Debugging Interactions between LLMs and Users in Conversational Process Modeling

Christian Horne[1,*], Matthias Ehrendorfer[1], Nataliia Klievtsova[1], Juergen Mangler[1] and Stefanie Rinderle-Ma[1]

[1]*Technical University of Munich, TUM School of Computation, Information and Technology, Garching, Germany*

Abstract

We demonstrate the *CPEE LLM Error Debugging Console*, an open-source web tool that renders the opaque AutoBPMN.AI pipeline transparent. AutoBPMN.AI enables conversational process modeling in the CPEE via a multi-format pipeline where syntax, conversion, and structural errors remain hidden, traceable only through verbose log files of interactions with the LLM agents. The console parses these conversational logs and exposes every intermediate representation step-by-step, contributing visibility into each transformation stage, cross-format traceability linking elements across representations, and structural verification via trace-based analysis. A longitudinal evaluation of 600 instances identified 1,637 errors; a user study with experts showed error identification accuracy rising from 13% (with raw log inspection) to 80% (with console use).

1. Introduction

AutoBPMN.AI [1] enables domain experts to create and redesign executable process models through natural-language conversation with the Cloud Process Execution Engine (CPEE) [1], lowering the barrier to process modeling for non-technical users. Each edit follows a three-stage pipeline: the executable CPEE-Tree [2] is translated into a Mermaid[2] intermediate, which the LLM then refines according to the user's instructions and finally the updated Mermaid is converted back to an executable CPEE-Tree (cf. "AutoBPMN.AI LLM service" box in Fig. 1). AutoBPMN.AI exemplifies the intermediate-representation strategy of LLM-based process model generation (cf. Tab. 1) by using a lightweight intermediate format rather than generating executable models directly or using multi-agent orchestration.

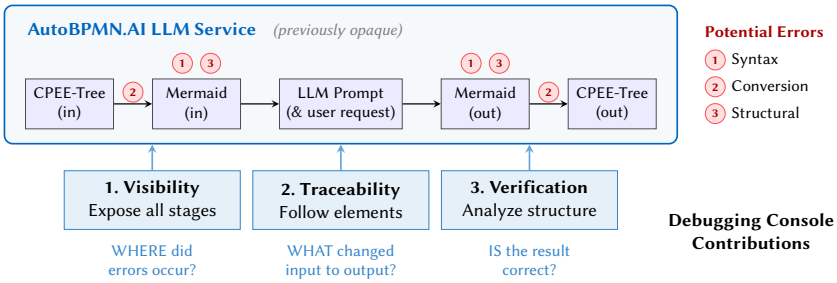


Figure 1: The opaque AutoBPMN.AI pipeline (top), with markers locating syntax (1), conversion (2), and structural (3) errors, and the three console contributions that render it transparent (bottom).

When using the intermediate representation strategy errors can arise at any stage in the multi-step pipeline, and since only the final result is shown to the user, intermediate failures remain hidden.

Proceedings of the BPM 2026 Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Forum (BPM 2026) , Toronto, Canada, September 27th to October 2nd, 2026.

*Corresponding author.

christian.horne@tum.de (C. Horne); matthias.ehrendorfer@tum.de (M. Ehrendorfer); nataliia.klievtsova@tum.de (N. Klievtsova); juergen.mangler@tum.de (J. Mangler); stefanie.rinderle-ma@tum.de (S. Rinderle-Ma)

0009-0003-1248-2684 (C. Horne); 0000-0002-7739-9123 (M. Ehrendorfer); 0009-0009-9010-2855 (N. Klievtsova); 0000-0002-6332-5801 (J. Mangler); 0000-0001-5656-6108 (S. Rinderle-Ma)

[1]https://cpee.org/, accessed: 2026-06-24

[2]mermaid.js, e.g., https://mermaid.js.org/, accessed: 2026-06-24

However, in order to observe the process of process modeling as described in [3] visualizing internal transformation steps is necessary when modeling with the help of an LLM agent to uncover at which point in the pipeline errors occur. The internal steps can be made transparent based on the recorded conversation with the LLM agent as well as additional information about the internal (i.e., Mermaid) and target (i.e., CPEE-Tree) representation of the created models. This is done based on the following information contained in the conversational log for each performed conversational process modification (possibly consisting of multiple atomic process model changes) following an interaction with the LLM: (1) textual user input, (2) selected LLM model, (3+4) input process model in target and intermediate representation (i.e., CPEE-Tree (in) and Mermaid (in) in Fig. 1) (4+5) output process model in target and intermediate representation (i.e., Mermaid (out) and CPEE-Tree (out) in Fig. 1).

During a conversational process modification three error classes can arise: *syntax errors* (1), where formal grammar of Mermaid flowchart syntax is violated; *conversion errors* (2), where the translation between CPEE-Tree and Mermaid intermediate introduces unwanted behavioral discrepancies; and *structural errors* (3), where a syntactically valid model violates structural properties like soundness or reachability (e.g. unreachable tasks, dead-end tasks). Existing transparency tools target generic LLM agents, such as VISTA [4] for multi-turn reasoning and XAgen [5] for multi-agent workflows.

Table 1
Strategies for LLM-based process model generation. AutoBPMN.AI (*) uses Mermaid.

Strategy	Representative Tools	Key Idea
Direct Generation	LLM4BPMNGen [6], InstruBPM [7]	LLM generates process models (e.g., BPMN XML) directly from text.
Intermediate Representation	BPMNGen [8], BPMN Assistant [9], BPMN-Chatbot [10], ProMoAI [11], Nala2BPMN [12], **AutoBPMN.AI*** [1]	LLM generates a structured intermediate format (e.g., JSON, Mermaid, POWL), which is then converted to process models.
Multi-Agent / RAG	MAO [13], BPLLM [14]	Multiple agents orchestrate generation and refinement, or retrieval augments the LLM with existing process knowledge.

We demonstrate the web-based open-source *CPEE LLM Error Debugging Console*, that makes the opaque AutoBPMN.AI pipeline transparent and debuggable. Based on conversational logs created with AutoBPMN.AI, pipeline internal intermediate representations are exposed through a step-by-step interface, addressing the transparency problem along three main contributions (cf. Fig. 1). *Visibility* exposes all transformation stages and makes them inspectable in different view modes, letting the user see *where* potential errors occur. *Traceability* links corresponding tasks and gateways across all representations, revealing *what* exactly changed, and allows to inspect behavioral differences via trace enumeration. *Verification* offers a trace-based approximation of structural soundness, boundedness and reachability, confirming *whether* the graphs are actually correct. To complete these goals, the *CPEE LLM Error Debugging Console* performs ex-post analysis based on conversational logs to identify potential errors but does not allow to correct these errors.

Section 2 describes the *CPEE LLM Error Debugging Console* capabilities along the three error classes, Sect. 3 reports evidence of maturity, and Sect. 4 provides the tool itself and a video of its usage.

2. Demonstration Scenarios

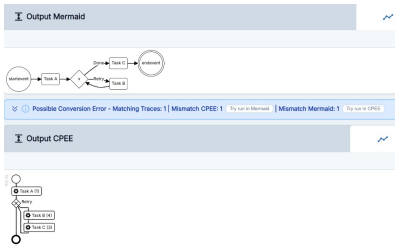
We demonstrate the debugging console using a single artificial AutoBPMN.AI instance that exhibits all three distinct error classes: (1) a **Syntax Error** after the first conversational process modification, (2) a **Conversion Error** after the second one, and (3) a **Structural Error** after the third one.

For each error, the console exposes the complete transformation pipeline (input CPEE-Tree, input Mermaid, textual user input, output Mermaid, output CPEE-Tree) enabling users to identify exactly where the errors occur. In Table 2, screenshots from the tool as well as interpretations what went wrong, are shown on the left while a simplified and shortened conversational log is shown for three consecutive conversational process modifications showcasing all three error types.

Table 2
Errors, their interpretation, and the corresponding conversational logs

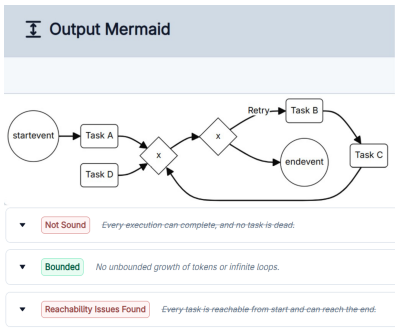
After the first conversational process modification the LLM generates a task node with an empty label. The console reports a **Syntax Error** on line 3 of the Mermaid output, showing the offending line. The parser expected label content (STR, TEXT, etc.) but received a closing parenthesis.

```
---
event:
 cpee:exposition: |-
  # User Input:
  Add two sequential tasks, A and B.
 time:timestamp: '2026-06-02T20:37:36+02:00'
---
event:
 cpee:exposition: |-
  # Used LLM:
  gemini-2.5-flash-lite
---
event:
 cpee:exposition: |-
  %% Output Intermediate
  graph LR
  0:startevent:((startevent)) --> 1:task:(Task A)
  1:task:(Task A) --> 2:task:()
  2:task:(Task B) --> 3:endevent:(((endevent)))
---
event:
 cpee:exposition: |-
  <!-- Output CPEE-Tree -->
  <description>
   <call id="a1"><label>Task A</label></call>
   <call id="a3"><label></label></call>
  </description>
 time:timestamp: '2026-06-02T20:37:36+02:00'
```



After the second conversational process modification the Mermaid output places Task C on the loop-exit path after the gateway, while the converted CPEE-Tree nests Task B then Task C inside the loop body. The console exposes this behavioral discrepancy through trace comparison. Since the computed trace sets of output Mermaid and output CPEE-Tree are different, the console flags a **Conversion Error**.

```
---
[... further events ...]
event:
 cpee:exposition: |-
  # User Input:
  Task B should be enclosed in a loop.
  On exit execute task C.
 time:timestamp: '2026-06-02T20:37:53+02:00'
---
event:
 cpee:exposition: |-
  %% Output Intermediate
  flowchart LR
  0:startevent:((startevent)) --> 1:task:(Task A)
  1:task:(Task A) --> 2:exclusivegateway:{x}
  2:exclusivegateway:{x} --> |Done| 3:task:(Task C)
  2:exclusivegateway:{x} --> |Retry| 4:task:(Task B)
  4:task:(Task B) --> 2:exclusivegateway:{x}
  3:task:(Task C) --> 5:endevent:(((endevent)))
---
event:
 cpee:exposition: |-
  <!-- Output CPEE-Tree -->
  <description>
   <call id="a1"><label>Task A</label></call>
   <loop condition="Retry">
    <call id="a7"><label>Task B</label></call>
    <call id="a5"><label>Task C</label></call>
   </loop>
  </description>
 time:timestamp: '2026-06-02T20:37:53+02:00'
```



After the third conversational process modification the LLM creates a structurally invalid model. The console surfaces a **Structural Error** and flags the model as Not Sound with reachability issues: Task D is an unreachable task, meaning that it cannot be reached from the start node and can reach the end node(s), as indicated in the structural analysis panel.

```
---
[... further events ...]
event:
 cpee:exposition: |-
  # User Input:
  Add Task D in an exclusive choice
  with Task A.
 time:timestamp: '2026-06-02T20:38:14+02:00'
---
event:
 cpee:exposition: |-
  %% Output Intermediate
  flowchart LR
  se:startevent:((startevent))-->a1:task:(Task A)
  a1:task:(Task A)-->gw1s:exclusivegateway:{x}
  a8:task:(Task D)-->gw1s:exclusivegateway:{x}
  gw1s:exclusivegateway:{x}-->gw1e:exclusivegateway:{x}
  gw1e:exclusivegateway:{x}-->|"Retry"|a7:task:(Task B)
  a7:task:(Task B)-->a5:task:(Task C)
  a5:task:(Task C)-->gw1s:exclusivegateway:{x}
  gw1e:exclusivegateway:{x}-->ee:endevent:((endevent))
---
event:
 cpee:exposition: |-
  <!-- Output CPEE-Tree -->
  <description>
   <call id="a1"><label>Task A</label></call>
   <loop condition="Retry">
    <call id="a7"><label>Task B</label></call>
    <call id="a5"><label>Task C</label></call>
   </loop>
  </description>
 time:timestamp: '2026-06-02T20:38:14+02:00'
```

2.1. System Capabilities

Each of the three error classes above is caught by one of the console's contributions (cf. Fig. 1), which map onto concrete system capabilities and each answer one diagnostic question.

Visibility. The console exposes every internal stage of the otherwise opaque pipeline. Step-by-step navigation through the conversational modification history reconstructs each transformation, and for every step the five intermediate representations (input CPEE-Tree, input Mermaid, user prompt, output Mermaid, output CPEE-Tree) are rendered side by side. Each representation supports multiple view

modes (interactive graph, syntax-highlighted source, and raw log), and a preprocessing pass repairs malformed LLM-generated Mermaid so that it stays renderable. This is what locates the **Syntax Error** after the first interaction with the LLM, showing the exact line and stage where it occurs.
Traceability. A task-mapping mechanism links equivalent tasks and gateways across all representations, so that selecting an element highlights its counterparts in every other view, letting users follow an element through the conversion. On top of this, a Graph Trace Analysis algorithm computes all possible traces of a graph and compares them, both between input and output to reveal behavioral changes and between the CPEE-Tree and Mermaid formats to detect conversion discrepancies. The diverging trace sets are exactly what flag the **Conversion Error** after the second interaction with the LLM.
Verification. Building on the computed traces, the console performs structural verification on every graph, checking workflow properties such as soundness, boundedness, and reachability. Property violations, such as the unreachable Task D after the third interaction with the LLM, are reported in a dedicated view mode that flags the model as containing a **Structural Error**. Verification is a trace-based approximation, not a formal WF-net proof: exploration is bounded by loop-iteration limits and timeouts, so boundedness holds by construction and soundness may miss violations outside the explored traces.
Impact. The console was evaluated through a longitudinal error analysis and a comparative user study. Applying it to 600 CPEE instances each representing a created process model with multiple conversational process modifications (563,163 exposition log lines) catalogued 1,637 errors, distributed as 838 syntax (51.2%), 561 conversion (34.3%), and 238 structural (14.5%) errors, with at least one error in 315 instances (52.5%): evidence that all three error classes occur frequently in practice. In a within-subjects study, five experts diagnosed one instance per error class, first via the raw log, then via the console, answering diagnostic questions, such as which error occurs where. Accuracy of error identification rose from 13% with raw log inspection to 80% with the console, while the mean perceived ease of use improved by 2.9 points on a five-point Likert scale, confirming that exposing, tracing, and verifying the pipeline turns errors that are opaque in the logs into ones that users can localize.

3. Maturity

Scientific Maturity. The concepts used for development of the *CPEE LLM Error Debugging Console* are based on conversational process modeling [15], as realized by AutoBPMN.AI [1]. The conversational logs generated during modeling were used for the conceptualization and implementation of the *CPEE LLM Error Debugging Console* as part of a bachelor's thesis. As described in Sect. 2.1 the *CPEE LLM Error Debugging Console* has been evaluated using (i) a longitudinal error analysis showcasing the occurrence of different error types and (ii) an initial comparative user study was conducted which indicates that error identification is easier using the *CPEE LLM Error Debugging Console*.
AutoBPMN.AI Maturity. The AutoBPMN.AI tool implements conversational process modeling [15] by letting users describe processes either from scratch or by describing desired adaptations and creating a process model (CPEE-Tree). AutoBPMN.AI has been used in multiple university courses.
CPEE Maturity. The cloud process execution engine (CPEE) which is the target system for the created models is an open source engine. The engine and its connected components have been downloaded 1,865,306 [3] times. The orchestration framework centurio.work which has been built on top of the CPEE currently realizes 16 different processes at 7 manufacturing companies [16].
Maturity of Debugging of LLM-based Process Modeling. Several tools for creating process models with the help of LLMs exist (cf. Fig. 1). However, only generic LLM agent transparency tools are available so far. The *CPEE LLM Error Debugging Console* presents a tool for debugging AutoBPMN.AI generated process models. However, the underlying concept can be transferred to other modeling pipelines using intermediate representations (cf. Fig. 1) as long as (1) the conversation log is available in sufficient detail, (2) it is easy to visualize final and intermediate formats (i.e., possible for Mermaid, POWL, ...), (3) final and intermediate formats are graph- or tree-based so that trace-based analysis is possible, and (4) tasks and gateways can be traced throughout the whole pipeline (e.g., by an ID).

[3] https://rubygems.org/profiles/eTM, accessed: 2026-06-24

4. Videos and Tool

CPEE LLM Error Debugging Console is hosted on autobpmn.ai where instances from the AutoBPMN.AI Modeller can be analyzed (cf. Sect. 2.1). Code and set-up instructions are available on github [4]. A demo video is available at https://autobpmn.ai/debugging-console.webm (accessed: 2026-07-01).

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] N. Klievtsova, M. Ehrendorfer, J. Mangler, S. Rinderle-Ma, AutoBPMN.AI: Conversational process modeling and automation, in: Proceedings of the Demonstration Track, BPM 2025, volume 4032 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025, pp. 304–311.

[2] J. Mangler, S. Rinderle-Ma, Cloud process execution engine: Architecture and interfaces, 2022. URL: https://arxiv.org/abs/2208.12214. arXiv:2208.12214.

[3] J. Claes, I. Vanderfeesten, J. Pinggera, H. A. Reijers, B. Weber, G. Poels, A visual analysis of the process of process modeling, Inf. Syst. E Bus. Manag. 13 (2015) 147–190.

[4] Y. Zhang, M. Lin, M. Dras, U. Naseem, Beyond the black box: Demystifying multi-turn LLM reasoning with VISTA, 2025. URL: https://arxiv.org/abs/2511.10182. arXiv:2511.10182.

[5] X. Wang, M. Yin, E. Koh, M. D. Dogan, XAgen: An explainability tool for identifying and correcting failures in multi-agent workflows, 2025. URL: https://arxiv.org/abs/2512.17896. arXiv:2512.17896.

[6] A. Costa, A. Wimmer, L. Pufahl, Llm4bpmngen: A tool for BPMN generation with llms, in: Companion Proceedings of ER 2025, volume 4099 of *CEUR Workshop Proceedings*, 2025, pp. 333–337.

[7] G. Çelikmasat, A. Özgövde, F. B. Aydemir, Instruction-tuning open-weight language models for bpmn model generation, 2025. URL: https://arxiv.org/abs/2512.12063. arXiv:2512.12063.

[8] L. F. Hörner, Introducing bpmngen: An llm-based conversational framework for bpmn 2.0 process model generation, in: EMISA 2025, Gesellschaft für Informatik e.V., Bonn, 2025, pp. 17–31.

[9] J. T. Licardo, N. Tankovic, D. Etinger, Bpmn assistant: An llm-based approach to business process modeling, 2025. URL: https://arxiv.org/abs/2509.24592. arXiv:2509.24592.

[10] A. Safan, J. Köpke, Bpmn-chatbot++: Llm-based modeling of collaboration diagrams with data, in: Proceedings of the BPM, 2025.

[11] H. Kourani, A. Berti, D. Schuster, W. M. P. van der Aalst, Promoai: Process modeling with generative ai, arXiv preprint arXiv:2403.04327 (2024). doi:10.48550/arXiv.2403.04327.

[12] A. Nour Eldin, N. Assy, O. Anesini, B. Dalmas, W. Gaaloul, Nala2bpmn: Automating bpmn model generation with large language models, in: M. Comuzzi, D. Grigori, M. Sellami, Z. Zhou (Eds.), Cooperative Information Systems, Springer Nature Switzerland, Cham, 2025, pp. 398–404.

[13] L. Lin, Y. Jin, Y. Zhou, W. Chen, C. Qian, Mao: A framework for process model generation with multi-agent orchestration, IEEE Transactions on Services Computing (2025) 1–14.

[14] M. L. Bernardi, A. Casciani, M. Cimitile, A. Marrella, Conversing with business process-aware large language models: the bpllm framework, Journal of Intelligent Information Systems 62 (2024) 1607–1629.

[15] N. Klievtsova, J. Benzin, T. Kampik, J. Mangler, S. Rinderle-Ma, Conversational process modelling: State of the art, applications, and implications in practice, in: Business Process Management Forum, 2023, pp. 319–336.

[16] S. Rinderle-Ma, J. Mangler, Process automation and process mining in manufacturing, in: International conference on business process management, Springer, 2021, pp. 3–14.

[4] https://github.com/Spunsel/cpee-log-error-console, accessed: 2026-06-24