AMIGO: Agentic Model Instance Generation

Maximilian Hummel
Karlsruhe Institute of Technology
Karlsruhe, Germany
ABB Corporate Research
Mannheim, Germany
maximilian.hummel@kit.edu

Julian Roßkothen
Karlsruhe Institute of Technology
Karlsruhe, Germany
julian.rosskothen@kit.edu

Nathan Hagel
Department of Informatics
Karlsruhe Institute of Technology
Karlsruhe, Germany
nathan.hagel@kit.edu

Fatih Çatalcaya
Karlsruhe Institute of Technology
Karlsruhe, Germany
fatih.catalkaya@student.kit.edu

Heiko Koziolek
ABB Corporate Research
Mannheim, Germany
heiko.koziolek@de.abb.com

Ralf Reussner
Karlsruhe Institute of Technology
Kalrsruhe, Germany
reussner@kit.edu

Abstract

Generating metamodel-conforming instance models is a recurring task in Model-Driven Engineering (MDE), yet it remains tedious and tool-bound: instances are serialized as verbose, deeply nested XMI that today's large language models (LLMs) cannot reliably produce or edit. We present **AMIGO**, a general, harness-agnostic architecture for *agentic instance model generation*. Instead of letting the LLM emit XMI, an LLM agent expresses modeling intent through a small set of schema-typed `query`, `manage`, and `validate` tools that a metamodel-aware server exposes over the Model Context Protocol (MCP). The server executes every operation deterministically and persists a conforming instance, and the agent uses constraint feedback from instance model validation to self-correct. We demonstrate AMIGO through a ready-to-use instantiation for the Palladio Component Model (PCM): the *PCM-MCP* server, driven by an LLM agent, turns natural language specifications into a PCM instance that opens in the Palladio Bench. On two microservice systems (Corona-Warn-App and TeaStore), the agent generation produces OCL-valid instances in 596 of 600 runs across three open-weight LLMs, with structures approaching references (per-view Jaccard up to 0.96). AMIGO shows MCP is a viable integration layer between standard agent harnesses and MDE tooling.

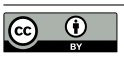
CCS Concepts

• **Software and its engineering** → **Software design engineering**; • **Computing methodologies** → *Artificial intelligence*.

Keywords

Instance Model Generation, LLM Agents, Model Context Protocol, Model-Driven Engineering, Palladio Component Model, Performance Modeling

ACM Reference Format:
Maximilian Hummel, Julian Roßkothen, Nathan Hagel, Fatih Çatalcaya, Heiko Koziolek, and Ralf Reussner. 2026. AMIGO: Agentic Model Instance Generation. In *ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion 2026), October 04–09, 2026, Málaga, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3837062.3838880



This work is licensed under a Creative Commons Attribution 4.0 International License.
MODELS Companion 2026, Málaga, Spain
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2903-4/26/10
https://doi.org/10.1145/3837062.3838880

Demo video (5 min, audio commentary):
https://doi.org/10.5281/zenodo.21759675
Artifact / source code:
https://doi.org/10.5281/zenodo.21759663

1 Introduction

Producing *instance models* which conform to their metamodels is a recurring, labor-intensive task. An engineer must translate informal design knowledge (e.g., natural-language descriptions, UML sketches, deployment notes) into a structurally valid model inside a dedicated workbench. Therefore, the task of producing instance models naturally lends itself to the strengths of LLM agents [3, 17]. However, conforming instance models are typically persisted as XMI in EMF-based tooling, and the verbosity and deep nesting of XMI cannot yet be generated or edited by LLM agents with any degree of reliability [14]. For example, cross-references are opaque identifiers, a single misplaced one breaks the whole file, and long serializations exceed the context budget within which LLMs can reliably reason [8]. On the other hand, agent harnesses like Claude Code and GitHub Copilot excel at working with tools exposed through schema-typed interfaces. Here, the de-facto standard is the *Model Context Protocol* (MCP). So far, however, MCP has only been proposed as a promising direction for modeling: A vision with a proof of concept rather than a realized, evaluated server that exposes a metamodel [6]. Consequently, LLM agents currently lack a metamodel-aware way of querying, modifying, and validating instance models. In this paper, we propose **AMIGO** - **A**gentic **M**odel **I**nstance **G**enerati**O**n. Agents can express modeling intent through a small set of schema-typed tools that a metamodel-aware server exposes over MCP: `query` (read the model), `manage` (create/update/delete), and `validate` (check OCL constraints). The server executes each call *deterministically* against an in-memory model that it persists as conforming XMI, and the agent builds the model incrementally, using `validate` feedback to repair violations before continuing. AMIGO is metamodel- and harness-agnostic: MCP for integration, an agent harness for orchestration, and a target metamodel for the tool surface apply to any EMF-based metamodel. The tool we demonstrate is a complete instantiation

of AMIGO for the *Palladio Component Model* (PCM) [4], a widely used metamodel for early-design performance prediction whose instances span five interrelated views with cross-view ID references. It comprises the *PCM-MCP* server exposing 21 PCM tools, driven by a single agent built with the OpenAI Agents SDK, turning informal inputs (natural language, UML/PlantUML, deployment manifests, load profiles) into a five-view PCM instance that opens directly in the Palladio Bench for simulation (Fig. 1). Envisioned users are software architects drafting performance models with an LLM assistant and educators teaching component-based modeling. The workflow is simple: mount a modeling workspace, point an agent harness at PCM-MCP, converse, inspect.

Contributions. (1) *AMIGO*, a general, harness-agnostic architecture for agentic instance model generation that replaces direct XMI synthesis with deterministic, schema-typed, self-validating tool calls over a target metamodel (§3). (2) A ready-to-use PCM instantiation — to our knowledge the first MCP server for the PCM, and the first realized and evaluated for agentic, multi-view instance-model generation — with a demo of a standard harness driving it (§4–§5). (3) An evaluation on Corona-Warn-App and TeaStore showing OCL-valid instances that approach expert references across three LLMs (§6).

2 Background

AMIGO rests on three technology pillars, which we introduce together with the target metamodel used in the demonstration.

Instance models and XMI. In EMF-based MDE, a metamodel defines element types and well-formedness constraints; an *instance model* is a graph of typed elements conforming to it, serialized as XMI [15]. Elements reference each other by generated identifiers.

Model Context Protocol (MCP). MCP [2] is a standardized interface for connecting LLMs with external tools [12]. An MCP server is, in effect, an LLM-optimized API: it advertises tools with JSON schemas for their arguments and structured results [10].

Agent harness (OpenAI Agents SDK). An LLM *agent* is an LLM in a loop that calls tools, observes the structured results, and decides the next action until the task is complete. We use the OpenAI Agents SDK [13] as the harness.

Palladio Component Model (PCM). PCM [4], the metamodel used in our demonstration, predicts response time, throughput, and resource utilization of component-based systems at the architecture level. A PCM instance comprises five views: the *repository* (components and interfaces), the *system* (assembly and wiring), the *resource environment* (nodes and links), the *allocation* (assemblies-to-nodes), and the *usage model* (workload).

3 The AMIGO Approach

AMIGO turns instance model generation into a sequence of small, schema-typed, deterministic tool calls made by LLM agents via MCP standard (Fig. 1). We describe its design principles and how the method instantiates for an arbitrary EMF metamodel; the concrete PCM realization follows in §4.

3.1 Design Principles

P1 — Intent through tools. The agent never reads or writes XMI; every interaction goes through a tool whose arguments are typed against the metamodel (creating a component takes a name and lists of provided/required roles, not raw XML). The server owns serialization, identifier generation, and cross-reference bookkeeping, and applies each call *deterministically* to an in-memory model that it persists as conforming XMI. This removes the failure potential of direct XMI synthesis and makes output reproducible and syntactically valid by construction. The LLM contributes only which elements to create and how to wire them.

P2 — A small, layered tool surface. Each advertised tool consumes context-window space, and performance degrades as that window fills [8]. AMIGO therefore exposes three layers over the metamodel: `create` tools that initialise an empty view, `query` tools that read model elements of a given type, and `manage` tools that fold create, update, and delete into one tool per model element kind via an `action` parameter.

P3 — Incremental validation and self-correction. AMIGO exposes constraint checking as a first-class `validate` tool, so the agent builds one view, validates it against the metamodel, and proceeds only once it is consistent; results are returned as structured issues with actionable messages, closing a feedback loop in which the agent repairs violations before they propagate. (Fig. 1).

3.2 Instantiating AMIGO for a Metamodel

Instantiating the method for a new EMF metamodel is mechanical. *(i) Tool surface:* group the metamodel into views and, per view, generate one `create` tool, one `query` tool, and a `manage` tool per top-level element kind, typed after the Ecore features; wrap the constraint set behind one `validate` tool. *(ii) Server:* back the tools with the metamodel's EMF API for in-memory mutation and XMI persistence and its validation framework for constraints. *(iii) Harness and prompt:* a system prompt frames the agent as a domain modeling expert and prescribes a modeling workflow with a mandatory validation step after each phase. §4 demonstrates the PCM instantiation of this recipe.

4 Tool: AMIGO for PCM

The demonstrated tool is AMIGO instantiated for PCM per §3.2: the *PCM-MCP* server exposing PCM as schema-typed tools, driven by a single OpenAI Agents SDK agent. From informal inputs it produces all five PCM views as OCL-valid XMI in a modeling workspace (`.repository`, `.system`, `.resourceenvironment`, `.allocation`, `.usagemodel`).

4.1 System Architecture

PCM-MCP is a stand-alone application that any of the common agent harnesses named in §2 connects to over MCP. The MCP server maps each tool onto factory methods of model elements in the generated PCM Ecore classes and onto the EMF validation framework for OCL checking.

4.2 Tool Surface

PCM-MCP exposes 21 tools (Table 1) realizing the three design principles defined in §3.1. Following the MCP specification, every tool returns structured output: the data the agent needs for subsequent steps (notably server-generated identifiers) plus a schema describing each field.

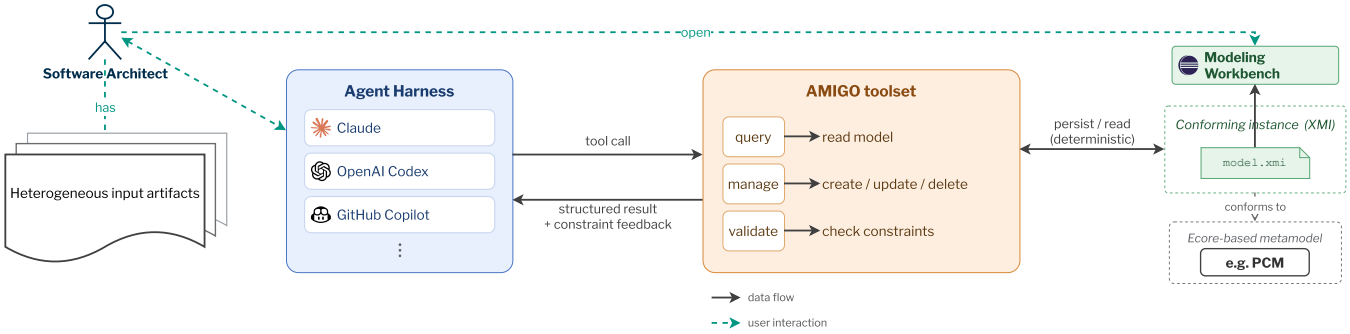


Figure 1: AMIGO architecture. A software architect interacts with an agent harness. The agent expresses modeling intent through tools over the Model Context Protocol (MCP), which executes operations deterministically and persists a conforming model instance. The persisted instance conforms to a pluggable Ecore-based metamodel and opens in the modeling workbench.

The tools fall into four categories. `create` tools instantiate an empty view file. `query` tools implement per-view reads; a `type` parameter filters the metamodel class and an optional `id` retrieves one element in detail, which the agent uses to recover identifiers for cross-view references. `manage` tools implement writes: each carries an `action` parameter (`CREATE`, `UPDATE`, or `DELETE`) plus view-specific parameters (e.g., component name, roles, and behavioral descriptions as JSON), keeping the surface small (P2). Finally, a single `palladio_validate` tool checks the PCM-OCL constraints; its `modelType` parameter restricts checking to one view, enabling the build-then-validate principle P3.

4.3 Agent Configuration

The agent and its system prompt are the only metamodel-specific configuration on the harness side. The task prompt, in contrast, is free-form: anything from a detailed specification with attached artifacts (§5) to a bare instruction to model the repository the harness is currently working in. The system prompt frames the agent as a precision-first PCM modeling expert and prescribes a five-phase workflow, one phase per view in dependency order: *repository* → *system* → *resource environment* → *allocation* → *usage model*. Its structure:

System prompt

 # Role You are a precision-first PCM modeler: translate the provided architecture artifacts into a complete, valid PCM instance by operating the PCM-MCP tools. Model only what the artifacts evidence — never invent elements, names, or values.

 # Instructions Read every artifact before the first tool call; on conflicts apply the precedence API spec → code → sequence diagram → architecture diagram → deployment. Recover identifiers via `query` tools — never guess an ID. After each phase, call `palladio_validate` and fix all errors before advancing.

 # Steps Phase 1 *repository* (evidence tiers for components, operations, and call chains; SEFF skeletons) → 2 *system* → 3 *resource environment* → 4 *allocation* (every assembly context mapped exactly once) → 5 *usage model* (workload derivation; entry-level calls only for actor-triggered flows).

5 Demonstration

Input. The demonstration replays the TeaStore login scenario (§6.1). The task prompt attaches a PlantUML system-context diagram, a login sequence diagram, a load profile text, and a textual deployment configuration. PCM-MCP is used with the OpenAI Agents harness. Harness independence follows from MCP rather than from a dedicated integration.

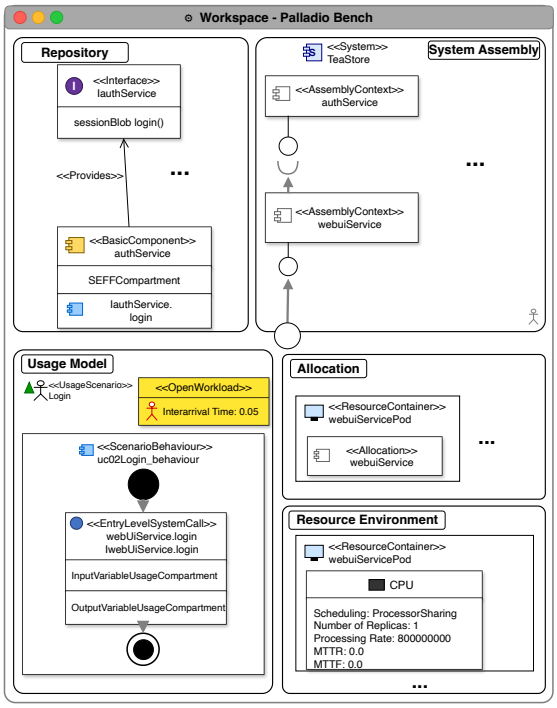


Figure 2: Generated PCM instance for a login scenario

Agent behavior. The Agent completes the task in 71 tool calls (Table 2): per view, one `create` call opens the file, `manage` calls add the elements, and `palladio_validate` closes the phase, while interleaved `query` calls recover the server-generated identifiers that wire cross-view references. The agent ends with a validation sweep over all five views; flagged violations surface as structured, repairable results.

Output. Fig. 2 shows parts of the generated valid instance opened in the Palladio Bench.

Table 1: Tools exposed by the PCM-MCP server (21 in total); {view} is one of five PCM views

Tool Name	#Tools	Description
`palladio_create_{view}`	5	Creates a new Palladio {view} file in the workspace.
`palladio_query_{view}`	5	Lists existing objects within a Palladio {view} file.
`palladio_manage_component`	1	Modifies a Palladio component.
`palladio_manage_interface`	1	Modifies a Palladio interface.
`palladio_manage_data_type`	1	Modifies a Palladio data type.
`palladio_manage_assembly_context`	1	Modifies an assembly context within the Palladio system.
`palladio_manage_connector`	1	Modifies a connector within the Palladio system.
`palladio_manage_system_role`	1	Modifies a provided or required system role.
`palladio_manage_resource_container`	1	Modifies a resource container within the Palladio resource environment.
`palladio_manage_linking_resource`	1	Modifies a linking resource within the Palladio resource environment.
`palladio_manage_allocation_context`	1	Modifies an allocation context.
`palladio_manage_usage_scenario`	1	Modifies a usage scenario. The workload and behavior actions are specified as JSON.
`palladio_validate`	1	Validates a PCM model against Palladio OCL constraints. Returns a list of validation issues if any exist, or confirms the model is valid.

Table 2: Tool-call chain of a demonstrated login run (71 calls; interleaved `query` calls omitted).

1 Repository	`create_repository` → `manage_interface` (6×) → `manage_component` (13×: 6 components + SEFFs) → `validate` ✓
2 System	`create_system` → `manage_system_role` → `manage_assembly_context` (6×) → `manage_connector` (7×) → `validate` ✓
3 Resources	`create_resource_environment` → `manage_resource_container` (6×) → `manage_linking_resource` → `validate` ✓
4 Allocation	`create_allocation` → `manage_allocation_context` (6×) → `validate` ✓
5 Usage	`create_usage_model` → `manage_usage_scenario` → `validate` ✓ → final sweep (5×)

6 Evaluation

We evaluate AMIGO's PCM instantiation with a Goal–Question–Metric plan. **Goal:** Assess PCM-MCP as a reliable way for an LLM agent to generate metamodel-valid, structurally faithful PCM instances. **Q1 (validity):** Do agents produce OCL-valid instances? **Q2 (structural fidelity):** How close are the instances to expert references per view? **Q3 (portability):** Do results hold across LLMs? For Q1 we run EMF's validation framework with the PCM-OCL constraints on every artifact. For Q2 we compute the per-view Jaccard similarity $JC = |R_{gen} \cap R_{ref}|/|R_{gen} \cup R_{ref}|$ over the typed elements, reusing the metric from AutoPCM [11]. The Jaccard metric element equality is topology-sensitive because it accounts for names (paired by semantic, embedding-based matching), structural properties, and cross-references. For Q3 we vary the LLM and inspect one full tool-call trace to open the agent black box.

6.1 Subjects and Setup

We reuse the subjects, use cases, and curated reference PCM instances from AutoPCM [11]: 20 use cases covering all five PCM views. **Corona-Warn-App (CWA)**, the official German SARS-CoV-2 contact-tracing app, contributes 11 use cases; **TeaStore** [16], microservice application, contributes 9. Each task prompt supplies the use case's diagrams, deployment information, and workload profile. The harness is the OpenAI Agents SDK; the system prompt prescribes the workflow of §4.3. We test PCM-MCP using the open-weight **Qwen3.5**, **Mistral-Small-4**, and **MiniMax-2.7** LLMs, all at $temperature = 0.1$. Each pair of LLM and use case is repeated 10 times, yielding 600 PCM instances.

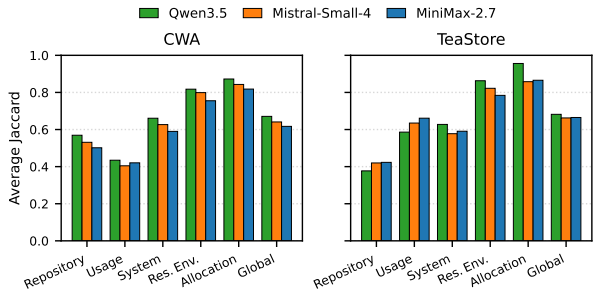


Figure 3: Average Jaccard per PCM view (mean over use cases and runs) for the three evaluated LLMs on CWA and TeaStore.

6.2 Results

Validity (Q1, Q3). 596 of 600 instances pass every PCM-OCL constraint (**99.3%**). In 260 of the 600 runs at least one intermediate `palladio_validate` call flagged violations that the agent then repaired before finishing. Regardless of the LLM, the deterministic tool surface plus the iterative `palladio_validate` loop yields near-complete conformance.

Structural fidelity (Q2, Q3). Fig. 3 reports the average per-view Jaccard scores. The three LLMs cluster tightly on both systems (global 0.62–0.67 on CWA, 0.66–0.68 on TeaStore), with Qwen3.5 slightly ahead. The allocation view scores highest (up to 0.96) and the resource environment follows, as deployment notes constrain them tightly. The repository (TeaStore) and usage (CWA) views score lowest, because they carry the richest typed structure (interfaces, signatures, behavior) and leave the most naming and granularity freedom: the agent places the right components and connections but names and decomposes them differently, so its defensible decisions diverge from the reference author's equally defensible choices.

Agent execution path. The Qwen3.5 trace of the demonstrated TeaStore login scenario (71 tool calls; Table 2) closely follows the five-phase workflow, invoking `palladio_validate` 10 times: once to close each view phase and five times as a final sweep, all passing. Together with the 260 runs in which flagged violations were repaired, this confirms that OCL-valid agentic modeling is driven by explicit constraint feedback (§3.1, P3).

6.3 Threats to Validity

Construct. OCL validity is necessary but not sufficient for semantic correctness, and Jaccard is only a lower bound on it. We mitigate reference subjectivity by reusing reviewed references from AutoPCM [11] and by reporting fidelity per view. **Internal.** LLM nondeterminism is addressed by 10 repetitions per pair and aggregated reporting. Also, since the system prompt could bias results, we provide it in our replication package. **External.** We evaluate two subject systems, one metamodel, three LLMs, and one harness; metamodel and harness generalization remains future work.

7 Related Work

LLM-based model generation. Several works generate models from informal input. Pan et al. [14] sidestep XMI by having the LLM emit an intermediate representation later compiled to XMI; others target specific notations such as graph models from text [7] — typically producing the model in a few prompts and treating it as text. Closest to our aim, EMF-Kaizen [1] embeds an LLM assistant in EMF that recommends metamodel-conforming model fragments at the abstract-syntax level, which users integrate via drag-and-drop; AMIGO complements this assistance with an autonomous agent that builds and validates the instance itself.

Agents for model-based engineering. Recent work applies LLM agents to MDE tasks such as model-transformation selection [9] and automatic system modeling [3]. The systematic study of Zhang et al. [17] reports that few LLM-based MDE approaches are agentic and most treat models as flat text, exactly the gap AMIGO addresses. The prior work AutoPCM [11] generates PCM with a fixed *multi-agent* pipeline; AMIGO is a complementary, simpler *single-agent* architecture, reusable across agent harnesses and metamodels.

Model Context Protocol. MCP is studied mainly from ecosystem and security perspectives [10, 12], and has only recently been brought to modeling. Cabot [6] proposes MCP as an integration layer between agents and modeling platforms and sketches a proof-of-concept, since developed into a fuller server for a specific low-code platform [5].

8 Conclusion and Future Work

We presented **AMIGO**, a general architecture for agentic instance model generation in which an LLM agent drives a metamodel-aware MCP server through schema-typed `query`, `manage`, and `validate` tools, building instances incrementally and self-correcting against constraint feedback. Its PCM instantiation generates OCL-valid instances in 99.3% of 600 runs across three LLMs, with structures approaching expert references. AMIGO turns standard agent harnesses into front-ends for instance modeling. Future work could automatically instantiate EMF metamodels and evaluate across harnesses.

Acknowledgments

This work was funded by Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF), the German Research Foundation (DFG) – SFB 1608 – 501798263, the Topic Engineering Secure Systems of the Helmholtz Association (HGF), and was supported by KASTEL Security Research Labs, Karlsruhe. This research was edited by our textician, Daniel Shea.

References

[1] Lissette Almonte, Jefferson Iván Rengifo, Esther Guerra, and Juan de Lara. 2026. EMF-Kaizen: An Intelligent Assistant for Domain-Specific Modelling and Meta-Modelling. *Journal of Object Technology* 25, 3 (2026). https://www.jot.fm/issues/issue_2026_03/a8.pdf

[2] Anthropic. 2024. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol. Accessed: 2026-06-20.

[3] Ludovic Apvrille and Bastien Sultan. 2024. System Architects Are not Alone Anymore: Automatic System Modeling with AI. In *Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering - MODELSWARD*. INSTICC, SciTePress, 27–38. doi:10.5220/0012320100003645

[4] Steffen Becker, Heiko Koziolek, and Ralf Reussner. 2009. The Palladio Component Model for Model-Driven Performance Prediction. *Journal of Systems and Software* 82, 1 (2009), 3–22. doi:10.1016/j.jss.2008.03.066

[5] Jordi Cabot. 2026. An MCP Server for Agentic Low-Code Development. https://modeling-languages.com/mcp-for-agentic-lowcode/. Accessed: 2026-08-02.

[6] Jordi Cabot. 2026. Vibe Modeling: Challenges and Opportunities. In *Advances in Conceptual Modeling*, Claudenir M. Fonseca, Anna Bernasconi, Sergio de Cesare, Ladjel Bellatreche, and Oscar Pastor (Eds.). Springer Nature Switzerland, Cham.

[7] Boqi Chen, Ou Wei, Bingzhou Zheng, and Gunter Mussbacher. 2025. Accurate and Consistent Graph Model Generation from Text with Large Language Models. arXiv:2508.00255 [cs.SE] https://arxiv.org/abs/2508.00255

[8] Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Babu Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A Huerta, and Hao Peng. 2025. Context Length Alone Hurts LLM Performance Despite Perfect Retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (Eds.). Association for Computational Linguistics, Suzhou, China, 23281–23298. doi:10.18653/v1/2025.findings-emnlp.1264

[9] Zakaria Hachm, Théo Le Calvar, Hugo Bruneliere, and Massimo Tisi. 2025. Towards LLM Agents for Model-Based Engineering: A Case in Transformation Selection. In *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 421–431. doi:10.1109/MODELS-C68889.2025.00061

[10] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. arXiv:2503.23278 [cs.CR] https://arxiv.org/abs/2503.23278

[11] Maximilian Hummel, Dominik Fuchß, Sophie Corallo, Nathan Hagel, Jan Keim, Heiko Koziolek, and Ralf Reussner. 2026. AutoPCM: Automated Generation of Architectural Performance Models from Early Design Artifacts using LLM Agents. In *20th European Conference on Software Architecture*.

[12] Zhiwei Lin, Bonan Ruan, Jiahao Liu, and Weibo Zhao. 2025. A Large-Scale Evolvable Dataset for Model Context Protocol Ecosystem and Security Analysis. arXiv:2506.23474 [cs.CR] https://arxiv.org/abs/2506.23474

[13] OpenAI. 2025. OpenAI Agents SDK. https://openai.github.io/openai-agents-python/. Accessed: 2026-06-20.

[14] Fengjunjie Pan, Nenad Petrovic, Vahid Zolfaghari, Long Wen, and Alois Knoll. 2025. Llm-enabled instance model generation. In *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 586–595.

[15] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. 2008. *EMF: Eclipse Modeling Framework*. Addison-Wesley.

[16] Jóakim von Kistowski, Simon Eismann, Norbert Schmitt, André Bauer, Johannes Grohmann, and Samuel Kounev. 2018. TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research. In *Proceedings of the 26th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems* (Milwaukee, WI, USA) *(MASCOTS '18)*.

[17] Weixing Zhang, Bowen Jiang, Yuhong Fu, Haowei Cheng, Maximilian Hummel, Vincenzo Scotti, Nathan Hagel, Jialong Li, Georg Grossmann, Markus Stumptner, et al. 2027. Large language models in model-driven engineering: a systematic mapping study. *Empirical Software Engineering* 32, 1 (2027), 3.