JOT
JOURNAL OF
OBJECT TECHNOLOGY

LLM-Powered Multi-Agent Systems: Exploring Documentation-Driven Metamodeling

James Pontes Miranda*, **Ansgar Radermacher***, **Fabien Baligand***, **Julie Bonnail***, **Kunal Suri***, **Pascal Bannerot***, **and Marcos Didonet Del Fabro***

*Université Paris-Saclay, CEA List, France

ABSTRACT In Model-Driven Engineering (MDE), metamodeling is a crucial activity and is often the starting point of a full MDE pipeline. A central aspect of this task is extracting domain knowledge from textual documentation and establishing the necessary classes, relationships, and constraints that will be later used to formalize the metamodel. This early stage is known to be demanding, error-prone, and influenced by individual modelers' expertise and bias. Recent advancements in the use of Large Language Models (LLMs), have stimulated research on interpreting textual information to generate models and metamodels. However, there is still limited work exploring the potential of LLM-powered agents, particularly LLM-powered Multi-Agent Systems (LLM-MAS), to support the metamodeling process in a structured manner. In this paper, we present an approach that leverages an LLM-MAS to assist in documentation-driven metamodeling. The proposed approach decomposes the task into multiple specialized agents responsible for activities such as domain analysis, terminology identification, normalization and deduplication, and textual serialization into a PlantUML class diagram. The system operates without human intervention and produces intermediate artifacts that support traceability and inspection. We report on the development of this LLM-MAS and its application in a case study involving the extraction of a draft metamodel from a set of agent framework documentations. We provide an exploratory qualitative evaluation focusing on the feasibility, stability, and structural plausibility of the generated artifacts. The results indicate that LLM-MAS can consistently produce structurally plausible model-like artifacts that may assist modelers in the early stages of metamodel creation. Rather than targeting full automation, the approach positions LLM-MAS as a modeling aid that supports early abstraction and helps modelers initiate metamodel development more systematically.

KEYWORDS Metamodeling, Large Language Models, LLM4MDE, Multi-Agent Systems.

1. Introduction

Metamodeling is a central activity in Model-Driven Engineering (MDE) (Brambilla et al. 2012), providing the conceptual foundations for Domain Specific Languages (DSLs) (Fowler 2010), tool development (Saraiva & Silva 2008), and analyses of software systems (Bjørner 2023). Systematic domain modeling is commonly realized via metamodeling, which captures domain concepts and constraints at a higher abstraction level (e. g., (Bork 2018; Levendovszky et al. 2009)).

In practice, modeling often relies on extensive textual documentation, such as specifications, textual requirements, framework manuals, and API descriptions, which must be interpreted and structured into efficient abstractions. This process is known to be time-consuming, error-prone, and highly dependent on individual modelers' expertise and bias (Roy Chaudhuri et al. 2019; Montrieux et al. 2013).

Recent advances in Large Language Models (LLMs) have renewed interest in automating or assisting modeling activities (LLM4MDE) (Cámara et al. 2023; Di Rocco et al. 2025; Conrardy & Cabot 2024). In fact, a growing body of work explores the use of LLMs to support modeling tasks, from generating

JOT reference format:
James Pontes Miranda, Ansgar Radermacher, Fabien Baligand, Julie Bonnail, Kunal Suri, Pascal Bannerot, and Marcos Didonet Del Fabro. *LLM-Powered Multi-Agent Systems: Exploring Documentation-Driven Metamodeling*. Journal of Object Technology. Vol. 25, No. 3, 2026.

http://dx.doi.org/10.5381/jot.2026.25.3.a29

models (Babaalla et al. 2025), and inferring constraints (Pan et al. 2024) to general modeling aid with LLMs (Brandolini et al. 2024; Hachm et al. 2025). While these solutions rely mostly on different Prompt Engineering (PE) strategies and single-agent setups, Multi-Agent Systems (MAS) re-emerged as a promising paradigm for structuring complex LLM-powered workflows (Li et al. 2024; Cheng et al. 2024). Despite this potential, the application of LLM-powered MAS (LLM-MAS) to metamodeling remains underexplored and with few structured analyses. An exception are examples of multi-agent architectures that distribute modeling subtasks among specialized agents to support the generation of UML (Unified Modeling Language ((OMG) 2024)) class diagrams (Giannouris & Ananiadou 2025).

This paper presents a modeling-centric report on the design, instantiation, and empirical exploration of an LLM-MAS pipeline to investigate whether it can systematically help with documentation-driven metamodeling. Rather than aiming for fully autonomous modeling, we explore using a MAS as a modeling aid that produces draft class diagrams that can be further refined by human engineers.

Concretely, we report on the development and evaluation of an automated MAS pipeline that transforms software documentation into a UML class diagram. The pipeline decomposes the process into documentation ingestion, terminology extraction, terminology consolidation, and structural modeling, with each stage handled by a dedicated agent implemented with a specific agentic framework, aiming to demonstrate that our work is agnostic to framework choice and ready to accommodate current and upcoming frameworks. This allows us to investigate both the feasibility of automated documentation-driven modeling and the role of framework heterogeneity in such pipelines. The system produces intermediate artifacts that support traceability and inspection. As a running example and reference artifact, the work builds on a previously developed metamodel (Miranda et al. 2026) for representing commercial code-centric LLM-powered multi-agent frameworks.

The contributions of this paper are threefold:

1. A LLM-MAS pipeline for documentation-driven metamodeling, composed of heterogeneous agents with specialized roles.
2. An exploratory and qualitative evaluation of the generated modeling artifacts, focusing on feasibility, stability, and structural plausibility.
3. An empirical exploration of lessons learned and limitations of LLM-MAS as modeling aids in MDE.

The remainder of the paper is organized as follows. Section 2 introduces the necessary background on metamodeling and LLM-MAS. Section 3 presents a manual baseline used throughout the paper. Section 4 describes the proposed LLM-MAS modeling pipeline, followed by a qualitative evaluation in Section 5. Section 6 discusses lessons learned, and in Section 7 we compare and contrast the literature with our work. Section 8 concludes the paper and outlines our future work. All the collected and generated data is provided in a companion repository via GitHub for transparency and verification.

2. Background

This section provides an overview of the two research dimensions of this work: Metamodeling in the MDE context and LLM-MAS. The goal is to situate our contributions within established modeling principles and emerging agentic Artificial Intelligence (AI) paradigms, rather than to survey these areas exhaustively.

2.1. Metamodeling in MDE

In the MDE context, a metamodel specifies the abstract syntax of a modeling language by formally defining domain concepts, relationships, and constraints. This enables structured abstraction, consistency verification, and a range of automated processes (e. g., validation, transformation, and code generation) that are central to MDE practices (Brambilla et al. 2012).

Given that, metamodeling is a well-known and crucial activity across different approaches. Metamodels enable the operationalization of domain modeling by defining and structuring domain-specific abstractions (de Lara et al. 2015). In this sense, metamodeling involves eliciting, structuring, and organizing domain concepts into a coherent conceptual schema that serves as the basis for subsequent modeling work.

Models and metamodels can be formalized using various languages (e. g. MOF-compliant languages). This paper uses PlantUML (Roques 2009), a Markdown-based UML language widely adopted for textual model serialization, due to its compatibility with LLM-powered solutions (Cámara et al. 2023; Petrovic et al. 2025).

2.2. LLM-Powered MAS

MAS are computational systems composed of multiple interacting agents, each capable of independent behavior, communication, and decision-making. In the classical AI literature, agents may be software components, autonomous processes, or goal-directed entities within distributed environments (Russell & Norvig 2020). With increasing interest in foundation models, primarily based on LLMs, the research and industrial communities have begun exploring how these models can serve as the primary reasoning mechanism for agents (Li et al. 2024), characterizing what we call LLM-MAS. Integrating LLMs into MAS is an active research area (Cheng et al. 2024; Jin et al. 2024).

LLM-MAS exploit LLMs' natural language reasoning, planning, and knowledge retrieval to enable collaborative problem-solving beyond isolated LLM capabilities. LLM-MAS often orchestrate the inner agents through explicitly designed workflows or communication protocols (e. g. Agent-to-Agent (A2A)[1]). By distributing subtasks among specialized agents, LLM-MAS can support complex workflows and mirror modular human processes in problem-solving and reasoning (Tran et al. 2025). In general, LLM-MAS exhibit advantages over monolithic single-agent approaches (Yan et al. 2025).

Despite this promise, the design and evaluation of LLM-MAS pose significant challenges, including issues of agent coordination and interpretability of intermediate reasoning

[1] https://a2a-protocol.org/latest/

steps (Tran et al. 2025). These considerations directly influence how such systems can be applied to structured tasks, such as metamodel generation.

2.3. Documentation-Driven Modeling

From an MDE perspective, textual documents (e. g., software documentation, informal requirements, etc.) is the primary input for early modeling (text-to-model, T2M) (Lucassen et al. 2017; Burgueño et al. 2025), though its informality, variability, and scale complicate automation (Mornie et al. 2023). As documentation evolves, keeping metamodels and derived models aligned requires repeated, ad-hoc extraction and refactoring steps that do not scale across large or heterogeneous document collections.

Despite these problems, natural-language requirements and technical documents remain the dominant means of capturing domain knowledge and system behavior in the MDE context, both in research and industry (Arora et al. 2016). Less strict models (e. g., a light collection of classes, attributes, and relations) can be systematically derived from requirements sentences using grammatical dependencies and extraction rules, confirming that textual documentation encodes the structural concepts later captured in more specific metamodels. This happens since textual documentation underpins higher-level abstractions such as patterns and semi-formal requirement representations, which may provide reusable templates for recurring behaviors and services in a domain (Kudo et al. 2023; Neubauer et al. 2019). Figure 1 provides a simplified overview of a process that goes from textual documentation to a metamodel, then to the domain models, and finally to the actual software system. This paper is focused on the first part of this process (i. e., the T2M).

Different Natural Language Processing (NLP) techniques were applied to handle T2M problems, cf. Bozyigit et al. for an overview and tool comparison (Bozyigit et al. 2024). Recent advances in NLP with LLMs have opened opportunities to handle the full pipeline from textual specifications to software models (Babaalla et al. 2025; Arulmohan et al. 2023). LLM-based metamodeling approaches highlight that iterative, pipeline-oriented automation can systematically refine metamodels from textual sources, but they also expose open challenges in robustness, traceability, and integration into existing MDE workflows (Petrovic et al. 2025). Our motivation is precisely to contribute to solving these challenges, which drive research on pipeline automation for metamodeling using LLM-MAS.

3. Manual Baseline

To illustrate the behavior and expected output of the proposed LLM-MAS metamodeling aid, we consider a running example drawn from the documentation of a set of frameworks designed for the development of LLM-powered agentic systems (e. g., LangChain, LlamaIndex, etc.). The goal of this section is to provide a concrete description of the input corpus, the manual baseline construction process, and the expected output artifacts, aligning them with established documentation-driven metamodeling practices.

Defining a common architecture for this kind of agentic framework is a demanding task, and we found efforts from different perspectives aimed at addressing it. These efforts range from less formal taxonomies (Di Sipio et al. 2025) and architectural references (L. Wang et al. 2024; Derouiche et al. 2025), to more formal and MDE approaches (Miranda et al. 2026; Hassouna et al. 2024). We adopt our previous work (Miranda et al. 2026), which is strongly aligned with an MDE mindset, as a manual baseline for evaluation. In that work, we conducted a systematic analysis of agentic frameworks and developed an Ecore[2] metamodel through a manual inspection process. Here, we reproduce a simplified version of that process using the same documentation corpus adopted for our automated pipeline.

As input, we considered the online documentation of 26 code-first LLM-powered agentic frameworks widely used across projects and referenced in technical media and industry-driven reports (ThoughtWorks 2025; Gartner, Inc. 2025). These documents are typically provided as heterogeneous web pages combining textual and non-textual information. For this paper, we restricted our analysis to textual content extracted directly from the documentation pages[3]. For each framework, the front page of the documentation was used as the entry point, and internal pages explicitly identified as API references (e. g., Github pages, Swagger documentation, etc.) were also incorporated. Each framework received an identifier, and its full textual documentation (main page + API references) composed the corpus for manual analysis. Table 1 presents an excerpt of the selected frameworks.

ID	Name	Initial URL (front-page)
1	Agno	https://docs.agno.com/
2	Atomic Agents	https://github.com/BrainBlend-AI/atomic-agents
3	AutoChain	https://autochain.forethought.ai/
4	BeeAI	https://framework.beeai.dev/
5	BESSER	https://besser-agentic-framework.readthedocs.io
6	CAMEL	https://www.camel-ai.org/
7	CrewAI	https://www.crewai.com/open-source
8	Eclipse LMOS	https://eclipse.dev/lmos/

Table 1 Excerpt of selected frameworks (Full table: 26 lines).

Following the manual process from our previous work, domain concepts were identified by constructing a terminology of recurring terms across the corpus. Relationships were extracted by inspection, and both concepts and relations were encoded into a textual PlantUML class diagram.

The resulting artifact reflects modeling decisions regarding which terms should be elevated to classes, how associations should be structured, and which abstractions best capture the domain across heterogeneous documentation sources. It is important to emphasize that this manual baseline is not treated as an absolute ground truth, but rather as a domain-expert inter-

[2] https://eclipse.dev/emf/
[3] All pages were visited in February 2026.

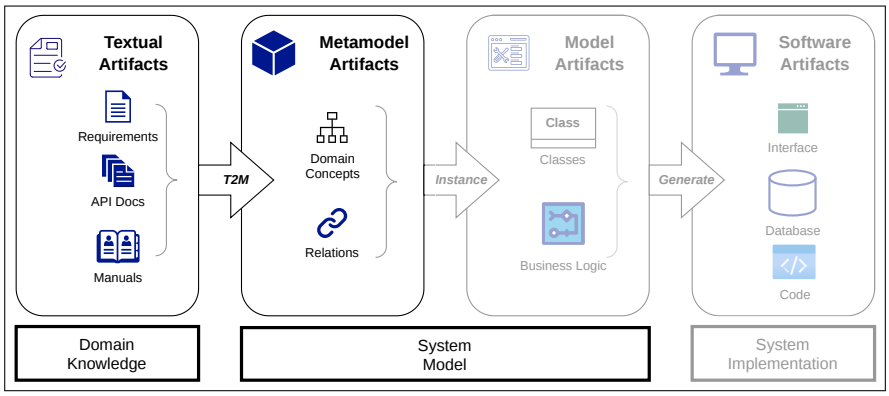

Figure 1 High-level overview of a classical MDE process from documentation to the final system implementation. This paper focuses on the T2M phase (highlighted)

pretation used as a reference artifact for comparison with the outputs produced by our automated approach.

Throughout the manual execution, we produced intermediate artifacts equivalent to those generated by the automated pipeline, enabling comparisons in Section 5. These include a list of extracted terminology with frequency indicators, as well as a consolidated set of candidate concepts aligned with domain semantics. Such artifacts are consistent with prior conceptual modeling workflows that rely on linguistic signals and structural cues to infer modeling elements from text (Arulmohan et al. 2023; Velardi et al. 2001).

The final artifact of this process is a textual PlantUML class diagram that can be converted into a visual representation. Figure 2 shows an excerpt of the generated diagram[4], illustrating how extracted concepts were mapped into classes and relationships.

The produced diagram reflects a draft metamodel derived solely from the documentation corpus. It includes both core domain concepts and structural relationships inferred during manual inspection. A simple analysis shows the centrality of the "Agent" concept as well as important concepts for agent engineering, such as "Tool" and "Memory", which are all essential during the conception of MAS. As such, it should be understood as a modeling starting point for further refinement, consistent with exploratory approaches to model generation from textual sources (Arulmohan et al. 2023).

4. LLM-Powered Multi-Agent Metamodeling Pipeline

This section presents the LLM-MAS developed to support the documentation-driven metamodeling. The goal is to act as a modeling aid by operationalizing a systematic process that transforms heterogeneous textual documentation into draft class diagrams. The pipeline is fully automated and produces inspectable intermediate artifacts that engineers can analyze and refine.

[4] Full PUML file and high-quality images are available in the companion repository: https://github.com/jameswpm/DATA_LLM-Powered_Multi-Agent_Systems_Exploring_Documentation-Driven_Metamodeling. This is valid for all excerpts from now on.

The system was designed as an MAS pipeline, where each agent is responsible for a well-scoped modeling task. This design aligns with traditional metamodeling and recent LLM agent practices, favoring task decomposition and artifact communication over monolithic prompts.

4.1. Pipeline Overview

Figure 3 illustrates the overall architecture of the proposed MAS. The pipeline follows a staged process, starting with raw documentation and ending with a textual PlantUML class diagram. For the scope of this paper, we do not detail how to collect the data, treating the input as a list of URLs pointing to the home pages of a set of agentic frameworks, as in the example proposed in Section 3.

At a high level, the pipeline consists of four main stages:

1. Documentation ingestion and preprocessing.
2. Terminology extraction and normalization.
3. Terminology scoring and consolidation.
4. Structural model generation.

Each stage is implemented by a dedicated agent, and communication between agents is mediated through explicit artifacts stored on disk (e. g., CSV files). This design enables traceability across the pipeline and allows intermediate results to be inspected independently of the final output.

The pipeline operates without necessary human intervention once initiated. Given a set of URLs as input, the system executes all stages sequentially and produces a draft class diagram as output. Although LLM-powered agents are inherently nondeterministic, their behavior is bounded within a structured workflow that enforces fixed stages, tool mediation, and explicit intermediate artifacts.

For each agent's prompt, we followed a prompt engineering approach based on the few-shot learning paradigm (Schulhoff et al. 2024), which defines a role for the agent, a detailed instruction for the task it should carry out, and a set of input/output

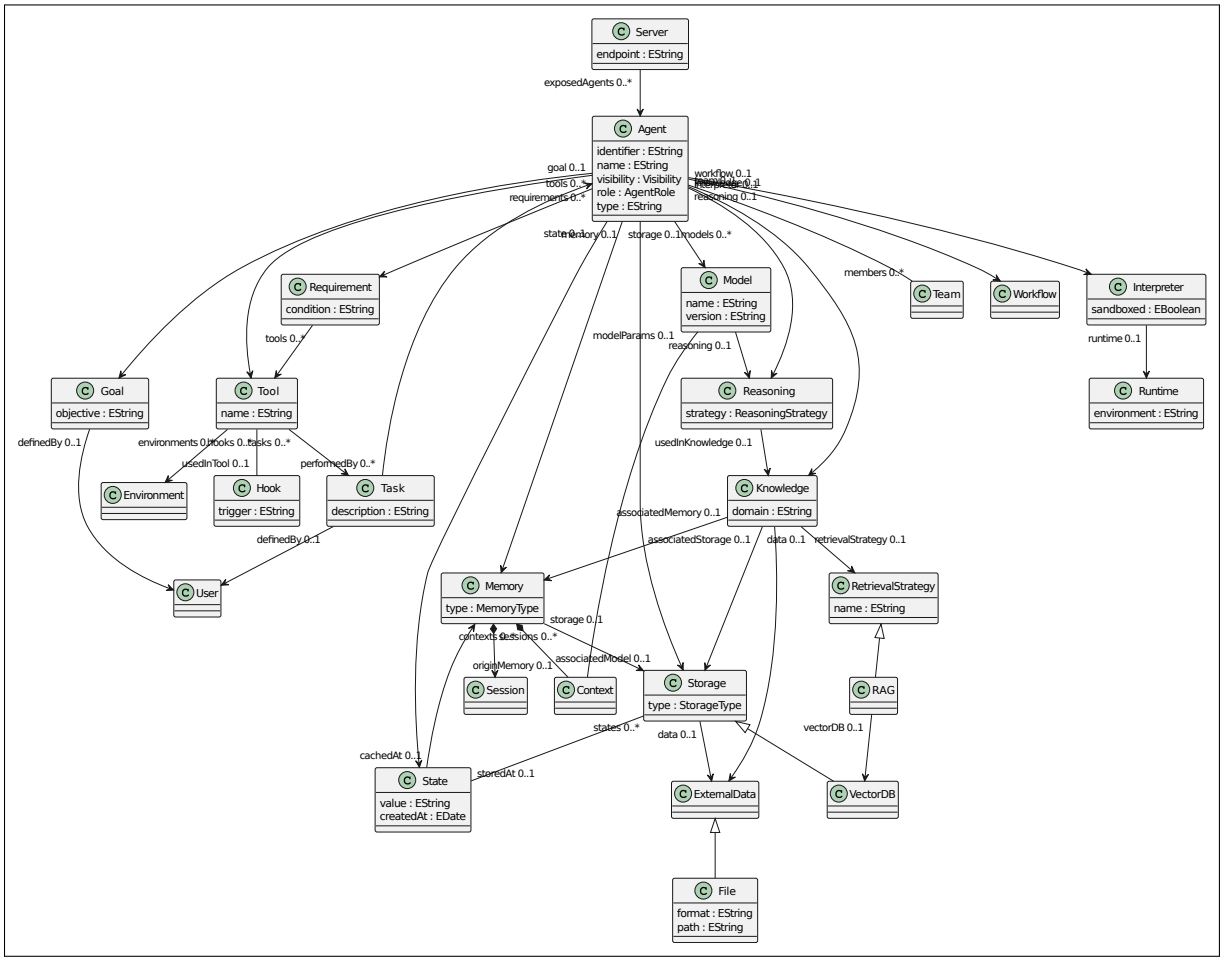

Figure 2 PlantUML class diagram of our manual baseline (Excerpt)

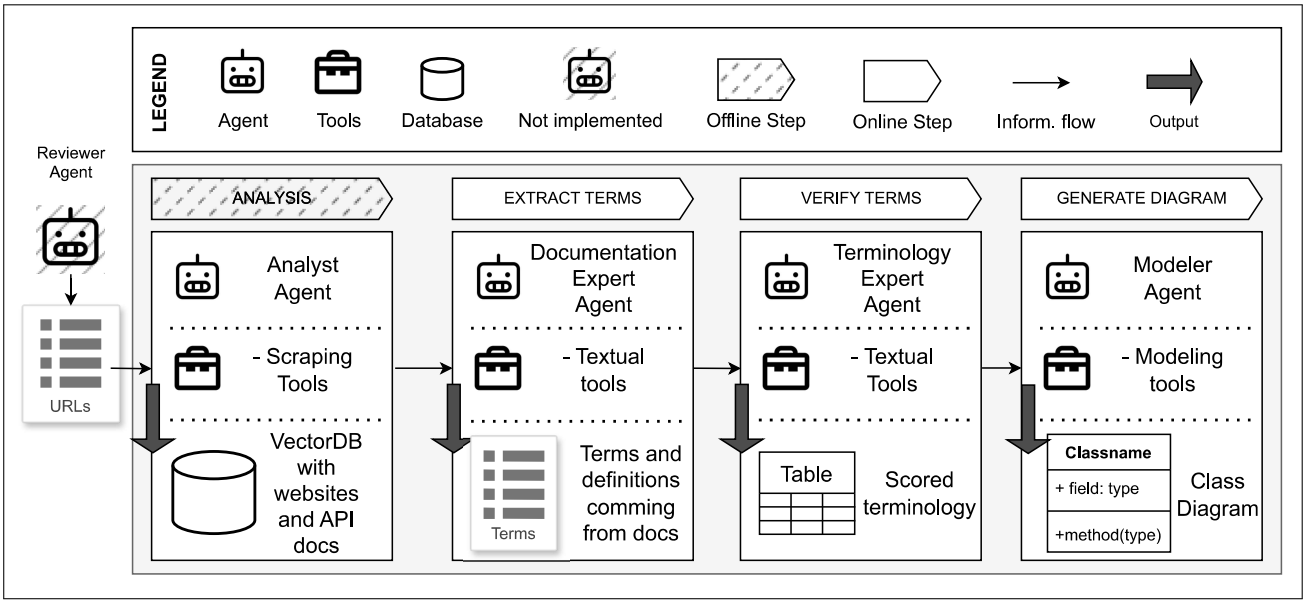

Figure 3 Illustrated full pipeline for the automated process of metamodeling creation (T2M phase)

examples. The companion repository[5] includes the input data, the generated artifacts and the agents' main prompts.

In the sequence, we describe each agent in terms of its modeling responsibility and the artifacts it consumes and produces.

4.1.1. Analyst Agent

The Analyst agent is responsible for ingesting and preprocessing documentation. Given a list of URLs, the agent retrieves the corresponding pages and stores their content in a Retrieval Augmented Generation (RAG)-ready persistent storage mechanism to support downstream processing.

From a modeling perspective, the Analyst agent establishes the raw knowledge base from which domain concepts may later be extracted. It does not perform any interpretation or modeling decisions beyond basic preprocessing. The process treats each given URL as the main source of truth for each evaluated framework, together with potential inner pages as helpers. To evaluate each page as a potential API-like page (i. e., containing domain knowledge), we use an LLM-powered step that classifies pages based on their textual content, as shown in the flowchart in Figure 4.

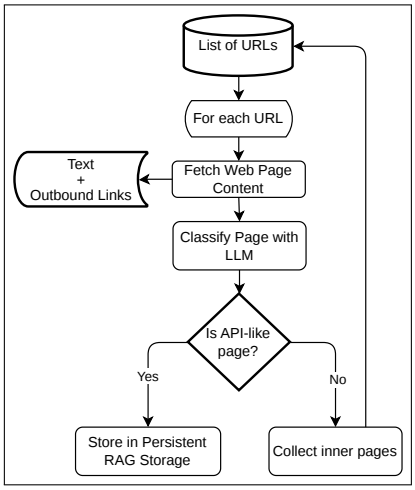

Figure 4 Flowchart for **Analyst Agent** data collection step

We decided to implement this agent using the Lanchain/LangGraph[6] framework for LLM-powered agents. This decision takes into consideration the ability enabled by the framework to define a fixed execution path (i. e., in the form of a graph/state-machine) with explicit LLM-powered nodes (e. g., the LLM page classifier). To improve the control and increase the agent's abilities, we implemented a set of simple web scraping tools based on HTML structure (e. g., collect text tags, collect link tags, etc.). For implementation, we choose ChromaDB[7] as the vector database.

4.1.2. Documentation Expert Agent

The Documentation Expert agent focuses on terminology extraction. Using the textual content collected by the Analyst agent, it identifies candidate terms that may correspond to domain concepts relevant for metamodeling.

The output of this agent is a preliminary terminology list, which may still contain redundant, ambiguous, or overlapping terms. At this stage, no attempt is made to assess the modeling relevance of the extracted terms, since its objective is coverage rather than precision.

This agent reflects a common early step in manual domain modeling, where engineers list candidate concepts before refining them.

Table 2 shows an excerpt of its output. The table is stored for later inspection and contains the term extracted from the documentation, a simple definition generated by the LLM, and the number of times it appears.

Regarding implementation details, we decided to use the Agno framework[8], which is intended for multi-agent systems and provides an easy-to-change set of tools, particularly well suited for this step, where we can provide a good set of textual tools to verify the LLM response and fix various syntactic errors.

Term	Definition	# of mentions in documentation
Agent	A component that provides[...].	48
Service	A component that provides [...].	38
API	Application Programming[...].	36

Table 2 An example output from **Documentation Expert Agent** step (Full table: about 302 rows).

4.1.3. Terminology Expert Agent

The Terminology Expert agent is responsible for evaluating and consolidating the extracted terminology. It assigns scores to candidate terms based on criteria such as frequency of occurrence, contextual relevance, and apparent semantic importance within the documentation.

In addition, the agent performs synonym consolidation, grouping together terms that appear to refer to the same underlying concept. The result is a reduced, structured terminology set intended to better reflect modeling-level abstractions.

Similarly to the previous step, the scoring process is explicit and materialized as an intermediate artifact (cf. 4.2), which can be inspected to understand why specific terms were retained or discarded, as well as to identify potential errors for future iterations.

Table 3 shows an excerpt of the agent output. It includes, again, the definition of each term and its count, along with a list of synonyms to identify which terms were considered duplicates by the LLM, allowing inspection and verification. The following columns are a sequence of scores as follows: *Frequency* Degree of cross-framework term agreement. *Relevance* Importance of the term for the target metamodel. *Consistency* Level of terminological uniformity across frameworks. Finally, *Score* is the simple sum of the previous rates. Except for the Relevance

[5] https://github.com/jameswpm/DATA_LLM-Powered_Multi-Agent_Systems_Exploring_Documentation-Driven_Metamodeling
[6] https://www.langchain.com/
[7] https://www.trychroma.com/
[8] https://www.agno.com/

Term	Definition	Count	Synonyms	Frequency	Relevance	Consistency	Score
Agent	[...]	6	['Sub-Agent']	1	2	1	4
Triggers	[...]	2	['Trigger']	1	3	1	5
Logger	[...]	1	[]	1	2	2	5

Table 3 An example output from **Terminology Expert Agent** step (Full table: about 60 rows).

score, which is subjective and so depends on the training data of the LLM (i. e., subject to hallucinations and misalignments), all other scores are enabled to be verified by a human-in-the-loop strategy in future implementations.

It was implemented using the SemanticKernel[9] agentic framework, mainly due to its modular architecture, which simplified its integration with tools. We also used textual tools for this step to fix various syntactic issues in the LLM output, including structured output, e. g., misalignments in the JSON formatting.

4.1.4. Modeler Agent

The Modeler Agent is responsible for generating the structural model. Using the consolidated terminology as input, it produces a PlantUML class diagram.

At this stage, the agent makes decisions regarding: which terms should be represented as classes, whose relationships should be introduce, and how attributes are assigned when applicable. To perform this inference, the main agent's prompt includes instructions and examples to guide its decisions. To support the production of syntactically valid artifacts, the agent uses UML tools for validation and can retry its operations when it produces invalid metamodels.

The output is a textual representation of a class diagram that can be rendered visually or further transformed into a concrete metamodel using standard MDE tooling. It captures structural hypotheses derived from documentation, but does not claim semantic completeness or correctness. Although syntactic validation is not claimed as well, the representation is intended to remain compatible with common modeling conventions, enabling subsequent refinement, validation, and integration within MDE workflows.

For the Modeler Agent, we implemented with LlamaIndex Workflows[10], which combines LLM-powered and code-generation steps, primarily used for syntax verification of the generated PlantUML.

4.2. Intermediate Modeling Artifacts

A key characteristic of the proposed pipeline is the explicit handling of intermediate artifacts. Each agent produces outputs that are persisted and passed to subsequent stages, rather than communicating exclusively through implicit prompts or internal states. These artifacts include raw extracted documentation content (stored as vector embeddings), lists of candidate terms, a scored and consolidated terminology table, and the final PlantUML class diagram.

This artifact-centered design supports traceability across the modeling process and enables engineers to analyze how specific modeling decisions emerge from the documentation. It also distinguishes the proposed approach from single-prompt or black-box LLM-powered modeling solutions, where intermediate reasoning steps are not explicitly accessible.

4.3. Implementation Details

Although the communication between agents is explicit and invoked in a pipeline, we also enabled an internal (optional) communication using the A2A protocol. This alternative aims to provide a future-proof implementation and easy coupling of our developed agents with different MAS architectures, consistently contributing to a collection of LLM-powered tooling in the MDE domain (cf. Section 8). In the same vein, the tool is ready for use in Dockerized environments and for deployment to Kubernetes instances for proper scalability. Finally, we decided to provide observability of the whole process, registering all the input-output pairs from the LLM as traces initially using Langsmisth[11], but always using OpenTelemetry[12] as the standard trace format, which enables it to be agnostic on the observability provider.

5. Evaluation

This section reports on our exploratory evaluation of the proposed LLM-MAS metamodeling pipeline. The objective is not to benchmark agent frameworks in isolation, nor to assess whether the generated metamodels are of better quality than human-created ones. Instead, we investigate whether the approach consistently produces model-like artifacts that can serve as a starting point for metamodeling activities. Given that, the choice of quantitative metrics used aims to enable the actual qualitative evaluation of the artifacts, excluding non-comparable artifacts (e. g. incomplete class-diagram, etc.).

In line with the tool's positioning as a modeling aid, the evaluation focuses primarily on structural characteristics and observable modeling behavior from a modeler's perspective, assessing the viability of LLM-powered agents for bootstrapping the metamodeling process.

5.1. Evaluation Objectives

The evaluation is guided by the high-level objectives shown in Table 4, assessing feasibility, stability, structural plausibility, and framework agnosticism (O1–O4). Although the evaluation

[9] https://learn.microsoft.com/en-us/semantic-kernel/overview/
[10] https://www.llamaindex.ai/workflows
[11] https://smith.langchain.com/
[12] https://opentelemetry.io/

is mainly qualitative, we complement it with a concise quantitative comparison against the manual baseline (cf. Section 3) and simpler prompting strategies.

To operationalize this evaluation, the automated pipeline described in Section 4 was executed across 26 distinct LLM-powered frameworks (i. e., the same dataset used for the manual baseline). The pipeline used a Local Ollama provider (for privacy preservation), with the `llama3.1:latest`[13] model, a temperature of 0 in all calls (hardcoded), and a CSV file containing a list of URLs as input (cf. Table 1). A total of 5 independent executions were performed, each producing intermediate artifacts (Section 4.2) and a final PlantUML class diagram.

ID	Objective
O1	Assess the feasibility of the proposed pipeline in generating renderable and syntactically valid class diagrams from documentation without human intervention during its execution.
O2	Observe the stability of the generated artifacts across multiple executions of the pipeline.
O3	Analyze the structural plausibility of the generated diagrams with respect to common metamodeling practices (e. g., presence of core abstractions, relationships, and hierarchies).
O4	Investigate whether the use of multiple LLM-powered agent frameworks influences the overall modeling outcomes in observable ways.

Table 4 Evaluation objectives

5.2. Quantitative Analysis

To support the qualitative observations with numerical clues, we compared four approaches: (i) manual baseline, (ii) LLM-MAS pipeline, (iii) prompt chaining without agent orchestration, (iv) a single direct prompt.

The manual baseline (i) serves as a domain-expert reference artifact. The purpose of the quantitative analysis is not to determine correctness, but to estimate structural alignment with this reference. To complement the evaluation, the prompt chaining (iii) is a simpler implementation of our proposal without the use of any agentic framework or agent orchestration (i. e. implemented as a single Python script using the same prompts executed in order). The input for (i), (ii), and (iii) is the same list of URLs (cf. Table 1). The purpose of this comparison is to assess the influence of agentic structural decisions on the artifacts produced. The execution we called single prompt (iv) is a simple prompt that receives no extra input to produce a metamodel for a given domain (e. g. agentic frameworks in our use case). This comparison aims to verify the influence of the input on the LLM's internal knowledge (i. e. the knowledge from its training) to ensure that the injection of documentation affects the final result.

The outputs of the 4 approaches are a textual representation of a class diagram (i. e. PlantUML), ready for later transformation into a metamodel. Artifacts were normalized before strict lexical matching. While semantic synonym alignment could increase recall, it would introduce additional subjective judgment into the evaluation process. Normalization included lowercasing identifiers, removing whitespace and special characters, and canonicalizing naming conventions (e. g., camelCase and snake_case).

Table 5 provides descriptive structural metrics. The LLM-MAS output exhibits greater structural richness (classes, relationships, attributes) than simpler prompting strategies, suggesting increased modeling capacity rather than mere verbosity.

Artifact	Baseline	MAS * Output	Prompt * Chaining	Single * Prompt
Classes	43	56	35.4	21.6
Relationships	72	89	68.8	18.2
Attributes	37	68	53.8	63.6

* Average of 5 runs.

Table 5 Numerical information about models used in the comparison

Matching was performed independently for classes, relationships, and attributes using normalized identifiers. We computed precision, recall, and F1-score relative to the reference artifact. These metrics quantify element-level overlap only and do not measure abstraction quality. Such aspects are addressed in the qualitative evaluation.

Table 6 summarizes the evaluation results[14].

The LLM-MAS pipeline consistently achieves higher F1 Scores across all element types than prompt chaining and single prompting. While recall remains moderate, the improvement in structural overlap supports the claim that agent orchestration contributes to more aligned and coherent artifacts. These numerical results should be interpreted as supportive indicators of structural plausibility rather than definitive performance benchmarks.

Artifact	Metric	MAS * Output	Prompt * Chaining	Single * Prompt
Classes	Precision	0.667	0.537	0.298
	Recall	0.837	0.372	0.130
	F1	0.720	0.371	0.174
Relationships	Precision	0.560	0.295	0.011
	Recall	0.675	0.147	0.002
	F1	0.588	0.118	0.004
Attributes	Precision	0.487	0.209	0.044
	Recall	0.876	0.189	0.054
	F1	0.612	0.149	0.042

* Average of 5 runs.

Table 6 Comparison of generated PlantUML artifacts against the manual baseline

Quantitative metrics are reported across five independent

[13] https://ollama.com/library/llama3.1, January 2026 version

[14] The companion repository includes scripts to automate this comparison and reproduce the results

executions in order to account for the stochastic nature of LLM-powered systems.

Performance of the LLM-MAS approach varies across runs, with a mean overall F1-score of 0.7697. This variability reflects the LLM's sensitivity to prompts and workflow organization, an aspect further discussed in Section 6.6.

5.2.1. Intermediate Artifacts We additionally compared intermediate artifacts using the same normalization protocol. The single-prompt approach is excluded because it produces no intermediate artifacts. Table 7 shows an overview of the number of collected terms.

Metric	MAS Output*	Prompt Chaining*
Terms extracted from documents	245.8	206
Terms after consolidation	34.6	100.2

* Average of 5 runs.

Table 7 Numerical information about intermediate artifacts used in the comparison

Although terminology extraction shows limited lexical overlap, the consolidation stage exhibits improved alignment in the LLM-MAS pipeline compared to prompt chaining. This suggests that the agent-based workflow contributes more significantly during abstraction and refinement phases than during raw term extraction. Table 8 summarizes our findings on this aspect.

Metric	MAS Output*	Prompt Chaining*
F1 (Terminology Extraction)	0.104	0.123
F1 (Consolidated Concepts)	0.176	0.173

* Average of 5 runs.

Table 8 Comparison of intermediate artifacts against the manual baseline

5.3. Evaluation Criteria

For the actual qualitative evaluation, we adopt a structural criteria that focus on observable properties of the generated artifacts to verify our evaluation objectives O1 to O4.

The criteria considered include:

- Diagram renderability: whether the pipeline produces a non-empty, renderable PlantUML class diagram.
- Concept coverage: the extent to which recurrent documentation concepts appear in the generated diagram.
- Redundancy and overlap: presence of duplicated or overly similar classes.
- Structural diversity: use of association, aggregation, or generalizations when applicable.
- Consistency across runs: degree of overlap between generated diagrams obtained from different executions.

These criteria are all accessed manually. Automation of this analysis using LLM judgment is expected for future iterations.

5.4. Comparative Observations

Based on manual inspection of five independent runs, the pipeline produced syntactically valid and renderable class diagrams in each case (O1).

We observed that certain core abstractions recur across runs, although variations in naming and organization are present (O2). These recurring elements suggest some degree of regularity in the generated structures, but no formal measure of stability is provided.

The generated diagrams also exhibit a range of structural constructs, including associations and inheritance relations, which appear generally consistent with common metamodeling practices (O3). This observation is qualitative and based on manual inspection rather than systematic measurement.

Finally, while differences exist between runs, the overall structural organization of the diagrams appears broadly similar at a high level (O4).

Overall, these observations should be interpreted as preliminary and qualitative. A more rigorous assessment of structural stability, diversity, and similarity would require dedicated quantitative metrics, which are left for future work.

5.5. Threats to Validity

First, the evaluation is limited to documentation-driven metamodeling and does not generalize to other modeling contexts. This limitation is increased by our use of a single case-study, making difficult to ensure a domain generalization and to the use of our own previous constructed metamodel as the baseline for initial comparisons, which also reduce potential generalizations. Second, reliance on LLM-MAS introduces inherent nondeterminism (even at temperature 0), which may affect reproducibility. We did not analyze how output metrics would change with better models (e. g., Google Pro 3 or Anthropic Opus) due to privacy constraints.

Finally, the evaluation does not aim to compare the proposed approach against human modeling efforts or existing metamodeling tools, as such comparisons would require a different experimental design.

These limitations are consistent with our exploratory and reporting-oriented scope. However, we propose some mitigation strategies in Section 8.

6. Discussion

This section discusses the main lessons learned from the design, implementation, and evaluation of the proposed LLM-MAS metamodeling pipeline. Our objective is to reflect on observed tendencies, identify design trade-offs, and outline implications for future work at the intersection of LLM-MAS and MDE, with particular focus on metamodeling.

6.1. LLM-Powered MAS as Modeling Aid

While the pipeline produces renderable class diagrams without human intervention during execution, the generated artifacts should be interpreted as drafts that support, rather than replace, human modeling activities. We expect the potential user to run

the pipeline multiple times and use the outcomes to support them in metamodeling activities.

By treating the output as a starting point for refinement, the system can tolerate partial inaccuracies, redundancy, or structural ambiguity, which is a phenomenon also common in early manual modeling phases. The generated intermediate artifacts (cf. Section 4.2) are intended to support refinement, including potential additional LLM-powered steps.

This observation reinforces the idea that LLM-powered systems can effectively assist in front-loading modeling effort, particularly in documentation-heavy contexts. A central lesson emerging from this is that LLM-MAS are better positioned as modeling aids rather than autonomous modeling solutions. This is complemented by scalability issues when dealing with large, more complex documentation, which potentially affect the performance of the outlined pipeline due to the LLMs' limited context window. While it felt out of the scope for the current project, we also pointed it out as a potential improvement for our future works (cf. Section 8)

6.2. Explicit Multi-Agent Decomposition

The decomposition of the metamodeling process into distinct agents provided several practical benefits. First, it enabled a clearer mapping between modeling activities and system components, making the pipeline easier to reason about and debug.

Additionally, the use of explicit intermediate artifacts improved traceability. Modeling decisions can be inspected rather than being embedded in opaque prompt chains.

Finally, the MAS architecture enabled the integration of multiple LLM-powered frameworks within a single system. This supports experimentation with heterogeneous agent implementations while maintaining a stable overall process, in a plug-and-play manner. The loosely coupled architecture of the MAS, open to extension through tools, also enables it to incorporate better tooling, targeting improvements at each step of the process.

6.3. Framework Heterogeinity

One of our design goals was to remain framework-agnostic at the architectural level. The use of four distinct LLM-powered frameworks across agents demonstrates that heterogeneous technologies can be composed into a coherent MAS for MDE, provided that agent responsibilities and artifacts are clearly defined.

Our experimentations suggest that framework-level differences may influence specific modeling outcomes. These differences, however, do not prevent the pipeline from producing structurally comparable artifacts. This indicates that the framework choice should be treated as a configuration parameter rather than a defining characteristic of the modeling approach. The quantitative comparison further suggests noticeable differences when no framework orchestration is used, raising research questions about how internal framework mechanisms affect final modeling results.

Similarly, the use of open standards for observability (e. g., OpenTelemetry) and communication (e. g., A2A) helped avoid lock-in to proprietary solutions. This proved important not only for enabling framework heterogeneity but also as a general design consideration when implementing agentic systems.

6.4. Limitations of Documentation-Driven Metamodeling

The proposed approach relies exclusively on textual documentation as input. While this makes the pipeline broadly applicable, it also introduces inherent limitations. Documentation may be incomplete, inconsistent, or written for purposes other than modeling, directly affecting the extracted concepts.

As a consequence, the generated metamodel tends to reflect explicitly stated documentation rather than implicit domain assumptions that human modelers often introduce based on experience. This further supports interpreting the output as a modeling aid and emphasizes the need for human validation and refinement.

A possible extension would be to couple the system with modeling and software repositories (e. g., Modelset[15]) or other curated knowledge sources (e. g., Wikidata[16]) to support disambiguation and enrichment.

6.5. Implications for MDE and Agentic Modeling Tools

From an MDE perspective, this work suggests that LLM-powered MAS can play a complementary role in early modeling phases, particularly in large or evolving ecosystems where documentation is abundant but underexploited. This is especially relevant in contexts similar to our running example (cf. Section 3), where heterogeneous agent frameworks often reflect opinionated or organization-specific design decisions rather than a shared standard.

Rather than embedding intelligence directly into modeling tools, the pipeline operates externally and produces standard modeling artifacts. This separation may ease integration with existing MDE toolchains and workflows, as well as with LLM-powered agents in broader ecosystems (e. g., A2A-based marketplaces).

More broadly, the work highlights the importance of process transparency and artifact-centered design when applying agentic systems to modeling tasks.

Based on the quantitative metrics, the MAS approach produced larger metamodels despite fewer consolidated terms (cf. Tables 6 and 8), likely because it can infer additional relationships and structural elements during the modeling phase. This behavior could inspire future research into this trade-off.

6.6. Importance of Agent Engineering Aspects

Although not formally evaluated due to our scope, modern LLM-powered Agent Engineering (Huyen 2025) is highly relevant to the development of LLM-MAS. Prompt engineering quality and agent configuration can influence outcomes as much as architectural decomposition. While prior work has investigated the impact of LLM hyperparameters and model selection on modeling activities (e. g., (Kebaili et al. 2024)), future research should also examine these factors within structured multi-agent orchestration settings.

[15] https://modelset.github.io/

[16] https://www.wikidata.org/wiki/Wikidata:Main_Page

7. Related Work

This section positions our work with respect to (i) pre-LLM approaches for model extraction from text, (ii) the use of LLMs for generating modeling artifacts, and (iii) agent-based approaches applied to modeling tasks.

Before the emergence of LLMs, several works investigated transforming textual artifacts into modeling representations using rule-based, NLP-driven, or heuristic approaches. Early studies focused on transforming user stories into UML artifacts through predefined linguistic patterns and syntactic analysis (Elallaoui et al. 2018). Subsequent work combined NLP with heuristics to generate UML models from textual requirements (Abdelnabi et al. 2020). Other approaches proposed systematic mappings from user stories to structural models using template-based extraction strategies (Kochbati et al. 2021). These works demonstrate that documentation-driven modeling is not new, but they are often coupled with handcrafted rules and domain-specific heuristics, which limit scalability and adaptability across heterogeneous documentation sources.

With the increasing interest in LLMs, research has explored their use for generating modeling artifacts directly from textual input and prompt engineering. Given that LLMs usually operate as text-in/text-out systems, they naturally lend themselves to interpreting requirements and producing textual representations of models. Recent works explore the generation of UML diagrams from natural language, e. g. NOMAD (Giannouris & Ananiadou 2025) aligns well with our proposal with a multi-agent architecture for generating UML class diagrams from structured requirements. Other studies investigate the use of LLMs as assistants for UML modeling tasks (B. Wang et al. 2024) or for generating sequence diagrams from requirements specifications (Ferrari et al. 2024). There are also proposals exploring the use of LLMs for domain-specific language construction (Alaoui Mdaghri et al. 2025). These works are complemented by streams investigating augmented approaches to overcome intrinsic LLM limitations, including human-in-the-loop (Silva et al. 2026) and automatic repair through self-refinement (Chen et al. 2026), for example.

While these approaches demonstrate the feasibility of LLM-driven (meta)model generation, most rely on single-agent or monolithic prompting strategies. Even when relying on MAS (e. g., NOMAD) or augmenting them with strategies to mitigate LLM errors, our work differs in explicitly structuring documentation-driven metamodeling as a workflow with intermediate artifacts that support traceability. Specifically, compared with NOMAD, which focuses on generating high-quality artifacts from structured requirements, our approach processes semi-structured and unstructured documentation and emphasizes the quality of intermediate artifacts (term extraction and classification) intended to support human modelers.

Finally, agent-based systems have long been studied in software engineering, including applications to modeling. In software engineering, recent work has begun investigating LLM-powered agents for structured development workflows. For example, approaches have been proposed for low-code or automated development pipelines leveraging coordinated LLM agents (Malamas et al. 2025). In a different context, MAS have been applied to digital twin environments, although not explicitly relying on LLMs (Pretel et al. 2025).

Specifically for modeling, the BESSER framework (Alfonso et al. 2024) provides an extensible modeling environment that could potentially serve as an integration point for intelligent assistants within MDE toolchains. Yang et al. also employ a strategy similar to ours. However, they rely on task decomposition and do not explicitly use an LLM-powered MAS. Such environments highlight the relevance of connecting agent-based reasoning with established modeling ecosystems.

In contrast to these works, our contribution focuses specifically on documentation-driven metamodeling and on the decomposition of this task into specialized LLM-powered agents. The emphasis is not on general software generation or development automation, but on supporting early-phase domain abstraction within MDE.

Agent engineering is relevant to industry and commerce outside academic environments as well. It is worth noting that recent industrial and open-source initiatives have introduced agent toolkits and orchestration environments (e. g., GitHub agent kits[17] and similar ecosystems) to structure LLM-powered workflows. Although not directly included on this paper, such tools are important for enhancing reproducibility, control, and lifecycle management of LLM-based pipelines and warrant future research.

8. Conclusions and Future Work

This paper proposes and evaluates an LLM-MAS for documentation-driven metamodeling, presenting a pipeline that systematically transforms heterogeneous textual documentation into draft class diagrams to support metamodeling. Our threefold contribution demonstrates the feasibility of agent-based modeling and its engineering challenges, as detailed in the following:

1. **Pipeline Design**: We compose LLM-powered agents into a fully automated pipeline generating renderable, structurally plausible class diagrams (preliminary drafts for refinement) compatible with MDE practices. The system's explicit modeling roles and intermediate artifacts ensure traceability, inspectability, and human-in-the-loop participation. Unlike related work focusing on prompt optimization, we prioritize problem decomposition and system design, while still leveraging prompt engineering.

2. **Evaluation**: We conduct a controlled quantitative evaluation comparing the LLM-MAS output to a reference artifact and simpler prompting baselines using element-level overlap metrics, which indicate structural convergence (rather than absolute correctness). The main exploratory qualitative analysis highlights the current state of maturity of LLM-agentic modeling and the lack of benchmarks for documentation-driven metamodeling.

3. **Lessons Learned on Agent Engineering for MDE**: Our multi-agent architecture integrates agents from diverse

[17] https://github.com/inngest/agent-kit

LLM frameworks, proving that heterogeneous technologies can coexist in a modeling pipeline. This suggests framework choice is a configurable parameter, not a hard constraint for MDE practices.

Overall, this work delivers empirical and methodological insights into LLM-MAS as modeling aids for MDE, balancing the potential of textual documentation with its limitations. Broadly, it opens directions for richer semantics, human validation, and large-scale empirical studies.

From an MDE perspective, future work includes larger empirical studies comparing the proposed pipeline with manual metamodeling in terms of time, quality, and completeness. To address generalizability, we plan to evaluate additional documentation-driven scenarios with similar input sparsity and modeling objectives, and study methods enabling domain generalization of the proposed tool across different documentation types rather than only API-like ones.

From an agent engineering perspective, improvements are needed to enhance reliability, reproducibility, and interoperability with workflow automation tools (e. g., n8n, Zapier, IFTTT) and other third-party agents. Given the absence of established benchmarks, we also aim to contribute evaluation artifacts by leveraging labeled public models and adapting existing workflow benchmarks (e. g., the one provided by the n8n team[18]). Still relevant for agent engineering, future work may address scalability issues posed by larger, more complex documentation, including non-API-like sources such as requirements.

Finally, at the tooling level, we plan to explore additional output formats (e. g., Ecore XML and Mermaid[19]) and tighter integration with modeling environments, including our tool under active development, Papyrus Web[20].

Acknowledgments

This work has received funding from the European Chips Joint Undertaking under Framework Partnership Agreement No 101139789 (HAL4SDV).

References

Abdelnabi, E. A., Maatuk, A. M., Abdelaziz, T. M., & Elakeili, S. M. (2020, December). Generating UML Class Diagram using NLP Techniques and Heuristic Rules. In *2020 20th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)* (pp. 277–282). (ISSN: 2573-539X) doi: 10.1109/STA50679.2020.9329301

Alaoui Mdaghri, A., Ouederni, M., & Chaari, L. (2025). MDE in the Era of Generative AI. In B. Ben Hedia, M. Ghazel, & B. Monsuez (Eds.), *Verification and Evaluation of Computer and Communication Systems* (pp. 113–127). Cham: Springer Nature Switzerland. doi: 10.1007/978-3-031-85356-2_8

Alfonso, I., Conrardy, A., Sulejmani, A., Nirumand, A., Ul Haq, F., Gomez-Vazquez, M., … Cabot, J. (2024). Building BESSER: An Open-Source Low-Code Platform. In H. van der Aa, D. Bork, R. Schmidt, & A. Sturm (Eds.), *Enterprise, Business-Process and Information Systems Modeling* (pp. 203–212). Cham: Springer Nature Switzerland. doi: 10.1007/978-3-031-61007-3_16

Arora, C., Sabetzadeh, M., Briand, L., & Zimmer, F. (2016, October). Extracting domain models from natural-language requirements: approach and industrial evaluation. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems* (pp. 250–260). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/2976767.2976769

Arulmohan, S., Meurs, M.-J., & Mosser, S. (2023, October). Extracting Domain Models from Textual Requirements in the Era of Large Language Models. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)* (pp. 580–587). Västerås, Sweden: IEEE. doi: 10.1109/MODELS-C59198.2023.00096

Babaalla, Z., Jakimi, A., & Oualla, M. (2025). LLM-Driven MDA Pipeline for Generating UML Class Diagrams and Code. *IEEE Access*, *13*, 171266–171286. doi: 10.1109/ACCESS.2025.3615828

Bjørner, D. (2023). Domain Modelling: A Foundation for Software Development. In *Theories of Programming and Formal Methods* (Vol. 14080, pp. 165–210). Springer. doi: 10.1007/978-3-031-40436-8_7

Bork, D. (2018). Metamodel-Based Analysis of Domain-Specific Conceptual Modeling Methods. In R. A. Buchmann, D. Karagiannis, & M. Kirikova (Eds.), *The Practice of Enterprise Modeling* (Vol. 335, pp. 172–187). Cham: Springer International Publishing. (Series Title: Lecture Notes in Business Information Processing) doi: 10.1007/978-3-030-02302-7_11

Bozyigit, F., Bardakci, T., Khalilipour, A., Challenger, M., Ramackers, G., Babur, O., & Chaudron, M. R. V. (2024, December). Generating domain models from natural language text using NLP: a benchmark dataset and experimental comparison of tools. *Softw Syst Model*, *23*(6), 1493–1511. doi: 10.1007/s10270-024-01176-y

Brambilla, M., Cabot, J., & Wimmer, M. (2012). *Model-Driven Software Engineering in Practice* (Vol. 1). (Journal Abbreviation: Synthesis Lectures on Software Engineering Publication Title: Synthesis Lectures on Software Engineering) doi: 10.2200/S00441ED1V01Y201208SWE001

Brandolini, L., Tisi, M., & Sottet, J.-S. (2024, July). A language workbench extension to generate conversational interfaces for domain-specific languages. In *First Large Language Models for Model-Driven Engineering Workshop (LLM4MDE 2024).* Enschede.

Burgueño, L., Di Ruscio, D., Sahraoui, H., & Wimmer, M. (2025, January). Automation in Model-Driven Engineering: A look back, and ahead. *ACM Trans. Softw. Eng. Methodol.*. (Just Accepted) doi: 10.1145/3712008

Chen, R., Shen, J., & He, X. (2026, January). MoRe: LLM-Based Domain Model Generation with Hybrid Self-Refinement. *Electronics*, *15*(6), 1239. doi: 10.3390/electronics15061239

[18] https://n8n.io/ai-benchmark/

[19] https://mermaid.js.org/

[20] https://gitlab.eclipse.org/eclipse/papyrus/org.eclipse.papyrus-web

Cheng, Y., Zhang, C., Zhang, Z., Meng, X., Hong, S., Li, W., … He, X. (2024, January). *Exploring Large Language Model based Intelligent Agents: Definitions, Methods, and Prospects.* arXiv. (arXiv:2401.03428 [cs]) doi: 10.48550/arXiv.2401.03428

Conrardy, A., & Cabot, J. (2024, July). From image to UML: First results of image-based UML diagram generation using LLMs. In *First Large Language Models for Model-Driven Engineering Workshop (LLM4MDE 2024).* Enschede.

Cámara, J., Troya, J., Burgueño, L., & Vallecillo, A. (2023, June). On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. *Softw Syst Model*, *22*(3), 781–793. doi: 10.1007/s10270-023-01105-5

de Lara, J., Guerra, E., & Cuadrado, J. S. (2015, February). Model-driven engineering with domain-specific meta-modelling languages. *Softw Syst Model*, *14*(1), 429–459. doi: 10.1007/s10270-013-0367-z

Derouiche, H., Brahmi, Z., & Mazeni, H. (2025, August). *Agentic AI Frameworks: Architectures, Protocols, and Design Challenges.* arXiv. (arXiv:2508.10146 [cs] version: 1) doi: 10.48550/arXiv.2508.10146

Di Sipio, C., De Oliveira, M. C. S., Di Ruscio, D., Nguyen, P. T., & Rubei, R. (2025, May). *Agentware in Software Engineering: A Taxonomy for Leveraging Llms-Based Multi-Agent Systems* [SSRN Scholarly Paper]. Rochester, NY: Social Science Research Network. doi: 10.2139/ssrn.5273078

Di Rocco, J., Di Ruscio, D., Di Sipio, C., Nguyen, P. T., & Rubei, R. (2025, June). On the use of large language models in model-driven engineering. *Softw Syst Model*, *24*(3), 923–948. doi: 10.1007/s10270-025-01263-8

Elallaoui, M., Nafil, K., & Touahni, R. (2018, January). Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques. *Procedia Computer Science*, *130*, 42–49. doi: 10.1016/j.procs.2018.04.010

Ferrari, A., Abualhaija, S., & Arora, C. (2024, June). Model Generation with LLMs: From Requirements to UML Sequence Diagrams. In *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)* (pp. 291–300). (ISSN: 2770-6834) doi: 10.1109/REW61692.2024.00044

Fowler, M. (2010). *Domain-specific languages*. Pearson Education.

Gartner, Inc. (2025). *Gartner Magic Quadrant for AI Application Development Platforms (document 7188230).* Gartner Research. Retrieved 2025-02-12, from https://www.gartner.com/en/documents/7188230 (Magic Quadrant research document, accessed via Gartner website)

Giannouris, P., & Ananiadou, S. (2025, November). *NOMAD: A Multi-Agent LLM System for UML Class Diagram Generation from Natural Language Requirements.* arXiv. (arXiv:2511.22409 [cs]) doi: 10.48550/arXiv.2511.22409

Hachm, Z., Calvar, T. L., Bruneliere, H., & Tisi, M. (2025, October). Towards LLM Agents for Model-Based Engineering: A Case in Transformation Selection. In *SAM 2025 - 17th System Analysis and Modelling conference.* Grand Rapids. doi: 10.1109/MODELSC68889.2025.00061

Hassouna, A. B., Chaari, H., & Belhaj, I. (2024, October). *LLM-Agent-UMF: LLM-based Agent Unified Modeling Framework for Seamless Integration of Multi Active/Passive Core-Agents.* arXiv. (arXiv:2409.11393 [cs]) doi: 10.48550/arXiv.2409.11393

Huyen, C. (2025). Chapter 6: Rag and agents. In *Ai engineering: Building applications with foundation models* (1st ed.). Sebastopol, CA: O'Reilly Media. Retrieved from https://www.oreilly.com/library/view/ai-engineering/9781098166298/

Jin, H., Huang, L., Cai, H., Yan, J., Li, B., & Chen, H. (2024, August). *From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future.* arXiv. (arXiv:2408.02479 [cs]) doi: 10.48550/arXiv.2408.02479

Kebaili, Z. K., Khelladi, D. E., Acher, M., & Barais, O. (2024, August). An Empirical Study on Leveraging LLMs for Metamodels and Code Co-evolution. *The Journal of Object Technology*, *23*(3), 1. doi: 10.5381/jot.2024.23.3.a6

Kochbati, T., Li, S., Gérard, S., & Mraidha, C. (2021, February). From user stories to models: A machine learning empowered automation. In *MODELSWARD 2022 - 9th International Conference on Model-Driven Engineering and Software Development* (Vol. 1, pp. 28–40). Online Streaming, France: SCITEPRESS - Science and Technology Publications. doi: 10.5220/0010197800280040

Kudo, T. N., Bulcão-Neto, R. d. F., Neto, V. V. G., & Vincenzi, A. M. R. (2023). Aligning requirements and testing through metamodeling and patterns: design and evaluation. *Requir Eng*, *28*(1), 97–115. doi: 10.1007/s00766-022-00377-5

Levendovszky, T., Lengyel, L., & Mészáros, T. (2009, September). Supporting domain-specific model patterns with metamodeling. *Softw Syst Model*, *8*(4), 501–520. doi: 10.1007/s10270-009-0118-3

Li, X., Wang, S., Zeng, S., Wu, Y., & Yang, Y. (2024, October). A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, *1*(1), 9. doi: 10.1007/s44336-024-00009-2

Lucassen, G., Robeer, M., Dalpiaz, F., van der Werf, J. M. E. M., & Brinkkemper, S. (2017, September). Extracting conceptual models from user stories with Visual Narrator. *Requirements Eng*, *22*(3), 339–358. doi: 10.1007/s00766-017-0270-1

Malamas, N., Tsardoulias, E., Panayiotou, K., & Symeonidis, A. L. (2025, November). Toward efficient vibe coding: An LLM-based agent for low-code software development. *Journal of Computer Languages*, *85*, 101367. doi: 10.1016/j.cola.2025.101367

Miranda, J. P., Radermacher, A., Baligand, F., Bonnail, J., Gérard, S., Bannerot, P., & Del Fabro, M. D. (2026). Bridging MDE and LLM-Based Agent Frameworks for Multi-Agent Systems: A Quasi-Systematic Review and Metamodel. In *Proceedings of the 14th international conference on model-based software and systems engineering* (pp. 33–44).

Montrieux, L., Yu, Y., Wermelinger, M., & Hu, Z. (2013, May). Issues in representing domain-specific concerns in model-driven engineering. In *2013 5th International Workshop on Modeling in Software Engineering (MiSE)* (pp. 1–6). San Francisco, CA, USA: IEEE. doi: 10.1109/

MiSE.2013.6595288
Mornie, M. N., Jali, N., Junaini, S. N., Mit, E., Shiang, C. W., & Saee, S. (2023, October). Visualisation of User Stories in UML Models: A Systematic Literature Review. *AIP*, *12*(2), 419–438. doi: 10.18267/j.aip.212
Neubauer, P., Bill, R., Kolovos, D. S., Paige, R. F., & Wimmer, M. (2019, September). Reusable Textual Styles for Domain-Specific Modeling Languages. *19th International Workshop in OCL and Textual Modeling, 2019, Munich, Germany, September 15-20, 2019*.
(OMG), O. M. G. (2024). *The unified modeling language (uml).* Retrieved from https://www.uml.org/ (Last accessed 24 January 2026)
Pan, F., Zolfaghari, V., Wen, L., Petrovic, N., Lin, J., & Knoll, A. (2024, October). Generative AI for OCL Constraint Generation: Dataset Collection and LLM Fine-tuning. In *2024 IEEE International Symposium on Systems Engineering (ISSE)* (pp. 1–8). (ISSN: 2687-8828) doi: 10.1109/ISSE63315.2024.10741141
Petrovic, N., Pan, F., Zolfaghari, V., & Knoll, A. (2025, March). LLM-based Iterative Approach to Metamodeling in Automotive. In *arXiv.org*.
Pretel, E., Zhinin-Vera, L., Navarro, E., López-Jaquero, V., & González, P. (2025, April). MAS4DT: A novel proposal for developing Digital Twins following a Multi-Agent system approach. *Journal of Systems and Software*, *222*, 112344. doi: 10.1016/j.jss.2025.112344
Roques, A. (2009). *PlantUML*. https://plantuml.com. (Accessed 2026)
Roy Chaudhuri, S., Natarajan, S., Banerjee, A., & Choppella, V. (2019, October). Methodology to develop domain specific modeling languages. In *Proceedings of the 17th ACM SIGPLAN International Workshop on Domain-Specific Modeling* (pp. 1–10). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3358501.3361235
Russell, S., & Norvig, P. (2020, May). The Foundations of Artificial Intelligence (Introduction). In *Artificial Intelligence: A Modern Approach* (4th edition ed., pp. 5–27). Hoboken: Pearson.
Saraiva, J. d. S., & Silva, A. R. d. (2008, October). Evaluation of MDE Tools from a Metamodeling Perspective. *JDM*, *19*(4), 21–46. doi: 10.4018/jdm.2008100102
Schulhoff, S., Ilie, M., Balepur, N., Kahadze, K., Liu, A., Si, C., … others (2024). The prompt report: A systematic survey of prompt engineering techniques. *arXiv preprint arXiv:2406.06608*.
Silva, J., Ma, Q., Cabot, J., Kelsen, P., & Proper, H. A. (2026). Towards Human-in-the-Loop LLM-Enabled Domain Modeling. In D. Bork, R. Lukyanenko, S. Sadiq, L. Bellatreche, & O. Pastor (Eds.), *Conceptual Modeling* (pp. 127–145). Cham: Springer Nature Switzerland. doi: 10.1007/978-3-032-08623-5_7
ThoughtWorks. (2025). *ThoughtWorks Technology Radar, vol. 32.* ThoughtWorks Technology Radar. Retrieved 2025-02-12, from https://www.thoughtworks.com/radar (ThoughtWorks, interactive report available online)
Tran, K.-T., Dao, D., Nguyen, M.-D., Pham, Q.-V., O'Sullivan, B., & Nguyen, H. D. (2025). Multi-agent collaboration mechanisms: a survey of LLMs. *arXiv preprint arXiv:2501.06322*.
Velardi, P., Missikoff, M., & Basili, R. (2001). Identification of Relevant Terms to Support the Construction of Domain Ontologies. In *Proceedings of the ACL 2001 Workshop on Human Language Technology and Knowledge Management.*
Wang, B., Wang, C., Liang, P., Li, B., & Zeng, C. (2024, July). How LLMs Aid in UML Modeling: An Exploratory Study with Novice Analysts. In *2024 IEEE International Conference on Software Services Engineering (SSE)* (pp. 249–257). doi: 10.1109/SSE62657.2024.00046
Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., … Wen, J. (2024, March). A survey on large language model based autonomous agents. *Front. Comput. Sci.*, *18*(6), 186345. doi: 10.1007/s11704-024-40231-1
Yan, B., Zhou, Z., Zhang, L., Zhang, L., Zhou, Z., Miao, D., … Zhang, X. (2025, June). *Beyond Self-Talk: A Communication-Centric Survey of LLM-Based Multi-Agent Systems.* arXiv. (arXiv:2502.14321 [cs]) doi: 10.48550/arXiv.2502.14321
Yang, Y., Chen, B., Chen, K., Mussbacher, G., & Varró, D. (2024, October). Multi-step Iterative Automated Domain Modeling with Large Language Models. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems* (pp. 587–595). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3652620.3687807

About the authors

James Pontes Miranda is research engineer at the CEA List. His work focuses on Generative AI for MDE. You can contact the author at james.pontesmiranda@cea.fr.

Ansgar Radermacher is a research director at CEA List. His work focuses on MDE tools, including code generation and AI assistance. You can contact the author at ansgar.radermacher@cea.fr.

Fabien Baligand is a research engineer at CEA List. His activities focus on agentic AI orchestration. You can contact the author at fabien.baligand@cea.fr.

Julie Bonnail is a research engineer at the CEA List. Her activities focus on integrating Generative AI into MDE tools. You can contact the author at julie.bonnail@cea.fr.

Kunal Suri is a research engineer at the CEA List. His work focuses on application of Generative AI to SDLC and Digital Twins. You can contact the author at kunal.suri@cea.fr.

Pascal Bannerot is a research engineer at the CEA List. His activities focus on architectural design and development of MDE tools. You can contact the author at pascal.bannerot@cea.fr.

Marcos Didonet Del Fabro is a research engineer at CEA List. His activities focus on developing and researching intelligent modeling architectures. You can contact the author at marcos.didonetdelfabro@cea.fr.