Structuring the Design Space of Graph Transformation Frameworks Across Graph Families

Vadim Zaytsev[1]

[1]Formal Methods & Tools, EEMCS, University of Twente, The Netherlands

Abstract

Graph transformation frameworks are often discussed one graph formalism or one tool ecosystem at a time. This makes it difficult to compare recent work on categorical rewriting, querying, verification, repair, and AI-assisted interaction. From a structured exploratory literature mapping of 97 papers drawn from ICGT 2021–2025, GCM 2021–2025, and a targeted journal harvest for 2021–2026, we derive support contracts that separate a common core for graph families: an object-and-morphism interface, replacement semantics, and a condition formalism interpreted over that interface, from theory extensions and support dimensions. The contracts give a compact vocabulary for comparing framework support across graph families without postulating a universal graph model.

1. Introduction

Graph transformation frameworks are often presented using one graph formalism at a time. Historically, the field has explored this space through systems such as AGG [1] and PROGRES [2], through model-transformation environments such as Fujaba [3], AToM^3 [4], and VIATRA [5], through high-performance rewriting tools such as GrGen.NET [6], through graph-programming languages such as GP 2 [7], and through more recent ecosystems such as Henshin [8], GROOVE [9], and eMoflon::IBeX [10]. These systems differ not only in engineering choices, but also in the graph families they treat as primitive, the morphisms and matches they admit, and the analysis or execution services they place around rewriting. Recent work suggests that this diversity is increasing rather than shrinking. The literature now ranges across generalised categorical rewriting constructions [11, 12, 13]; conditions and conflict analysis [14, 15]; quantitative termination arguments [16, 17, 18]; query and execution infrastructure, including RETE-style discrimination networks for incremental pattern matching [19, 20, 21]; verification and property specification [22, 23, 24, 25]; repair and bidirectionality [26, 27]; and AI-mediated workflows or search [28, 29, 30, 31]. Recent work on model transformation languages also argues for sharper value propositions than simply offering another standalone language [32].

We argue that the design space is easier to understand if frameworks are described by *what they assume and what they support*. We use *graph family* as the broad term. We use *substrate* as shorthand for a graph family together with its typing discipline, admissible morphisms, and rule-application assumptions. On this reading, the key question is not whether a framework "supports graphs", but which graph families it supports at which semantic level. The strongest common layer we found is a core formed by an object-and-morphism interface, replacement semantics, and a condition language. Querying, verification, repair, and AI-facing interaction are better treated as layers above that core.

The argument is grounded in a conservative construction of the corpus and a screening process, following the structured exploratory literature mapping methodology [33]. Section 2 presents the methodology, reduction passes that were undertaken, and the resulting corpus. Section 3 extracts the recurring abstraction boundary. Section 4 defines the support contracts. Section 5 tests those contracts against three positive exhibits and one deliberate boundary case. The conclusion returns to the implications for describing and comparing frameworks.

Joint Proceedings of the STAF 2026 Workshops: AgileMDE, GCM, ICMM, LLM4SE, TTC. Rennes, France, June 29-July 3, 2026
vadim@grammarware.net (V. Zaytsev)
https://grammarware.net (V. Zaytsev)
0000-0001-7764-4224 (V. Zaytsev)

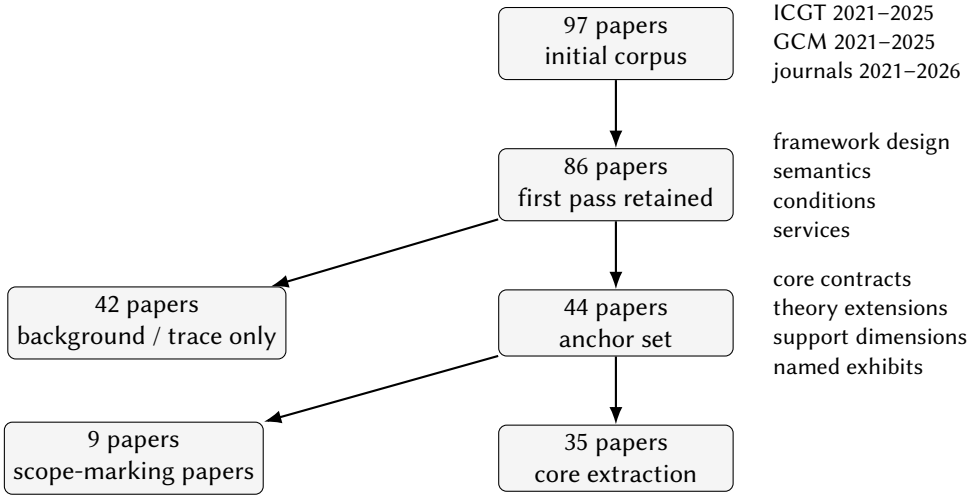

Figure 1: Corpus construction and staged retention process.

2. Corpus Construction Methodology

In order to avoid falling too much into wishful thinking and proposing a classification rooted in personal biases, let us first collect some evidence to build it upon. This could have been done in a more targeted way, such as starting with recent highlights and snowballing back, but any such focused starting points we could think of, would also embed some kind of bias. So, instead, we start as broad as we can, and limit ourselves only to keep it realistic.

We follow the structured exploratory literature mapping methodology, which differs from well-known systematic maps in that it seeks comprehensiveness and rigour but not exhaustiveness [33, p.324]. The contract family proposed in this paper is intended as an analytic abstraction over recent graph transformation research, rather than as a purely speculative taxonomy. To reduce dependence on the author's prior expectations, we constructed a bounded literature corpus and analysed it through a staged exploratory mapping process. The process starts from broad venue-based coverage, applies explicit screening and coding passes, and retains an anchor set used to derive the abstraction boundary.

Thus, our initial corpus covers all papers from five editions of ICGT 2021–2025 and all four proceedings of GCM 2021–2025, plus journal papers from 2021–2026. The proceedings component was venue-based, while the journal component was deliberately narrower. For journals, our initial DBLP probing exploration identified venues such as *Automated Software Engineering*, *Software and Systems Modeling*, *Natural Computing*, *Software Quality Journal*, and *Science of Computer Programming* as relevant venues. We harvested papers from these journals from 2021 onwards. Since that produced too many false positives, we retained only papers with "graph transformation" explicitly in the title. We then manually added one 2026 SoSyM paper as contextual motivation because it is directly about transformation-language design [32]. This entire process yielded 97 papers in total.

We then applied three staged screening and coding passes. Pass A screened for relevance to framework design, graph-family assumptions, replacement semantics, conditions, analysis services, or tool/language architecture. This reduced the corpus to 86 papers. Pass B coded the retained papers along dimensions such as graph family, typing and attribution discipline, rewrite discipline, morphism restrictions, application conditions, relabelling, concurrency/conflict results, abstraction/verification support, incremental querying, bidirectionality, repair orientation, AI interface, and contribution mode. Pass C clustered the 86 coded papers into six themes and selected a 44 paper anchor set. A paper was retained in the anchor set if it met at least one of three criteria: (i) it informed the core contracts in sections 3–4 directly, (ii) it represented one of the theory extensions or support dimensions that reappears later in the paper, or (iii) it served as one of the named exhibits in section 5. The remaining 42 papers were kept as background or trace-only material. A final confidence audit separated 35 papers used for core

Table 1
Anchor-set clusters after Pass C.

Cluster	Count	Cluster	Count
Rewrite contracts and semantics	19	Consistency, repair, and AI-facing workflows	9
Analysis and verification services	22	Graph-family exhibits and boundary cases	18
Execution, querying, and tooling	15	Field-framing and meta-level discussion	3

extraction from 9 scope-marking papers, which were retained to test the limits of the abstraction rather than to justify the core contracts.

Most of the work in these passes was done in an admittedly superficial (let us say "intentionally lightweight") way by looking at abstracts and conclusions, perhaps methodological summaries of the papers, and diving deeper into the core sections only when the real need arose. Full reading was reserved for papers that directly affected the proposed contracts, served as exhibits, or created uncertainty during coding. This is still appropriate for the paper's purpose: we do not claim any quantitative findings about the field as a whole, but use the corpus to discipline the construction of a design space vocabulary, and to develop an intuition of trends and genres of graph transformation research.

Figure 1 records the staged retention counts, and Table 1 reports the non-exclusive thematic clusters assigned during Pass C. The 44-paper anchor set is still broad, but its centre of gravity is clear. Most anchor papers are foundational rather than purely application-driven, and the largest clusters concern rewrite contracts and semantics, analysis and verification services, and graph-family exhibits. Those distributions matter because they show that the resulting contract family is not being inferred from one or two isolated theoretical papers. Rewrite contracts and semantics form a substantial cluster, but they do not dominate the anchor set on their own. Analysis and verification papers are the most common, graph-family exhibits are also quite prominent, and execution-, repair-, and AI-facing work is still present in enough quantity to force a separation between the reusable core and the layers built on top of it. The claim is that the contract family we present in the following section will then reflect a distribution across several recurring lines of work, instead of a retrospective abstraction from one favoured subcommunity.

The confidence audit served a narrower purpose. It did not challenge the relevance of the 9 scope-marking papers, just established that several of them fit the coding scheme only partially. They were useful for testing the breadth and limits of the proposed abstraction, but not as direct evidence for the core contracts. We therefore use them to delimit scope, not to justify the core contracts.

3. From Coded Evidence to the Abstraction Boundary

The main outcome of the coding was a layered prerequisite structure rather than a flat feature list. Across the 35 papers used for core extraction, the most reusable layer consists of three contracts:

- C0: an object-and-morphism interface for a graph family;
- C1: replacement semantics;
- C2: a condition or constraint formalism interpreted over the object-and-morphism interface and compatible with replacement steps.

On top of these we place three theory extensions: E1 for composition and concurrency, E2 for quantitative and termination-style reasoning, and E3 for program or control layers. Besides them, we position four support dimensions: S1 querying and execution, S2 verification and property support, S3 repair and bidirectionality, and S4 search, testing, and AI-facing interaction.

We use three terms consistently. *Core contracts* are the assumptions that define reusable support for a graph family: C0 fixes the object-and-morphism interface, C1 fixes replacement semantics, and C2 fixes a condition or constraint formalism over that interface. *Theory extensions* are additional theoretical

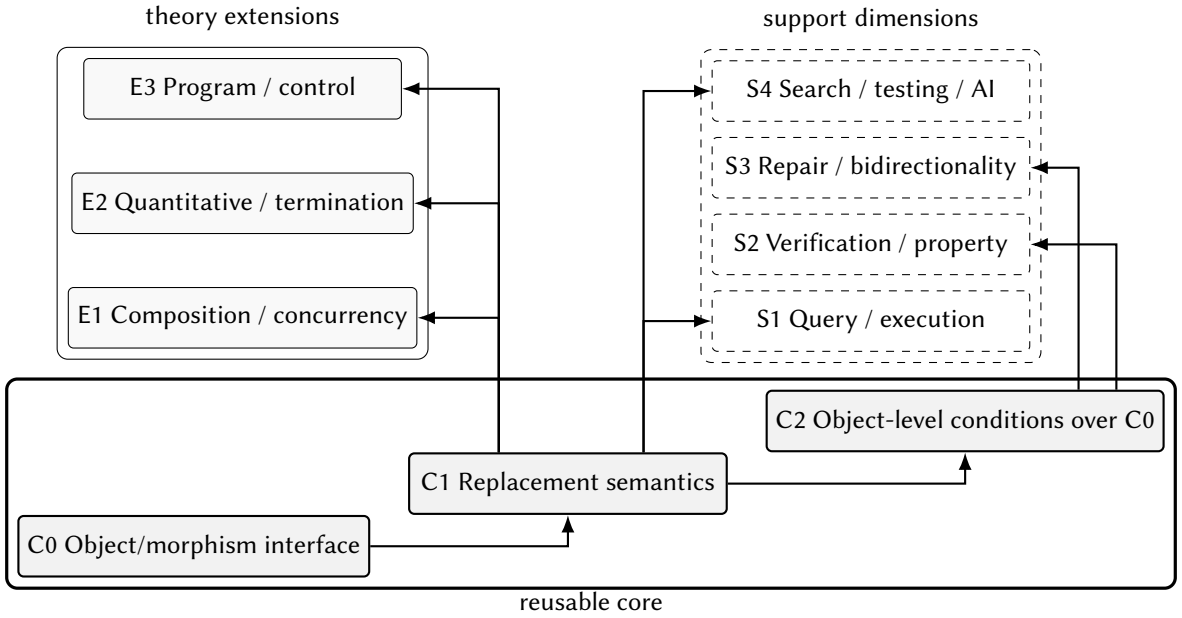

Figure 2: Layered dependency model derived from the 35 core-extraction papers.

layers that strengthen the core with assumptions needed for families of results, such as concurrency, termination, or graph programming. *Support dimensions* are framework-facing capabilities, such as querying, verification, repair, or AI-assisted interaction, that consume the core and possibly some theory extensions but also require execution, property, or interaction commitments of their own.

Figure 2 summarises this layered prerequisite structure. The key boundary is not between one graph kind and another, but between a reusable replacement core and higher layers that require additional assumptions. Core theory papers repeatedly revolve around interfaces for graph families and replacement constructions [11, 12, 13, 34]. Condition-aware analyses recur as a distinct next step [14, 15]. Quantitative and concurrency results require visibly stronger assumptions [35, 16, 17, 18]. Query engines, automata-based property checking, repair ranking, or AI interfaces depend on additional commitments beyond that core [20, 23, 27, 29]. This separation also makes a heterogeneous recent literature easier to compare. A localised RETE network, an automata-based property checker, higher-order shortcut repair, and notebook or chatbot support do not all belong to the same semantic layer. They consume a lower-level replacement semantics in different ways.

4. A Family of Support Contracts

The contract family follows from the layered dependency model. It does not force every graph family into one universal model. It describes what a framework must assume before it can offer a capability in a well-defined way.

For the purposes of this paper, an abstract graph-family interface is a tuple

$$\Sigma = (\mathcal{C}, \mathcal{M}, \mathsf{Pat}, \mathsf{Ty}, \mathsf{Cond}, \models)$$

where $\mathcal{C}$ is a category of graph-like objects, $\mathcal{M} \subseteq \mathrm{Mor}(\mathcal{C})$ is a distinguished class of admissible matches, Pat specifies admissible patterns or rule sides, Ty specifies typing or attribution structure when present, Cond is a language of conditions, and $\models$ is a satisfaction relation. Not every framework needs to expose all of this structure explicitly, but the papers in the core extraction repeatedly make these ingredients visible in one form or another.

A graph family satisfies a contract C, written $\Sigma \Vdash C$, if it provides the structures and laws required by that contract. A framework F therefore does not simply "support" a graph family. It offers a "support

Table 2
Support contracts, theory extensions, and support dimensions.

ID	Informal requirement	Enables
C0	Graph-like objects, admissible morphisms, interfaces or patterns, and an explicit typing/attribution discipline (or an explicit statement that none is assumed)	A common vocabulary for a graph family
C1	A rule representation, a distinguished match class, and a well-defined replacement construction yielding derivations	Reusable replacement semantics
C2	A condition or constraint formalism interpreted over the graph-family objects, morphisms, patterns, and matches fixed by C0, with a satisfaction semantics that can be related to C1 replacement steps	Application conditions, constraint preservation, conflict-style reasoning, condition-aware repair and analysis
E1	Stronger compositionality or overlap assumptions on top of C1	Concurrent and parallel rule application, composition results
E2	Additional quantitative structure on top of C1	Termination and ranking arguments
E3	Program or control syntax over replacement steps	Graph programs, control algebras, executable transformation languages
S1	Execution-specific pattern/query machinery	Incremental pattern matching, localised RETE, query solving
S2	A state/property semantics over induced systems	Temporal reasoning, automata-based checking, abstraction/refinement
S3	An explicit consistency and repair layer	Synchronisation, restoration, ranking of repairs
S4	Stable interfaces for interaction, search, or learning	Notebooks, chatbots, reinforcement-learning-guided testing, fuzzing-style exploration

profile" which is basically just a subset of the set of all possible contracts and extensions we define below and list in Table 2. This profile records which contracts, theory extensions, and support dimensions are available for graph family Σ under the assumptions fixed by framework F.

Contract C0. $\Sigma \Vdash$ C0 if $\mathcal{C}$ and $\mathcal{M}$ are fixed explicitly, if patterns and interfaces are well formed in $\mathcal{C}$ or interpretable in it, and if typing or attribution is preserved by admissible morphisms whenever such structure is present. This is the lowest level worth distinguishing in the design space. It defines what counts as a legitimate graph family for the framework, but it does not yet guarantee rule application. Evidence for this level comes not only from mainstream typed-graph settings but also from papers that broaden the family space without directly yielding a common framework, such as hypergraph grammars, bigraphs, or nearby categorical formalisms [36, 37, 38]. There exist feature models which are suitable to structure the design space within C0 further [39, Fig.1.9].

Contract C1. $\Sigma \Vdash$ C1 if rules are representable as spans or an equivalent replacement specification,

$$r = (L \xleftarrow{l} K \xrightarrow{r} R),$$

and for each admissible match $m\colon L \to G$ the replacement construction is defined whenever the side conditions of the graph family hold, yielding a derivation relation $G \Rightarrow_r H$. This is the threshold for reusable replacement semantics. In a DPO-centred instance, replacement is realised through pushout complements and pushouts in adhesive or related settings [40, 41]. This also leaves room for algebraic presentations in which graphs are given by expressions and rewriting is formulated at the expression level, rather than directly through a single categorical construction [42]. Papers on PBPO+, computational category-theoretic rewriting, traced monoidal settings, and mechanised DPO theory all support that reading [11, 13, 34, 43].

Table 3
Support profiles for the selected exhibits.

ID	Exhibit and support profile	Main role in the paper	Evidence
X1	Typed attributed graphs. Strong fit for C0–C2; optional support for E1–E3; support dimensions depend on framework choices.	Canonical full-core positive exhibit.	[45, 17, 27, 15]
X2	Partial triple graphs / TGG-style correspondence structures. Strong fit for C0–C2; especially useful for S3.	Shows why repair and bidirectionality should be treated as services built on top of the core.	[26, 46, 3]
X3	Hypergraph-oriented or traced-monoidal settings. Strong fit for C0–C1; cautious fit for C2.	Shows that the contract family is not tied to mainstream typed graphs.	[13, 37]
B1	Bigraphs. Useful as a boundary case for C0, but not used for core extraction.	Marks the edge of the present argument without forcing a stronger claim than the evidence warrants.	[38, 47, 48]

Contract C2. $\Sigma \Vdash$ C2 if, in addition, there is a condition language Cond and a satisfaction judgement

$$(G, m) \models \varphi$$

for conditions φ over matches or contexts, compatible with the replacement semantics in the sense needed by the target theoretical layer. This is the upper edge of the reusable core. Conditions repeatedly reappear where replacement semantics extends into conflict reasoning, repair ranking, and parts of verification [14, 15, 27]. For that reason, we place C2 inside the core rather than outside it.

Theory extensions and support dimensions. The evidence does not support treating concurrency, quantitative reasoning, graph programming, verification, repair, or AI interaction as parts of one universal substrate contract. Concurrency and termination are theory extensions of the replacement core: they require the C1 derivation semantics, but add further assumptions or structures for reasoning about overlapping derivations, parallel application, composition, or well-founded measures [17, 35, 44]. Querying, verification, repair, and AI interfaces rest on execution, property, or interaction commitments that go beyond substrate support [20, 23, 26, 29]. Table 2 therefore separates theory extensions (E1–E3) from support dimensions (S1–S4).

5. Three Exhibits and One Boundary Case

We check the contract family against three positive exhibits and one deliberate boundary case. They are not exhaustive evaluations, but evidence-carrying cases selected from the anchor set.

X1: typed attributed graphs. Typed attributed graphs are the safest positive exhibit. They are close to mainstream GT practice and still rich enough to host conditions, repair ranking, and quantitative variants without combining support services into the core graph family itself. Older work on attributed graph transformation with type inheritance remains relevant here [45]. In the recent evidence, weighted type graphs, application-condition reasoning, and granular conflict analysis all fit comfortably in this setting [17, 27, 15].

X2: partial triple graphs and TGG-style correspondence structures. This exhibit makes the service boundary clear. Triple-graph or correspondence-based settings clearly inherit a replacement core, but their main added value lies in synchronisation, restoration, and repair. The higher-order shortcut-rule paper is especially useful here because it shows that sophisticated restoration mechanisms can be built above the core rather than folded into the universal graph-family contract [26]. This is also where systems such as Fujaba and eMoflon::IBeX become particularly informative as comparison points [3, 46].

X3: hypergraph-oriented or traced-monoidal settings. This exhibit shows family breadth without becoming boundary-only material. Rewriting for traced monoidal closed categories provides strong evidence for C1 beyond standard typed graphs [13]. Hypergraph parsing provides a second example that the object-and-morphism interface cannot be phrased too narrowly [37]. The evidence is more cautious on whether a full C2 story should be claimed uniformly here, so Table 3 marks this case as a partial rather than complete fit.

B1: bigraphs as a deliberate boundary case. The bigraphs paper in the anchor set was retained as a scope-marking case. It is useful because it widens the family space, not because it carries the fine-grained extraction of replacement or condition contracts. It shows that the proposed vocabulary is not merely a renaming of typed attributed graphs, while also marking where the current evidence stops [38]. Directed bigraphs strengthen this point: their algebraic presentation makes location, connection, and resource structure explicit at the object/interface level, showing that C0 may itself require nontrivial algebraic structure before replacement-style reasoning is considered [48]. Bigraphs can be compared with gs-monoidal theories and gs-graphs, suggesting that even before replacement semantics is fixed, the object/interface layer may admit several nontrivial algebraic presentations [47].

6. Closing Discussion

Support profiles make differences among graph transformation frameworks more explicit. AGG and PROGRES illustrate early environment-oriented lines of work [1, 2]. AToM^3, Fujaba, and VIATRA show how graph transformation has been embedded in broader model-driven settings [4, 3, 5]. GrGen.NET emphasises expressive and high-performance rewriting [6]. Henshin is a strong reference point for in-place EMF transformation [8], GROOVE for state-space generation and analysis [9], and eMoflon::IBeX for incremental uni- and bidirectional model transformation [10, 46]. Describing such systems by support profiles over graph families would make these differences easier to compare without forcing them into one graph model.

The same separation also helps to read the recent literature. Localised RETE networks and rule-based query solving belong primarily to execution and querying [20, 21]. Alternating graph automata and CEGAR belong to verification and property support [23, 24, 25]. Higher-order shortcut rules and repair ranking belong to repair and bidirectionality [26, 27]. Computational notebooks, chatbot interfaces, reinforcement-learning-guided testing, and related work belong to interaction, search, and experimentation [28, 29, 30, 31]. This reading preserves the diversity of the recent literature while making its layers easier to compare.

6.1. Threats to validity

The main threat is corpus-selection bias. The proceedings part of the corpus is venue-based, but the journal part is different, since it is a targeted title-based harvest from selected journals. This choice made the journal search tractable and conservative, yet also left many false negatives behind that did not explicitly have “graph transformation” in the title (think of neighbouring terminology such as graph rewriting, model transformation, bigraphs, reactive systems, string diagrams, categorical rewriting, graph grammars, etc). The corpus should therefore be read as a bounded exploratory corpus, not as an exhaustive systematic map of recent work.

A related threat concerns the manually added 2026 SoSyM paper on model transformation languages. We included this exception because it directly informs the paper’s framework-design question: whether transformation languages and frameworks should be compared by their explicit value propositions and support profiles, rather than simply as standalone languages. To avoid overstating its role, we use it as contextual motivation for the design-space framing, not as evidence for the extraction of the C0–C2 core contracts.

A second threat concerns the status of the papers retained after the confidence audit. The 35 core-extraction papers provide the evidence from which the core contracts and theory extensions were

abstracted. The remaining 9 scope-marking papers were retained for a different purpose: they test the edges of the vocabulary by showing graph-like settings that fit the coding scheme only partially. They are therefore not used to justify fine-grained claims such as "this family satisfies C1" or "this family supports C2". When section 5 discusses such cases, the claim is deliberately weaker: they show where the vocabulary appears useful, where it becomes only partial, and where additional structure would be needed before assigning a full support profile.

A third threat is abstraction loss. Support profiles deliberately compress framework and graph-family differences into a small set of contracts, theory extensions, and support dimensions. This is useful for comparison, but it can hide distinctions that matter in particular families. Bigraphs are the clearest example in this paper. Treating them merely as graph-like objects with morphisms risks losing the separation between place structure and link structure, and treating them as a simple instance of replacement semantics risks ignoring the specific reactive-system assumptions under which bigraph rewriting is usually studied. Similar losses can occur for hypergraph, traced-monoidal, or correspondence-based settings, where the relevant interfaces, composition operations, or consistency relations are part of the technical substance rather than incidental details. The proposed contracts should therefore be read as a comparison vocabulary, not as a substitute for family-specific semantics.

Finally, the coding scheme reflects the question asked in this paper: what must a framework assume before it can claim support for a graph family or for services built on top of rewriting? This makes the scheme more sensitive to framework design, semantics, conditions, verification, repair, and execution support than to application domains or empirical tool adoption. A different study organised around user communities, benchmark performance, or industrial uptake would likely produce a different classification.

6.2. Conclusion

The recent graph transformation literature supports a more precise organising principle than either a catalogue of named tools or a search for one universal graph model. Across the retained evidence, the strongest common layer is a family of contracts for an object-and-morphism interface, replacement semantics, and condition formalisms interpreted over that interface. Concurrency, quantitative reasoning, and graph programming are found naturally above that core as theory extensions. Querying, verification, repair, and AI-facing interaction belong to support dimensions that depend on additional commitments.

A practical next step for framework design is to describe support in terms of explicit profiles over graph families and to compare systems by those profiles. This is neither a replacement for established tools nor a claim that all graph transformation work should converge on one calculus. It is a way of stating assumptions and capabilities more clearly, while preserving the family-specific semantics that make graph transformation frameworks technically interesting.

References

[1] G. Taentzer, AGG: A Graph Transformation Environment for Modeling and Validation of Software, in: AGTIVE'03, 2004. doi:10.1007/978-3-540-25959-6_35.

[2] B. Böhlen, U. Ranger, Concepts for Specifying Complex Graph Transformation Systems, in: ICGT'04, 2004. doi:10.1007/978-3-540-30203-2_9.

[3] L. Grunske, L. Geiger, M. Lawley, A Graphical Specification of Model Transformations with Triple Graph Grammars, in: ECMDA-FA'05, 2005. doi:10.1007/11581741_21.

[4] J. de Lara, H. Vangheluwe, AToM^3: A Tool for Multi-formalism and Meta-modelling, in: FASE'02, 2002. doi:10.1007/3-540-45923-5_12.

[5] 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, SoSyM (2016). doi:10.1007/s10270-016-0530-4.

[6] R. Geiß, M. Kroll, GrGen.NET: A Fast, Expressive, and General Purpose Graph Rewrite Tool, in: AGTIVE'07, 2008. doi:10.1007/978-3-540-89020-1_38.
[7] D. Plump, The Design of GP 2, in: S. Escobar (Ed.), Proceedings 10th International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2011, EPTCS, 2011, pp. 1–16. doi:10.4204/EPTCS.82.1.
[8] T. Arendt, E. Biermann, S. Jurack, C. Krause, G. Taentzer, Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations, in: MoDELS'10, 2010. doi:10.1007/978-3-642-16145-2_9.
[9] A. H. Ghamarian, M. de Mol, A. Rensink, E. Zambon, M. Zimakova, Modelling and Analysis Using GROOVE, STTT (2012). doi:10.1007/s10009-011-0186-x.
[10] N. Weidmann, A. Anjorin, P. Robrecht, G. Varró, Incremental (Unidirectional) Model Transformation with eMoflon::IBeX, in: ICGT'19, 2019. doi:10.1007/978-3-030-23611-3_8.
[11] K. Brown, E. Patterson, T. Hanks, J. P. Fairbanks, Computational Category-Theoretic Rewriting, in: ICGT'22, 2022. doi:10.1007/978-3-031-09843-7_9.
[12] H.-J. Kreowski, A. Lye, A. Windhorst, Extension and Restriction of Derivations in Adhesive Categories, in: ICGT'24, 2024. doi:10.1007/978-3-031-64285-2_4.
[13] A. Di Giorgio, D. R. Ghica, F. Zanasi, Rewriting for Traced Monoidal Closed Categories, in: ICGT'25, 2025. doi:10.1007/978-3-031-94706-3_2.
[14] S. Schneider, L. Lambers, Evaluation Diversity for Graph Conditions, in: ICGT'21, 2021. doi:10.1007/978-3-030-78946-6_7.
[15] A. Lauer, J. Kosiol, G. Taentzer, Granular Conflict Analysis for Transformation Rules with Application Conditions, in: ICGT'25, 2025. doi:10.1007/978-3-031-94706-3_4.
[16] R. Overbeek, J. Endrullis, Termination of Graph Transformation Systems Using Weighted Subgraph Counting, in: ICGT'23, 2023. doi:10.1007/978-3-031-36709-0_5.
[17] J. Endrullis, R. Overbeek, Generalized Weighted Type Graphs for Termination of Graph Transformation Systems, in: ICGT'24, 2024. doi:10.1007/978-3-031-64285-2_3.
[18] Q. Qiu, Termination of Injective DPO Graph Rewriting Systems Using Subgraph Counting, in: ICGT'25, 2025. doi:10.1007/978-3-031-94706-3_1.
[19] M. Barkowsky, H. Giese, Host-Graph-Sensitive RETE Nets for Incremental Graph Pattern Matching, in: ICGT'21, 2021. doi:10.1007/978-3-030-78946-6_8.
[20] M. Barkowsky, H. Giese, Localized RETE for Incremental Graph Queries, in: ICGT'24, 2024. doi:10.1007/978-3-031-64285-2_7.
[21] D. Duval, R. Echahed, F. Prost, A Rule-Based Procedure for Graph Query Solving, in: ICGT'23, 2023. doi:10.1007/978-3-031-36709-0_9.
[22] F. Gadducci, A. Laretto, D. Trotta, Specification and Verification of a Linear-Time Temporal Logic for Graph Transformation, in: ICGT'23, 2023. doi:10.1007/978-3-031-36709-0_2.
[23] B. König, A. Rensink, L. Stoltenow, F. Urrigshardt, Counterexample-Guided Abstraction Refinement for Generalized Graph Transformation Systems, in: ICGT'25, 2025. doi:10.1007/978-3-031-94706-3_8.
[24] F. Drewes, B. Hoffmann, M. Minas, Graph Formulas and Their Translation to Alternating Graph Automata, in: ICGT'25, 2025. doi:10.1007/978-3-031-94706-3_6.
[25] F. Drewes, B. Hoffmann, M. Minas, Specifying and Checking Graph Properties with Alternating Graph Automata, in: ICGT'25, 2025. doi:10.1007/978-3-031-94706-3_5.
[26] L. Fritsche, J. Kosiol, A. Möller, A. Schürr, Advanced Consistency Restoration with Higher-Order Short-Cut Rules, in: ICGT'23, 2023. doi:10.1007/978-3-031-36709-0_10.
[27] L. Fritsche, A. Lauer, A. Schürr, G. Taentzer, Using Application Conditions to Rank Graph Transformations for Graph Repair, in: ICGT'24, 2024. doi:10.1007/978-3-031-64285-2_8.
[28] J. H. Weber, *GrapePress* — A Computational Notebook for Graph Transformations, in: ICGT'21, 2021. doi:10.1007/978-3-030-78946-6_16.
[29] R. Heckel, I. Al-Azzoni, Can I Teach Graph Rewriting to My Chatbot?, in: ICGT'24, 2024. doi:10.1007/978-3-031-64285-2_12.
[30] S. Ghasemi, V. Rafe, M. Mehrabi, R. Heckel, I. Al-Azzoni, Test Case Generation from Graph

Transformation Systems Using Deep Reinforcement Learning, in: ICGT'25, 2025. doi:10.1007/978-3-031-94706-3_9.
[31] S. Ghasemi, M. Asgari Araghi, V. Rafe, R. Heckel, MoTDeReL: Model-Based Testing Through Deep Reinforcement Learning for Software Systems Specified Through Graph Transformation, ASE (2026). doi:10.1007/s10515-026-00610-3.
[32] J. de Lara, E. Guerra, J. S. Cuadrado, Have Model Transformation Languages Failed?, SoSyM (2026). doi:10.1007/s10270-026-01360-2.
[33] A. M. Soaita, B. Serin, J. Preece, A Methodological Quest for Systematic Literature Mapping, International Journal of Housing Policy 20 (2020) 320–343. doi:10.1080/19491247.2019.1649040.
[34] R. Overbeek, J. Endrullis, A. Rosset, Graph Rewriting and Relabeling with PBPO$^+$, in: ICGT'21, 2021. doi:10.1007/978-3-030-78946-6_4.
[35] J. Kosiol, G. Taentzer, A Generalized Concurrent Rule Construction for Double-Pushout Rewriting, in: ICGT'21, 2021. doi:10.1007/978-3-030-78946-6_2.
[36] G. Taentzer, S. John, J. Kosiol, A Generic Construction for Crossovers of Graph-Like Structures, in: ICGT'22, 2022. doi:10.1007/978-3-031-09843-7_6.
[37] G. Costagliola, F. Vastarini, Parsing Hypergraphs Using Context-Free Positional Grammars, in: GCM'25, 2025. doi:10.4204/EPTCS.440.1.
[38] B. Archibald, M. Sevegnani, A Bigraphs Paper of Sorts, in: ICGT'24, 2024. doi:10.1007/978-3-031-64285-2_2.
[39] R. Heckel, G. Taentzer, Graph Transformation for Software Engineers, Springer, 2020. doi:10.1007/978-3-030-43916-3.
[40] H. Ehrig, A. Habel, J. Padberg, U. Prange, Adhesive High-Level Replacement Categories and Systems, in: ICGT'04, 2004. doi:10.1007/978-3-540-30203-2_12.
[41] H. Ehrig, U. Golas, A. Habel, L. Lambers, F. Orejas, $\mathcal{M}$-Adhesive Transformation Systems with Nested Application Conditions, MSCS (2014). doi:10.1017/S0960129512000357.
[42] M. Bauderon, B. Courcelle, An Algebraic Formalism for Graphs, in: CAAP'86, Springer, 1986, pp. 74–84. doi:10.1007/BFb0022660.
[43] R. Söldner, D. Plump, Mechanised DPO Theory: Uniqueness of Derivations and Church-Rosser Theorem, in: ICGT'23, 2023. doi:10.1007/978-3-031-36709-0_7.
[44] H.-J. Kreowski, A. Lye, Parallel Rule Application with Doubling Avoidance, in: ICGT'25, 2025. doi:10.1007/978-3-031-94706-3_3.
[45] J. de Lara, R. Bardohl, H. Ehrig, K. Ehrig, U. Prange, G. Taentzer, Attributed Graph Transformation with Node Type Inheritance, TCS (2007). doi:10.1016/j.tcs.2007.02.001.
[46] N. Weidmann, A. Anjorin, L. Fritsche, G. Varró, A. Schürr, E. Leblebici, Incremental Bidirectional Model Transformation with eMoflon::IBeX, in: BX'19, 2019. URL: https://ceur-ws.org/Vol-2355.
[47] R. Bruni, U. Montanari, G. D. Plotkin, D. Terreni, On Hierarchical Graphs: Reconciling Bigraphs, gs-monoidal Theories and gs-graphs, Fundamenta Informaticae 134 (2014) 287–317. doi:10.3233/FI-2014-1103.
[48] D. Grohmann, M. Miculan, An Algebra for Directed Bigraphs, TERMGRAPH'07, ENTCS 203 (2008) 49–63. doi:10.1016/j.entcs.2008.03.033.

Declaration on Generative AI

The author used ChatGPT versions 5.3–5.5, with Extended Thinking enabled, as an interactive research assistant during the preparation of this paper. The tool was used to explore and challenge research ideas, organise the staged literature mapping process, draft intermediate summaries and tables, and suggest revisions to wording and structure. Its role was not to replace the author's judgement, but to support a more systematic and reflective understanding of the design space studied in the paper. The author selected the final argument, checked and revised the substance of the claims, verified the cited sources to the best of his ability, and takes full responsibility for the content of the paper.