Agentic Model Transformation: Service-Mediated Orchestration for Transformation Development

Duy Dao
Mälardalen University
Sweden
khanh.duy.dao@mdu.se

Alessio Bucaioni
Mälardalen University
Sweden
alessio.bucaioni@mdu.se

Antonio Cicchetti
Mälardalen University
Sweden
antonio.cicchetti@mdu.se

Abstract

Model transformation is a core mechanism in Model-Driven Engineering (MDE), but its use requires expertise in metamodels, transformation languages, execution engines, runtime configuration, and diagnostics. Although LLMs can generate and revise transformation artifacts, direct prompt-to-output generation does not by itself provide an executable MDE workflow grounded in metamodel conformance and transformation rules execution.

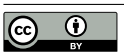
This paper proposes *agentic model transformation*: a service-mediated workflow in which an LLM agent generates candidate transformation specifications and coordinates MDE services for metamodel access, example retrieval, checking, execution, diagnostics, and repair. Model Context Protocol is used to expose these services to the agent. When execution succeeds, target artifacts are produced through the configured transformation infrastructure, serialized by the target modeling environment, and loadable against the registered target metamodel.

Early results from adopting the proposed framework for the UML-to-Java and SysMLv2-to-AAS cases show that constructing and repairing executable transformations is possible. The results also show that executable success and metamodel conformance are only first checks; structural completeness, mapping coverage, and semantic adequacy require additional MDE services. The framework, therefore, shifts LLM support for model transformation from one-off artifact generation towards service-supported transformation development, where specifications, diagnostics, and execution traces can be inspected, reused, and extended.

Keywords

Agentic Model Transformation, Service-Mediated Orchestration, Large Language Models, Execution Diagnostics

ACM Reference Format:
Duy Dao, Alessio Bucaioni, and Antonio Cicchetti. 2026. Agentic Model Transformation: Service-Mediated Orchestration for Transformation Development. In *ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS 2026), October 04–09, 2026, Málaga, Spain.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3822455.3838763



This work is licensed under a Creative Commons Attribution 4.0 International License.
MODELS 2026, Málaga, Spain
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2809-9/2026/10
https://doi.org/10.1145/3822455.3838763

1 Introduction

Model transformations are a core mechanism in Model-Driven Engineering (MDE), supporting derivation, synchronization, migration, and generation across modeling languages and technical spaces [7, 23]. However, using transformations effectively requires expertise in metamodels, transformation languages, execution engines, runtime configuration, serialization, and diagnostics [16]. This dependence remains an adoption barrier for domain experts and engineers who need to define, adapt, or understand transformations without specialized MDE expertise [19].

Large Language Models (LLMs) can support transformation development by interpreting modeling artifacts, generating transformation specifications, explaining diagnostics, and revising outputs based on feedback [9, 17, 20, 21]. However, direct prompt-to-output generation does not by itself provide an executable MDE workflow. Generated artifacts may appear plausible but fail to execute, or they may execute while only partially realizing the intended source-to-target mappings [8]. Retrieval augmentation and fine-tuning may improve generation quality, but they do not by themselves connect LLMs to the metamodels, transformation engines, runtime configuration, and diagnostics required for transformation development [15, 18].

This paper envisions addressing this gap through *agentic model transformation*: a service-mediated workflow in which an LLM agent coordinates MDE services rather than replacing transformation languages or execution engines. The agent retrieves metamodel and example information, generates candidate Model Transformation Language (MTL) specifications, invokes checking and execution services, inspects diagnostics, and revises failed attempts. Transformation operations remain delegated to MDE infrastructure, including metamodel access, runtime preparation, execution, serialization, and loading in the selected language workbench.

The workflow treats LLM-generated transformations as candidate specifications that must pass through the configured transformation infrastructure. Target artifacts are produced by the transformation engine rather than by direct LLM text generation. Thus, successful execution produces artifacts serialized by the target modeling infrastructure and loadable against the registered target metamodel. This provides an operational basis for metamodel conformance in the configured environment, while structural completeness, mapping coverage, and semantic adequacy can be assessed separately.

We implement the workflow through a framework prototype using the Model Context Protocol (MCP) as an integration mechanism for exposing model-aware capabilities as structured services [13]. The prototype provides services for metamodel inspection, example retrieval, runtime support, checking, compilation, execution,

and diagnostic feedback. It also keeps generated transformations, service calls, diagnostics, and repair steps inspectable outside the LLM interaction, so transformation specifications and diagnostic traces can be retained, reused, and extended.

We examine the framework in two transformation cases: UML-to-Java and SysMLv2-to-AAS. The assessment uses the Eclipse Modeling Framework (EMF) language workbench, ATL, and QVTo model transformation languages with GPT-5, DeepSeek R1, and Sonnet 4, resulting in 120 runs. The focus is on executable success, diagnostic-supported repair, and structural similarity to ground-truth artifacts. The assessment is used to study whether transformation development can be organized as explicit MDE service coordination, not to rank LLMs or transformation languages.

As an outcome, this paper contributes an agentic framework for LLM-assisted model transformation development and preliminary evidence that such a workflow can support executable transformation generation, diagnostic-based repair, and inspection of adequacy limits. The results also show that executable success and metamodel conformance are not sufficient to ensure structural completeness or semantic adequacy. The framework, therefore, shifts LLM support for model transformation and related MDE tasks from one-off artifact generation toward repeatable transformation development, where specifications, diagnostics, and execution traces can be retained, inspected, reused, and extended through the use of service-mediated approach.

2 Background and Related Work

This section positions the proposed framework with respect to model transformation environments and recent LLM-based support for MDE. The main distinction is architectural: the LLM agents do not directly produce the target artifact; instead, they coordinate existing MDE infrastructure through explicit services.

2.1 Model transformation

Model transformations automate derivation, synchronization, migration, and generation across modeling languages, abstraction levels, and technical spaces. Their use depends on metamodels, transformation specifications, execution engines, runtime configuration, serialization conventions, and diagnostics [7, 23]. This dependence makes transformation development difficult for users without dedicated MDE expertise [16].

In this paper, this dependence is not bypassed but exposed through services. A transformation result is not treated as a textual artifact that only resembles the target language. It must be produced through the configured transformation infrastructure and loaded against the registered target metamodel. This keeps metamodel conformance, execution, and serialization within the MDE workflow.

2.2 LLM support for MDE

LLMs have recently been applied to MDE tasks involving natural-language requirements, metamodel fragments, serialized models, examples, generated artifacts, and diagnostic messages. Arulmohan et al. [1] use LLMs to extract domain models from textual requirements. Chen et al. [5] compare LLMs for automated domain modeling. Clarisó and Cabot [6] propose model-driven prompt engineering for structuring prompts in modeling tasks. These works show how LLMs can support model construction and modeling assistance.

Other work connects LLMs with Model-Driven Architecture (MDA)-style and reverse-engineering pipelines. Babaalla et al. [2] generate UML class diagrams and Python code from textual specifications using an intermediate DSL for validation, editing, and traceability. Siala and Lano [24] study model-driven reverse engineering with LLM4Models, extracting UML class diagram representations from Java programs through abstraction rules, fine-tuning, and pre- and post-processing. These works connect LLMs with model extraction and generation, but they do not address the development of model transformation specifications through the transformation workflow itself.

Hachm et al. [14] are closest to the agentic setting. They expose existing ATL transformations as tools through an MCP-based ATL server and use tool filtering to help an agent select an appropriate transformation. Our work addresses a different activity: the agent generates and repairs the transformation specification itself, while MDE services provide the tool support needed to check, execute, and diagnose candidate specifications.

Existing work mainly studies LLMs as generators of modeling artifacts, components in MDA-style pipelines, reverse-engineering aids, or agents for selecting existing transformations. This paper instead focuses on transformation development itself. The LLM proposes candidate MTL specifications, while checking, execution, serialization, loading, and diagnostics remain delegated to MDE infrastructure.

2.3 Why generation support is not enough

The difference can be seen in a typical transformation request: a user may ask for a Java model from a UML class model, or an AAS model from a SysMLv2 model. In a direct prompt-to-output setting, the LLM generates the target artifact directly, for example, as XMI or another serialized representation. The result may contain plausible Java or AAS elements, but still include invalid references, wrong containment structures, unsupported target elements, or serialization details that prevent loading against the registered target metamodel. In this setting, metamodel conformance and loadability to the selected language workbench (e.g., EMF) are not ensured during artifact production.

Retrieval augmentation and fine-tuning can improve generation by adding examples, documentation, or adaptation to a target syntax [15, 18]. However, they do not by themselves change where the target artifact is produced. As shown in Figure 1, the final artifact may still be generated outside the configured transformation infrastructure.

The envisioned workflow changes this path. The LLM proposes a transformation specification in the selected model transformation language, while the configured MDE infrastructure performs checking, execution, and serialization. As shown in Figure 2, the agent accesses MDE capabilities as explicit services. The produced target artifact is generated by the transformation engine and checked through EMF loading against the registered target metamodel. In

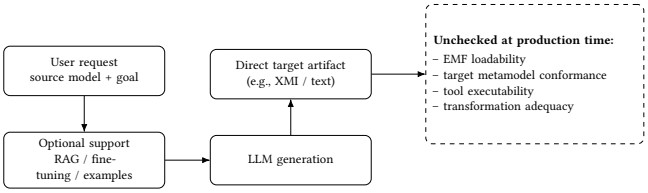


Figure 1: Direct prompt-to-output generation.

this sense, the proposed idea is worked out as an executable workflow: candidate specifications are generated, checked, executed, diagnosed, and revised through explicit service calls.

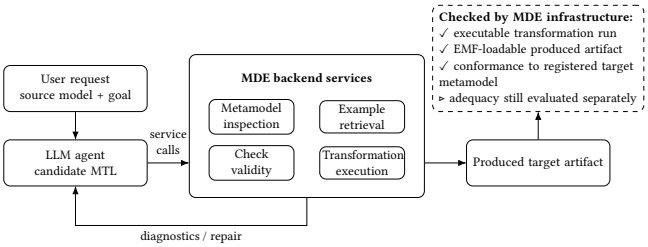


Figure 2: Service-mediated agentic transformation.

This distinction separates tool-supported execution from transformation adequacy. Execution and EMF loading show that the candidate specification can run in the configured environment and produce a loadable target artifact. They do not, by themselves, show that the transformation captures all intended source-to-target mappings. Adequacy must therefore be evaluated with additional checks, as discussed in the results and future work.

2.4 Paper positioning

We position this work between direct LLM generation and traditional model transformation environments. Direct generation can lower the entry barrier, but may leave artifacts outside metamodel checking and execution [9, 17]. Traditional environments provide transformation languages, engines, conformance checks, execution support, and diagnostics, but require users to coordinate these steps manually.

To the best of our knowledge, as of June 2026, the MDE literature has not focused on transformation development as a service-mediated workflow in which LLM-generated MTL specifications are checked, executed, diagnosed, and repaired through MDE infrastructure. The contribution is therefore not a new transformation language or execution engine. It is a framework that organizes transformation development as service operations around existing MDE infrastructure. MCP is used only as the integration mechanism for exposing these services to the agent.

The agent does not replace metamodels, transformation languages, execution engines, or transformation tooling. It orchestrates them while supporting task interpretation, candidate transformation generation, diagnostic interpretation, and repair. This gives the LLM access to the model transformation workflow while preserving inspectability and tool support expected in MDE [4, 13, 22]. It also points beyond one-off generation: generated specifications, service calls, diagnostics, and repair steps can be retained and reused in later transformation tasks.

3 Research Approach

This study follows an iterative engineering research process to design and preliminarily assess the framework [3]. The objective is not to make definitive claims about individual LLMs or MTLs, but to examine whether transformation operations can be externalized as services that LLM agents can coordinate. The process focuses on making transformation operations explicit: metamodel inspection, checking, execution, diagnostics, and repair are delegated to MDE services rather than left to implicit LLM behavior.

We designed and refined the framework through successive iterations. The assessment examines whether an LLM agent can coordinate explicit MDE services to generate executable ATL and QVTo specifications, use diagnostics to revise failed attempts, and produce target artifacts that are EMF-loadable against the registered target metamodel. It therefore provides feasibility evidence and initial insight into the relation between executable success and structural completeness.

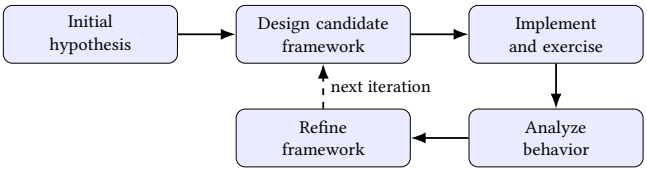


Figure 3: Iterative engineering research process.

Figure 3 summarizes the process. Across the iterations, the framework was refined around four concerns: separating LLM coordination from transformation execution, making artifacts and diagnostics inspectable, supporting diagnostic-based repair, and clarifying service boundaries between the agent, transformation services, and execution support.

The framework evolved through four iterations. The first established feasibility with a centralized prototype in which orchestration, transformation handling, and execution support were placed in one component. The second introduced service interfaces for transformation concerns, artifacts, and diagnostics. The third separated runtime support from transformation services, clarifying the boundary between orchestration, transformation functionality, and execution. The fourth revised the client environment to support multiple API-based LLMs, reducing dependence on a single LLM interface.

The preliminary assessment uses the final framework iteration. It considers executable success, first-attempt success, retries to success, recovery after failed initial attempts, and structural similarity to ground-truth artifacts. These measures characterize executable transformation generation, diagnostic-based repair, and structural output quality. Repeated runs for stochastic variation are left for future work.

4 Agentic Model Transformation Framework

This section defines the framework and the allocation of responsibilities between LLM-based coordination and MDE services. The LLM interprets the user request, invokes services, generates candidate MTL specifications, and revises them using diagnostics. MDE services provide metamodel access, examples, runtime preparation, checking, execution, and diagnostic feedback. Thus, model transformation is exposed as an inspectable workflow while transformation operations remain delegated to MDE infrastructure.

The design follows three principles.

- First, operations that affect transformation validity are delegated to explicit MDE services.
- Second, generated transformations, service calls, diagnostics, and repair steps remain inspectable.
- Third, the framework should not be tied to one client, LLM backend, or transformation language implementation.

The goal is to keep transformation validity checkable through MDE services while the LLM supports coordination, candidate generation, diagnostic interpretation, and repair.

4.1 System Framework

The framework's architecture is organized into four layers, shown in Figure 4: orchestration, MCP, backend services, and infrastructure.

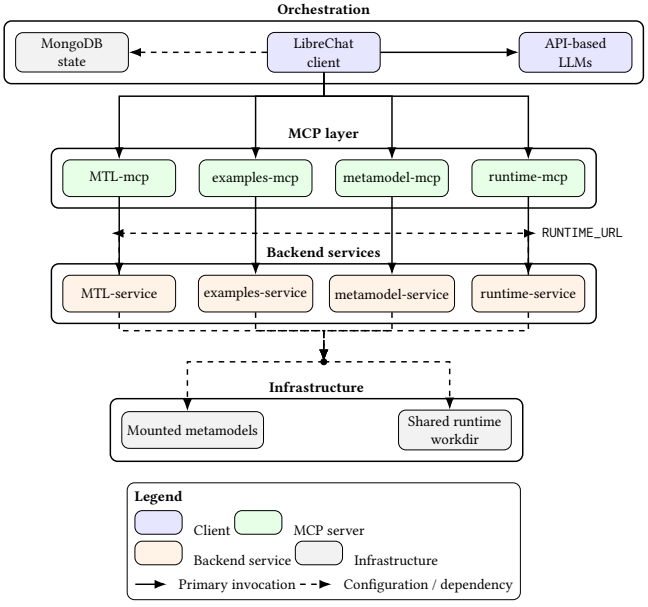


Figure 4: Final system framework

The orchestration layer contains the LLM-enabled client, API-based LLMs, and client state. In this framework, LibreChat serves as the LLM-enabled client, providing the user interface and coordinating interactions between users, LLMs, and the available services. The layer supports user interaction and workflow coordination, but does not execute transformations. The LLM obtains context, generates candidate specifications, and revises them through service calls.

The MCP layer exposes model-aware capabilities as callable services for MTL support, examples, metamodels, and runtime support. The MTL MCP provides interfaces for checking, execution, and diagnostic feedback. In the prototype, these interfaces are realized through ATL- and QVTo-specific operations. The MCP layer, therefore, defines the protocol boundary through which the agent invokes the MDE services.

The backend layer implements the MDE services. The `metamodel-service` provides source and target metamodel access, the `examples-service` retrieves source-target examples, the `MTL-service` checks and executes generated transformations, and the `runtime-service` prepares inputs and shared resources.

The infrastructure layer provides mounted metamodels and a shared runtime working directory. Together, these layers expose MDE infrastructure through services while preserving existing transformation engines and EMF-based resources.

4.2 Workflow

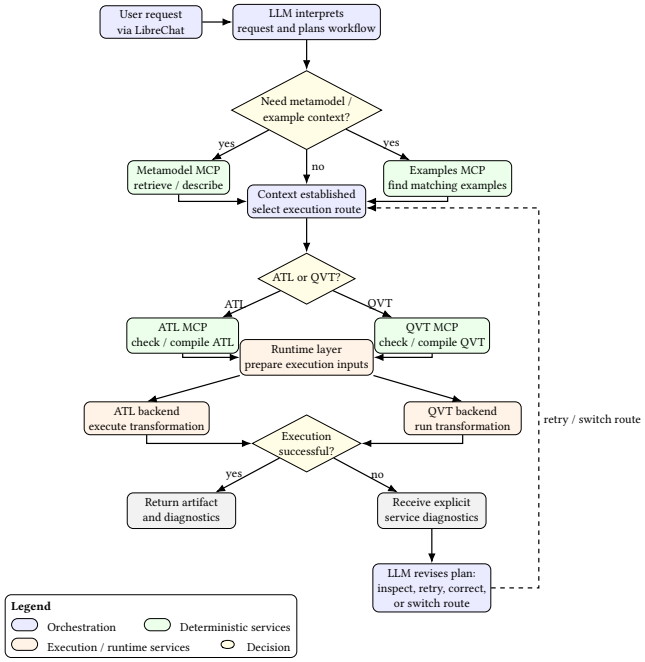


Figure 5: Workflow in the final setup.

Figure 5 shows the transformation workflow. A session starts with a user request in the LLM-enabled client. The LLM interprets the request, plans the next step, and invokes metamodel or example services when additional context is needed. After context is established, the LLM selects the ATL or QVTo path and generates a candidate transformation specification in the selected MTL. The corresponding service checks or compiles the candidate transformation, the runtime service prepares inputs and resources, and the backend executes the transformation.

A run is counted as an executable success only when the backend reports successful execution, and the produced artifact can be serialized and loaded as an EMF resource against the registered

target metamodel. This provides an operational check for syntactic well-formedness and metamodel conformance in the configured environment. If checking, execution, serialization, loading, or metamodel conformance fails, the service returns diagnostics that the LLM can use to revise the transformation, request more context, retry execution, or switch between MTLs when allowed by the task setting.

The workflow defines the scope of the framework: the LLM coordinates transformation development, while metamodels, transformation languages, engines, runtime resources, and EMF loading remain part of the MDE infrastructure. This reduces manual coordination while preserving inspectable tool-supported execution.

5 Emerging Results

This section reports emerging results used to assess the feasibility and limits of the proposed workflow in two transformation scenarios. The assessment examines whether generated transformation specifications can be executed, whether service diagnostics support repair, and how closely produced target artifacts match corresponding ground-truth artifacts. The goal is to study workflow behavior across tasks, MTLs, and LLMs, not to benchmark them.

5.1 Assessment Setup

The assessment uses two tasks: UML-to-Java as a model-to-text setting, and SysMLv2-to-AAS as a model-to-model setting with a domain-specific target structure. For UML-to-Java, we selected 10 UML class diagram models from the Lindholmen dataset and generated Java ground-truth artifacts using Modelio[1]. For SysMLv2-to-AAS, we selected 10 SysMLv2 models and corresponding AAS ground-truth models from the SysMLv2-to-AAS integration work by Ferko et al. [12].

We used three LLMs: GPT-5, DeepSeek R1, and Sonnet 4. Each input model was used with ATL and QVTo, resulting in 120 runs: two tasks, two transformation languages, three LLMs, and 10 input models per task. Each run had a fixed task, input model, LLM, and transformation language; ATL runs did not switch to QVTo, and QVTo runs did not switch to ATL.

During a run, the LLM could inspect metamodels, retrieve examples, generate a candidate transformation, invoke checking or compilation, execute the transformation, and revise it using returned diagnostics. A run ended when execution succeeded, the retry budget was exhausted, or no new candidate was produced. The retry budget was five repair attempts after the first failed execution. All runs used the same task template, tool descriptions, and service interfaces. Parameters were fixed to temperature = 0, maximum output tokens = 64000, and top-p = 1, when supported.

We report three measures. *Executable success* counts runs in which the generated ATL or QVTo transformation passed the backend workflow and the service returned ok. It does not imply structural completeness, mapping coverage, or semantic adequacy. *Recovery* counts initially failed runs that later reached executable success after diagnostic-based repair. *Structural F1* compares normalized element sets extracted from executable outputs and ground-truth artifacts. For UML-to-Java, these include Java classes, enums, fields, and enum literals. For SysMLv2-to-AAS, they include AAS-level node and attribute signatures. Formatting, generated identifiers, namespace prefixes, schema locations, and ordering are ignored. Failed runs are excluded because they do not produce executable target artifacts. Structural F1 is therefore an approximation of structural overlap, not a measure of transformation-script quality, maintainability, behavioral preservation, or full semantic correctness.

[1] http://models-db.com/

5.2 Initial Findings

Table 1 shows three observations. First, ATL led to more executable runs than QVTo in this setup. Across all configurations, ATL reached 54/60 executable successes, while QVTo reached 27/60. This suggests that service-mediated repair is sensitive to the diagnostics and execution conventions of the selected MTL.

Second, the task influenced structural quality. UML-to-Java produced higher Structural F1 values than SysMLv2-to-AAS. The UML-to-Java configurations mostly fall between 0.53 and 0.78 on average, while SysMLv2-to-AAS remains below 0.22. This indicates that executable and metamodel-conformant transformations can still omit important domain structures, especially for a domain-specific target structure such as AAS.

Third, diagnostics supported repair in several configurations, but not uniformly. GPT-5 and Sonnet often required repair before executable success, while DeepSeek often succeeded on the first attempt for ATL. Sonnet recovered 9/9 initially failed UML-to-Java QVTo runs and 9/10 SysMLv2-to-AAS ATL runs, but did not recover SysMLv2-to-AAS QVTo runs. Repair is therefore not only an LLM capability; it is an interaction between the generated specification, the transformation language, the execution engine, and the diagnostic information exposed by services.

Overall, the assessment supports the feasibility of exposing transformation development as MDE services for LLM-based coordination. It also shows why this workflow must go beyond executability and metamodel conformance. Future services need to assess structural completeness, mapping coverage, and semantic adequacy.

6 Discussion: Promise, Limits, and Questions

The findings suggest that LLM support for model transformation should be connected to the executable MDE workflow. In this setting, metamodels, examples, transformation engines, diagnostics, and adequacy checks are not passive prompt context; they are services that the agent can invoke during transformation development. This differs from approaches that mainly improve generation, such as retrieval augmentation or fine-tuning, because the LLM can use feedback from transformation tools while checking and execution remain part of the MDE infrastructure.

The results separate executable success from transformation adequacy. Successful execution shows that a transformation can run and produce an EMF-loadable, metamodel-conformant target artifact. It does not show that the transformation captures all intended source-to-target mappings. This was visible in SysMLv2-to-AAS, where several executable runs produced low-F1 outputs containing mainly top-level AAS structures while omitting expected submodels, properties, or domain-specific correspondences.

The current assessment is output-oriented. Structural F1 is computed on produced target artifacts, not on the generated ATL or QVTo specifications. It estimates structural overlap with the ground

LLM	Task	Transformation Languages	Exec. success	First-attempt success	Avg. retries to succeed	Recovery	Avg. F1	Highest F1	Lowest F1
GPT-5	UML→Java	ATL	9/10	6/10	1	3/4	0.530	**1.000**	0.000
		QVTo	6/10	0/10	1	6/10	**0.775**	**1.000**	0.421
	SysMLv2→AAS	ATL	8/10	0/10	2	8/10	0.161	0.424	0.046
		QVTo	6/10	0/10	2	6/10	0.144	0.333	0.083
DeepSeek	UML→Java	ATL	**10/10**	**10/10**	0	0/0	0.698	**1.000**	0.000
		QVTo	1/10	1/10	0	0/9	0.727	0.727	0.727
	SysMLv2→AAS	ATL	8/10	8/10	0	0/2	0.172	0.447	0.032
		QVTo	4/10	4/10	0	0/6	0.213	0.667	0.043
Sonnet	UML→Java	ATL	**10/10**	7/10	1	3/3	0.763	**1.000**	0.500
		QVTo	**10/10**	1/10	2	9/9	0.653	**1.000**	0.000
	SysMLv2→AAS	ATL	9/10	0/10	5	9/10	0.215	0.350	0.065
		QVTo	0/10	0/10	0	0/10	0.000	0.000	0.000

Table 1: Summary of preliminary experiment results. Executable success and recovery are measured at the transformation-run level; Structural F1 is computed only on produced target artifacts from executable runs.

truth, but does not assess the transformation specification itself. Work on model transformation testing and debugging treats the transformation as the system under test, using input models, expected outputs, oracles, traces, coverage criteria, mutation, and fault localization [25]. Future versions should therefore connect agentic repair with model transformation testing services, including rule coverage, metamodel coverage, oracle satisfaction, traces, mutation, and regression-oriented checks.

The findings also indicate that service-mediated repair depends on both the selected MTL and the task. ATL and QVTo expose different diagnostics, runtime failures, and execution conventions, which affect repair behavior in this setup. The task also matters: UML-to-Java reached moderate structural overlap in several configurations, whereas SysMLv2-to-AAS remained structurally weak despite executable success. Future services should therefore assess mapping coverage and detect trivial or incomplete target models rather than only checking executability.

A further implication is reusability. Generated specifications, service calls, diagnostics, and repair steps can be retained as inspectable MDE artifacts. For related metamodels, later tasks should reuse existing transformations and extend them when new metamodel elements, mappings, or examples are introduced. This paper does not yet implement incremental coverage over metamodel elements, but the framework allows such checks to be introduced as services. In this direction, transformation development becomes a reusable and incrementally improved process rather than repeated prompt-to-output generation.

These findings raise five questions for the next stage of the work:

- How should diagnostics be structured so that they support repair rather than only report failure?
- What adequacy checks are needed beyond execution and EMF loading to assess mapping completeness?
- How can model transformation testing services be integrated to assess generated transformation specifications?
- How should transformation services be exposed across different MTLs, given the differences between ATL and QVTo?
- How should the workflow support engineer validation when an executable and EMF-loadable transformation remains structurally incomplete or semantically ambiguous?

Limitations of the Preliminary Assessment

The assessment has several limitations. The dataset is small, with 10 input models per task, and each input was executed once per configuration; repeated runs are needed to study variation and stability. Executable success depends on the implemented services, runtime configuration, metamodel registrations, and serialization conventions. Structural F1 captures normalized structural overlap, but not full semantic correctness, behavioral preservation, transformation maintainability, or all domain-specific mapping constraints. The prototype also does not yet assess incremental metamodel coverage or reuse generated transformation specifications across related tasks. Finally, the assessment is limited to two MTLs and two tasks; future work should examine additional languages, metamodels, and more complex transformation scenarios as bidirectional [11] and round-trip [10] transformations.

7 Conclusion and Future Plans

This paper introduced agentic model transformation as a service-mediated workflow for LLM-assisted model transformation development. The workflow does not replace transformation languages, execution engines, or EMF tooling. It exposes them as services so that generated MTL specifications can be checked, executed, diagnosed, and repaired within the configured MDE infrastructure.

The emerging results on UML-to-Java and SysMLv2-to-AAS show that agents can obtain executable ATL and QVTo transformations and recover from some failures through service diagnostics. Successful runs produced EMF-loadable target artifacts conforming to the registered target metamodel. However, executable success did not guarantee structural completeness, especially in SysMLv2-to-AAS. Execution, EMF loading, and metamodel conformance are therefore useful first checks, but not sufficient evidence of structural completeness, mapping coverage, or semantic adequacy.

Future work will address the questions raised above in four directions. First, we will improve diagnostics by relating failures to metamodel elements, transformation rules, source locations, runtime traces, and produced target fragments. Second, we will extend adequacy checks beyond execution and EMF loading, including structural coverage, mapping completeness, domain-specific constraints, and detection of trivial but executable target models. Third, we will integrate model transformation testing services to assess generated specifications through rule coverage, metamodel coverage, oracle satisfaction, traces, mutation, and regression-oriented

checks. Fourth, we will study how transformation services should be exposed across MTLs, and how the workflow should support engineer validation when executable transformations remain structurally incomplete or semantically ambiguous.

The full study will also repeat configurations with fixed inputs, prompts, tool descriptions, model parameters, and retry budgets to analyze variation, service calls, repair steps, and failure categories. It will broaden the assessment with more scenarios, larger model sets, additional metamodels, and further MTL implementations. Finally, we will investigate how generated specifications, diagnostics, and service traces can be retained, assessed, reused, and extended across related transformation tasks, rather than regenerated from scratch. The same service-mediated principle may also support other MDE tasks that depend on explicit tool feedback, including model validation, model repair, model comparison, consistency checking, and code generation. The replication package in Section 8 will include models, metamodels, prompts, tool descriptions, generated transformations, service-call logs, scripts, and normalized comparison outputs.

8 Data Availability

We have made a public replication package available at the following repository: https://zenodo.org/records/20095491

Acknowledgments

This work is supported by the Swedish Agency for Innovation Systems through the project “iSecure: Developing Predictable and Secure IoT for Autonomous Systems” (2023-01899), by the Key Digital Technologies Joint Undertaking through the project “MATISSE: Model-based engineering of digital twins for early verification and validation of industrial systems” (101140216), and by the Clean Energy Transition Partnership through the project “FLEXI: Human-centered AI and digital twin powered energy system integration for flexibility markets” (101069750).

References

[1] Sathurshan Arulmohan, Marie-Jean Meurs, and Sébastien Mosser. 2023. 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)*. 580–587. doi:10.1109/MODELS-C59198.2023.00096

[2] Zakaria Babaalla, Abdeslam Jakimi, and Mohamed Oualla. 2025. LLM-Driven MDA Pipeline for Generating UML Class Diagrams and Code. *IEEE Access* 13 (2025), 171266–171286. doi:10.1109/ACCESS.2025.3615828

[3] Victor R. Basili. 1993. The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, H. Dieter Rombach, Victor R. Basili, and Richard W. Selby (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–12.

[4] Dimitrios Brodimas, Alexios Birbas, Dimitrios Kapolos, and Spyros Denazis. 2025. Intent-Based Infrastructure and Service Orchestration Using Agentic-AI. *IEEE Open Journal of the Communications Society* PP (01 2025), 1–1. doi:10.1109/OJCOMS.2025.3600706

[5] Kua Chen, Yujing Yang, Boqi Chen, José Antonio Hernández López, Gunter Mussbacher, and Daniel Varro. 2023. Automated Domain Modeling with Large Language Models: A Comparative Study. In *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 162–172. doi:10.1109/MODELS58315.2023.00037

[6] Robert Clarisó and Jordi Cabot. 2023. Model-Driven Prompt Engineering. In *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 47–54. doi:10.1109/MODELS58315.2023.00020

[7] Krzysztof Czarnecki and Simon Helsen. 2003. Classification of Model Transformation Approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of Model-Driven Architecture*.

[8] K. Czarnecki and S. Helsen. 2006. Feature-based survey of model transformation approaches. *IBM Systems Journal* 45, 3 (2006), 621–645. doi:10.1147/sj.453.0621

[9] Duy Dao, Alessio Bucaioni, and Antonio Cicchetti. 2025. Learning to Transform: Evaluating LLMs on Model Transformation by Example. In *MDE Intelligence 2025*. http://www.es.mdu.se/publications/7236-

[10] Duy Dao, Alessio Bucaioni, and Antonio Cicchetti. 2026. Change-aware round-trip benchmarking of LLMs for reliable and efficient artifact co-evolution. *Journal of Systems and Software* (2026), 113014.

[11] Romina Eramo and Alessio Bucaioni. 2013. Understanding bidirectional transformations with tggs and jtl. *Electronic Communications of the EASST* 57 (2013).

[12] Enxhi Ferko, Luca Berardinelli, Alessio Bucaioni, Moris Behnam, and Manuel Wimmer. 2025. From Engineering Models to Digital Twins: Generating AAS from SysML v2 Models. *Journal of Systems and Software* (November 2025). http://www.es.mdu.se/publications/7290-

[13] Mingming Gao, Longjie Li, Xiayi Ming, and Guoxin Shen. 2025. Research on Standardized Encapsulation of Model Context Protocol Tools and AI Agent Integration. In *2025 International Conference on Aerospace Information Perception and Intelligent Processing (AIPIP)*. 328–331. doi:10.1109/AIPIP66876.2025.11299220

[14] Zakaria Hachm, Théo Le Calvar, Hugo Bruneliere, and Massimo Tisi. 2025. Towards LLM Agents for Model-Based Engineering: A Case in Transformation Selection. doi:10.1109/MODELS-C68889.2025.00061

[15] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685 [cs.CL] https://arxiv.org/abs/2106.09685

[16] Nafiseh Kahani, Mojtaba Bagherzadeh, James Cordy, Juergen Dingel, and Daniel Varro. 2019. Survey and Classification of Model Transformation Tools. *Software and Systems Modeling* 18 (2019). doi:10.1007/s10270-018-0665-6

[17] Gabriel Kazai, Ronnie Agyeiwaa Osei, Alessio Bucaioni, and Antonio Cicchetti. 2025. Model Transformations Using LLMs Out-of-the-Box: Can Accidental Complexity Be Reduced?. In *First Workshop on Large Language Models For Generative Software Engineering*. http://www.es.mdu.se/publications/7201-

[18] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401 [cs.CL] https://arxiv.org/abs/2005.11401

[19] Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. 2014. Assessing the State-of-Practice of Model-Based Engineering in the Embedded Systems Domain. In *Model Driven Engineering Languages and Systems*. 166–182. doi:10.1007/978-3-319-11653-2_11

[20] Zahra Moezkarimi, Kevin Eriksson, Albin Alm Johansson, Alessio Bucaioni, and Marjan Sirjani. 2024. Harnessing ChatGPT for Model Transformation in Software Architecture: From UML State Diagrams to Rebeca Models for Formal Verification. In *4th International Workshop of Model-Driven Engineering for Software Architecture*. http://www.ipr.mdu.se/publications/7130-

[21] Ipek Ozkaya. 2023. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software* 40, 3 (2023), 4–8. doi:10.1109/MS.2023.3248401

[22] Anjana Sarkar and Soumyendu Sarkar. 2025. Survey of LLM Agent Communication with MCP: A Software Design Pattern Centric Review. arXiv:2506.05364 [cs.SE] https://arxiv.org/abs/2506.05364

[23] S. Sendall and W. Kozaczynski. 2003. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software* 20, 5 (2003), 42–45. doi:10.1109/MS.2003.1231150

[24] Hanan Abdulwahab Siala and Kevin Lano. 2025. Using Large Language Models to Extract UML Class Diagrams from Java Programs. In *2025 8th International Conference on Software and System Engineering (ICoSSE)*. 70–74. doi:10.1109/ICoSSE65712.2025.00020

[25] Javier Troya, Sergio Segura, Lola Burgueño, and Manuel Wimmer. 2023. Model Transformation Testing and Debugging: A Survey. *Comput. Surveys* 55, 4, Article 72 (2023). doi:10.1145/3523056