universität
uulm

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Enhancing BPMNGen: Improving LLM-based BPMN 2.0 Process Model Generation through Natural Language Processing

Bachelor thesis at Ulm University.
Submitted for the degree of Bachelor of Science (B.Sc) in Informatik.

Submitted by:
Zhe Shi
zhe.shi@uni-ulm.de
Matriculation number:1036630

Thesis advisor:
Prof. Dr. Manfred Reichert

Supervisor:
Luca Hörner

Ulm, April 2025

Acknowledgements

I would like to express my sincere gratitude to my thesis advisor, Prof. Dr. Manfred Reichert, for his valuable guidance. His expertise in business process management and information systems provided me with the direction needed to tackle this challenging topic.

My deepest appreciation goes to my supervisor, Luca Hörner, whose continuous support, constructive suggestions, and dedication made this work possible. Her expertise in AI applications and willingness to engage in detailed technical discussions significantly shaped this research.

I am grateful to the entire team at the Institut für Datenbanken und Informationssysteme for creating a supportive academic environment and providing access to the necessary resources and infrastructure.

Finally, I would like to thank my family and friends for their unwavering support, patience, and encouragement throughout my academic journey. Their belief in my abilities has been a constant source of motivation.

Abstract

This thesis explores the enhancement of an AI-powered chatbot that transforms natural language descriptions into Business Process Model and Notation (BPMN 2.0) diagrams. Building upon a previous prototype that utilized ChatGPT's capabilities, we addressed limitations to improve both diagram quality and overall implementation. Our approach includes a validation framework verifying component properties, connections, spatial organization, and process integrity; specialized event type handling; and improved flow rendering algorithms. The user interface features intuitive input handling and better diagram organization. Comparative analysis demonstrates significant improvements in accuracy, readability, and standards compliance while maintaining good performance. The research shows how targeted improvements to AI-driven solutions can bridge the gap between natural language descriptions and technical diagram requirements, making business process modeling more accessible to domain experts without specialized knowledge. The results highlight the growing capabilities of large language models in technical domains and the promising future of natural language processing for professional BPMN 2.0 diagram generation.

Version October 30, 2025

Set: PDF-LaTeX 2$_\varepsilon$

Contents

1 Introduction

1.1 Motivation

In the modern business environment, the ability to document, analyze, and optimize business processes is crucial for maintaining competitive advantage. Business Process Model and Notation (BPMN 2.0) [1] has become the standard for visualizing these processes, providing a clear and standardized method for describing workflows. However, creating accurate and standards-compliant BPMN 2.0 diagrams requires specialized technical knowledge that many domain experts lack.

With recent progress in artificial intelligence, particularly Large Language Models (LLMs) [2], natural language descriptions can now be analyzed and transformed into structured BPMN 2.0 components. This technological advancement has the potential to make process modeling accessible to those without specialized expertise.

An initial chatbot prototype developed before this project began demonstrated the basic feasibility of AI-assisted BPMN 2.0 generation. However, this early implementation faced significant challenges including diagram accuracy, BPMN 2.0 standards compliance, and performance limitations with complex diagrams. Professional BPMN 2.0 modeling must adhere to established guidelines such as those proposed by Mendling et al. [3], which emphasize clarity, simplicity, and consistency in process diagrams. The existing prototype did not sufficiently address these quality aspects.

This thesis focuses on optimizing and enhancing the existing BPMN 2.0 generation chatbot powered by ChatGPT [4]. By refining the interaction between the application and the OpenAI API, implementing robust validation mechanisms, and enhancing the user interface, this project aims to transform a promising concept into a reliable tool for professional use. The enhanced chatbot will enable domain experts to generate professional-quality BPMN 2.0 diagrams without specialized technical training, potentially transforming how organizations document and optimize their business processes.

1.2 Goals

Specifically, this work aims to address critical limitations in the current chatbot by implementing robust validation mechanisms that ensure compliance with BPMN 2.0 standards, developing algorithms like the TurnPoint method to properly structure diagram flows, and optimizing chatbot performance for handling complex diagrams. The refined chatbot will maintain proper element positioning within pools and lanes, prevent crossing lines, validate connection completeness, and enforce correct flow direction.

Additionally, this thesis seeks to enhance the user experience by resolving existing bugs, improving interface elements, and streamlining the process through which users can view, create, and edit diagrams. By improving both the AI instructions and the technical implementation, this project will deliver a reliable tool that enables domain experts to generate professional-quality BPMN 2.0 diagrams without specialized technical training, effectively bridging the gap between natural language process descriptions and standardized BPMN 2.0 models.

1.3 Thesis Structure

The thesis structure is as follows. First, we will introduce the fundamentals related to functionality enhancements, including the BPMN 2.0 coordinate, newly added BPMN 2.0 component types, and the meaning of AI assistant debugging parameters such as token size. Then, we will define the requirements that are critical for project development. After briefly introducing the areas the chatbot should improve, we examine the enhancements in prompt design, which is a core aspect of development that affects the quality of generated diagrams. Here, we discuss development strategies driven by ChatGPT's capabilities and functionalities that need code implementation due to ChatGPT's limitations. Subsequently, the architecture chapter will discuss the new structures added to the chatbot. The next chapter further explains these chatbot components, demonstrating the implementation process. Finally, we review some results produced by the chatbot and discuss the project's limitations and future work.

2 Fundamentals

2.1 Business Process Model and Notation (BPMN 2.0)

Business Process Model and Notation (BPMN 2.0) is a standardized graphical language used to document business processes in a way that is easily understood by both business users and technical developers. Similar to how an architect uses blueprints to represent buildings, business analysts use BPMN 2.0 to create visual representations of organizational workflows and procedures.

BPMN 2.0 diagrams show the sequence of business activities from start to finish, including what happens, when it happens, and who performs each task. For example, a customer order process might start with receiving an order, proceed through inventory checking, payment processing, and end with shipping the product.

The BPMN 2.0 standard includes several basic components:

- **Events:** Circle-shaped elements representing something that happens, such as a start event, end event, or timer event
- **Activities:** Rectangle-shaped elements showing tasks performed in the process
- **Gateways:** Diamond-shaped elements indicating decision points or branches in the process
- **Connections:** Lines connecting elements to show the flow of the process
- **Pools and Lanes:** Containers that organize activities by departments or roles

2.1.1 Flow Objects and Connecting Objects

For the further development of our BPMN 2.0 modeling tool, more event types and connecting objects will be added to the chatbot.

Flow Objects are the elementary building blocks of BPMN 2.0. They are located within the pool in their respective swimlanes and include events, activities, and gateways. New events types like message events and timer events are added to the chatbot. Connecting Objects serve to connect all elements within the pool, even across the boundaries of swimlanes.

They include sequence flows and message flows. The sequence flow connects elements within a pool with arrows to show the flow of the process. The message flow connects elements from different pools.

As shown in Figure 2.1, the diagram depicts a simple process with a start event, two tasks, two gateways and an end event.

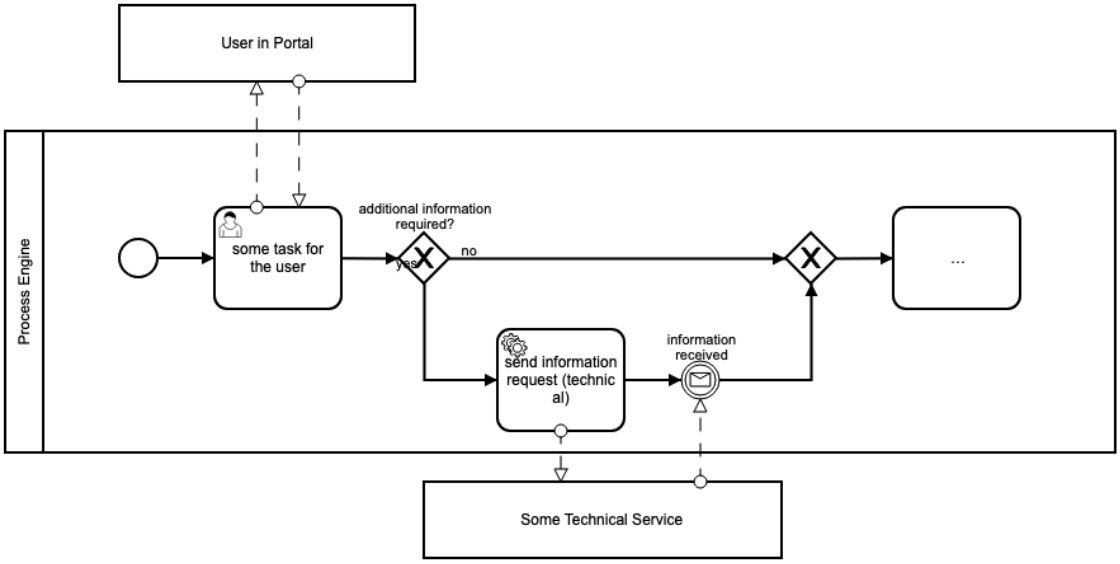

Figure 2.1: Basic BPMN 2.0 components

2.1.2 BPMN 2.0 Coordinate System in bpmn.js

Bpmn.js [5] is a JavaScript library for rendering and interacting with BPMN 2.0 diagrams in web applications. It serves as the visualization engine in our implementation, converting the XML representation of BPMN diagrams into interactive visual elements on a web page.

To properly position elements in diagrams using bpmn.js, we need to understand its coordinate system. As shown in Figure 2.2, bpmn.js uses a Cartesian grid with the origin (0,0) at the top-left corner of the diagram, X-axis extending right and Y-axis extending downward.

This coordinate system impacts four critical aspects of diagram creation:

- Element placement with precise positioning
- Connection path calculations between elements
- Containment of elements within pools and lanes
- Spacing between elements for diagram readability

For our chatbot implementation, understanding this coordinate system is essential since all JSON-defined element positions must follow these conventions to render correctly in

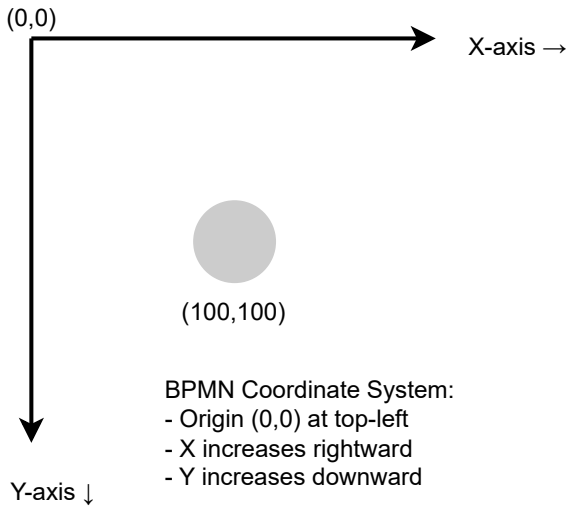

Figure 2.2: BPMN 2.0 coordinate system in bpmn.js

bpmn.js. Without proper coordinate management, elements might overlap, connections might cross inappropriately, or elements might appear outside their intended containers.

2.2 OpenAI Assistant

OpenAI's AI assistant [6] technology provides powerful natural language processing capabilities through APIs that can be integrated into various applications. To ensure efficient and error-free operation, it is crucial to manage token usage effectively and optimize file processing strategies. To enhance the AI assistant we can uploaded files to the AI assistant, which can be used for document search and retrieval. This feature is particularly useful for BPMN 2.0 modeling tools, where users can upload process documentation or guidelines to assist in creating accurate process models.But the way can manage and optimize file processing strategies in the AI assistant is important.

3 Requirements

This chapter outlines the essential requirements for the BPMN 2.0 modeling chatbot. The requirements are categorized into functional and non-functional aspects, detailing the specific features, capabilities, and quality attributes that the chatbot must possess to meet user needs and industry standards. The following sections provide a comprehensive overview of these requirements, which serve as the foundation for the chatbot's design and implementation.

3.1 Functional Requirements

Requirements FR1 through FR10 represent entirely new capabilities being introduced to the chatbot. These requirements address fundamental diagram accuracy and validation concerns not previously implemented, such as pool and lane boundary management, flow line clarity, and connection completeness. These additions form the core of the enhanced BPMN 2.0 modeling capabilities and significantly improve the quality of generated diagrams.

ID	**FR1**
Title:	Pool Boundary Management
Description:	Ensure elements are positioned correctly within pools and do not fall outside pool boundaries. The chatbot shall enforce proper containment of all BPMN 2.0 elements within their designated pools to maintain diagram structure integrity and prevent visual confusion. This includes automatic repositioning of elements that exceed pool boundaries.

ID	**FR2**
Title:	Lane Boundary Protection
Description:	Prevent elements or lines from crossing lane boundaries. The chatbot must maintain clear separation between lanes by implementing boundary protection mechanisms that ensure elements remain within their assigned lanes, preserving the organizational structure of the process diagram.

ID	**FR3**
Title:	Flow Line Clarity
Description:	Avoid lines crossing each other to improve readability. The chatbot shall implement intelligent routing algorithms to minimize line intersections and maintain clear visual paths. This includes automatic line adjustment features and suggestion of optimal connection routes between elements.

ID	**FR4**
Title:	Connection Completeness
Description:	Ensure no missing connections (for example from gateways to subsequent elements) all elements (except start and end events) must have both incoming and outgoing flows. The chatbot shall implement real-time validation checks to identify incomplete connections and provide visual indicators for missing flows, helping users maintain process continuity.

ID	**FR5**
Title:	Label Format Validation
Description:	Conduct semantic checks to enforce that activity labels follow a verb-object format in English and a norm-object format in German. The chatbot must include natural language processing capabilities to analyze label syntax and provide immediate feedback when labels don't conform to the specified format guidelines.

ID	**FR6**
Title:	Gateway Logic Verification
Description:	Verify gateway types, ensuring correct split and merge logic. The chatbot shall automatically validate gateway patterns, checking for proper matching of splits and joins, and verify that the control flow logic maintains process integrity. This includes validation of exclusive, parallel, and inclusive gateway combinations.

ID	**FR7**
Title:	Flow Direction Control
Description:	Prevent diagonal flows, only allow horizontal and vertical flows. The chatbot must enforce orthogonal routing of sequence flows, automatically adjusting flow directions to maintain a structured layout that enhances diagram readability and professional appearance.

ID	**FR8**
Title:	Event Management
Description:	Try to keep diagrams for a single start and end event. The chatbot should guide users towards creating processes with clear entry and exit points, providing warnings when multiple start or end events are introduced, while still allowing flexibility when business requirements necessitate multiple events.

ID	**FR9**
Title:	Complex Event Integration
Description:	Integrate message events and other complex events (Timer, Message events). The chatbot must support the full range of BPMN 2.0 event types, providing proper validation for their usage contexts and ensuring correct interaction patterns between different event types.

ID	**FR10**
Title:	Path Completion
Description:	Ensure all paths end at end event. The chatbot shall implement path analysis algorithms to verify that all process flows terminate properly, identifying and highlighting any incomplete or hanging paths that don't lead to an end event.

Requirements FR11 through FR15 build upon and improve features that already existed in the original chatbot implementation. These enhancements focus on optimizing the AI agent functionality, improving diagram organization, and refining the user interaction experience. Rather than introducing completely new capabilities, these requirements represent targeted improvements to existing features to make them more robust, user-friendly, and effective.

ID	**FR11**
Title:	AI Enhancement
Description:	Improve AI agent functionality by refining its instructions. The chatbot shall continuously optimize the AI component through refined instruction sets and improved learning capabilities, ensuring more accurate and context-aware responses to user inputs and diagram modifications.

ID	**FR12**
Title:	Diagram Organization
Description:	Enable sorting and grouping of diagrams by creation date. The chatbot shall provide robust organization capabilities including flexible sorting options, customizable grouping criteria, and efficient search functionality to help users manage large collections of process diagrams.

ID	FR13
Title:	Input Handling
Description:	Ensure the description submission can be triggered using the Enter key. The chatbot must implement intuitive keyboard shortcuts and input handling mechanisms that align with common user expectations, improving the overall efficiency of diagram creation and editing.

ID	FR14
Title:	Process Control
Description:	Allow users to interrupt the submission in creation and update phase process. The chatbot shall provide clear cancel/interrupt mechanisms during all operational phases, ensuring users maintain control over their modeling activities and can safely abort operations without losing work.

ID	FR15
Title:	Bug Resolution
Description:	General bug fixing: default display of latest diagram, file delete bug. The chatbot must implement robust error handling and maintain a stable operational state, including proper file management and consistent display of diagram versions.

3.2 Non-functional Requirements

The non-functional requirements (NFR1 through NFR4) address optimizations that enhance the overall quality, performance, and maintainability of the chatbot. These requirements ensure that the chatbot not only provides the necessary functionality but does so in an efficient, user-friendly, and standards-compliant manner. By meeting these non-functional requirements, the chatbot achieves a level of professional quality necessary for practical business use. The specific non-functional requirements for this chatbot are outlined below:

ID	NFR1
Title:	Usability
Description:	Ensure optimal layout with no text/component overlap and clear visual organization. The chatbot must provide an intuitive user interface that maintains proper spacing between elements, implements clear visual hierarchies, and ensures all text remains readable. Response times for user interactions should not exceed 2 seconds to maintain a fluid user experience.

ID	NFR2
Title:	Performance
Description:	Optimize token size and chunk handling for efficient processing of large diagrams. The chatbot shall maintain responsive performance even with complex diagrams, implementing efficient data structures and processing algorithms to handle large models without degradation in user experience. Load times should not exceed 3 seconds for standard diagrams.

ID	NFR3
Title:	Maintainability
Description:	Maintain clean code architecture with optimized JSON response format and minimal code redundancy. The chatbot must follow software engineering best practices, implementing modular design patterns, comprehensive documentation, and efficient data structures that facilitate future maintenance and extensions.

ID	NFR4
Title:	Standards Compliance
Description:	Ensure full compatibility with BPMN 2.0 specification. The chatbot must strictly adhere to BPMN 2.0 standards, including proper implementation of all diagram elements, attributes, and behaviors as defined in the specification. This ensures interoperability with other BPMN 2.0 tools and maintains diagram validity.

3.3 System Functionality

Before delving into the technical implementation details, it is valuable to establish a clear understanding of the chatbot's expected behavior. This section outlines the core functionality of the BPMN 2.0 modeling chatbot. The chatbot's feature set has been carefully crafted to balance modeling accuracy with user experience, ensuring that BPMN 2.0 diagrams can be created efficiently while adhering to established standards and best practices.

3.3.1 Core Features

The chatbot provides a comprehensive set of tools for creating and validating BPMN 2.0 diagrams. Users can create pools and lanes with automatic boundary management, add BPMN 2.0 elements through an intuitive interface, and connect elements using smart routing for sequence flows. Real-time validation feedback is provided to ensure diagram accuracy.

Intelligent flow management features are implemented to enforce horizontal and vertical flow directions, prevent line crossings through automatic path optimization, maintain proper gateway connections and logic, and ensure complete paths from start to end events. These features help maintain the integrity and clarity of the diagrams, as illustrated in Figure 3.1.

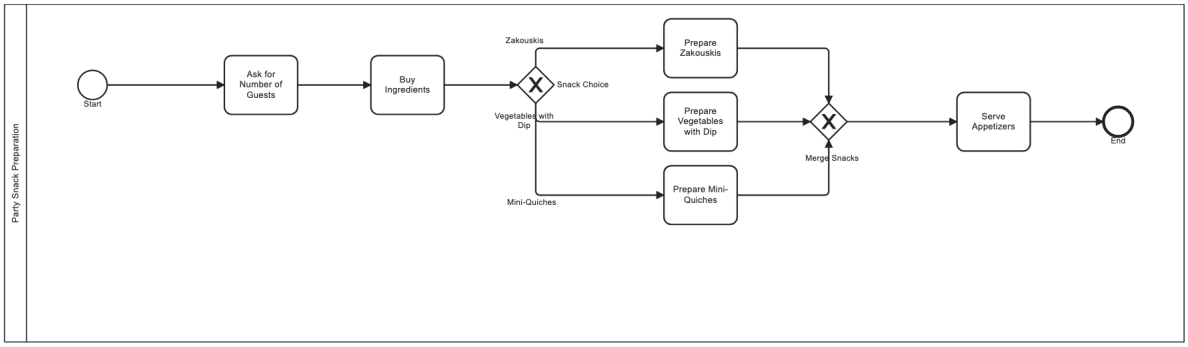

Figure 3.1: Generated result from given user instructions

The AI component enhances the modeling experience by suggesting corrections for improper element placements, validating label formats in both English and German, optimizing diagram layout for clarity, and providing context-aware recommendations. This intelligent assistance ensures that users can create accurate and well-structured BPMN 2.0 diagrams with ease.

3.3.2 User Interaction

A typical user interaction flow begins with the initiation of a new BPMN 2.0 diagram creation. The user adds a pool and defines lanes for different participants, places BPMN 2.0 elements within the lanes, and the chatbot automatically validates element placement and connections. Real-time feedback is provided for any validation issues, allowing the user to make corrections based on chatbot suggestions. The final diagram is then validated for completeness and correctness.

3.3.3 Performance and Standards

The chatbot maintains high performance standards by processing diagrams efficiently with optimized token handling, providing quick response times for user actions, ensuring strict adherence to BPMN 2.0 specifications, and maintaining clean and organized diagram layouts. This functionality implementation ensures that users can create accurate, well-structured BPMN 2.0 diagrams while benefiting from intelligent assistance and validation features.

4 Architecture

In this chapter, we introduce the architecture of the BPMN 2.0 modeling chatbot, with particular focus on recent structural enhancements.

4.1 Overview

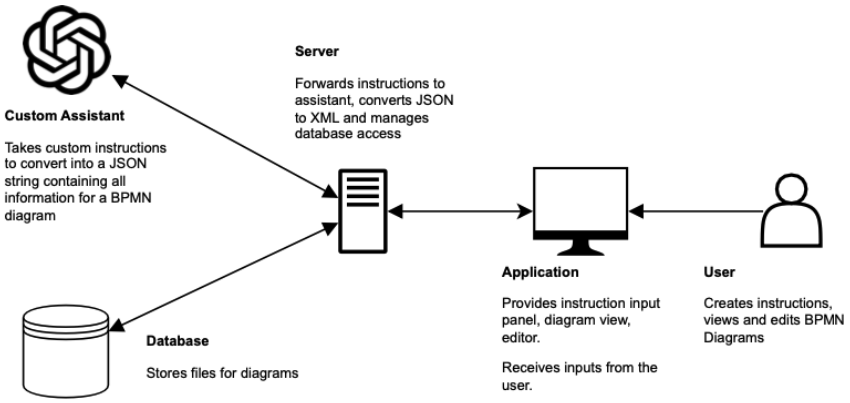

Figure 4.1: Basic system component overview

The chatbot project architecture integrates four main components that collaborate to deliver BPMN 2.0 modeling capabilities. Figure 4.1 shows the basic system component overview. The user interacts with the frontend application, which communicates with the server to retrieve and update diagram data. The server manages the database and communication with the OpenAI API to generate and update diagrams. The AI assistant processes user input and generates JSON structures that the server converts to BPMN 2.0 compliant XML files.

This project adopts a Client-Server Architecture for its scalability and ability to support concurrent users accessing shared resources. We focused on targeted enhancements to existing components, with one significant addition: the XMLValidator module on the backend server. This new component performs crucial validation and modification of generated XML files, ensuring strict compliance with BPMN 2.0 standards while preserving integrity.

4.2 Diagram Processing Workflow

The chatbot supports two fundamental operations: creating new BPMN 2.0 diagrams and modifying existing ones. While both operations have similar AI-assisted generation processes, the update workflow incorporates additional complexity through file retrieval and change management. We focus on this more complex update scenario to illustrate the chatbot's capabilities.

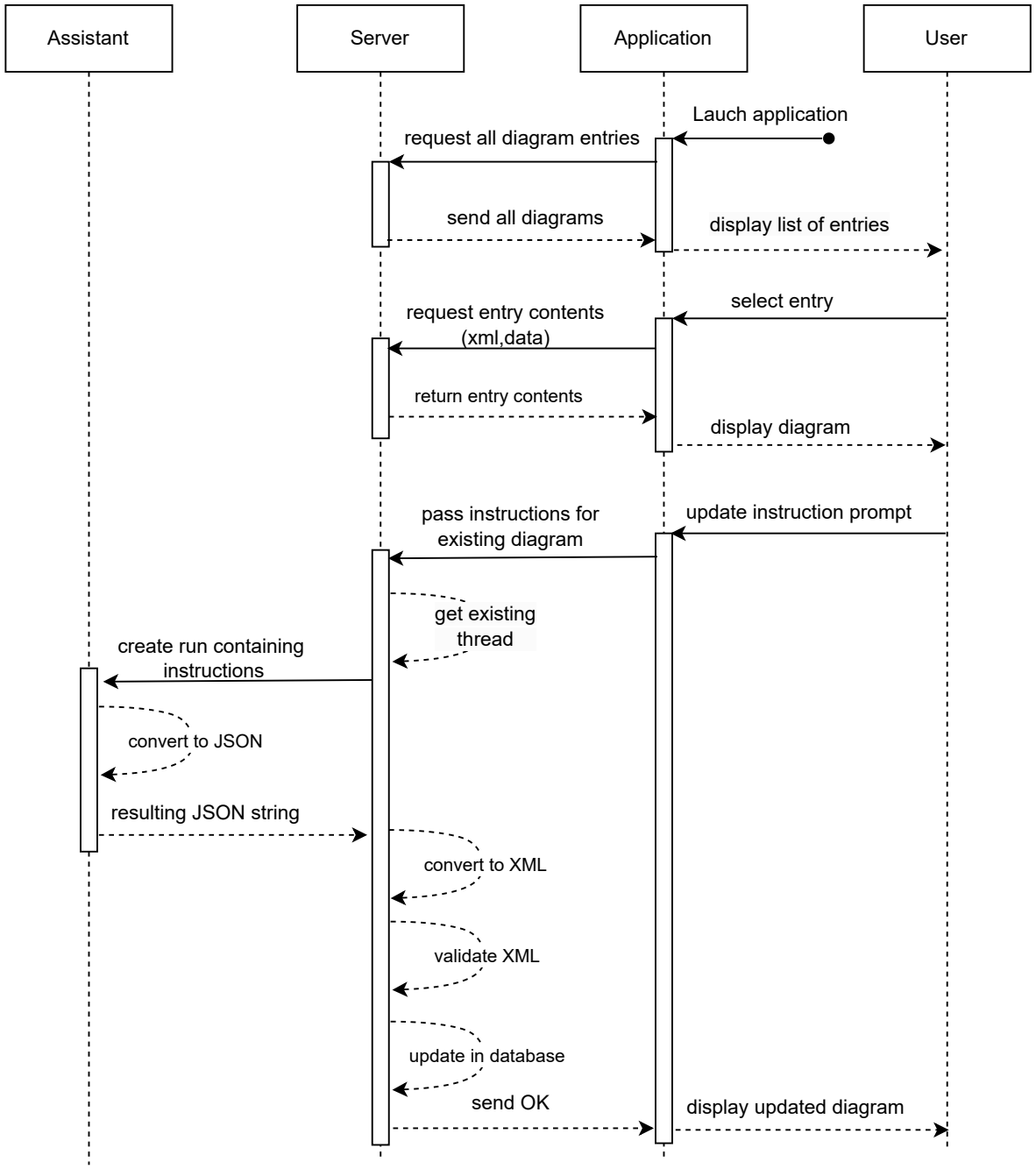

Figure 4.2: Sequence diagram for updating a diagram via prompt

Figure 4.2 illustrates the complete update workflow between user, application, server and

assistant. The process begins when a user launches the application and is presented with available diagrams. After selecting an existing diagram, the user provides modification instructions through natural language prompts. The chatbot first validates both the diagram identifier and the modification request for completeness and correctness. Upon validation, the server creates a new run with the user instructions using the associated OpenAI thread. The assistant generates a new JSON structure containing the update data, which the server then processes and converts to XML. After XML validation ensures BPMN 2.0 compliance, the database attributes are updated to match the current diagram information. The server confirms the successful update, and the application refreshes to display the updated diagram. This process can be repeated iteratively until the user achieves the desired result. If validation fails at any stage, appropriate error messages are generated while preserving the original diagram state.

4.3 Server Component Architecture

The server consists of three primary components working together to manage the chatbot's core functionality. These components are:

- OpenAI API Controller: Responsible for orchestrating communication with the OpenAI Assistant
- Conversion Manager: Processes and transforms data structures received from the assistant
- Database Manager: Handles persistence operations for the file-based database

Figure 4.3 provides an overview of the server's functional architecture. As shown, the diagram update function requires both diagram ID and content as inputs, performs validation through the XMLValidator component, and then either returns the updated diagram or appropriate error messages. The OpenAI API Controller serves as the central coordination point, initiating calls to the Conversion Manager, XMLValidator, and Database Manager as needed to process requests.

The XMLValidator component represents a significant architectural enhancement in this project. Located in the processing pipeline after JSON-to-XML conversion, this module implements a three-tiered validation approach: syntactic validation ensures well-formed XML structure, schema validation confirms adherence to the BPMN 2.0 specification, and semantic validation verifies logical consistency between diagram elements. By intercepting and correcting potential errors before they reach the database or user interface, the XMLValidator substantially improves diagram quality while maintaining backward compatibility with the existing architecture.

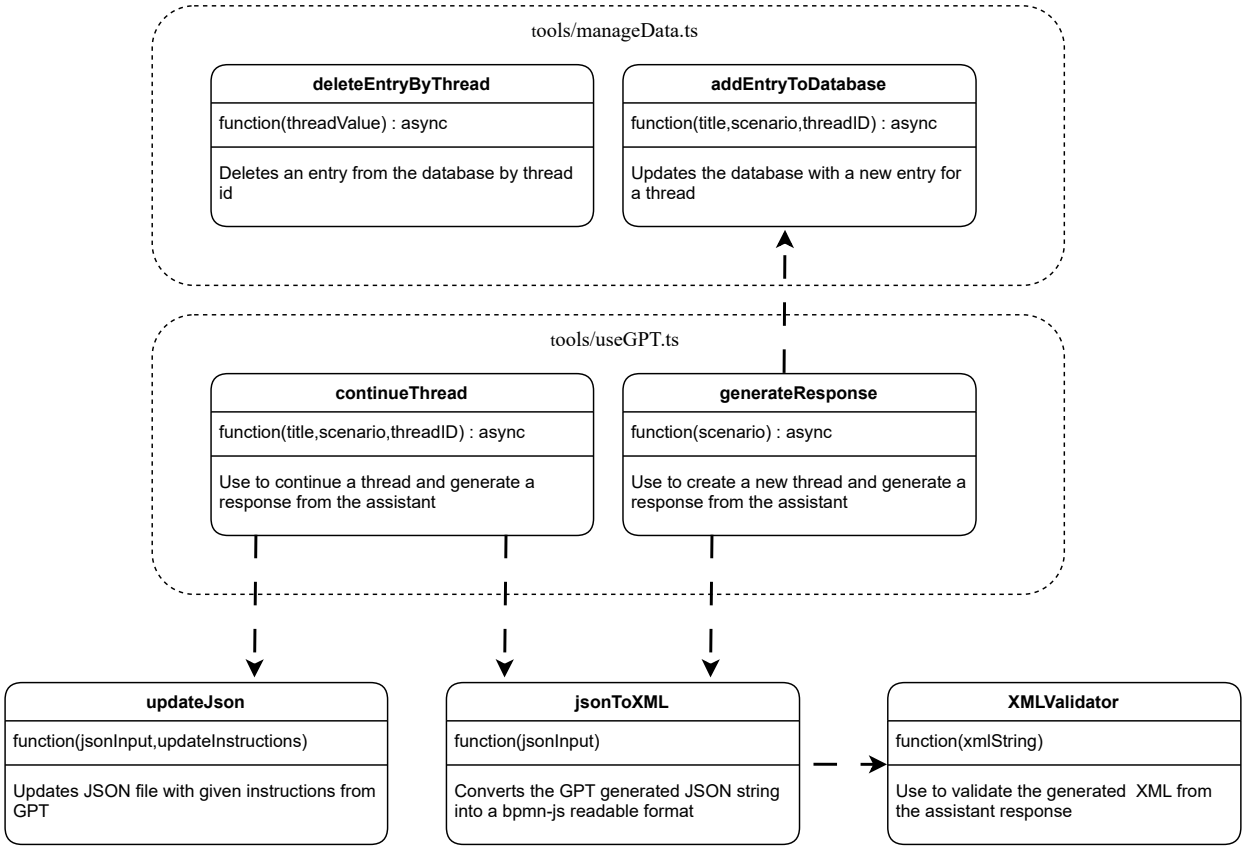

Figure 4.3: Server function overview

5 Prompt Design and Improvement

This chapter examines our approach to optimizing AI prompting for BPMN 2.0 diagram generation. We present a progression from basic prompt engineering techniques to advanced knowledge integration methods, showing how structured interactions with AI models significantly enhance diagram quality. We also address critical technical considerations including token management and search optimization that ensure reliable operation.

5.1 Prompt Analysis

Prompt engineering forms the foundation of effective BPMN 2.0 diagram generation with AI. As established in the first version of project, the quality of AI responses directly determines the success of the entire project. Creating accurate BPMN 2.0 diagrams requires precise guidance through carefully structured prompts.

Our tests with the chatbot revealed that prompt design influences diagram quality across several critical dimensions: ensuring compliance with BPMN 2.0 standards, optimizing spatial layout, maintaining connection integrity, preserving gateway logic, and supporting coherent representation of complex processes. These factors form the foundation for effective AI generated business process visualizations.

Building on the foundational capabilities identified in the initial implementation, our work now focuses on refining these capabilities for professional grade diagram generation.

5.2 Prompt Improvement

Our prompt strategy for the AI assistant consists of two complementary parts that work together to produce BPMN 2.0 diagrams:

First, we improved a component identification guide that serves as a comprehensive reference for BPMN symbols. This document contains detailed descriptions and examples of each BPMN element, enabling the AI to recognize appropriate symbols for different process scenarios.

Our second refinement to the prompt engineering establishes strict guidelines for diagram generation. This covers essential aspects including component properties, connection rules, spatial organization, and process integrity. These rules ensure the AI produces diagrams that are not only semantically correct but also technically valid according to the BPMN 2.0 specification.

5.2.1 Component Identification Improvement

Our first refinement to the prompt engineering approach involved expanding the BPMN 2.0 component identification capabilities. The first version of implementation covered basic elements but lacked certain specialized event types, which limited some diagram creation options. We extended the component identification prompts by adding two additional event types:

Additional BPMN 2.0 Event Types

4. Timer Events: These events represent specific times or durations within a process. They can start a process, delay it, or trigger actions at defined times.
- Examples:
- Timer Start Event: Starting a monthly payroll process on the last day of each month.
- Timer Intermediate Event: Triggering a reminder email if a task is not completed within 24 hours.
- Timer End Event: Ending a promotional campaign at midnight on a specific date.
5. Message Events: These events represent interactions with external communication parties. They can be used to start, delay, or complete a process based on message exchanges.
- Examples:
- Message Start Event: Starting a support ticket process when a customer email is received.
- Message Intermediate Event: Sending a request for approval and waiting for a response before proceeding.
- Message End Event: Completing a process by sending a final report to a client.

To evaluate these improvements, we tested the enhanced component identification using a practical business scenario:

Sample Process with Timer Event

A company sends an invoice to a customer. If the payment is not received immediately, a timer event starts counting. After 7 days, the system automatically sends a payment reminder to the customer. The process then waits for the payment. Once the payment is received, the process ends.

When processing this scenario, the AI successfully identified and implemented the timer event in its response:

Extracted Timer Event from JSON Response

```
{
    // ... components omitted for brevity
    "ID": "e2",
    "Name": "Timer",
    "Type": "Intermediate Timer Event",
    "x": 500,
    "y": 100,
    "Incoming": ["c2"],
    "Outgoing": ["c3"]
    // ...
}
```

While the AI correctly identified the timer event in its JSON response, we discovered that the event didn't appear in the final rendered diagram. This discrepancy between the JSON structure and the visual output points to an implementation issue rather than a prompt design problem, which we'll address in the implementation chapter. Nevertheless, our testing confirmed that these prompt additions enhanced the AI's understanding of temporal aspects and communication flows in business processes. This improvement represents an initial step toward more comprehensive BPMN 2.0 modeling capabilities, particularly for processes involving scheduled activities and message based interactions.

5.2.2 Prompting Improvements

Unlike dedicated diagramming tools, large language models such as ChatGPT operate as general purpose assistants with no inherent understanding of visual layouts or technical standards. After analyzing the initial prompt structure, we identified several limitations affecting diagram quality. The original design lacked specific guidance in critical areas, leading to consistent errors in the generated diagrams. Through evaluation of the previous chatbot version, we discovered multiple opportunities for improvement that needed to be addressed through enhanced prompting techniques.

- Undersized pools and lanes that couldn't properly contain their elements
- Missing or incomplete connections between components
- Incorrect gateway type selection and usage
- Elements defined in JSON but not appearing in rendered diagrams
- Incomplete process paths that didn't properly terminate
- Inconsistent positioning of components within the coordinate system

Coordinate System Introduction An improvement in our prompt design addressed spatial positioning of elements. The BPMN 2.0 coordinate system has important differences from the standard Cartesian system that affect element placement. The original implementation lacked explicit positioning guidelines, leading to overlapping elements and poor spatial organization. To address this, we introduced formal coordinate system instructions:

Coordinate System Guidelines

Understanding X, Y Coordinates in BPMN 2.0 (bpmn-js):
In bpmn-js, element positions are based on the SVG coordinate system, where:
X-axis (horizontal) $\rightarrow$ Increases from left to right (moving right)
Y-axis (vertical) $\rightarrow$ Increases from top to bottom (moving down)

Valid BPMN 2.0 Component Types A problem in the previous implementation was the inconsistent naming of component types. The AI used descriptive names like "Start Event", "Sequence Flow", or "Gateway" that didn't match BPMN 2.0 standards. This required extra processing steps, including converting names to camelCase format and removing spaces, which increased processing time and created potential for errors. To eliminate these unnecessary steps, we provided standard compliant component names directly in the prompt. This improvement included comprehensive definitions of valid BPMN 2.0 components, ensuring the AI generated JSON responses after being transformed into XML could be used directly by the diagram rendering engine:

Valid BPMN 2.0 Component Types

This are valid components types that you should use in json response.
1. Events:
Start Events: startEvent ,messageStartEvent, timerStartEvent
Intermediate Events: intermediateCatchEvent, intermediateThrowEvent, messageCatchEvent, messageThrowEvent timerIntermediateEvent
End Events: endEvent, messageEndEvent
2. Tasks: task
3. Gateways: exclusiveGateway, parallelGateway, inclusiveGateway
4. Flows: sequenceFlow, messageFlow

Response Format Standardization After adding new event types and standardizing component naming conventions, we needed to establish a consistent JSON response format. This standardization ensured reliable interpretation by both the AI and the diagram rendering engine, eliminating ambiguities in the output structure.

Listing 5.1: Response Format

```
You must strictly adhere to this format, only answer with this format:
  {
    "Pools": [
      {
        "ID": "p1",
        "Name": "Customer Pool",
        "XY": [0, 50],
        "width": 1600,
        "height": 600,
        "Lanes": [
          {
            "ID": "l1",
            "Name": "Customer Lane",
            "XY": [0, 50],
            "width": 1600,
            "height": 600,
            "Components": [
              {
                "ID": "e1",
                "Name": "Start",
                "Type": "startEvent",
                "x": 100,
                "y": 100,
                "Incoming": [],
```

```
            "Outgoing": ["f1"]
          },
          {
            "ID": "t1",
            "Name": "Place Order",
            "Type": "task",
            // ... components omitted for brevity
          },
          {
            "ID": "e2",
            "Name": "Receive Order Confirmation",
            "Type": "intermediateCatchEvent",
            // ...
          },
          // ...
          {
            "ID": "e4",
            "Name": "End",
            "Type": "endEvent",
            // ...
          }
        ],
        "Flows": [
          {
            "ID": "f1",
            "Start": "e1",
            "Target": "t1",
            "Type": "sequenceFlow",
            "StartXY": [100, 100],
            "TargetXY": [300, 100],
            "Descriptor": ""
          },
          // ...
          {
            "ID": "mf3",
            "Start": "t2",
            "Target": "e6",
            "Type": "messageFlow",
            "StartXY": [700, 100],
            "TargetXY": [500, 750],
            "Descriptor": ""
          }
          // ...
```

```
    {
      "ID": "p2",
      // ...
      "Lanes": [
        {
          "ID": "l2",
          // ...
          "Components": [
            {
              "ID": "e5",
              "Name": "Receive Order",
              "Type": "messageStartEvent",
          // ...
}
```

To evaluate these improvements, we tested the enhanced instruction by using a snack preparing scenario.

Snack Preparing Scenario

To prepare snacks for a party, you ask for the number of guests, buy the ingredients, and then prepare the snacks. You can make one, two, or three of the following types of snacks: zakouskis, vegetables with dip sauce, and mini quiches. Once everything is ready, you serve the appetizers.

When processing this scenario, the AI successfully used valid components types in its response.

Extracted JSON Response

```
{
"Pools": [
  {
    "ID": "p1",
    "Name": "Party Preparation Pool",
    "XY": [0, 50],
    "width": 800,  // ISSUE: Too small, should be at least 1600
    "height": 300,
    "Lanes": [
      {
        "ID": "l1",
        "Name": "Snack Preparation Lane",
        // ...
          {
            "ID": "g1",
                   // ISSUE: Should be "e3" (incorrect ID prefix)
            "Name": "Choose Snacks",
            "Type": "exclusiveGateway",
                            // ISSUE: Should be inclusiveGateway
          // ...
          {
            "ID": "g2",
            // ISSUE: Should be "e8"(incorrect ID prefix)
            "Name": "All Snacks Prepared",
            "Type": "parallelGateway",
                                 // ISSUE: Should match g1's type
          // ...
}
```

Analysis of the JSON Response Analysis of the snack preparation scenario JSON response revealed both improvements and persisting challenges in AI generated diagrams. While the AI successfully implemented valid component types and created a logical process flow, several technical issues remained:

- **Sizing Issues:** Insufficient pool width (800 units versus the required 1600+) that couldn't properly contain all elements
- **Spacing Problems:** Component spacing inconsistencies that violated the minimum 200 unit guideline

- **Gateway Logic Errors:** Improper gateway pairing (mixing exclusiveGateway split with parallelGateway merge)

These findings demonstrate why comprehensive component rules and JSON validation frameworks are essential. Without explicit guidance on spatial organization, gateway logic, and component naming conventions, the AI produces diagrams that appear structurally sound at first glance but contain critical technical violations that could affect diagram rendering and interpretation. The validation framework we developed addresses precisely these issues by verifying component properties, connections, spatial organization, and process integrity before diagram generation.

Component Rules Enhancement To ensure professional diagrams, we developed comprehensive component rules covering Basic Element Types, Gateway and Flow Rules, and Container and Process Organization. This approach allows for verification of diagram correctness at multiple levels of abstraction.

1. Basic Element Types The first section defines the core elements and their usage requirements, establishing the building blocks for any business process diagram:

Listing 5.2: Basic Element Types

```
1. Start Events:
  - Generally, a process should have only one Start Event
  - In multiple pools, each can have a Start Event
  - Important for marking where a process begins

2. End Events:
  - A process should conclude with an End Event
  - Multiple End Events allowed in a single process
  - Each End Event signifies a different conclusion path
  - Every lane can have only one Start and one End Event

3. Timer Events:
  - Represent specific times or time-based delays
  - Formats: date-specific, duration-based, or cyclical
  - Can be used as Start or Intermediate events

4. Message Events:
  - Represent communication with external entities
  - Only allowed between different pools, never within same pool
  - Can be sent (black envelope) or received (white envelope)

```

```
5. Tasks and Sub-Processes:
  - Tasks for atomic activities that cannot be broken down
  - Sub-Processes for complex activities with internal steps
  - Must have clear, descriptive labels
  - Task types: User, Service, Manual, Script, etc.
```

2. Gateway and Flow Rules The second section addresses how elements connect and how process flow is controlled, focusing on decision points and sequence management:

Listing 5.3: Gateway and Flow Rules

```
1. Gateways:
  - Types: Exclusive (XOR), Inclusive (OR), Parallel (AND)
  - Only gateways can have multiple outgoing sequence flows
  - Gateway Pairing Rules:
    - XOR split: XOR merge optional
    - AND split: AND merge mandatory
    - OR split: OR merge mandatory
  - All paths from a split must reach corresponding merge

2. Sequence Flows:
  - Connect elements within the same pool (even across lanes)
  - Avoid crossing flows for readability
  - Can create loops by flowing back to earlier components
  - Component-specific connection rules:
    - Start Events: No incoming flows, one outgoing flow
    - End Events: Can have multiple incoming flows, no outgoing
    - Tasks: One incoming flow, one outgoing flow
    - Intermediate Events: One incoming flow, one outgoing flow

3. Message Flows:
  - Only allowed between different pools, never within same pool
  - Forbidden in single-pool diagrams
  - Must originate from: Tasks, Message Events
  - Must target components in a different pool
```

3. Container and Process Organization The third section covers structural organization and process integrity, ensuring proper containment and logical completeness:

Listing 5.4: Container and Process Organization

```
1. Pools and Lanes:
  - Pools represent different participants or organizations
```

```
  - Lanes represent roles or departments within a pool
  - Elements must never extend beyond pool/lane boundaries
  - Each pool must be properly sized to contain all elements

2. Process Integrity:
  - Every path must end properly at an End Event or message to another pool
  - No dead-end flows allowed in the process
  - Cross-pool communication uses message flows only
  - If process contains loops, must have at least one path to End Event

3. Parallel vs. Sequential Processes:
  - Use Parallel Gateways for simultaneous activities
  - Use Exclusive/Inclusive Gateways for conditional paths
  - Sequential activities should follow logical order

4. Artifacts and Data:
  - Data Objects show information used or produced
  - Annotations provide additional clarification
  - Artifacts should not affect the flow but enhance understanding
```

JSON Validation To ensure diagram integrity, we developed a comprehensive four part validation for the JSON output. This validation covered component properties, connections and flows, spatial organization, and process logic. By verifying each aspect of the JSON response, we could identify and correct errors before rendering the diagram.

1. Component Property Validation The first validation layer focuses on individual component correctness:

Listing 5.5: Component Property Validation

```
1. Component Property Requirements:
  - ID:
    - Must be unique
    - Prefixed appropriately:
      - General components: "e" (e.g., e1, e2)
      - Gateways: "g" (e.g., g1, g2)
      - Tasks: "t" (e.g., t1, t2)
  - Name:
    - Specify component name (e.g., "Start", "Task", "End")
    - For Timer Events use specific formats:
      - Date: "Start (timeDate: 2025-03-01 08:00:00)"
      - Duration: "Wait (timeDuration: PT5M)"
```

```
      - Cycle: "Repeat (timeCycle: R3/PT10M)"
  - Type:
    - Must be valid BPMN 2.0 component type
    - Examples: "startEvent", "task", "endEvent", "messageStartEvent"
  - Activity Labels:
    - English: verb-object format (e.g., "Approve Request")
    - German: noun-object format (e.g., "Antragsgenehmigung")
  - Flow ID:
    - Must be unique across all lanes/pools
    - Sequence flows: "f1", "f2", etc.
    - Message flows: "mf1", "mf2", etc.
```

2. Connection and Flow Validation The second validation layer ensures proper connections between components:

Listing 5.6: Connection and Flow Validation

```
2. Connection Requirements:
  - Incoming:
    - Array of flow IDs leading to component
    - Only merging gateways or task can have multiple incoming sequence flows
  - Outgoing:
    - Array of flow IDs from component
    - Only splitting gateways can have multiple outgoing sequence flows
  - Flow Table Requirements:
    - Must include all flows from Component Table
    - Required fields:
      - Start: Source component ID
      - Target: Destination component ID (single target only)
      - Type: "sequenceFlow" or "messageFlow"
      - StartXY: Start point coordinates
      - TargetXY: End point coordinates
      - Descriptor: Only for XOR Gateway outgoing flows
  - Connection Validation:
    - Every flow must have valid source and target components
    - All flows listed in Component's "Incoming" and "Outgoing" arrays must
    exist in Flow Table
    - If component A has a flow to component B, component B must have that flow
    in its "Incoming" array
    - Flow endpoints must match component positions
    - No missing or incomplete connections allowed
  - Flow Validation Rules:
    - Each flow ID must be unique across entire diagram
```

```
    - If flow f1 appears in any component's "Outgoing" (except startEvent), it must:
      - Exist exactly once in another component's "Incoming"
      - Match its Flow Table entry
```

3. Spatial Organization and Layout The third validation layer governs positioning and spatial relationships:

Listing 5.7: Spatial Organization and Layout

```
3. Spatial Requirements:
  - Coordinate Validation:
    - All x,y coordinates must be non-negative integers
    - Component coordinates reference its left upper point
  - Minimum spacing between components:
    - Horizontal: 200 units
    - Vertical: 200 units (if on same x-axis)
  - Flow coordinates:
    - StartXY must match source component's position
    - TargetXY must match target component's position
  - Alignment:
    - Align components at a consistent y-coordinate within a lane to maintain uniform height
    - Ensure similar y-coordinates for components connected by flows to improve readability
  - Container Structure:
    - Place Components into Lanes:
      - Assign each component and its corresponding flows to the correct lane
      - Ensure all components and flows are fully contained within their lane boundaries
    - Create Lanes and Validate Sizing:
      - Ensure each lane fully contains its components and flows
      - Adjust width and height of lanes as necessary to fit contents
    - Create Pools and Validate Sizing:
      - Use largest component coordinates for pool sizing
      - Update pool size to width = x + 200 and height = y + 200
      - Each pool must fully contain all its lanes and their contents
  - Multi-Pool Layout:
    - Position next pool at least 200 units below previous pool
    - Adjust positions to prevent overlap and maintain clarity
    - Ensure sequence flows do not cross unnecessarily
```

4. Process Logic and Flow Integrity The fourth validation layer ensures diagram correctness according to BPMN 2.0 standards:

Listing 5.8: Process Logic and Flow Integrity

```
4. Process Flow Requirements:
  - Start and End Events:
    - Only one Start Event per diagram without incoming flow
    - In multiple pools, each can have a Start Event
    - End Events can be multiple, but one is preferred
  - Component-Specific Rules:
    - Start Events: No incoming sequence flows, one outgoing flow
    - End Events: Can have multiple incoming flows, no outgoing flows
    - Tasks & Sub-Processes: Must have one incoming flow, one outgoing flow
    - Intermediate Events: One incoming flow, one outgoing flow
  - Gateway Rules:
    - Splitting Gateway: One incoming flow, multiple outgoing flows
    - Merging Gateway: Multiple incoming flows, one outgoing flow
  - Process Completion:
    - Every path must end properly with End Event or message to another pool
    - No dead-end flows allowed in the process
    - Cross-pool communication uses message flows only
    - If process loops back, must have at least one path to End Event
  - Connection Completeness:
    - Every component (except Start/End events) must have incoming and outgoing
    connections
    - No dangling connections allowed
  - Message Flow Rules:
    - When message flow sent from Pool A to B, Pool B must handle that message
    - If process starts with messageStartEvent, must terminate with
    messageEndEvent
  - Self-Correction Logic:
    - If component extends outside parent, expand parent or reposition item
    - Verify after changes that all boundaries remain respected
```

To evaluate these improvements, we use the snack preparing scenario again. When processing this scenario, the AI successfully used valid components types in its response.

Extracted JSON Response

```
{
  "Pools": [
    {
      "ID": "p1",
      "Name": "Party Preparation Pool",
      "XY": [0, 50],
      "width": 2000,  // IMPROVED: Adequate size for all elements
      "height": 600,  // IMPROVED: Proper height for content
      "Lanes": [
        {
          "ID": "l1",
          "Name": "Snack Preparation Lane",
          "XY": [0, 50],
          "width": 2000,
          "height": 600,
          "Components": [
            // Components omitted for brevity
            {
              "ID": "g1",
              "Name": "Decide Snacks",
              "Type": "inclusiveGateway",
            // ...
            },
            // ...
            {
              "ID": "g2",
              "Name": "Merge Snacks",
              "Type": "inclusiveGateway",
                          // IMPROVED: Consistent gateway types
          // ...
}
```

Analysis of Improved JSON Response After implementing our comprehensive validation framework, we observed significant improvements in the quality of the AI generated diagrams:

- **Pool Sizing Corrected:** The pool width increased from 800 to 2000 units, providing adequate space for all elements

- **Component Spacing Improved:** All components now maintain the minimum 200 unit horizontal spacing guideline for readability
- **Gateway Consistency Achieved:** Both gateways now use the same type, maintaining logical correctness in the process flow
- **ID Naming Conventions:** Component IDs now follow the proper prefixing conventions specified in our validation framework (e.g., "g" for gateways, "t" for tasks)

These improvements demonstrate the effectiveness of our validation framework in guiding the AI to produce technically valid diagrams. The four layer approach systematically verifies component properties, connections, spatial organization, and process integrity, resulting in diagrams that not only look visually correct but also fully comply with BPMN 2.0 technical standards.

5.3 Technical Implementation Optimization

To build a reliable and efficient BPMN 2.0 diagram generation chatbot, we addressed several critical technical implementation factors beyond prompt design. These optimizations ensured the AI could effectively process complex business process scenarios while maintaining performance.

5.3.1 Enhanced Knowledge through File Search

While our validation framework provided comprehensive rules, the AI required access to additional technical documentation for advanced cases. We used File Search [7] that enabled the system to:

- Access specialized BPMN 2.0 documentation and examples on demand
- Process technical specifications by segmenting them into retrievable chunks
- Perform both similarity based and keyword searches during diagram generation

This approach allowed the AI assistant to reference standard compliant practices during generation without requiring excessive prompt content, significantly improving output quality for complex processes.

5.3.2 Performance Optimization

To maintain reliable performance even with complex diagrams, we implemented several technical optimizations:

Token Management We carefully balanced available context between instructions, examples, and generation capacity:

- **Context Window Allocation:** Reserved sufficient tokens for model responses without exceeding the 128k token limit of GPT-4-turbo
- **Instruction Efficiency:** Reduced redundancy in prompts while preserving critical validation requirements
- **Dynamic Response Sizing:** Adjusted max_tokens parameter based on diagram complexity

Chunk Size and Overlap Configuration For document search functionality, we determined optimal configurations through empirical testing:

- **Optimal Chunking:** Breaking down large inputs into semantically coherent segments of 400 tokens
- **Strategic Overlap:** Maintained 50-100 token overlap between segments to preserve context
- **Retrieval Balance:** Limited search results to maintain performance while ensuring sufficient information retrieval

These technical optimizations complemented our prompt design improvements, creating a robust end to end solution for generating standards compliant BPMN 2.0 diagrams with high performance and reliability.

5.4 Conclusion: Bridging Prompt Design and Code

Our improvements to prompt design yielded significant enhancements in BPMN 2.0 diagram quality but also revealed inherent limitations of prompt engineering alone. The snack preparation scenario demonstrated this clearly: while AI excelled at logical elements like gateway selection based on business context, technical aspects like pool sizing remained challenging despite explicit validation rules.

This suggests an optimal approach combines AI's strengths in semantic interpretation with code based safeguards for technical aspects. For instance, consistent gateway typing was achieved through prompting, while pool sizing issues may require programmatic solutions. Our four layer validation framework provides a strong foundation, but translation from JSON to XML and subsequent rendering remain technical challenges best addressed through code implementation.

6 Implementation

Following the prompt improvements discussed in Chapter 5, we implemented most of functional requirements defined in the system requirements specification. While AI-powered capabilities enhanced our implementation, particularly for complex validation rules and logical constraints (covering functional requirements FR2-FR6, FR8-FR10, FR11, FR4, and non-functional requirements NFR1-4), we encountered limitations with ChatGPT's spatial reasoning abilities. These limitations required additional code-based adjustments to properly generate and format the XML output.

This chapter provides an overview of the key implementation aspects, focusing on three primary areas: JSON to XML transformation, XML validation, and user interface components. Specifically, we will address how we implemented functional requirements related to dynamic pool sizing (FR1), flow connection handling (FR7), XML validation (FR9), diagram organization (FR12), input handling (FR13), and component size considerations (FR15). By the end of the development phase, all functional and non-functional requirements were successfully implemented.

6.1 JSON to XML Transformation

The core implementation challenge involved converting the AI-generated JSON data structures into valid BPMN 2.0 XML documents that could be rendered by the bpmn.js library. The bpmn.js viewer is also online accessible at https://demo.bpmn.io/, which serves as an excellent tool to verify the generated XML.

When a user submits a text description of a business process, the AI assistant generates a JSON structure that must be transformed into BPMN 2.0 compatible XML. Let us examine the key aspects of this transformation process.

6.1.1 Dynamic Pool Size Calculation (FR1)

When users create complex diagrams, the pool size could be incorrectly defined in JSON response. Our implementation finds the maximum x and y values of all components in the pool to adjust its dimensions:

Listing 6.1: Dynamic pool size calculation

```
    function resetPoolsize(pool: any): void {
        let maxX = 0, maxY = 0;
        pool.Lanes.forEach((lane: any) => {
            lane.Components.forEach((component: any) => {
                if (component.x > maxX) maxX = component.x;
                if (component.y > maxY) maxY = component.y;
            });
        });
        pool.width = maxX + 200;
        //pool.height = maxY + 200;
    }
```

This function ensures that the diagram canvas is appropriately sized for all components with sufficient margin space.

6.1.2 Flow Classification and Handling (FR7)

When rendering connections between diagram elements, we need to determine how to route these flows. Our implementation classifies flows by direction and handles them accordingly:

Listing 6.2: Vertical and horizontal flow handling

```
function addVerticalHorizontalFlow
    (item: any, start: any, target: any): void {
    const up = (start.y - target.y) > 0;
    const down = (target.y - start.y) > 0;
    const right = (target.x - start.x) > 0;
    const left = (start.x - target.x) > 0;

    if (up) {
        // flow is going up
        if (start.Type.toLowerCase().includes('event')) {
            item.ele('omgdi:waypoint', {
                'x': start.x + 20,
                'y': start.y + 20
            });
        }
    // Additional positioning logic for different component types
    }
    // Similar handling for down, right, and left directions
}
```

This approach allows for clean, orthogonal connections that make diagrams more readable. When a user views their generated diagram, connections follow professional BPMN 2.0 conventions rather than crossing through elements or creating unclear paths.

6.1.3 Diagonal Flow Improvement (FR7)

In some complex diagrams, diagonal flows are unavoidable. For these cases, we implemented a more sophisticated routing algorithm that creates proper waypoints with appropriate turns:

Listing 6.3: Diagonal flow handling

```
function addDiagonalFlow(item: any, start: any, target: any,
    startTemp: any, targetTemp: any, turnPoint: any): void {
    const up = (startTemp.y - targetTemp.y) > 0;
    const down = (targetTemp.y - startTemp.y) > 0;
    const right = (targetTemp.x - startTemp.x) > 0;
    const left = (startTemp.x - targetTemp.x) > 0;

    if (up && right) {
    // Handle up-right diagonal with appropriate turn points
    // Component-specific adjustments
    }
    // Similar handling for other diagonal directions
}
```

We route diagonal flows using L-shaped paths with appropriate turning points rather than direct diagonal lines. This creates more professional diagrams.

6.1.4 Component Size Consideration (FR15)

Different element types have different shapes and sizes, which affects how connections should attach to them:

Listing 6.4: Component-specific anchor point calculation

```
 // Finding the middle point of components for better flow anchoring
 if (start.Type.toLowerCase().includes('event')) {
     startTemp.x = start.x + 20;
     startTemp.y = start.y + 20 + 20;
 } else if (start.Type.toLowerCase().includes('task')) {
     startTemp.x = start.x + 50;
     startTemp.y = start.y + 40;
```

```
} else if (start.Type.toLowerCase().includes('gateway')) {
    startTemp.x = start.x + 25;
    startTemp.y = start.y + 15 + 25;
}
```

These calculations ensure that when a user views their diagram, connections attach properly to each component type—connecting to the edge of circles for events, the middle of rectangles for tasks, and the appropriate points of diamonds for gateways.

6.2 XML Validation (FR9)

Before displaying the final diagram to users, we need to ensure the generated XML complies with BPMN 2.0 standards. Let's examine how this validation process works.

When a diagram is generated from JSON, the XML undergoes validation before being rendered. This process begins when the 'convertJsonToXml' function calls 'validateXML' with the raw XML string:

Listing 6.5: XML validation process

```
export function validateXML(xmlString: string): string {
    const parser = new XMLParser({ ignoreAttributes: false });
    const builder = new XMLBuilder({ ignoreAttributes: false });

    // Parse XML to JSON
    const jsonObj = parser.parse(xmlString);

    // Move message flows into collaboration
    const updatedJsonObj = moveMessageFlows(jsonObj);

    // Transform events format
    const transformedJsonObj = transformEvents(updatedJsonObj);

    // Convert back to XML
    return builder.build(transformedJsonObj);
}
```

The validator addresses two common issues that would otherwise prevent proper rendering:

First, message flows must be located in the correct XML section:

Listing 6.6: Message flow correction

```
function moveMessageFlows(jsonObj: any): any {
    const messageFlows: any[] = [];
    // Get all processes
    const processes = Array.isArray(jsonObj.definitions.process)
        ? jsonObj.definitions.process
        : [jsonObj.definitions.process];
    // Collect and remove message flows from processes
    processes.forEach((process: any) => {
        if (process.messageFlow) {
            const flows = Array.isArray(process.messageFlow)
                ? process.messageFlow : [process.messageFlow];
            messageFlows.push(...flows);
            delete process.messageFlow;
        }
    });
    // Add collected flows to collaboration
    if (messageFlows.length > 0) {
        if (!jsonObj.definitions.collaboration) {
            jsonObj.definitions.collaboration = {};
        }
        jsonObj.definitions.collaboration.messageFlow = messageFlows;
    }
    return jsonObj;
}
```

Second, specialized event types need to be properly formatted according to BPMN 2.0 specifications:

Listing 6.7: Event standardization

```
function transformEventType(process: any,
    sourceType: string, config: any) {
    if (process[sourceType]) {
        const events = Array.isArray(process[sourceType])
            ? process[sourceType]: [process[sourceType]];

        const transformedEvents = events.map((event: any) => {
            // Create new event with correct structure
            const newEvent: {
                '@_id': any;
                '@_name': any;
                incoming?: any;
                outgoing?: any;
                messageEventDefinition?: any;
```

```
15                  timerEventDefinition?: any;
16              } = {
17                  '@_id': event['@_id'],
18                  '@_name': event['@_name']
19              };
20
21              if (event.incoming) newEvent.incoming = event.incoming;
22              if (event.outgoing) newEvent.outgoing = event.outgoing;
23              if (config.needsMessageDef) {
24                  newEvent.messageEventDefinition = {};
25              }
26              if (config.needsTimerDef) {
27                  newEvent.timerEventDefinition =
28                  event.timerEventDefinition || {};
29              }
30              return newEvent;
31          });
32
33          if (!Array.isArray(process[config.targetName])) {
34              process[config.targetName] =
35              process[config.targetName] ? [process[config.targetName]] : [];
36          }
37
38          process[config.targetName].push(...transformedEvents);
39          delete process[sourceType];
40      }
41  }
```

These validation steps ensure that when users view their diagrams, all elements render correctly according to the BPMN 2.0 specification, regardless of complexity.

6.3 User Interface Implementation

Let's examine how users interact with the application through its interface components.

6.3.1 Input Handling (FR13)

When users want to create or modify a diagram, they need a convenient way to enter instructions. We implemented keyboard shortcuts to streamline this process:

Listing 6.8: Text input with keyboard shortcuts

```
<textarea matInput [(ngModel)]="diagramInstructions"
          cdkTextareaAutosize
          cdkAutosizeMinRows="1"
          cdkAutosizeMaxRows="16"
          (keydown.enter)="createDiagram()">
</textarea>
```

When a user presses the Enter key after typing their instructions, the application automatically triggers the diagram creation process, enhancing efficiency compared to requiring a mouse click on a submit button.

6.3.2 Diagram Organization (FR12)

As users create numerous diagrams over time, effective organization becomes essential. Our implementation includes a date-based sorting and grouping functionality.

Listing 6.9: Date sorting function

```
// Sort diagrams by date
sortDiagramsByDate(files: any[]): any[] {
 return files.sort((a, b) =>
 new Date(b.time).getTime() - new Date(a.time).getTime());
}
```

To categorize diagrams into time periods, we implemented helper functions:

Listing 6.10: Time categorization functions

```
// Check if diagram is older than one week
isOlderThanOneWeek(date: string): boolean {
 const fileDate = new Date(date);
 const oneWeekAgo = new Date();
 oneWeekAgo.setDate(oneWeekAgo.getDate() - 7);
 return fileDate < oneWeekAgo;
}
// Check if diagram is older than thirty days
isOlderThanThirtyDays(date: string): boolean {
 const fileDate = new Date(date);
 const thirtyDaysAgo = new Date();
 thirtyDaysAgo.setDate(thirtyDaysAgo.getDate() - 30);
 return fileDate < thirtyDaysAgo;
}
```

These functions power the dynamic diagram list display:

Listing 6.11: Dynamic diagram list with time categories

```
<div class="content">
  <mat-list>
    <!-- Start with "Last 7 days" header -->
    <mat-list-item>
      <mat-card-title>Last 7 days</mat-card-title>
    </mat-list-item>

    <!-- Show all diagrams in a loop -->
    <div *ngFor="let file of files; let i = index">
      <!-- Each diagram with its title and date -->
      <mat-list-item (click)="onFileClick(file)">
        {{ file.title }}
        <p class="file-time">
        {{file.time | date:'dd.MM.y, HH:mm':'CET'}}</p>
      </mat-list-item>
      <mat-divider></mat-divider>

      <!-- Add "Last 30 days" header when needed -->
      <div *ngIf="i < files.length - 1 && isOlderThanOneWeek(files[i + 1].time)
      && !isOlderThanOneWeek(file.time)">
        <mat-list-item>
          <mat-card-title>Last 30 days</mat-card-title>
        </mat-list-item>
      </div>

      /... similar code for older than 30 days ...
```

When a user opens the application, the diagram list is loaded and automatically organized. As they create new diagrams or as existing diagrams age, the categories update dynamically without manual reorganization.

6.3.3 Diagram Interaction Flow

Interaction and updating diagrams is handled via API calls to the server. Let's walk through a typical user interaction process:

When the application launches, it fetches the list of diagrams from the server using Angular's constructor. Once the diagram list is loaded, users can interact with it by clicking on items in the list. This functionality is implemented using Angular's (click) event binding.

If a user clicks the "New Diagram" button, the application opens a dialog popup where they can enter their instructions. When the dialog closes with confirmation, the application refreshes the file list and assigns all necessary attributes to display the newly created diagram.

When viewing an existing diagram, users have two options: they can manually edit it in the bpmn.js viewer or write a new update prompt. When updating through a prompt, the application refreshes the file list and updates the current diagram by pushing the latest instructions to the content and refreshing the display.

Users can also download their diagrams by clicking the button in the top bar. This feature leverages a function of the bpmn.js component to export the current state of the diagram for download.

Figure 6.1 shows the final user interface with organized diagrams by creation date. The list is easy to navigate, with clear headers indicating the time period of each diagram.

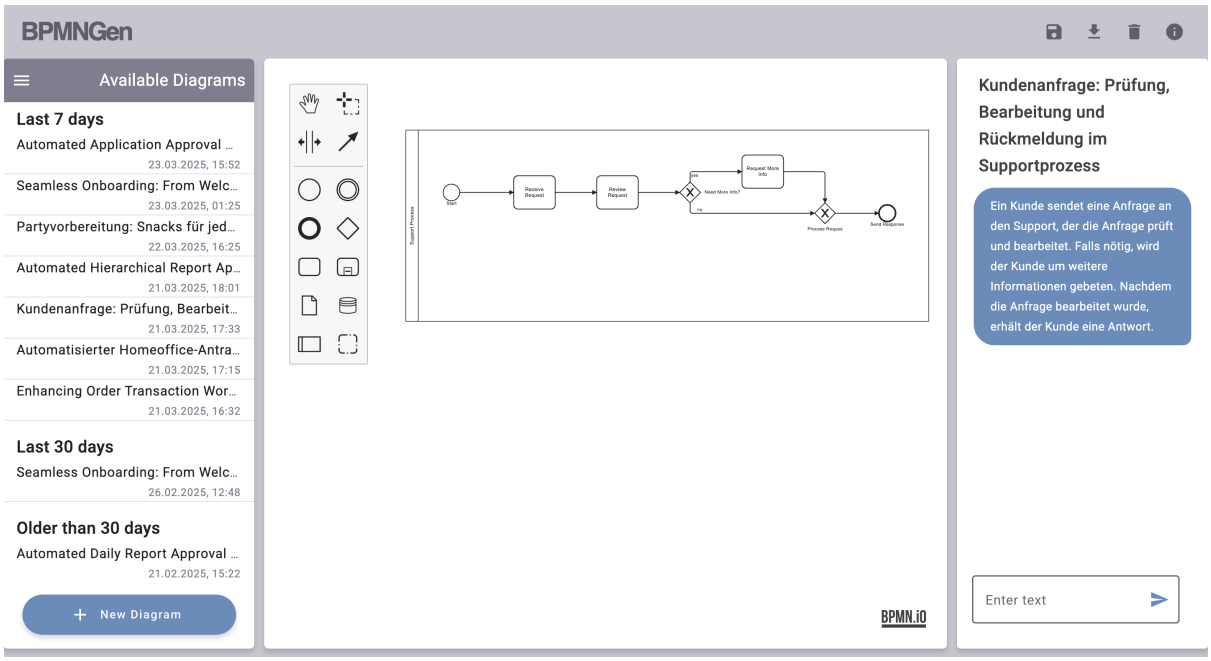

Figure 6.1: Final user interface showing organized diagram list

7 Results

This chapter presents and analyzes the results of the improvements made to our diagram generation chatbot. To demonstrate the chatbot's capabilities, we selected five user cases, which were processed by both the previous and the new versions of our diagram generation chatbot. Compared to the text used in the previous project, the given texts are more complex. Each case study focuses on two key aspects:

- **Accuracy:** We evaluate how well the generated diagrams match the intended requirements and assess the quality of their formatting and visual presentation.
- **Performance:** We analyze the time efficiency of the generation process and discuss the potential for further optimization from AI-based approaches.

The following sections present these case studies and their corresponding analyses.

7.1 Examples of Generated Diagrams

7.1.1 Party Snack Preparing

User Scenario

Um Snacks für eine Party vorzubereiten, fragst du nach der Anzahl der Gäste, kaufst die Zutaten und bereitest dann die Snacks zu. Du kannst eine, zwei oder drei der folgenden Snackarten zubereiten: Zakouskis, Gemüse mit Dip-Sauce und Mini-Quiches. Wenn alles fertig ist, servierst du die Vorspeisen.

Note: This text is in German, in order to test if the chatbot can conduct semantic checks to enforce that activity labels follow a verb-object format in English and a norm-object format in German.

Figure 7.1 illustrates the diagram generated by the previous version of chatbot, which had issues such as improper sequencing of activities and incorrect placement of elements. In contrast, Figure 7.2 shows the improved diagram, featuring properly sequenced activities,

correct placement of all elements within the pool, and appropriate gateway types for decision points. Initially, XOR was used, but this did not meet the user's requirements. After multiple iterations, a satisfactory diagram was produced. The enhanced visual organization, with clear labels and properly formatted symbols, improves readability and comprehension. Performance analysis shows comparable generation times between both chatbots (18-20 seconds), indicating that the quality improvements did not compromise efficiency.

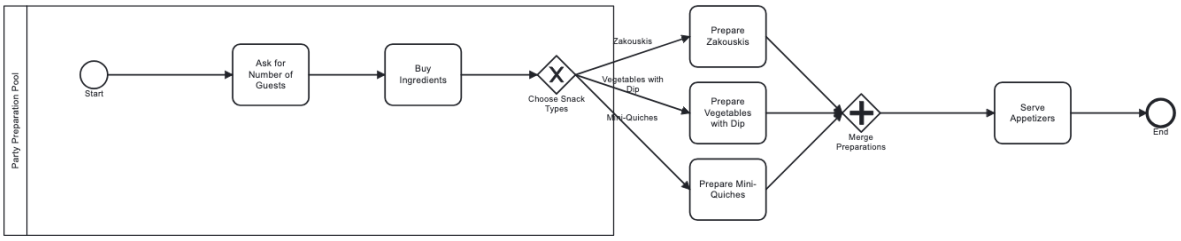

Figure 7.1: Party Snack Preparing previous version

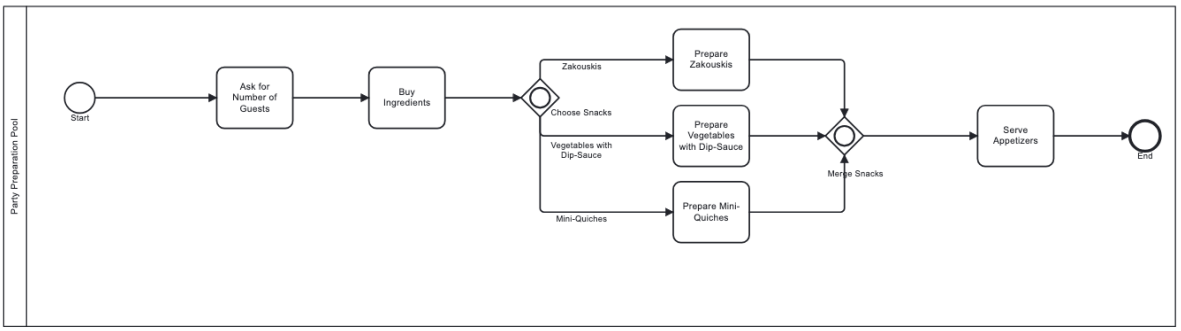

Figure 7.2: Party Snack Preparing new version

7.1.2 Application Process with Budget Verification

User Scenario

Two process roles are involved in the home office application process: the applicant (employee) and the department head. The process starts when the employee submits the application. If the previously determined budget is sufficient, the application is automatically approved without requiring review by the department head. However, if the budget is insufficient, the department head must review the application and make a decision. Upon approval, the applicant receives an automated email notification and the process is completed.

Figure 7.3 shows the diagram produced by the previous version of chatbot, which exhibits several flaws including crossed flow lines and poorly positioned connection points. In contrast, Figure 7.4 displays the enhanced visualization generated by our improved implementation, which offers a cleaner, more logically structured representation. The process flow is more logical, with clear decision points for budget verification and department head review. Additionally, all paths are properly terminated with end events, ensuring process integrity. Performance wise, generation for the diagram took 18 seconds.

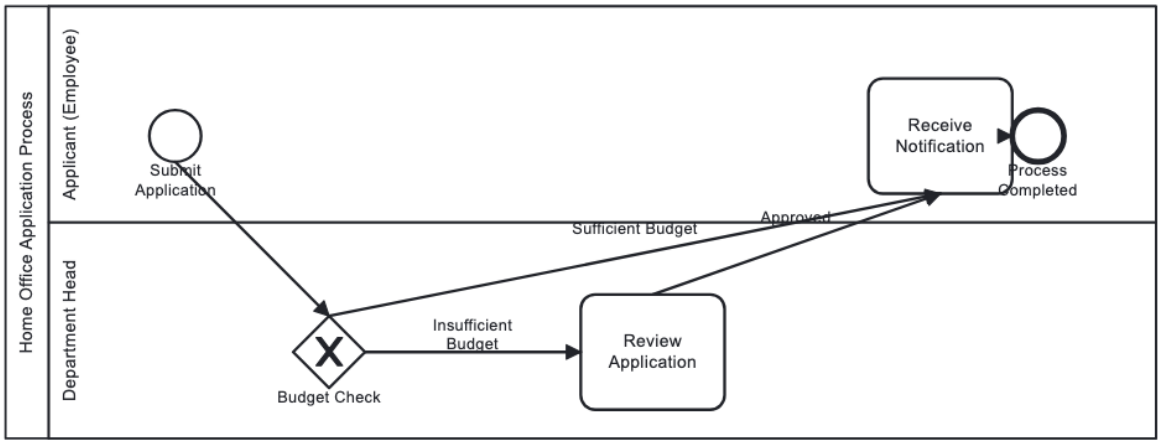

Figure 7.3: Application Process with Budget Verification previous version

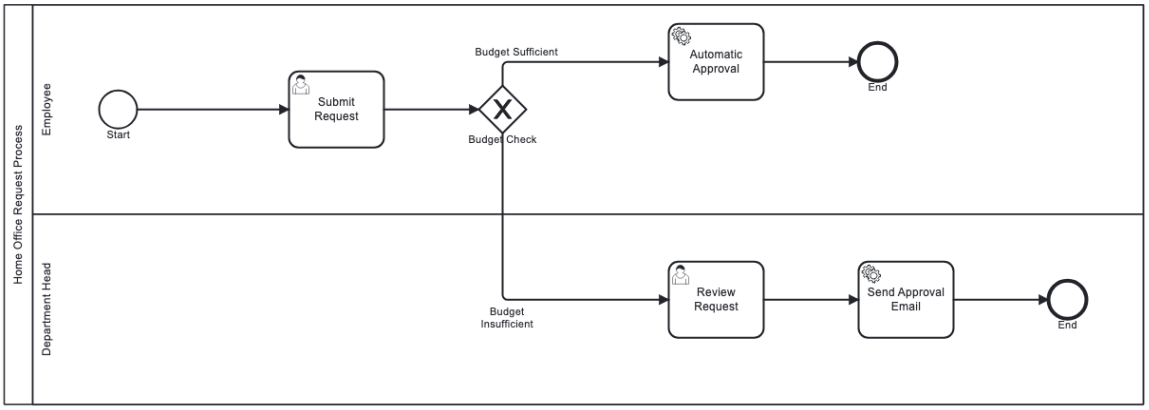

Figure 7.4: Application Process with Budget Verification new version

7.1.3 Smooth Onboarding Process

User Scenario

When a new employee accepts a job offer, the HR department sends them an onboarding welcome email. The IT department sets up the employee's workstation and provides the necessary equipment. The HR department then schedules an orientation session. If there are scheduling conflicts, the orientation session is rescheduled. Otherwise, the new employee will attend the orientation session, get introduced to their team, and be given a tour of the office. The employee then completes necessary paperwork and HR formalities, completing the onboarding process and starting their duties.

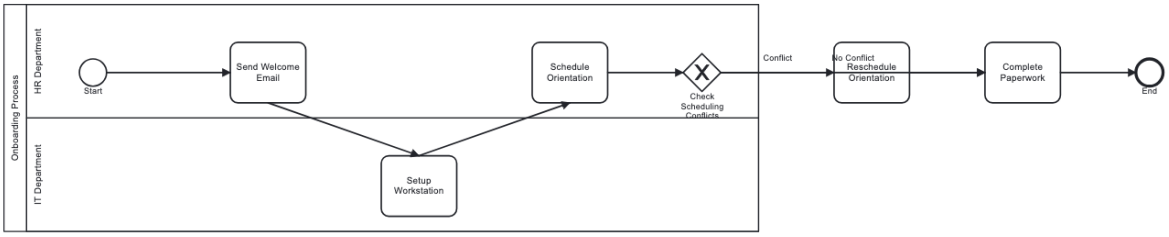

Figure 7.5: Smooth Onboarding Process previous version

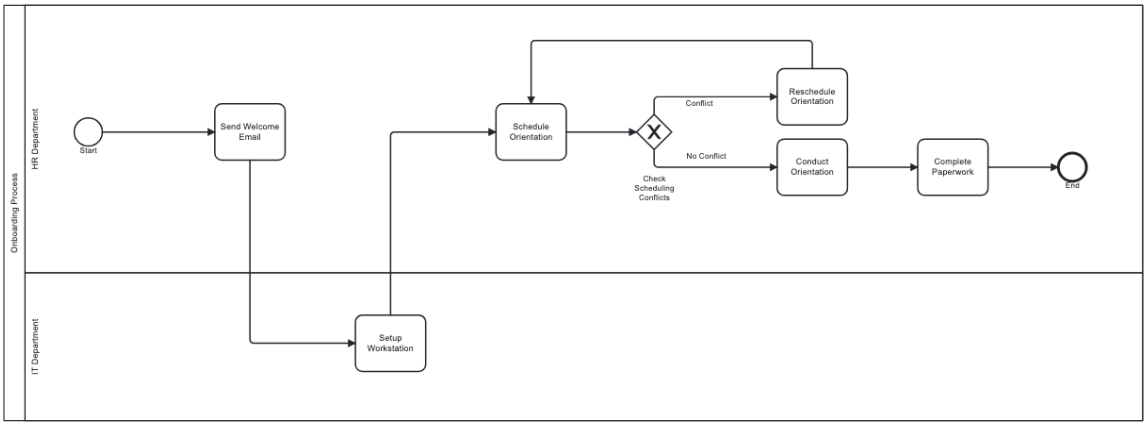

Figure 7.6: Smooth Onboarding Process new version

Figure 7.5 shows the diagram produced by the previous version of chatbot, which had issues such as unclear flow and improperly placed elements. By comparison, Figure 7.6 displays the improved diagram, which features a better layout with pool size correction and the usage of vertical and horizontal flows. The missing flow from the gateway was fixed, but there is an outgoing flow from the task "reschedule orientation" to the task "schedule

orientation," meaning the task "schedule orientation" has two incoming flows. Using a gateway like XOR or OR would be better for understanding. The generation time for this diagram was 15 seconds.

7.1.4 Coordinated Order Processing

User Scenario

In this business process, a Customer and Supplier coordinate an order transaction through message exchanges. The flow begins with the Customer placing an order, which the Supplier validates upon receipt. After processing, the Supplier sends a confirmation back to the Customer, who then proceeds with payment. However, there's a logical gap as the process should include a payment confirmation step where the Supplier acknowledges receipt before moving to the next phase. Only after payment verification should the Supplier prepare and ship the order, followed by sending shipping confirmation with tracking details. The Customer then tracks and confirms receipt of goods, completing the transaction. This process effectively demonstrates inter-organizational workflow coordination through structured message exchanges, though it requires the addition of the missing payment verification step for full logical integrity.

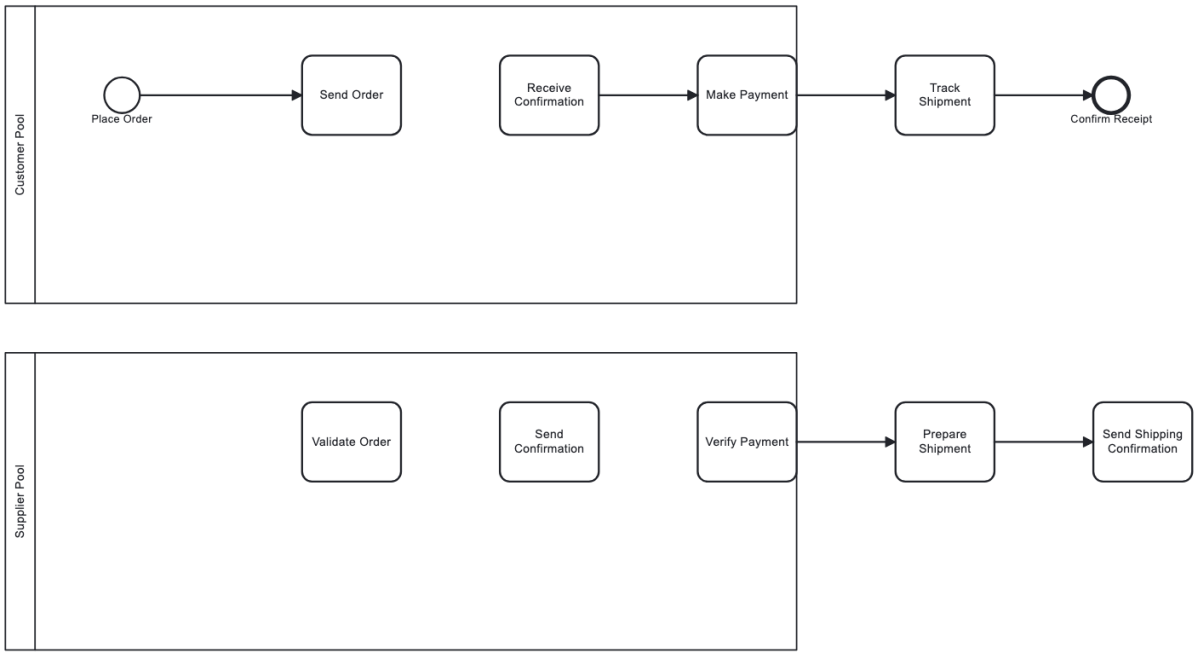

Figure 7.7: Coordinated Order Processing previous version

In the previous version, the diagram was generated as Figure 7.7 demonstrates. With the improvements implemented, the newly generated diagram is shown in Figure 7.8. After adding new event types, the chatbot is able to illustrate communications between pools by using message events and message flows. The diagram now accurately represents the user's requirements, with a clear sequence of events and proper message exchange between the Customer and Supplier. The diagram's structure is more coherent, with improved readability and logical flow. However, to achieve satisfactory results, sometimes multiple diagram generations or manual modifications are needed, as message events and message flows significantly increase the complexity of diagram generation. The generation time for this diagram was 18 seconds.

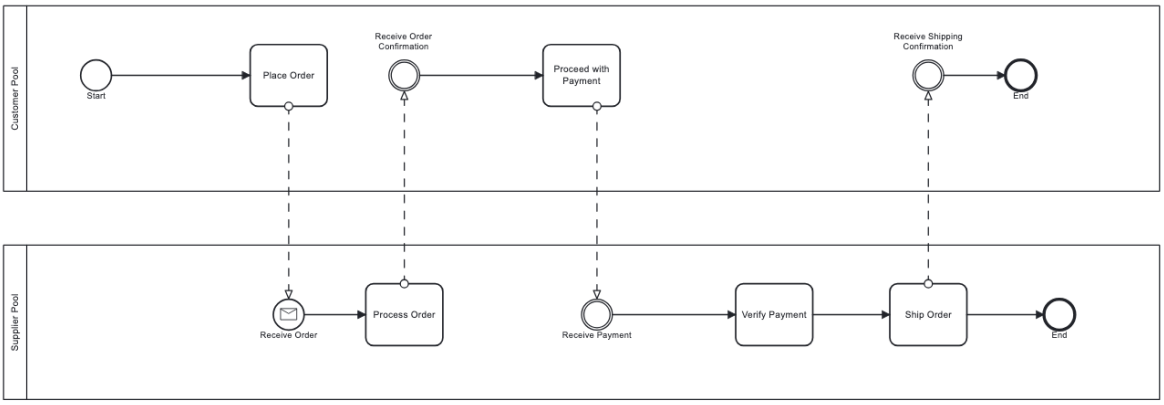

Figure 7.8: Coordinated Order Processing new version

7.1.5 Automated Report Approval

User Scenario

Every day at 8:00 AM, the chatbot automatically starts the daily report approval process. The report is sent to the manager, who must review the report and decide to approve it or not. If the manager approves the report, the process is complete. If the manager does not approve the report, a timer will count, and after 6 hours, the system automatically escalates it to the director for review. The director then takes over the review and decides whether to approve or reject it. If the director approves the report, the process ends. If the director rejects it, the report may be sent for corrections before going through the approval process again.

Figure 7.9 shows the diagram produced by the previous version of chatbot, which had issues such as unclear escalation paths and improperly placed elements. In contrast, Figure 7.10 displays the improved diagram, which features the use of timer events to represent

cases with time constraints. This enhancement allows for a more accurate representation of the process, ensuring that time-sensitive tasks are clearly indicated. The diagram's structure is more coherent, with improved readability and logical flow. The enhancements in diagram structure and clarity are evident in the new version. The generation time for this diagram was 15 seconds.

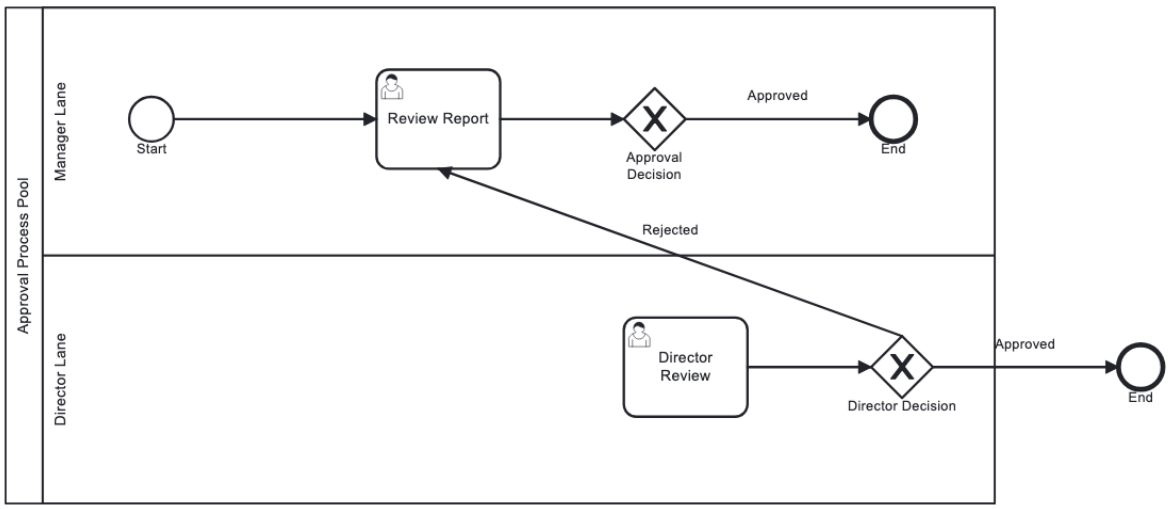

Figure 7.9: Automated Report Approval previous version

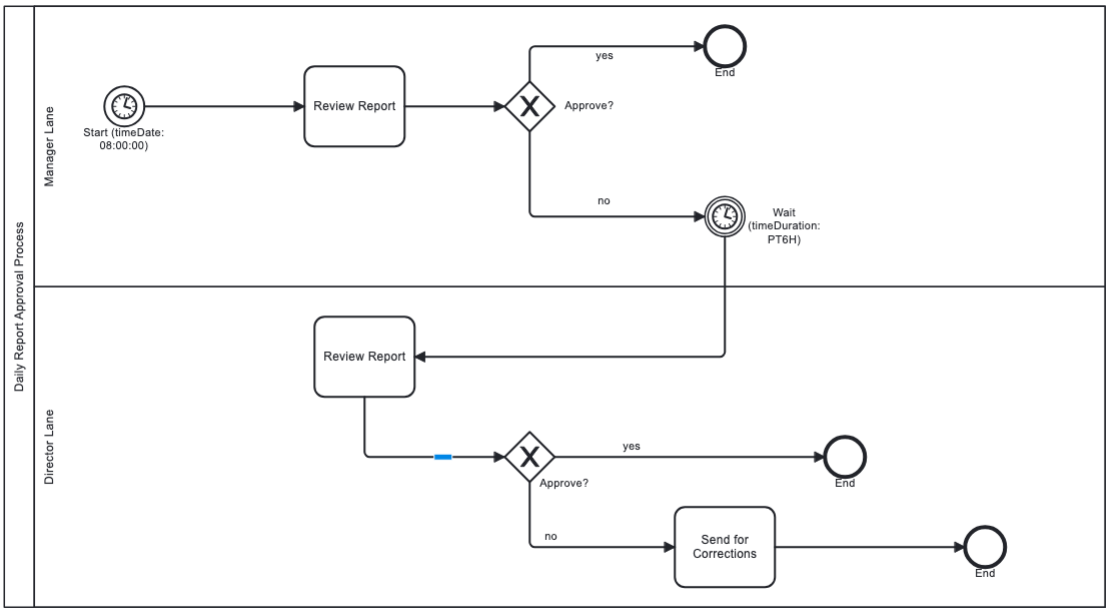

Figure 7.10: Automated Report Approval new version

7.2 Key Findings

Our analysis of the improved diagram generation chatbot reveals significant advancements in several key areas. The new version consistently produces diagrams with enhanced readability, more coherent structure, and closer alignment with user requirements. The BPMN 2.0 rendering now correctly handles complex paths and ensures proper diagram coherence. Despite utilizing more tokens for generation due to enhanced self-checking mechanisms and expanded search capabilities, the chatbot maintains comparable processing times, indicating efficient implementation of new features. We observed a direct correlation between input quality and diagram accuracy, with clear instructions reliably producing better representations. The chatbot's effective iterative refinement capabilities allow users to achieve progressively better results when regenerating diagrams after instruction refinements. This approach proves especially valuable for domain experts who can precisely articulate requirements. Additionally, improved UI/UX provides a more user-friendly experience while maintaining technical robustness.

8 Discussion

In this chapter, we revisit the system requirements, evaluate the chatbot's limitations, and discuss potential future developments for this project.

8.1 Reevaluation

In Chapter 3, we defined comprehensive functional requirements and non-functional requirements. Our BPMN 2.0 diagram generation chatbot has successfully fulfilled all the specified requirements, demonstrating its effectiveness and reliability in practical applications.

- Functional Requirements ✓
  - Diagram Accuracy and Validation ✓
  - Optimisation and Performance ✓
- Non-functional Requirements ✓

All the specified requirements were successfully fulfilled, demonstrating the chatbot's comprehensive functionality and reliability in practical applications.

8.2 Limitations

Despite the chatbot's achievements, several limitations should be acknowledged:

- **Flow Routing Complexity:** While our TurnPoint algorithm successfully converts diagonal flows to horizontal and vertical paths, highly complex diagrams may require additional turning points. This increases diagram complexity and can reduce readability, while also presenting implementation challenges.
- **Performance Constraints:** Despite runtime optimization through refined instructions and reduced token usage, complex diagrams may still experience noticeable response delays sometimes. Additionally, obtaining optimal results sometimes requires multiple generation attempts, which increases token consumption and consequently raises operational costs when using commercial AI services.

- **AI Instruction Sensitivity:** The AI assistant exhibits high sensitivity to prompt engineering. Minor modifications to the AI instructions can significantly alter outputs or cause failures. The semantic gap between human intention and AI interpretation presents a fundamental challenge in development.
- **Black-Box Nature of LLMs:** Working with large language models presents inherent limitations due to their opaque nature. We cannot precisely determine which rules must be explicitly defined versus which constitute tacit knowledge already embedded in the model. This ambiguity complicates systematic improvement of the chatbot's performance.

These limitations, while not preventing the chatbot from meeting its core requirements, highlight areas where future research and development efforts could be directed.

8.3 Future Work

While the chatbot functions well in its current state, there are several areas for potential improvement and expansion:

- **Diagram Management:** Adding file management features would improve organization, storage, and retrieval of diagrams, allowing users to maintain diagram libraries and version histories for complex projects.
- **Enhanced User Guidance:** Integrating contextual pop-up messages or tooltips during diagram generation would improve user experience by providing just-in-time assistance and reducing the learning curve for new users.
- **Interactive Interface:** Developing a chat-like layout similar to ChatGPT would create a more intuitive interaction model, potentially increasing user engagement and simplifying the diagram creation process.
- **Modular Diagram Handling:** For complex scenarios, implementing functionality to decompose large models (exceeding 50 elements) into smaller, manageable sections would improve both performance and diagram readability.
- **Diagram Optimization:** Enhancing the chatbot to automatically use the minimum number of elements required while avoiding complex routing elements would improve structural clarity and processing efficiency.

These enhancements would significantly improve both functionality and user experience, making the chatbot more practical for diverse real-world applications.

8.4 Conclusion

This project focused on further development and optimization of an existing BPMN 2.0 diagram generation chatbot. Through systematic improvements to diagram accuracy and validation processes, we successfully enhanced the chatbot's performance in several key areas and improved the quality and reliability of the generated diagrams.

Our contribution demonstrates that even in complex AI-driven chatbot, targeted optimization can yield substantial benefits. By refining the AI instructions and implementing robust validation mechanisms, we resolved persistent bugs related to diagram rendering and improved the overall user experience. The optimized chatbot now produces more accurate BPMN 2.0 diagrams while adhering to industry standards and best practices.

The improvements made to the chatbot's diagram accuracy, validation, and performance optimization showcase the importance of continuous refinement in AI-assisted modeling tools. As LLM technology rapidly evolves, optimization work like this remains crucial for bridging the gap between raw AI capabilities and practical, user-friendly applications. With growing interest from both research and industry, optimized BPMN 2.0 generation chatbot will continue to improve, making process modeling more accessible and efficient for a wider range of users.

Bibliography

[1] Object Management Group. *Business Process Model and Notation (BPMN) Specification*. Accessed: 2025-03-18. 2024. URL: https://www.omg.org/spec/BPMN/.

[2] Wikipedia contributors. *Large Language Model*. Accessed: 2025-03-18. 2025. URL: https://de.wikipedia.org/wiki/Large_Language_Model.

[3] Jan Mendling, Hajo A Reijers, and Wil MP van der Aalst. "Seven process modeling guidelines (7PMG)". In: *Information and software technology* 52.2 (2010), pp. 127–136.

[4] OpenAI. *Introducing ChatGPT*. Accessed: 2025-03-18. 2022. URL: https://openai.com/index/chatgpt/.

[5] Camunda Services GmbH. *bpmn-js BPMN 2.0 viewer and editor.* Accessed: 2025-03-18. 2023. URL: https://bpmn.io/toolkit/bpmn-js/.

[6] OpenAI. *Assistants API overview*. Accessed: 2025-03-18. URL: https://platform.openai.com/docs/assistants/overview.

[7] OpenAI. *Assistants File Search*. Accessed: 2025-03-18. URL: https://platform.openai.com/docs/assistants/tools/file-search.

Name: Zhe Shi Matriculation number:1036630

Declaration

I hereby assure that I wrote this work independently and did not use any other than the denoted sources and tools.

Ulm, ...

Zhe Shi