Interweaving Graphical and Textual Modelling using GLSP and Langium

Bofan Zhang*
bofan.zhang@kcl.ac.uk
Department of Informatics
King's College London
London, United Kingdom

Andreas Hell*
e12123458@student.tuwien.ac.at
Business Informatics Group
TU Wien
Vienna, Austria

Steffen Zschaler
szschaler@acm.org
Department of Informatics
King's College London
London, United Kingdom

Dominik Bork
dominik.bork@tuwien.ac.at
Business Informatics Group
TU Wien
Vienna, Austria

Abstract

Many domain-specific modelling languages (DSMLs) combine graphical and textual elements. While simple textual labels are often sufficient, some situations (e.g., condition expressions in flowchart-like languages) require more complex textual sub-languages with their own grammar and the ability to reference elements defined elsewhere in the model, providing full scoping support for such references. Developing tool support for DSMLs that integrate expressive graphical and textual elements requires combining two distinct paradigms for language processing: projectional editing for graphical elements and a parsing-based approach for textual elements. A purely projectional approach quickly becomes inconvenient, and parsing-based approaches currently do not handle graphical languages well. For modern web-based language workbenches, such integrations are not well supported. We present a reusable framework for integrating text-based grammar support using Langium into the graphical model of GLSP-based languages. The backend maintains a complete abstract syntax graph, regardless of whether a particular part of the model is edited graphically or textually. Langium-managed text elements automatically handle scoping across the entire model, enabling scenarios where a text node's position in the graphical model can influence which elements are available for linking. The framework is available open source, and we invite the community to integrate it into their own languages.

CCS Concepts

• **Software and its engineering → Visual languages**; **Syntax**; **Development frameworks and environments**.

Keywords

Language Engineering, GLSP, Langium, Blended modelling, Hybrid modelling, Modelling tool

ACM Reference Format:
Bofan Zhang, Andreas Hell, Steffen Zschaler, and Dominik Bork. 2026. Interweaving Graphical and Textual Modelling using GLSP and Langium. In *ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion 2026), October 04–09, 2026, Málaga, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3837062.3838863

1 Introduction

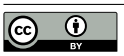
Many domain-specific modelling languages (DSMLs) combine graphical and textual elements. While it is often possible to get away with simple textual labels, many situations require more complex textual sub-languages. For example, consider the use of condition expressions in flowchart-like languages or action specifications in statechart-like languages. These sub-languages have their own grammar and must be able to reference elements defined elsewhere in the model, providing full scoping support for such references.

Developing tool support for such DSMLs can be complicated, not least because textual and graphical modelling languages are typically supported via different language workbenches [6]. As language engineering evolves towards web-based language workbenches—such as Langium[1] for textual languages and the Graphical Language Server Platform (GLSP)[2] [12] for graphical languages—language engineers need to address the challenges of integrating these distinct software packages.

In this tool demonstration, we present a reusable framework for efficiently integrating Langium's text-based grammar support into the graphical model of GLSP-based languages. Our integration allows Langium grammars to be attached to string-typed attributes of model elements, with the abstract syntax graphs generated by parsing these textual elements integrated into the model's overall abstract syntax graph. Textual elements can thus reference other model elements, and their scoping can be constrained by their position in the graphical model. The framework is available as open

*Both authors contributed equally to this research.

[1] langium.org
[2] www.eclipse.dev/glsp/



MODELS Companion 2026, Málaga, Spain

ACM ISBN 979-8-4007-2903-4/2026/10
https://doi.org/10.1145/3837062.3838863

source[3]. This repository also includes a short video showcasing the framework in action and how to create a language and editor with it.[4] The purpose of this demonstration is to make the framework more generally available and invite feedback from the community.

In the remainder of this paper, we briefly discuss related work in Section 2. Section 3 then provides a motivating example for interweaving textual and graphical modelling whose abstraction yields general interweaving challenges. Our framework, aimed at addressing these challenges, is presented in Section 4 before we conclude the paper in Section 5.

2 Related Work

It has been recognised that modellers benefit from using the most appropriate notation for different aspects of a model [3]. There is a spectrum of possibilities for combining such different notations. On one end, multi-view modelling [4] focuses on providing specialised concepts and notations for different stakeholders who offer different perspectives on the same system.

On the other end, blended modelling [5] integrates different notations for the same set of concepts, aiming to provide a better user experience for modellers. Often, these approaches take the form of a textual modelling language combined with a graphical visualisation that can be statically generated from the textual model (e.g., PlantUML[5], Mermaid[6]). Tools like UMPLE [11], BIGER [7], and cross-model[7]provide a typical blended-modelling experience, where both the textual and graphical representations can be edited concurrently. Latifaj establishes core principles for the systematic development of such tools in her thesis [10].

Projectional editing approaches, such as those supported in JetBrains MPS [2] and Gentleman [9], provide multiple notations for the same set of concepts within a single editor, allowing users to seamlessly switch between graphical and textual representations. They provide different notations *in the same view*, and thus conceptually sit somewhere between multi-view modelling (different notations for different concepts in different views) and blended modelling (different notations for the same concepts in different views). This provides more fine-grained control over the notational choices—for example, it becomes possible to mix graphical, textual, and tabular notations inside the same editor and even switch between them on a per-element basis. However, projectional editors come at a high development cost. For example, they do not natively support traditional parsing-based editing of textual languages, though extensions such as grammar cells in MPS can help to alleviate this [14]. This has inspired research into fine-grained integration of graphical and textual notations in traditional language workbenches—a recent example being Graphite [13].

However, as language workbenches increasingly move to web-based platforms [1, 8, 15], the community lacks a reusable framework for interweaving graphical and textual modelling *in the same view* in web-based environments. In this tool paper, we present exactly such a framework, based on two prominent web-based language workbenches: GLSP for graphical modelling and Langium for textual modelling.

3 Motivating Example and Challenges

In the following, we first illustrate a motivating example (Section 3.1) before introducing generalized challenges for interweaving graphical and textual modelling (Section 3.2).

3.1 The Workflow Modelling Example

Figure 1 shows the official GLSP workflow example[8], which we extend with textual condition expressions and use as an example of interweaving graphical and textual modelling[3]. A workflow is a graph of step nodes connected by control edges, where decisions branch the flow along alternative paths. A step node may declare variables: the `ChkWt` step in Figure 1 declares a `water` variable with a `level` attribute. We would like to record *why* the flow takes one branch rather than another, annotating each decision edge with the condition under which it is followed—e.g., `water.level` ≥ 50.

In GLSP—and in most common workflow modelling tools—, however, an edge carries only a simple label, which cannot express a condition of this kind: it has its own grammar and, to be meaningful, must refer to variables declared elsewhere in the model and within its scope. Which variables it may reference depends on where the edge sits in the control flow, since a variable is only in scope once a preceding step has declared it. Supporting such conditions thus means interweaving a textual sub-language—with its own grammar and model-wide scoping—into a graphical host language.

With our framework, each decision edge carries its condition in a text editor (a Monaco editor[9]) embedded directly inside the graphical diagram, and the condition references variables declared in *other*, graphical nodes, so the text scopes into the surrounding model rather than only over its own contents. As the modeller types, the editor offers only the attributes that are in scope at that edge (Figure 1), as determined by the node's position in the control flow. When a reference cannot be resolved—because the variable is not declared on any path reaching the edge—the editor flags it in place with the usual validation markers.

3.2 Interweaving Challenges

To interweave graphical and textual modelling, with a graphical host language and textual sub-languages, we can generalise the following set of challenges from the motivating example presented above:

- **C1.** *Integration of language workbenches:* Graphical languages are typically supported via projectional editing, while textual languages are best supported through parsing. Language workbenches rarely support both paradigms well, so we need to technically integrate two different paradigms in a single editing environment.
- **C2.** *Integration of abstract syntax:* The abstract syntax elements of the graphical and textual sub-languages must be integrated into a single abstract syntax graph, so that the model

[3] https://github.com/YourHarbour/glsp-langium-integration
[4] https://github.com/YourHarbour/glsp-langium-integration/blob/main/demo/demo.mp4
[5] plantuml.com
[6] mermaid.js.org
[7] https://github.com/eclipsesource/crossmodel
[8] https://github.com/eclipse-glsp/glsp-examples
[9] https://microsoft.github.io/monaco-editor/

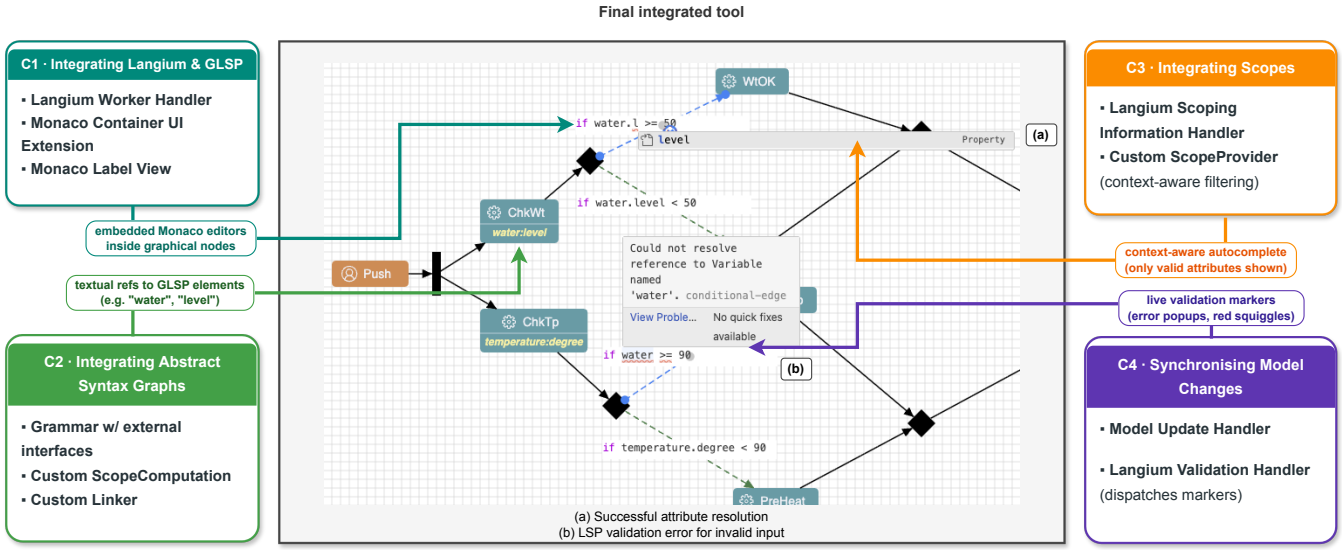


Figure 1: Our framework applied to our running example: components C1–C4 (left) and the tool capabilities they enable (right).

can be processed as a whole by downstream model management activities, regardless of whether particular elements were created via graphical or textual editing.

C3. *Integration of scopes:* Scoping in the textual sub-languages must be able to reference elements defined in the graphical model, and vice versa. It should be further possible to constrain the scoping rules of the textual sub-languages based on the position of the textual element in the graphical model, e.g., so that a condition expression attached to an edge in a flowchart can only reference elements defined in the source and target nodes of that edge.

C4. *Synchronisation of model changes:* Changes to the model made via graphical editing must be reflected in the textual elements, and changes made via textual editing must be reflected in the graphical elements. In particular, scopes in relevant textual elements must be dynamically reevaluated after changes to the graphical model, and the user must be notified of any resulting issues (e.g., dangling references).

As a language engineer, we would like to develop a language in an integrated manner, minimising the need to implement custom code to address these challenges. In the following sections, we present our reusable framework for addressing these challenges, using GLSP and Langium as the technological foundation.

4 A Reusable Framework for Interweaving Graphical and Textual Modelling

In this section, we discuss how our framework addresses the four interweaving challenges introduced in Section 3.2. The framework targets languages with a graphical host language and one or more textual sub-languages, such as the workflow language from Section 3.1, in which decision edges contain condition expressions. For each challenge, we describe what a language engineer needs to provide when developing a new language, as well as what the framework contributes. Our open-source repository[3] also contains a statemachine example with alternative integration styles.

4.1 C1: Integrating Langium and GLSP

Challenge C1 concerns integrating two editing paradigms: GLSP provides projectional editing for graphical model elements, while Langium provides parsing, validation, linking, and editor services for textual languages. Our framework keeps these responsibilities separate but connects them in the browser. The GLSP client remains the primary diagram editor and communicates with the GLSP server as usual. For each textual sub-language, the framework starts a Langium language server in a web worker and embeds Monaco editors as editable labels inside the GLSP diagram. A textual element is therefore still part of the graphical diagram, but its contents are edited with the services expected from a textual language, such as syntax checking, completion, and diagnostics.

To instantiate the framework for a new language, a language engineer identifies the graphical element types that should contain grammar-controlled text, provides the corresponding Langium grammars, and registers how those textual elements are rendered in the diagram. In the workflow example, the graphical host language provides workflow nodes and edges, while the textual sub-language is attached to decision edges and describes condition expressions. The framework handles recurring infrastructure tasks: it creates and reuses Monaco editors, connects them to the appropriate Langium language server, and routes validation information back to the GLSP client. This addresses the integration of language workbenches without requiring the language engineer to reimplement either GLSP's graphical editing or Langium's textual editing services.

4.2 C2: Integrating Abstract Syntax Graphs

Challenge C2 requires the graphical and textual parts of a model to form a single abstract syntax graph. In our framework, the language engineer defines each textual sub-language as a Langium grammar

and attaches it to selected string-valued properties of graphical model elements. The attachment determines which Langium language is used for a particular textual element. For example, a condition property on a decision edge can use the condition grammar, while another property in the same diagram could use a different grammar. The framework uses this mapping to create a virtual Langium document for each grammar-controlled textual element and to route it to the correct Langium module.

When a modeller edits such a textual element, Langium parses the text and produces the corresponding abstract syntax information. The framework sends both the edited text and the parsed information to the GLSP side, where the language engineer constructs or updates the domain-specific abstract syntax graph in the same way as for a pure GLSP language. This design keeps the framework reusable: it does not prescribe a particular meta-model for conditions, expressions, or other textual sub-languages. In practice, it is often useful to store the concrete text alongside the abstract syntax elements produced by parsing, because Langium does not currently provide an unparsing mechanism to reconstruct source text from an abstract syntax tree. This can introduce inconsistencies between the text and its parsed structure, so the framework treats the parsed information as the result of the current text edit and updates both together. A language engineer can implement a custom unparser for stricter separation, but this is not required.

4.3 C3: Integrating Scopes

Challenge C3 is the main semantic challenge. A textual sub-language must be able to refer to elements that are not parsed from the text itself, but are defined in the surrounding graphical model. In Langium, such external elements can be represented in the grammar using `interface` declarations. A condition grammar for the workflow example can therefore declare the shape of externally visible variables and attributes, even though these elements are created graphically as part of workflow steps. The framework then injects descriptions of the relevant graphical elements into Langium's scope computation, so references in the text can be linked against model elements outside the textual document.

We distinguish two kinds of scoping rules. *Local textual scoping* is implemented with standard Langium scope providers and determines which references are valid at a particular point in a grammar-controlled text. *Graphical context scoping* is implemented on the GLSP side and determines which graphical model elements are visible to a particular textual element as a whole. In the workflow example, graphical context scoping computes which variables are available before a decision edge, based on the edge's position in the control flow. Local textual scoping can then, within the condition expression, determine whether an attribute reference is valid for the selected variable. This separation keeps Langium responsible for language-internal scoping while the GLSP model contributes the position-dependent context required by the host language.

4.4 C4: Synchronising Model Changes

Challenge C4 arises because edits in either paradigm can affect the other. A graphical edit may add, remove, rename, or move an element referenced from a textual condition; a textual edit may change the abstract syntax graph stored in the model. The framework treats both cases as model changes that must be propagated through GLSP, Langium, and the embedded label editors. The language engineer does not need to implement a separate synchronisation protocol: the framework observes GLSP model updates, refreshes the graphical context scoping information sent to Langium, and asks Langium to revalidate the affected textual elements.

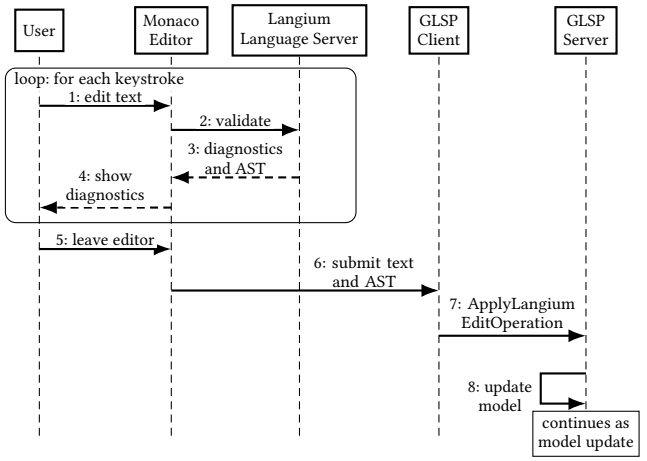


Figure 2: Synchronisation after an edit in a grammar-controlled textual element.

Figure 2 shows the sequence for edits in a grammar-controlled textual element. While the modeller types, the Monaco editor sends the current text to the Langium language server. Langium parses and validates the text on every keystroke and returns diagnostics, together with the parsed abstract syntax information required by the framework. Monaco can therefore show immediate textual feedback without waiting for the whole GLSP model to be saved. When the modeller leaves the editor, the framework submits the text and parsed information to the GLSP client, which forwards an edit operation to the GLSP server. The server then updates the domain model, including the textual element's contribution to the overall abstract syntax graph.

Figure 3 shows the corresponding sequence for graphical edits. A GLSP editing action is sent to the GLSP server and applied to the server-side model. The updated model is sent back to the GLSP client as an *UpdateModelAction*. On the client, the framework creates Monaco editors for newly created grammar-controlled textual elements. It then recomputes the graphical context scoping information and sends it to the Langium language server, before triggering validation of the affected textual elements. A graphical edit can thus invalidate a textual reference immediately: for example, if a workflow step is renamed or no longer precedes a decision edge, the condition on that edge is revalidated against the new context.

The framework also manages the interaction between validation results and diagram feedback. Monaco shows immediate textual diagnostics while the modeller edits, and the GLSP client receives validation results that can be shown as markers on the corresponding graphical elements. Internally, the framework uses a message bus to communicate among the GLSP client, the embedded label components, and the Langium language server worker, so that

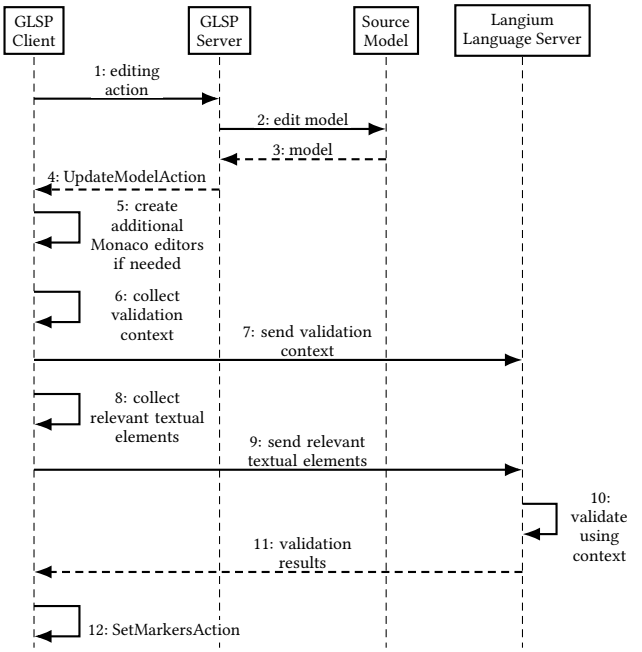


Figure 3: Synchronisation and validation after a graphical model edit.

validity annotations are updated when either the text or the surrounding model changes. The same mechanism is used after textual and graphical edits, keeping the terminology consistent: both kinds of edits update the interwoven model, refresh graphical context scoping where needed, and revalidate grammar-controlled textual elements. Synchronisation is therefore not a one-off transformation between text and graphics, but a continuous propagation of changes across the interwoven model.

None of these mechanisms is tied to string-valued properties: as the language engineer controls how parsed information enters the abstract syntax graph, a textual sub-language may instead be modelled in the meta-model, leaving the string-valued property to carry only concrete syntax. Scoping is likewise computed over the entire model, so referable elements may originate from textual as well as graphical elements, with changes propagating between textual elements alike.

5 Conclusion

In this tool demonstration paper, we showed how Langium-based textual editors can be easily interweaved into GLSP-based graphical languages using our reusable framework. The framework addresses the key challenges of interweaving language workbench paradigms, abstract syntax, scopes, and model change synchronisation, enabling language engineers to develop interweaved graphical and textual languages with minimal custom implementation effort. A demonstration video is available in the tool repository[3].

Future work includes an empirical evaluation with language engineers, addressing adoption effort and scalability to larger meta-models with many textual elements.

Acknowledgments

This study was partially funded by the Austrian Research Agency (FFG)-funded project Smart GLSP – Facilitating Large Language Models for Smart GLSP-based Modelling (Grant #FO999925707) and by the Bachelor with Honours program of TU Wien. BoFan Zhang's contribution was partly funded by a CSC Scholarship (Grant #202408890021).

References

[1] Dominik Bork, Philip Langer, and Tobias Ortmayr. 2023. *A Vision for Flexible GLSP-Based Web Modeling Tools.* Springer Nature Switzerland, 109–124. doi:10.1007/978-3-031-48583-1_7

[2] Antonio Bucchiarone, Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio (Eds.). 2021. *Domain-Specific Languages in Practice: with JetBrains MPS.* Springer International Publishing. doi:10.1007/978-3-030-73758-0

[3] Paulo Carreira, Vasco Amaral, and Hans Vangheluwe (Eds.). 2020. *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems.* Springer International Publishing. doi:10.1007/978-3-030-43946-0

[4] Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio. 2019. Multi-view approaches for software and system modelling: a systematic literature review. *Software and Systems Modeling* 18, 6 (Feb. 2019), 3207–3233. doi:10.1007/s10270-018-00713-w

[5] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. 2019. Blended Modelling – What, Why and How. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C).* IEEE, 425–430. doi:10.1109/models-c.2019.00068

[6] Sebastian Erdweg, Tijs van der Storm, Markus Völter, Laurence Tratt, Remi Bosman, William R. Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh, Gabriël Konat, Pedro J. Molina, Martin Palatnik, Risto Pohjonen, Eugen Schindler, Klemens Schindler, Riccardo Solmi, Vlad Vergu, Eelco Visser, Kevin van der Vlist, Guido Wachsmuth, and Jimi van der Woning. 2015. Evaluating and comparing language workbenches: Existing results and benchmarks for the future. *Computer Languages, Systems & Structures* 44 (2015), 24–47. doi:10.1016/j.cl.2015.08.007 Special issue on the 6th and 7th Int'l Conf Software Language Engineering (SLE 2013 and SLE 2014).

[7] Philipp-Lorenz Glaser and Dominik Bork. 2021. The bigER Tool - Hybrid Textual and Graphical Modeling of Entity Relationships in VS Code. In *25th International Enterprise Distributed Object Computing Workshop, EDOC Workshop 2021, Gold Coast, Australia, October 25-29, 2021.* IEEE, 337–340. doi:10.1109/EDOCW52865.2021.00066

[8] Lennart C.L. Kats, Richard G. Vogelij, Karl Trygve Kalleberg, and Eelco Visser. 2012. Software development environments on the web: a research agenda. In *Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (SPLASH '12).* ACM, 99–116. doi:10.1145/2384592.2384603

[9] Louis-Edouard Lafontant and Eugene Syriani. 2020. Gentleman: a light-weight web-based projectional editor generator. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings.* Association for Computing Machinery, Article 1, 5 pages. doi:10.1145/3417990.3421998

[10] Malvina Latifaj. 2024. *Systematic Development of Collaborative Blended Modeling Environments.* Ph. D. Dissertation. Mälardalen University, Västerås, Sweden.

[11] Timothy C. Lethbridge, Andrew Forward, Omar Badreddin, Dusan Brestovansky, Miguel Garzon, Hamoud Aljamaan, Sultan Eid, Ahmed Husseini Orabi, Mahmoud Husseini Orabi, Vahdat Abdelzad, Opeyemi Adesina, Aliaa Alghamdi, Abdulaziz Algablan, and Amid Zakariapour. 2021. Umple: Model-driven development for open source and education. *Science of Computer Programming* 208 (2021), 102665. doi:10.1016/j.scico.2021.102665

[12] Haydar Metin and Dominik Bork. 2025. A reference architecture for the development of GLSP-based web modeling tools. *Software and Systems Modeling* (jan 2025). doi:10.1007/s10270-024-01257-y

[13] Ionut Predoaia, Dimitris Kolovos, and Antonio García-Domínguez. 2025. Graphite: Automated Development of Hybrid Graphical-Textual DSL Editors. In *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C).* IEEE, 89–93. doi:10.1109/models-c68889.2025.00021

[14] Markus Voelter, Tamás Szabó, Sascha Lisson, Bernd Kolb, Sebastian Erdweg, and Thorsten Berger. 2016. Efficient development of consistent projectional editors using grammar cells. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering (SLE'16).* ACM, 28–40. doi:10.1145/2997364.2997365

[15] Jos Warmer and Anneke Kleppe. 2022. Freon: An Open Web Native Language Workbench. In *Proceedings of the 15th ACM SIGPLAN International Conference on Software Language Engineering (SLE '22).* ACM, 30––35. doi:10.1145/3567512.3567515