BXAgent: LLM-Driven Code Generation for Bidirectional and Incremental EMF Model Transformations

Thomas Buchmann[1,*]

[1]Hof University of Applied Sciences, Hof, Germany

Abstract

Bidirectional model transformations (BX) are notoriously difficult to implement correctly. Developers must ensure that forward and backward directions remain consistent, that incremental changes are propagated without overwriting unrelated model elements, and that concurrent modifications on both sides can be reconciled. In this paper, we present BXAgent, an LLM-driven code generator that automatically produces fully functional Java classes for bidirectional, incremental, and synchronizing EMF model transformations from two Ecore metamodels and a natural-language mapping description. BXAgent employs a five-stage pipeline comprising metamodel parsing, LLM-based mapping extraction, bidirectionality checking, template-driven code generation, and an optional LLM-based compilation repair loop. The generated transformation code supports batch execution, incremental change propagation in both directions, and a novel fingerprint-based synchronization algorithm that handles concurrent modifications, conflict resolution, and fuzzy matching of independently created objects. We evaluate BXAgent on the well-known Families-to-Persons benchmark and a simpler database synchronization scenario, demonstrating that the approach produces correct transformations covering complex cases such as role changes and bidirectional synchronization.

Keywords

Bidirectional Model Transformation, Large Language Models, Code Generation, EMF, Incremental Synchronization, Model-Driven Engineering

1. Introduction

Model-Driven Engineering (MDE) relies on model transformations as a central mechanism to bridge different levels of abstraction or to keep heterogeneous models consistent with each other [1]. When two metamodels represent overlapping concerns — for instance, a structural family model and a flat person registry — changes to one model must be propagated to the other and vice versa. Such *bidirectional transformations* (BX) are a long-standing research topic with well-known theoretical foundations [2, 3], yet implementing them correctly remains a significant engineering challenge.

A correct BX implementation must satisfy several properties simultaneously: the forward and backward directions must be consistent (the *round-tripping laws* [4]), incremental updates must propagate only the changed portions of a model rather than reconstructing the target from scratch, and deletions must be detected and handled according to a configurable policy. The most demanding scenario arises when both models have been modified independently and must be *synchronized* — requiring conflict detection, resolution, and the ability to match independently created objects that represent the same real-world entity.

Existing BX frameworks such as BiGUL [5], JTL [6], BXtend [7], BXtendDSL [8, 9] and QVT-R [10] require developers to learn specialized languages or DSLs and to manually encode the transformation logic, which creates a steep learning curve and a high maintenance burden. Recent advances in Large Language Models (LLMs) have demonstrated remarkable code generation capabilities across a wide range of programming tasks [11, 12]. This raises the question of whether LLMs can be leveraged to *generate* the intricate transformation logic that BX demands.

Joint Proceedings of the STAF 2026 Workshops: AgileMDE, GCM, ICMM, LLM4SE, TTC. Rennes, France, June 29-July 3, 2026

*Corresponding author.

thomas.buchmann@hof-university.de (T. Buchmann)

0000-0002-5675-6339 (T. Buchmann)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-26

CEUR-WS.org/Vol-4338/LLM4SE-paper1.pdf

In this paper, we present BXAgent, an LLM-driven tool that automatically generates complete, compilable Java classes implementing bidirectional, incremental, and synchronizing EMF model transformations. Given two Ecore metamodels and a natural-language description of the intended mapping, BXAgent orchestrates a five-stage pipeline: (1) metamodel parsing, (2) LLM-based mapping extraction, (3) bidirectionality checking, (4) template-based code generation, and (5) optional LLM-based compilation repair. The output is a fully compilable transformation class and a JUnit test skeleton. The generated code supports three execution modes: *batch transformation* (creating the target from scratch), *incremental transformation* (propagating partial changes), and *bidirectional synchronization* (reconciling concurrent modifications on both sides).

A key technical contribution is the *fingerprint-based synchronization algorithm* embedded in the generated code. Rather than relying on persistent object identifiers, it uses content-based fingerprints derived from natural key attributes to detect changes, identify deletions via EMF's automatic null-reference mechanism, resolve conflicts according to a configurable policy, and perform fuzzy matching of independently created objects.

The remainder of this paper is structured as follows. Section 2 provides background on bidirectional transformations and EMF. Section 3 presents the BXAgent pipeline in detail. Section 4 describes the synchronization algorithm. Section 5 evaluates the approach on two case studies. Section 6 discusses related work. Section 7 concludes and outlines future work.

2. Background

2.1. Bidirectional Model Transformations

A bidirectional transformation (BX) between a source metamodel S and a target metamodel T consists of a pair of functions: a *forward* transformation $\overrightarrow{f} : S \to T$ and a *backward* transformation $\overleftarrow{f} : T \to S$. To be well-behaved, these functions must satisfy consistency laws. The most common formulation requires *round-tripping*: applying the forward transformation followed by the backward transformation (or vice versa) must yield a result that is consistent with the original model [4, 13].

In practice, BX implementations must also handle *incremental* scenarios where only a subset of the source (or target) model has changed. A naïve approach of re-running the batch transformation would discard any target-side information not derivable from the source. Instead, incremental BX must identify which elements have changed and propagate only the corresponding updates.

The most complex scenario is *synchronization*, where both models may have been modified independently. Synchronization requires detecting which elements changed on each side, identifying conflicts (elements changed on both sides), and resolving them according to a configurable policy.

Throughout this paper, we use the well-known Families-to-Persons benchmark [14] as a running example. It involves transforming a family register (with `Family` containers holding `FamilyMember` objects in roles such as `father`, `mother`, `sons`, `daughters`) into a flat person register (with `Male` and `Female` objects carrying a combined name attribute).

2.2. Eclipse Modeling Framework (EMF)

The Eclipse Modeling Framework (EMF) [15] is the de facto standard for metamodeling in the academic world as well as in the Java ecosystem. An *Ecore* metamodel defines classes (`EClass`), typed attributes (`EAttribute`), and references (`EReference`) with cardinality and containment semantics. Model instances are graphs of `EObject` nodes serialized as XMI.

A key EMF mechanism exploited by BXAgent is *automatic reference nullification*: when an `EObject` is deleted via `EcoreUtil.delete(obj, true)`, EMF sets all `EReferences` pointing to that object to `null`. As we show in Section 4, this mechanism provides a zero-cost deletion detection strategy for the correspondence model.

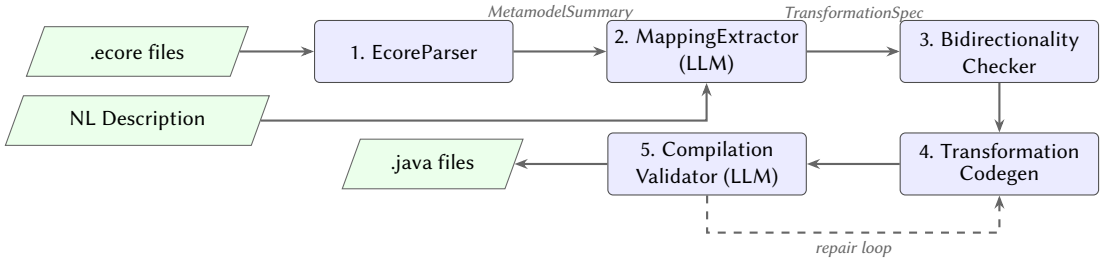


Figure 1: The BXAgent pipeline. Stages 2 and 5 involve LLM interaction; all other stages are deterministic.

3. The BXAgent Approach

Figure 1 gives an overview of the BXAgent pipeline. The input consists of two Ecore metamodels (source and target) and a natural-language description of the desired mapping. The output is a fully compilable Java class that implements batch, incremental, and synchronizing transformations, along with a JUnit test skeleton.

3.1. Stage 1: Metamodel Parsing

The `EcoreParser` reads both `.ecore` files and produces a `MetamodelSummary` for each. This summary contains, for every `EClass`, its name, attributes with types, containment references, and cross-references, each annotated with cardinality and opposite-reference information. The summary is a compact textual representation designed to fit within an LLM's context window while retaining all information necessary for mapping extraction.

3.2. Stage 2: LLM-Based Mapping Extraction

The `MappingExtractor` sends a carefully crafted prompt to the configured LLM backend (any backend compatible with the Ollama or OpenAI API is supported) consisting of:

1. A system prompt specifying the exact JSON schema for the response, including constraints on expression syntax (e.g., variable naming conventions, avoidance of error-prone patterns like `split(" ")[n]`).
2. The source and target metamodel summaries.
3. The user-provided natural-language mapping description.

The LLM responds with a `TransformationSpec` in JSON format containing:

`TypeMapping`: Direct EClass-to-EClass correspondences with natural key attributes for fingerprinting.

`AttributeMapping`: Per-attribute correspondences with Java expressions for both directions (`forwardExpression` reading from `source`, `backwardExpression` reading from `target`).

`ReferenceMapping`: EReference correspondences annotated with containment and opposite-reference flags.

`RoleBasedTypeMapping`: For cases where a source type maps to *different* target types depending on its structural role (e.g., a `FamilyMember` in the role `father` maps to `Male`, while in the role `mother` it maps to `Female`).

`TransformationOption` and `BackwardConfig`: Configurable parameters for non-deterministic backward mappings.

The system prompt encodes domain-specific constraints, for instance: types appearing as `sourceType` in a `RoleBasedTypeMapping` must *not* also appear in `typeMappings` (to avoid duplicate processing), and root container types must always be included in `typeMappings`.

If the LLM response fails to parse, up to three retry attempts are made with the parsing error fed back as additional context.

3.3. Stage 3: Bidirectionality Checking

The `BidirectionalityChecker` validates that every `AttributeMapping` has a `backwardExpression`. For attributes where the backward direction is non-trivial or ambiguous (e.g., splitting a full name "Simpson, Homer" back into family name and given name), the LLM may leave the `backwardExpression` as `null`. In such cases, the checker either prompts the user interactively for the missing expression or inserts a configurable default behavior via `BackwardConfig` parameters.

3.4. Stage 4: Template-Based Code Generation

The `TransformationCodegen` uses Apache FreeMarker templates[1] to produce the final Java source code. Rather than asking the LLM to generate the entire transformation class (which would be error-prone for complex metamodels), BXAgent uses the LLM *only* for the high-level mapping specification. The actual transformation code — including all incremental and synchronization logic — is generated deterministically from the `TransformationSpec` via templates. This separation of concerns ensures that the structural correctness of the generated code is guaranteed by the template, while the LLM contributes only the domain-specific mapping knowledge.

The generated class provides the following public API:

- `transform(source, target, [corr], [options])` — forward transformation
- `transformBack(source, target, [corr], [options])` — backward transformation
- `sync(source, target, corr, [policy], [options])` — bidirectional synchronization

Each method is overloaded to support batch mode (without correspondence resource) and incremental mode (with correspondence resource).

3.5. Stage 5: Compilation Validation and Repair

The `CompilationValidator` compiles the generated Java class in a temporary directory. If compilation errors occur, the error messages are sent to the LLM along with the generated source code, and the LLM is asked to produce a corrected version. This repair loop runs for up to three iterations. This limit was chosen pragmatically: in our experience, compilation errors caused by incorrect LLM-generated expressions are typically simple type mismatches or missing imports that the LLM can fix within one or two attempts; errors that persist beyond three rounds usually indicate a fundamental mapping problem that requires human intervention. In practice, compilation errors are rare because the code structure is template-generated; they mainly arise from incorrect Java expressions in the LLM-provided `forwardExpression` or `backwardExpression` fields.

4. Fingerprint-Based Synchronization

The most technically involved aspect of BXAgent's generated code is the `sync()` method, which handles the case where both models have been modified independently and must be reconciled. This section describes the algorithm in detail.

4.1. Correspondence Model

BXAgent generates a dynamic EMF-based correspondence model persisted as XMI. Each `CorrespondenceEntry` stores:

- `sourceObject` and `targetObject` as EMF cross-references
- `sourceFingerprint` and `targetFingerprint` — the fingerprints at the time of the last synchronization

[1] https://freemarker.apache.org/index.html

Table 1
The four-partition classification of correspondence entries.

Partition	sourceObject	targetObject	Interpretation
1	$\neq$ null	$\neq$ null	Both sides exist
2	$\neq$ null	= null	Target was deleted
3	= null	$\neq$ null	Source was deleted
4	= null	= null	Both deleted

- `sourceType` and `targetType` — the EClass names
- `sourceContainmentRole` and `targetContainmentRole`

A critical design decision is that `sourceObject` and `targetObject` are *EReferences*, not string-based identifiers. When an EObject is deleted via `EcoreUtil.delete(obj, true)`, EMF automatically nullifies all references to it — including those in the correspondence model. This provides a zero-overhead deletion detection mechanism: a correspondence entry with `sourceObject == null` indicates that the source element was deleted (and analogously for the target side). Note that this mechanism operates on in-memory EMF models; when models are loaded from disk (XMI), the correspondence model must be loaded into the same `ResourceSet` so that EMF can resolve and subsequently nullify cross-references correctly.

4.2. Content-Based Fingerprinting

Instead of relying on persistent object identifiers (such as XMI IDs), BXAgent uses *content-based fingerprints* computed from natural key attributes.

For each mapped type, the generated code includes a `computeFingerprint(EObject)` method that concatenates the values of the configured key attributes into a string of the form `"ClassName:val1|val2|"`. For role-based types, a *composite fingerprint* includes the container's key attributes, e.g., `family.getName() + "|" + computeFingerprint(member)`, enabling detection of container renames.

Two additional methods are generated for the synchronization algorithm:

- `computeExpectedTargetFingerprint(srcObj)`: computes the fingerprint that a target object *would have* if created from `srcObj` via forward transformation.
- `computeExpectedSourceFingerprint(tgtObj)`: the symmetric counterpart.

These methods are the foundation of the fuzzy matching step (Section 4.4).

4.3. The Four-Partition Synchronization Algorithm

The `sync()` method classifies all correspondence entries into four partitions based on the nullity of their object references (Table 1).

Partition 1 — Fingerprint Comparison. For entries where both objects exist, the algorithm compares current fingerprints against the stored fingerprints to determine what changed:

- *Only source changed*: propagate forward via `updateTargetAttributes`.
- *Only target changed*: propagate backward via `updateSourceAttributes`.
- *Both changed (conflict)*: resolved according to the `SyncConflictPolicy`: `SOURCE_WINS` (default), `TARGET_WINS`, or `LOG_AND_SKIP`.
- *Neither changed*: skip.

For role-based types, structural changes (e.g., a role change from `father` to `mother`) cannot be handled by simple attribute propagation. The algorithm employs a *fingerprint neutralization* technique: the stored fingerprint of the "losing" side is updated to match the current value, causing the subsequent incremental step to see "no change" on that side and process only the "winning" side.

Partitions 2 and 3 — Re-creation. Missing target objects (Partition 2) are re-created via forward transformation; missing source objects (Partition 3) are re-created via backward transformation. This handles the scenario where one side deletes an object that the other side still expects to exist.

Partition 4 — Garbage Collection. Entries where both references are null are removed from the correspondence model.

4.4. Fuzzy Matching

Before creating new objects for unmatched elements, the algorithm performs a *fuzzy matching* step to detect independently created objects that represent the same real-world entity.

Consider the scenario: in the Families model, a new `FamilyMember` "Homer" is added as `father` of family "Simpson". Independently, in the Persons model, a new `Male` object with name "Simpson, Homer" is created. Both represent the same person, and naïve synchronization would create a duplicate on each side.

The fuzzy matching algorithm works as follows:

1. Index all unmatched target objects by their fingerprint.
2. For each unmatched source object, compute `computeExpectedTargetFingerprint(src)`.
3. If a match is found in the index, create a correspondence entry linking both objects — no new object is created.
4. Repeat symmetrically for the backward direction.

Only source objects that remain unmatched after this step trigger the creation of new target objects (and vice versa).

4.5. Deletion Policies

The generated code supports three configurable deletion policies:

CASCADE: The corresponding object on the other side is deleted, and the correspondence entry is removed. This is the default for most scenarios.

ORPHAN: The correspondence entry is removed, but the other-side object is preserved (becoming an orphan). This is useful when domain rules require that objects survive the deletion of their counterpart, e.g., a person record that must persist even after its family entry is removed.

TOMBSTONE: The correspondence entry is marked as orphaned but retained for later reconciliation. This supports offline or deferred synchronization scenarios where deletions need to be reviewed before they take effect.

5. Evaluation

We evaluate BXAgent along two dimensions: (1) the correctness of the generated transformations on established benchmarks, and (2) the role of the LLM in the pipeline.

5.1. Case Study 1: Families ↔ Persons

The Families-to-Persons benchmark (cf. Section 2.1) poses several challenges: role-dependent type mappings, disappearing containers, and composite attribute splitting in the backward direction.

BXAgent was provided with the two Ecore metamodels and the following natural-language description:

> *"Transform FamilyMember: combine member.eContainer.name + ", " + member.name into single name field of Person, in backward transformation split name at the comma. Retrieve the matching family. two configuration parameters: prefer existing family and prefer parent over child role. Depending on the role in the family: father + sons a Male object from the persons model is created, for mother + daughters a Female object is created."*

The LLM successfully produced a `TransformationSpec` that correctly identified the role-based mapping, the name concatenation and splitting logic, and the necessary backward configuration parameters. From this specification, BXAgent generated a `Families2PersonsTransformation` class supporting forward transformation, backward transformation, and concurrent synchronization.

In order to check the generated transformation class for correctness, we did not only run the tests generated by BXAgent, but more importantly we used the extended Benchmarx Families-to-Persons test suite [16]. The extended test suite retains all original 34 forward, backward, and incremental test cases which are presented in [14]. Furthermore, it adds 13 concurrent synchronization scenarios ranging from conflict-free monotonic edits to genuine move/delete, rename/rename, and delete/rename conflicts, and adds three round-trip cases as well. BXAgent is applied to this extended suite without any manual post-editing of the generated code.

5.1.1. Test Results

All concurrent tests in this evaluation were run with the `SOURCE_WINS` conflict policy, meaning that whenever the source-side (family register) and target-side (person register) edits are in tension, the source-side change takes precedence.

Table 2 summarizes BXAgent's performance on the complete test suite.

Table 2
BXAgent test results on the extended Families-to-Persons benchmark (conflict policy: `SOURCE_WINS`).

Category	Tests	Passed
Batch (forward / backward)	16	16
Incremental (forward / backward)	18	18
Round-trip	3	3
Concurrent – Monotonic Creating	4	4
Concurrent – Monotonic Deleting	3	1
Concurrent – Non-Monotonic	2	0
Concurrent – Conflicts	4	4
Total	**50**	**46**

BXAgent passes all batch, incremental, round-trip, conflict, and monotonic creation tests. All four failing tests are concentrated in the *MonotonicDeleting* and *NonMonotonic* concurrent categories. Notably, BXAgent handles all four genuine conflict scenarios correctly under the `SOURCE_WINS` policy. Note that Table 2 reports only the results of the external benchmark suite [16]; BXAgent's own generated test skeleton (covering batch and incremental scenarios) passes completely but is not included here, as the benchmark suite is more comprehensive and independently defined.

All four failing tests share the same root cause: in BXAgent's `sync()` algorithm, attribute propagation (forward and backward) is performed *before* deletion propagation. This ordering means that a deletion on one side can be undone by the attribute-propagation step, which still sees the to-be-deleted object in the correspondence model and faithfully recreates its counterpart on the other side. A fix would require reordering the synchronization phases so that deletions are processed before attribute updates, which we plan to address in a future version.

5.2. Role of the LLM

The LLM is involved in exactly two pipeline stages: mapping extraction (Stage 2) and compilation repair (Stage 5). Crucially, the LLM never generates the transformation *code* directly — it only produces a structured mapping specification in JSON format. The deterministic template engine then generates all code, including the complex incremental and synchronization logic.

This design has several advantages:

- The LLM's task is reduced to a structured information extraction problem rather than open-ended code generation, making it more reliable.
- The structural correctness of the generated code (e.g., correct iteration patterns, deletion handling, reference resolution) is guaranteed by the template.
- The LLM's contribution (the mapping specification) is inspectable and editable by the developer before code generation.
- Different LLM backends can be used interchangeably without affecting the code generation logic.

We tested BXAgent with a self-hosted small LLM: devstral-small-2-24B [17] run by Ollama[2] on a NVidia RTX 4090 consumer GPU. We chose this model to demonstrate that BXAgent works with a lightweight, open-source, locally hosted coding model — without requiring access to commercial cloud APIs. For the Families-to-Persons case, the LLM produced a correct mapping specification on the first attempt. The compilation repair loop (Stage 5) was not triggered, as the generated code compiled successfully without errors.

6. Related Work

BX Frameworks and Languages. The BX community has produced a rich body of work on languages and frameworks for bidirectional transformations. QVT-R [10] is the OMG standard but has seen limited adoption due to its complexity. Triple Graph Grammars (TGGs) [18, 19] offer a declarative approach with strong formal foundations but require developers to specify correspondences in a specialized graph grammar formalism. BiGUL [5] takes a "putback"-based approach where the backward transformation is written explicitly and the forward direction is derived. JTL [6] supports non-bijective transformations with a relation-based approach. BXtend [7] is based on imperative programming and uses an internal DSL embedded in Xtend, while BXtendDSL [8, 9] is a hybrid language which provides an external DSL, as well as the option to specify algorithmic details of the transformation on the imperative layer if required. All of these approaches require manual specification of the transformation logic. BXAgent differs fundamentally by using an LLM to extract the mapping from natural language and generating the implementation automatically. A natural question is why BXAgent generates plain Java rather than targeting a dedicated BX language such as QVT-R or TGGs. Generating into a high-level BX language would enable formal analyzability [20], but it would also tie BXAgent to a specific BX runtime and limit its applicability to environments where that runtime is available. By generating self-contained Java code with no external BX framework dependency, BXAgent maximizes portability. Furthermore, there are transformation problems, which can not be solved completely with TGGs [21] or QVT-R [22]. However, supporting additional target languages is planned as future work.

LLMs for Model-Driven Engineering. The application of LLMs to MDE tasks is a rapidly growing research area. Existing work has explored using LLMs for metamodel generation [23], model completion [24], and model transformation specification [25]. In our previous work, we tried to use prompting techniques for generating bidirectional and incremental transformations [26]. While we successfully obtained a working batch transformation, the approach was not able to generate a fully functional bidirectional and incremental transformation for the Families-to-Persons example. BXAgent now has proven to be the much better approach, which additionally supports concurrent synchronization.

Incremental and Synchronizing Transformations. Incremental model transformation has been addressed by reactive approaches such as VIATRA [27] and eMoflon [28, 29]. Model synchronization has been studied theoretically [30] and implemented in TGG-based tools. BXAgent's synchronization algorithm is most closely related to the state-based synchronization approach [31], where changes are detected by comparing model states rather than recording edit operations. The fingerprint-based change detection and fuzzy matching are, to our knowledge, novel contributions in the context of generated BX code.

[2] https://ollama.com/

7. Conclusion and Future Work

We have presented BXAgent, an LLM-driven code generator for bidirectional, incremental, and synchronizing EMF model transformations. BXAgent is publicly available as open source.[3] By separating the LLM's role (structured mapping extraction) from the code generation (deterministic template expansion), BXAgent combines the flexibility of natural-language specification with the reliability of template-based code generation.

The generated code supports batch, incremental, and synchronization modes, the latter featuring a novel four-partition algorithm with content-based fingerprinting, configurable conflict resolution, and fuzzy matching. Evaluation on the Families-to-Persons benchmark demonstrates that the approach produces correct transformations covering complex scenarios including role changes and bidirectional synchronization with concurrent modifications.

Several directions for future work are planned:

- **Persistent object IDs**: Replacing fingerprinting with XMI IDs for more robust correspondence tracking, eliminating ambiguity in delete-recreate scenarios.
- **Schema evolution**: Handling metamodel changes and migrating existing correspondence models.
- **Broader evaluation**: Applying BXAgent to additional benchmarks from the BenchmarX repository [14] and industrial case studies, as well as evaluating the effects of different LLM backends and configurations on mapping quality.
- **Multi-platform code generation**: Supporting additional target languages beyond Java, such as QVT-R or TGG specifications, to enable formal analyzability.
- **User studies**: Conducting controlled experiments to assess how effectively developers can use natural-language descriptions to specify complex bidirectional mappings.
- **Semantic validation**: Verifying that the LLM-generated forward and backward expressions are truly inverse before code generation.

Declaration on Generative AI. The BXAgent tool itself uses LLMs as part of its pipeline to extract mapping specifications from natural-language descriptions, as described in Section 3.2. The manuscript text was prepared by the author with AI-assisted editing (gpt-oss-120B, self-hosted) for language polishing.

References

[1] K. Czarnecki, S. Helsen, Feature-based survey of model transformation approaches, IBM Syst. J. 45 (2006) 621–646. doi:10.1147/SJ.453.0621.

[2] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, J. F. Terwilliger, Bidirectional transformations: A cross-discipline perspective, in: R. F. Paige (Ed.), Theory and Practice of Model Transformations - 2nd International Conference, ICMT@TOOLS 2009, Zurich, Switzerland, June 29-30, 2009. Proceedings, volume 5563 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 260–283. doi:10.1007/978-3-642-02408-5_19.

[3] P. Stevens, Bidirectional model transformations in QVT: semantic issues and open questions, in: Model Driven Engineering Languages and Systems, 10th International Conference, MoDELS 2007, Nashville, USA, September 30 - October 5, 2007, Proceedings, 2007, pp. 1–15. doi:10.1007/978-3-540-75209-7_1.

[4] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, A. Schmitt, Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem, ACM Trans. Program. Lang. Syst. 29 (2007) 17. doi:10.1145/1232420.1232424.

[5] H. Ko, T. Zan, Z. Hu, Bigul: a formally verified core language for putback-based bidirectional programming, in: M. Erwig, T. Rompf (Eds.), Proceedings of the 2016 ACM SIGPLAN Workshop

[3] https://https://github.com/tbuchmann/bx-agent — includes source code and shell scripts for generating Pdb1-to-Pdb2 and Families-to-Persons examples

on Partial Evaluation and Program Manipulation, PEPM 2016, St. Petersburg, FL, USA, January 20 - 22, 2016, ACM, 2016, pp. 61–72. doi:10.1145/2847538.2847544.

[6] A. Cicchetti, D. D. Ruscio, R. Eramo, A. Pierantonio, JTL: A bidirectional and change propagating transformation language, in: B. A. Malloy, S. Staab, M. van den Brand (Eds.), Software Language Engineering - Third International Conference, SLE 2010, Eindhoven, The Netherlands, October 12-13, 2010, Revised Selected Papers, volume 6563 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 183–202. doi:10.1007/978-3-642-19440-5_11.

[7] T. Buchmann, Bxtend - A framework for (bidirectional) incremental model transformations, in: S. Hammoudi, L. F. Pires, B. Selic (Eds.), Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2018, Funchal, Madeira - Portugal, January 22-24, 2018, SciTePress, 2018, pp. 336–345. doi:10.5220/0006563503360345.

[8] T. Buchmann, M. Bank, B. Westfechtel, Bxtenddsl: A layered framework for bidirectional model transformations combining a declarative and an imperative language, J. Syst. Softw. 189 (2022) 111288. doi:10.1016/J.JSS.2022.111288.

[9] T. Buchmann, M. Bank, B. Westfechtel, Bxtenddsl at work: Combining declarative and imperative programming of bidirectional model transformations, SN Comput. Sci. 4 (2023) 50. doi:10.1007/S42979-022-01448-8.

[10] OMG, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, formal/2015-02-01 ed., Needham, MA, 2015.

[11] A. El-Kishky, A. Wei, A. Saraiva, B. Minaiev, D. Selsam, D. Dohan, F. Song, H. Lightman, I. C. Gilaberte, J. Pachocki, J. Tworek, L. Kuhn, L. Kaiser, M. Chen, M. Schwarzer, M. Rohaninejad, N. McAleese, o3 contributors, O. Mürk, R. Garg, R. Shu, S. Sidor, V. Kosaraju, W. Zhou, Competitive programming with large reasoning models, CoRR abs/2502.06807 (2025). doi:10.48550/ARXIV.2502.06807. arXiv:2502.06807.

[12] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. Canton-Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, G. Synnaeve, Code llama: Open foundation models for code, CoRR abs/2308.12950 (2023). doi:10.48550/ARXIV.2308.12950. arXiv:2308.12950.

[13] P. Stevens, Bidirectional transformations in the large, in: 20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2017, Austin, TX, USA, September 17-22, 2017, IEEE Computer Society, 2017, pp. 1–11. doi:10.1109/MODELS.2017.8.

[14] A. Anjorin, T. Buchmann, B. Westfechtel, Z. Diskin, H. Ko, R. Eramo, G. Hinkel, L. Samimi-Dehkordi, A. Zündorf, Benchmarking bidirectional transformations: theory, implementation, application, and assessment, Softw. Syst. Model. 19 (2020) 647–691. doi:10.1007/S10270-019-00752-X.

[15] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF Eclipse Modeling Framework, The Eclipse Series, 2nd ed., Addison-Wesley, Boston, MA, 2009.

[16] A. Anjorin, T. Buchmann, L. Fritsche, Benchmarx 2.0: A benchmark for concurrent model synchronisation approaches, in: M. Wimmer, A. Egyed, B. Combemale, M. Chechik (Eds.), Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, MODELS Companion 2024, Linz, Austria, September 22-27, 2024, ACM, 2024, pp. 950–959. doi:10.1145/3652620.3688217.

[17] M. AI, A. H. AI, Devstral: Fine-tuning language models for coding agent applications, CoRR abs/2509.25193 (2025). doi:10.48550/ARXIV.2509.25193. arXiv:2509.25193.

[18] A. Schürr, Specification of graph translators with triple graph grammars, in: E. W. Mayr, G. Schmidt, G. Tinhofer (Eds.), Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG '94, Herrsching, Germany, June 16-18, 1994, Proceedings, volume 903 of *Lecture Notes in Computer Science*, Springer, 1994, pp. 151–163. doi:10.1007/3-540-59071-4_45.

[19] A. Anjorin, E. Leblebici, A. Schürr, 20 years of triple graph grammars: A roadmap for future research, Electron. Commun. Eur. Assoc. Softw. Sci. Technol. 73 (2015). doi:10.14279/TUJ.ECEASST.73.1031.

[20] J. de Lara, E. Guerra, J. S. Cuadrado, Have model transformation languages failed?, Softw. Syst.

Model. (2026). doi:10.1007/s10270-026-01360-2.

[21] A. Anjorin, T. Buchmann, On the practical expressiveness of triple graph grammars, in: Proceedings of the 14th International Conference on Model-Based Software and Systems Engineering - MODELSWARD, INSTICC, SciTePress, 2026, pp. 410–417. doi:10.5220/0014448100004058.

[22] B. Westfechtel, Case-based exploration of bidirectional transformations in QVT relations, Softw. Syst. Model. 17 (2018) 989–1029. doi:10.1007/S10270-016-0527-Z.

[23] J. Cámara, J. Troya, L. Burgueño, A. Vallecillo, On the assessment of generative ai in modeling tasks: an experience report with chatgpt and uml: Jj. cámara et al., Software and Systems Modeling 22 (2023) 781–793.

[24] M. B. Chaaben, L. Burgueño, H. Sahraoui, Towards using few-shot prompt learning for automating model completion, in: 2023 IEEE/ACM 45th international conference on software engineering: New ideas and emerging results (ICSE-NIER), IEEE, 2023, pp. 7–12.

[25] D. Dao, A. Bucaioni, A. Cicchetti, Learning to transform: Evaluating llms on model transformation by example, in: 2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2025, pp. 576–585. doi:10.1109/MODELS-C68889.2025.00081.

[26] T. Buchmann, Prompting bidirectional model transformations - the good, the bad and the ugly, in: M. Wimmer, A. Egyed, B. Combemale, M. Chechik (Eds.), Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, MODELS Companion 2024, Linz, Austria, September 22-27, 2024, ACM, 2024, pp. 550–555. doi:10.1145/3652620.3687802.

[27] D. Varró, G. Bergmann, Á. Hegedüs, Á. Horváth, I. Ráth, Z. Ujhelyi, Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework, Softw. Syst. Model. 15 (2016) 609–629. doi:10.1007/S10270-016-0530-4.

[28] A. Anjorin, M. Lauder, S. Patzina, A. Schürr, Emoflon: leveraging EMF and professional CASE tools, in: H. Heiß, P. Pepper, H. Schlingloff, J. Schneider (Eds.), 41. Jahrestagung der Gesellschaft für Informatik, Informatik schafft Communities, INFORMATIK 2011, Berlin, Germany, October 4-7, 2011, Abstract Proceedings, volume P-192 of *LNI*, GI, 2011, p. 281.

[29] N. Weidmann, A. Anjorin, emoflon: : Neo - consistency and model management with graph databases, in: L. Iovino, L. M. Kristensen (Eds.), STAF 2021 Workshop Proceedings: 9th International Workshop on Bidirectional Transformations, volume 2999 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 54–64.

[30] Z. Diskin, Y. Xiong, K. Czarnecki, H. Ehrig, F. Hermann, F. Orejas, From state- to delta-based bidirectional model transformations: The symmetric case, in: J. Whittle, T. Clark, T. Kühne (Eds.), Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings, volume 6981 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 304–318. doi:10.1007/978-3-642-24485-8_22.

[31] Y. Xiong, D. Liu, Z. Hu, H. Zhao, M. Takeichi, H. Mei, Towards automatic model synchronization from model transformations, in: Proceedings of the 22nd IEEE/ACM international conference on Automated Software Engineering, 2007, pp. 164–173.