, where the first column states the fragment name and the second provides a corresponding BeePath string. As the table caption also suggests, it is easy to see fragments having formal parameters referring to activities and/or sub-process identifiers. Consider the description from Fig. 2. Its possible translation using BeePath is presented in Fig. 3.

There are two crucial observations to make about BeePath. First, the BeePath language cannot automatically resolve (semantic) ambiguities present in source texts, but offers a structured approach to force *users* to do so. Second, the language also offers ways to address incomplete (source) specifications, including missing or unclear connections between activities. For instance, in Fig. 2, if a “sample analysis” was deemed invalid, the original text only mentioned that “a new sample was required” without specifying which activity should follow (e.g., “take sample” or “arrange appointment”). However, when converting the description into BeePath, the user must explicitly resolve such issues by specifying which

Table 1. Instances of identified fragments and corresponding strings generated in the BeePath language. Fragment arguments appear in **red**.

Fragment instance	Example of a BeePath language string
Sequence (A , B)	After " A " ends, immediately start " B ".
Parallel Split (A , B , C)	After " A " ends, immediately start " B " and start " C ".
Synchronization (A , B , C)	After " A " ends and " B " ends, immediately start " C ".
Exclusive Choice (A , B , C)	After " A " ends, immediately either start " B " or start " C ".
Simple Merge (A , B , C)	After either " A " ends or " B " ends, immediately start " C ".
Repeat Since (A , B , C)	After " A " ends, immediately repeat since " B " or start " C ".
Eventually (A , B)	After " A " ends, eventually start " B ".
AND-split in XOR-split (A , B_and_C , D)	After " A " ends, immediately either start (B_and_C) or start " D ".
AND-join in XOR-join (A_and_B , C , E)	After either (A_and_B) ends or " C " ends, immediately start " E ".
XOR-split in AND-split (A , B_or_C , D)	After " A " ends, immediately start (B_or_C) and start " D ".
XOR-join in AND-join (A_or_B , C , E)	After (A_or_B) ends and " C " ends, immediately start " E ".
OR-subprocess (A_or_B , A , B)	(A_or_B): " A " or " B "
AND-subprocess (A_and_B , A , B)	(A_and_B): " A " and " B "

activity should be repeated. Without this, the resulting model would be incomplete and disconnected. Finally, the current version of the BeePath language exhibits some limitations: (1) The current BeePath grammar has an imperative bias. This means that, when describing a process in BeePath, all possible execution paths must be explicitly defined. (2) The current BeePath grammar does not allow specifying nested subprocesses (e.g., an AND subprocess within an OR subprocess). This restriction was a deliberate choice to avoid adding unnecessary complexity to the grammar, the underlying patterns, and their translation into formal process models. (3) The lack of (process) resources and roles.

4.2 Parsing BeePath

Once the textual description is available, the parsing algorithm is executed. The parsing algorithm transforms sentences written in the BeePath language into a target (formal) process model expressed in one of the supported languages. In a nutshell, the parsing algorithm goes through the following steps:

1. Iterate through all input sentences in the order in which they are written and generate the corresponding abstract syntax tree (AST).
2. Use the AST to identify for each sentence which fragment type it represents and its arguments (activity names and/or sub-process identifiers). Store the obtained information in a "global storage".⁴
3. For each identified fragment, invoke a corresponding translation sub-routine yielding a fragment of the target formal model.
4. Produce the global model by joining the outputs for all identified fragments.⁵

In the current implementation of the BeePath framework, the parsing is targeting Petri nets, BPMN, and DECLARE. We opted for these languages as representative of different modeling paradigms: imperative and declarative.

⁴ This is needed to keep the information about encountered activities in order to maintain the referential integrity between parsed sentences (sentences that "re-use" the same activities will be parsed into formal models with the same activity labels)

⁵ The join operation is specific to each target formalism.

```

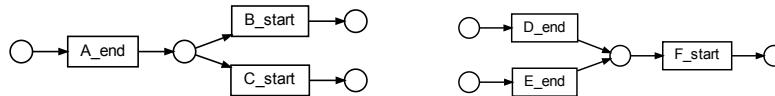
1 The following textual description follows the closed-world assumption,
  meaning that only the activities...
2 Initially start "examine female patient".
3 (s1): "order examination with follow up treatment" and "fill out request form
  ".
4 (s2): "conduct follow up treatment" and "diagnose with prescribe therapy".
5 After "examine female patient" ends, immediately either start "fill out
  examination form" or start (s1).
6 After "order examination with follow up treatment" ends and "fill out request
  form" ends, immediately start "inform patient about risks".
7 After "inform patient about risks" ends, immediately either start "sign
  consent" or start "deny consent".
8 After "sign consent" ends, immediately start "arrange appointment".
9 After "arrange appointment" ends, immediately start "update HIS selecting".
10 After "update HIS selecting" ends, eventually start "prepare sampling".
11 After "prepare sampling" ends, eventually start "take sample".
12 After "take sample" ends, immediately start "send sample to lab" and start "
  conduct follow up treatment".
13 After "send sample to lab" ends, immediately start "validate sample".
14 After "validate sample" ends, immediately repeat since "arrange appointment"
  or start "perform analysis".
15 After "perform analysis" ends, immediately start "validate results".
16 After "validate results" ends, immediately start "diagnose with prescribe
  therapy".
17 After either (s2) ends or "deny consent" ends or "fill out examination form"
  ends, the process finishes.

```

Fig. 3. Conversion in BeePath language of the Hospital treatment example.

The fragments identified in Section 4.1 are translated into Petri nets (and BPMN) and DECLARE according to translation rules.

In the following, we report just two fragments, but the complete set of translation rules is available in the accompanying technical report [5]. Let's focus on Exclusive Choice (A, B, C) and Simple Merge (D, E, F). These two complementary fragments are translated into two Petri nets $\mathcal{N}_1$ and $\mathcal{N}_2$:



$\mathcal{N}_1$ ensures that when an activity ends, its *end* transition (e.g., *A_end*) is connected to the *start* transitions (e.g., *B_start* and *C_start*) of multiple alternative activities or subprocesses, with only one of them being able to execute. No other transitions can occur between them. $\mathcal{N}_2$ ensures that multiple alternative activities or subprocesses can *end* (e.g., *D_end* and *E_end*), with only one of them leading to the *start* of the next activity (e.g., *F_start*). The DECLARE representation $\mathcal{D}_1$ for Exclusive Choice requires a complex combination of DECLARE templates, including an “artificial” end event that forces $\mathcal{D}_1$ to complete, even though their corresponding automata-based representation is straightforward:

$$\begin{aligned}
 \mathcal{D}_1 = \{ & \text{Not CoExistence}(B, C); \text{Alternate Precedence}(B, A); \\
 & \text{Alternate Precedence}(C, A); \text{Exactly One}(A); \\
 & \text{Alternate Response}(B, \text{end}); \text{Alternate Response}(C, \text{end}); \\
 & \text{Not Chain Succession}(A, \text{end}); \text{Not Chain Succession}(\text{end}, A) \}
 \end{aligned}$$

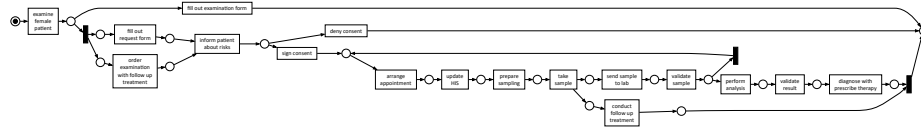
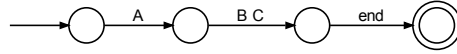
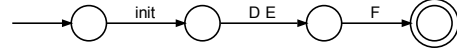


Fig. 4. Petri net resulting from the automatic conversion of Fig. 3. Please note that the layout has been redrawn for representation purposes and that “start” and “end” activities have been collapsed into a single transition.



The DECLARE representation $\mathcal{D}_2$ for Simple Merge requires the dual “artificial” initialisation event (marked as init) and is as follows:

$$\mathcal{D}_2 = \{ \text{Alternate Response}(D, F); \text{Alternate Response}(E, F); \\ \text{Not CoExistence}(D, E); \text{Not Chain Succession}(\text{init}, F); \\ \text{Not Chain Succession}(F, \text{init}); \text{Exactly One}(F); \\ \text{Init}(\text{init}); \text{Exactly One}(\text{init}) \}$$



Following the approach described in the beginning of the section, the final formal model is constructed by joining models obtained for each local fragment. In the case of DECLARE, the final model is the union of all local specifications. In the case of Petri nets, a slightly more elaborate treatment is necessary: local fragments are merged using Petri nets composition based on transition labels, i.e., the activity names. In Fig. 4, we show the result of translating the text from Fig. 3 into a Petri net.

5 Tool Support

A tool implementing the BeePath framework has been built as a web application using TypeScript and NodeJS.⁶ The grammar for BeePath and its corresponding parsers were implemented using ANTLR. The output is rendered using Graphviz (for Petri nets and BPMN) and declare-js [21] (for DECLARE) and can also be downloaded as BPMN, TPN, or DECL files. The connection to the LLM was established through a REST API request to the OpenAI platform where all their models can be chosen and used (a valid key is required in the web application, but the platform proposes some templates in BeePath language).

⁶ Source code available at <https://github.com/processintelligence/beepath> and deployment available at <https://processintelligence.github.io/beepath/>.

Table 2. Coverage of the BeePath fragments from the Model Judge process descriptions (manual conversion).

Process	Tot frag. covered	Frag. covered	Frag. with workaround not covered	Frag. covered
Court Hearing	6	5	1	0
Credit Rating	7	7	0	0
Dispatch	7	5	2	0
Web Frontend	10	9	1	0
Zoo	12	12	0	0

Table 3. Automatic conversion of the textual descriptions into BeePath text.

Process	Tot frag. covered	Frag. covered not covered	Frag. Avg time (sec.)
Court Hearing	6	4	2 3.62
Credit Rating	7	5	2 4.97
Dispatch	7	4	3 4.01
Web Frontend	10	10	0 5.90
Zoo	12	8	4 5.56

6 Evaluation

This section presents the evaluation of the BeePath framework. Following the stages shown in Fig. 1, we assess: the expressiveness of the constrained natural language fragments; the parser that converts constrained natural language into formal process models; and the LLM that translates natural language into constrained natural language. We also report the results of a pilot study with a domain expert.

6.1 Synthetic Experiments

The first evaluation assesses whether the proposed fragments can represent business processes beyond the examples used in this paper. We used process descriptions and BPMN models from Model Judge [6], a small but carefully curated dataset. Treating the BPMN models as ground truth, we modeled each process in BeePath using the fragments from Table 1 and, when necessary, explored possible workarounds for unsupported constructs.

As shown in Table 2, out of the five business processes analyzed, two were fully covered, two had one fragment not explicitly covered but with an available workaround, and one process had two uncovered fragments, both of which also have applicable workarounds.

The second evaluation assessed the alignment between the generated models and the ground truth. Using the same five Model Judge processes, we compared the Petri nets and BPMN models produced by BeePath against the reference BPMN solutions.⁷ All observed discrepancies correspond to fragments identified in the first evaluation (Tab. 2) that are only supported through workarounds.

The final evaluation assessed the LLM’s ability to convert process descriptions into BeePath descriptions and measured generation time. For each of the five Model Judge descriptions, we counted the fragments correctly identified with respect to the manually created BeePath specification from the first evaluation. Minor syntactic differences and small spelling variations in activity names were ignored. Since LLM outputs are nondeterministic, each conversion was repeated

⁷ Petri nets (from BeePath), textual conversions, and BPMN models (from Model Judge) at <https://github.com/processintelligence/beepath/blob/main/paper-material/conversion-against-model-judge.pdf>

10 times and the best result was retained to estimate the model’s potential performance rather than the effects of prompt engineering. The results are reported in Table 3.⁷ All experiments were performed using ChatGPT4o. Most LLM-generated conversions contained minor syntax errors that occasionally affected parsing, but these were generally easy for users to correct. For simpler processes, results were consistent across runs and closely matched the reference solutions, whereas more complex processes exhibited greater variability. Table 3 also reports the average conversion time over 10 runs. As expected, execution time generally increased with the length of the process description (Court Hearing: 446 chars; Credit Rating: 568; Dispatch: 603; Web Frontend: 911; Zoo: 1124).

6.2 Pilot Study on a Real-World Application

In addition to the synthetic experiments, we conducted a pilot study with an information security consultant at the research institute of one of the authors. The consultant used BeePath to model the high-level risk management process described in ISO 27001 [12, chap. 6]. Their group consists of information security experts with no formal process modeling background and is interested in tools that simplify process modeling across departments.

After a brief introduction to the tool and its modeling fragments, the consultant provided a natural language description of the process and iteratively refined the LLM-generated constrained description.⁸ The initial output was incomplete, containing missing activities, incorrectly identified patterns, and syntax errors. Nevertheless, it provided a useful starting point. With the support of the tool’s syntax validation, the consultant refined the description until obtaining a satisfactory model comprising 11 activities and six different fragments.

In a follow-up interview, the consultant highlighted the potential value of maintaining a formal repository of process models and noted that BeePath could improve modeling efficiency by allowing domain experts to focus on process concepts rather than formal notations such as BPMN or Petri nets.

Two main limitations were identified: (1) the LLM often failed to generate a correct constrained description, and (2) effective use of the tool requires a process-oriented mindset. We acknowledge the first limitation as an important direction for future work (though the scope of the paper has never been to fine-tune the LLM or the prompt). Regarding the second, we agree that process-oriented thinking is essential; however, this requirement is not specific to BeePath, as it is equally necessary for formal modeling and for writing process descriptions that can be translated into formal representations.

7 Conclusions and Future Work

In this paper, we propose BeePath, a framework designed to assist domain experts in formalizing business processes using a constrained natural language –

⁸ Complete description available at <https://github.com/processintelligence/beepath/blob/main/paper-material/risk-management-process.md>

thus closer to a natural language description of processes. BeePath consists of a structured natural language and an automated model generator and makes use of LLMs to transform any process description into the given structured language. The structured language, which comprises a set of process fragments, aims to force the users to resolve inconsistencies present in unstructured natural language descriptions. The automated model generator provides process analysts with consistent and reliable models derived from BeePath descriptions. BeePath is a framework that can be easily extended with new process fragments to capture a broader range of process behaviors.

Future work will focus on extending the BeePath grammar to support more fragments and more target languages. Additionally, refining the LLM prompt engineering and post-processing steps could enhance output consistency and reduce syntax errors by using, for example, GrammarLLM [26]. Finally, validation with real users represents an important next step as well, necessary to validate the entire approach.

References

1. van der Aalst, W.M., ter Hofstede, A.H., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. *Distributed and Parallel Databases* **14**(1), 5–51 (2003). <https://doi.org/10.1023/A:1022883727209>
2. Alexander, C., Ishikawa, S., Silverstein, M.: *A Pattern Language*. Oxford University Press, New York, NY, USA (1977)
3. Baïna, K., El Hamlaoui, M., Kabbaj, H.: Business Process Modelling Augmented: Model Driven transformation of User Stories to Processes. In: *ACM ICPS* (2020). <https://doi.org/10.1145/3419604.3419793>
4. Bender, E.M., Gebru, T., McMillan-Major, A., Shmitchell, S.: On the Dangers of Stochastic Parrots. In: *Proc. of ACM FAccT*. ACM (2021). <https://doi.org/10.1145/3442188.3445922>
5. Burattin, A., Grama, A., Sima, A., Rivkin, A., Weber, B.: A framework for processing textual descriptions of business processes using a constrained language - technical report. *CoRR* (2025). <https://doi.org/10.48550/ARXIV.2508.15799>
6. Delicado, L., Sanchez-Ferreres, J., Carmona, J., Pardo, L.: The Model Judge - A Tool for Supporting Novices in Learning Process Modeling. In: *BPM 2018 Demonstration Track* (2018)
7. Di Ciccio, C., Montali, M.: Declarative Process Specifications: Reasoning, Discovery, Monitoring. In: *Process Mining Handbook*. Springer (2022). https://doi.org/10.1007/978-3-031-08848-3_4
8. Friedrich, F.: *Automated Generation of Business Process Models from Natural Language Input*. Master's thesis, Humboldt Universität zu Berlin (2010)
9. Giannakopoulou, D., Pressburger, T., Mavridou, A., Schumann, J.: Automated formalization of structured natural language requirements. *Inf. Softw. Technol.* **137**, 106590 (2021). <https://doi.org/10.1016/J.INFSOF.2021.106590>
10. Hildebrandt, T.T., Mukkamala, R.R.: Declarative Event-Based Workflow as Distributed Dynamic Condition Response Graphs (10 2011). <https://doi.org/10.4204/EPTCS.69.5>
11. Honkisz, K., Kluza, K., Wiśniewski, P.: A Concept for Generating Business Process Models from Natural Language Description. pp. 91–103 (2018). https://doi.org/10.1007/978-3-319-99365-2_{_}8

12. International Standards Organization International - Electrotechnical Commission: ISO/IEC 27001. Tech. rep. (2022)
13. Ivanchikj, A., Serbout, S., Pautasso, C.: From text to visual BPMN process models: Design and evaluation. In: Proceedings of MODELS (2020). <https://doi.org/10.1145/3365438.3410990>
14. Ivanchikj, A., Serbout, S., Pautasso, C.: Live process modeling with the BPMN Sketch Miner. *Software and Systems Modeling* **21**(5), 1877–1906 (10 2022). <https://doi.org/10.1007/s10270-022-01009-w>
15. Lanz, A., Weber, B., Reichert, M.: Time patterns for process-aware information systems. *Requirements Engineering* **19**(2), 113–141 (6 2014). <https://doi.org/10.1007/s00766-012-0162-3>
16. Leopold, H., Mendling, J., Polyvyanyy, A.: Generating natural language texts from business process models. In: Proceedings of CAiSE 2012 (2012). https://doi.org/10.1007/978-3-642-31095-9_5
17. Leopold, H., Mendling, J., Polyvyanyy, A.: Supporting process model validation through natural language generation. *IEEE Transactions on Software Engineering* **40**(8) (2014). <https://doi.org/10.1109/TSE.2014.2327044>
18. Licardo, J.T., Tanković, N., Etinger, D.: A Method for Extracting BPMN Models from Textual Descriptions Using Natural Language Processing. *Procedia Computer Science* **239**, 483–490 (2024). <https://doi.org/10.1016/j.procs.2024.06.196>
19. López, H.A., Debois, S., Hildebrandt, T.T., Marquard, M.: The process highlighter: From texts to declarative processes and back. In: CEUR-WS (2018)
20. Murata, T.: Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* **77**(4), 541–580 (1989)
21. Nagel, S., Amann, E., Delfmann, P.: declare-js: A web-based viewer and editor for declarative process models. In: ICPM Doctoral Consortium/Demo (2023)
22. Nasiri, S., Adadi, A., Lahmer, M.: Automatic generation of business process models from user stories. *International Journal of Electrical and Computer Engineering* **13**(1) (2023). <https://doi.org/10.11591/ijece.v13i1.pp809-822>
23. Pešić, M., Schonenberg, H., van der Aalst, W.M.: DECLARE: Full Support for Loosely-Structured Processes. In: Proceedings of EDOC. pp. 287–298. IEEE (2007). <https://doi.org/10.1109/EDOC.2007.14>
24. Sánchez-Ferreres, J., Burattin, A., Carmona, J., Montali, M., Padró, L.: Formal Reasoning on Natural Language Descriptions of Processes. In: In Proceedings of BPM 2019 (2019). https://doi.org/10.1007/978-3-030-26619-6_8
25. Sholiq, S., Sarno, R., Astuti, E.S.: Generating BPMN diagram from textual requirements. *Journal of King Saud University - Computer and Information Sciences* **34**(10) (2022). <https://doi.org/10.1016/j.jksuci.2022.10.007>
26. Tuccio, G., Bulla, L., Madonia, M., Gangemi, A., Mongiovì, M.: GRAMMAR-LLM: Grammar-Constrained Natural Language Generation. In: Findings of ACL (2025). <https://doi.org/10.18653/v1/2025.findings-acl.177>