Low-Code AI

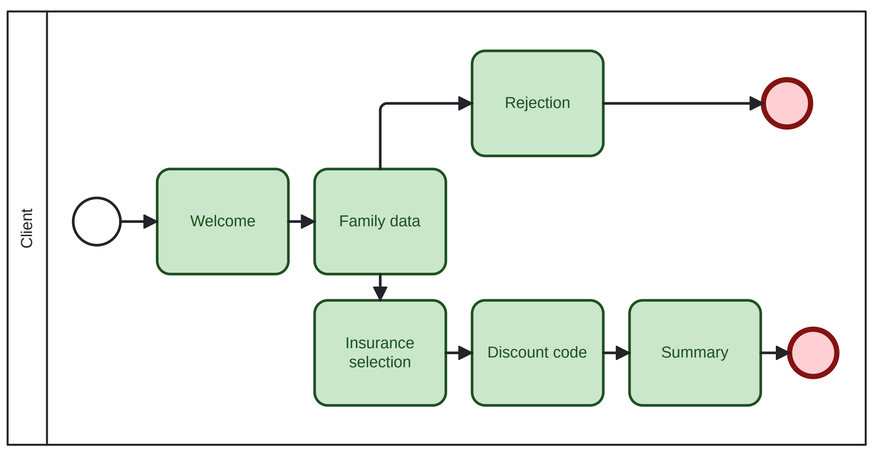
Improving Code Generation Accuracy in Low-Code Platforms with Hierarchical Contexts

Adam Mickiewicz University Poznań

Introduction

The project is being carried out in collaboration with **DOMDATA** and their **Ferryt** platform. Our project has also been awarded a grant under the **Study@Research** program.

Low-code and no-code platforms have gained popularity as cost-effective solutions to the scarcity of skilled developers. These systems enable users to build applications with **little or no programming**, focusing on business logic rather than technical complexities. They accelerate development significantly —often 5 to 10 times faster—by offering tools like data model designers, GUI builders, API integrations, and BPMN-based process editors. Despite their advantages, low-code platforms still often require some technical knowledge. Our project addresses this by developing a multi-agent AI plugin to **transform Ferryt low-code platform into true no-code solution.**

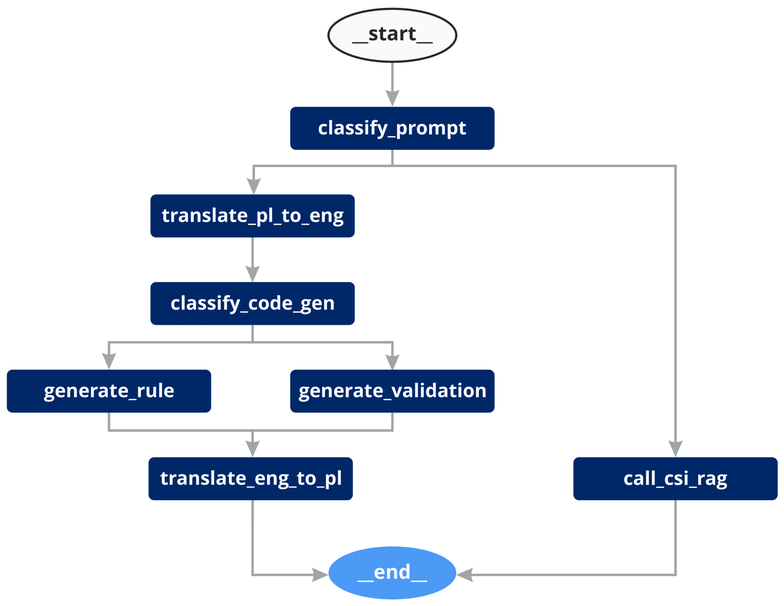


Example BPMN diagram

Multi-agent Architecture

Key Nodes in the multi-agent system:

- **start**: Entry point.
- **classify_prompt**: Decides the type of user query (code generation or documentation content) using a language model.
- **call_csi_rag**: Accesses Ferryt Navigator's documentation and API, leveraging RAG architecture and the BIELIK model.
- **translate_pl_to_eng**: Translates the user query into English, as codestral has limited understanding of Polish.
- **classify_code_gen**: Determines the type of code to generate:
  - JavaScript validation code (e.g., PESEL or NIP validation).
  - C# business rule code (e.g., making a field non-editable based on BPMN diagram context).
- **generate_rule**: Extracts relevant nodes from a BPMN diagram, leveraging research methods, to generate C# business rules using a language model.
- **generate_validation**: Generates JavaScript validators.
- **translate_eng_to_pl**: Translates the response back into Polish.
- **end**: Exit point.

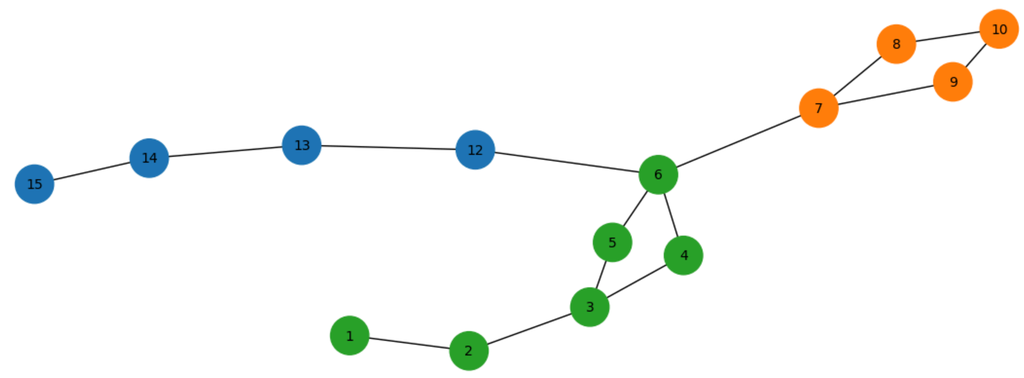


Actual information flow in the **multi-agent** system, generated using the LangGraph framework based on real code.

Context-Aware Graph Search

The research investigates leveraging user prompts to extract contextual information from nodes in **BPMN (Business Process Model and Notation)** diagrams for **automatic code generation** in low-code platforms. Unlike static knowledge-based approaches, our method dynamically queries the BPMN graph using natural language prompts processed by a Large Language Model (LLM). The extracted contextual information supports dynamic process modeling without requiring a predefined knowledge base.

In this approach, BPMN diagrams are represented as graphs, where nodes denote tasks, gateways, and events, and edges depict transitions. The graph is decomposed into community structures using **Graph Spectral Clustering**, which analyzes the spectrum (eigenvalues) of the similarity matrix to identify clusters of densely connected nodes.



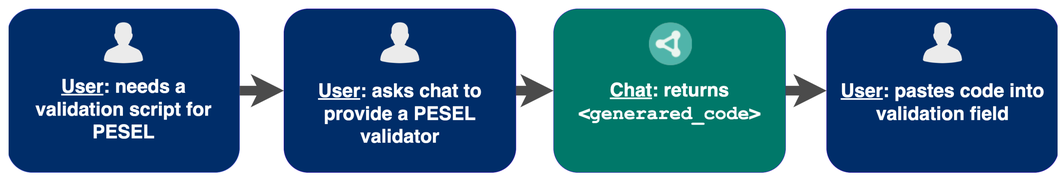
Clustered context graph

Finding relevant nodes in a graph using an LLM is enhanced by **parallel processing** with the **MapReduce** framework, enabling simultaneous searches across graph communities. This approach boosts performance, with user prompts refining results for real-time BPMN graph querying, surpassing traditional methods like RAG and GraphRAG that rely on static databases.

We used **Bielik-11B-v2.2-Instruct-FP8** for context extraction, **Codestral-22B-v0.1** for code generation, **LangGraph** 0.2.39 for multi-agent architecture, **LangChain** 0.3.0 for LLM integration, and **Ollama** for improved model interaction.

Evaluation

Assessing the quality of LLM-generated code is challenging due to the unique requirements of integrating concise snippets into a larger codebase. Traditional evaluation methods, such as unit or E2E tests, are not feasible. Evaluation will instead involve designing diverse test scenarios, **reviewed by five domain experts** for quality and functionality, supplemented by validation from another **LLM** to ensure a robust assessment.



Actual Use Case Example

Conclusions

- **Observability** and **tracing** are critical for understanding, monitoring, and optimizing multi-agent systems, enabling effective troubleshooting and performance evaluation.
- Evaluating LLM-generated code remains **complex**, requiring robust frameworks and **domain expertise**.
- The model offers **potential** for **future** development, including integrating additional Ferryt **coding** aspects to expand its applicability.

Supervisor: prof. UAM dr hab. Patryk Żywica

Team: Dawid Korzępa, Arden Wołowiec, Anna Śmigiel, Krzysztof Raczyński

Uczelnia Badawcza Inicjatywa Doskonałości

DOMDATA