



jjodel – A reflective cloud-based modeling framework

Juri Di Rocco
Università degli Studi dell'Aquila
L'Aquila, Italy
juri.dirocco@univaq.it 

Davide Di Ruscio
Università degli Studi dell'Aquila
L'Aquila, Italy
davide.diruscio@univaq.it 

Amleto Di Salle
European University of Rome
Rome, Italy
amleto.disalle@unier.it 

Damiano Di Vincenzo
Università degli Studi dell'Aquila
L'Aquila, Italy
damiano.divincenzo@student.univaq.it

Alfonso Pierantonio
Università degli Studi dell'Aquila
L'Aquila, Italy
alfonso.pierantonio@univaq.it 

Giordano Tinella
Università degli Studi dell'Aquila
L'Aquila, Italy
giordano.tinella@student.univaq.it

Abstract—Model-Driven Engineering (MDE) relies on modeling frameworks that have proven scalable and successful. However, the accumulated technical debt has reduced effectiveness, among other quality factors. *jjodel* is a cloud-based, reflective modeling software that aims to reduce unnecessary complexity while enhancing usability. To make modeling frameworks more accessible for new users and students, *jjodel* eliminates the need for costly installation and maintenance (“zero-setup”). *jjodel* is reflective, meaning it can reason about its structure and behavior, mitigating the problems due to the generation, compilation, and deployment pattern that typically characterizes model-driven platforms. Thus, it is flexible and customizable and adapts to changing needs. Model editors are living components that naturally co-evolve to any change to the metamodel and related syntactical viewpoints (multiple concrete syntaxes). Cloud-based *jjodel* allows collaborative modeling, metamodeling, and syntax definition. Finally, *jjodel* allows layout-based syntax definition notations, which are needed in many engineering domains, such as railways.

Index Terms—MDE, model editor, collaborative modeling

I. INTRODUCTION

Model-Driven Engineering [1] (MDE) is an engineering methodology founded on modeling frameworks that have been shown to be successful and scalable. Because such platforms, e.g., EMF¹, are complex and consist of highly dependant components, they exposed over the years considerable technical debt [2], [3]. Moreover, they are based on technological stacks considered state-of-the-art decades ago but may not be the ultimate choice today. All this can result in accidental or unintended complexity [4], making such systems less approachable for new users and students, with reduced usability hindering conceptual tidiness and intuition. In other words, we need the same leap forward that has led us from 4GLs and case tools to nowadays low-code development platforms.

The authors of this paper have long taught MDE in academic courses. EMF and MPS require skills and expertise beyond metamodeling and model transformations, making their adoption difficult. As a result, we started the *jjodel* project

to facilitate the adoption of model-driven platforms promoted in various contexts, including academic courses.

jjodel [5] is an Ecore compatible cloud-based, reflective modeling platform that tries to minimize unnecessary complexity and improves usability. *jjodel* does not require expensive installation or maintenance, making modeling frameworks more approachable for new users and trainees. *jjodel* is reflective and can reason about its structure and behavior, reducing costly operations like generation, compilation, and deployment of artifacts. Therefore, it can smoothly adapt to changing requirements and offer users a more flexible and customized experience. Thus, model editors become living components that naturally evolve with metamodel and syntactical viewpoint changes. *jjodel*'s cloud infrastructure supports collaborative modeling. Thus, modelers can collaboratively define metamodels, models, and syntactical viewpoints. Specifically, *jjodel* allows layout- and geometry-based notations with projectional editors, making it more expressive than topological notations like diagrams. Railway engineering uses these notations. *jjodel* implementation is available online at <https://github.com/MDEGroup/jjodel>.

The paper is structured as follows. The next section discusses the requirements a modeling platform should meet. In Sect. III, an overview of *jjodel* is given. Finally, in Sect. IV, a brief discussion and conclusions are drawn.

II. REQUIREMENTS

This section discusses and motivates requirements for a modern model-driven platform. Part of them comes from the observation that cloud-based infrastructures and advanced web application front-end frameworks can improve several quality factors related to the user experience. Others are based on the authors' expertise in applying model-driven techniques.

A. Req #1: Minimizing accidental complexity

A fundamental principle in software engineering is to minimize the amount of accidental complexity introduced into engineering solutions due to misalignments between a problem

¹<https://projects.eclipse.org/projects/modeling.emf.emf>

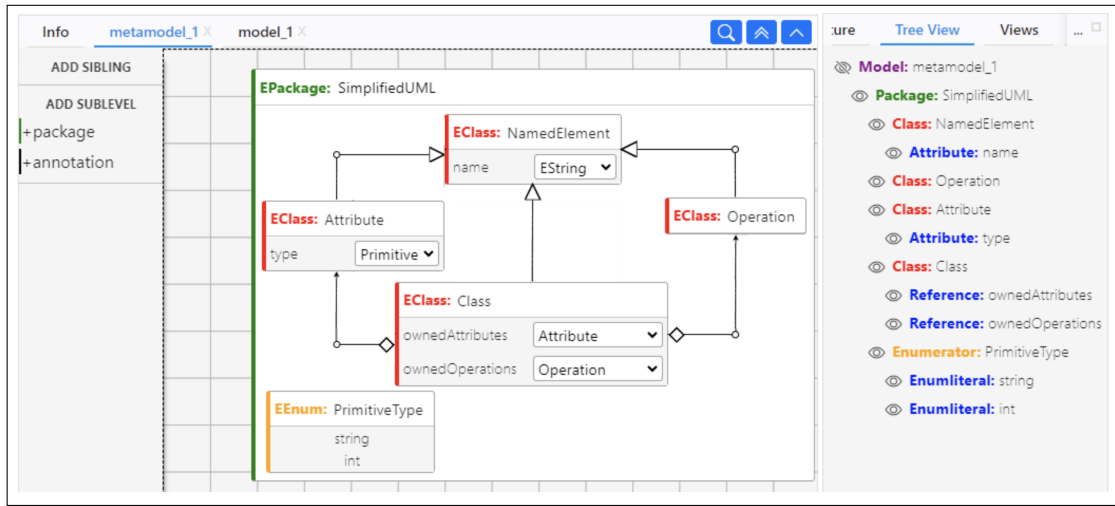


Fig. 1. Metamodeling with default syntax

and the technology used to represent the problem [4]. According to a recent analysis [6], several factors can mitigate the accidental complexity in modeling, including:

- designing modeling tools with advanced and intuitive user interfaces, clear documentation, and helpful error messages can significantly reduce the cognitive load on users, especially if newcomers; by making it easier to understand and manipulate the models, users can concentrate on their specific domain expertise rather than grappling with the intricacies of the modeling framework;
- cloud-based infrastructure to ease installation and upgrading of versioned components (“zero-setup”), artifact sharing, and application deployment and maintenance;
- machine learning techniques to provide modelers with automatic assistance tools in (meta)modeling activities, such as a recommendation system.

Thus, pursuing such factors should be considered highly beneficial when designing a new modern modeling platform.

B. Req #2: Unobtrusive adaptation to changing requirement/language evolution

One of the most daunting challenges in Model-Driven Engineering is keeping the collection of artifacts and the metamodels they are based on consistent while such metamodels undergo modifications. As extensively discussed in the current corpus of literature (see e.g. [7]), similarly to software, metamodels are subject to the same evolutionary pressure. However, changing a metamodel comes at the price of invalidating existing models, the concrete syntax (and the related editors), and code generators. Then the question is, how do we make functionality for adaptation unobtrusive for modelers so that restoring the modeling ecosystem consistency is sustainable, scalable, and with little or no cognitive impedance? Flexibility and responsiveness should be key for faster iteration and adaptation, releasing the human-in-the-loop from generative stages and improving the overall efficiency and agility of the modeling process.

C. Req #3: Enhanced Syntax Specification

Current meta-editors, such as GMF, Sirius, and Eugenia², support the generation of topological notations and editors, i.e., a type of modeling tool that focuses on the topology of a system instead of its geometry [5]. Atom³ [8] and Gentleman [9] have been proposed as a light-weight web-based projectional editor generator. The user defines the relationships between components in these editors rather than specifying their exact locations and shapes. Thus, the model semantics is entirely agnostic of the model layout. However, providing support to layout can bridge the gap between model editors and technical editors. It is necessary in many engineering domains as it conveys important information about the structure and organization of the model, e.g., railway.

Another relevant aspect (which can be realized on existing platforms) is modeling notations with multiple syntactical viewpoints³. Like this, purpose-specific representations and abstractions can align with the specific needs of different stakeholders, such as developers, domain experts, or business analysts, by improving collaboration and ensuring that the model conveys meaningful and relevant information to the stakeholders.

Finally, the usability of the syntax definition notation (and its support) is of crucial relevance. By providing intuitive syntax rules, the modeling language becomes more accessible and easier to learn, use, and maintain.

D. Req #4: Collaborative modeling

Collaborative modeling enables multiple individuals to work simultaneously on the same modeling artifact in a distributed environment. It entails employing approaches, strategies, and tools wherein diverse stakeholders collectively oversee, cooperate, and possess different aspects of the system. Access-control and coordination policies, such as locking mechanisms,

²<https://eclipse.dev/epsilon/doc/eugenia/>

³The terms *syntactical viewpoint* and *concrete syntax* are intended as synonyms. However, the former denotes a more purpose- or stakeholder-specific nature.

model versioning systems, conflict management, and support for presence awareness, allow participants to notice who else is actively working on which specific fragment of the model to mitigate the risk of conflicts and divergence.

E. Req #5: Continuous and automated co-evolution

The adaptation aspects described in Sect. II-B should prevent model editors from being regenerated whenever the associated metamodel undergoes modifications. When a metamodel is modified, the model editor should be manually or semi-automatically adapted (e.g., see [10]) accordingly, along with the model instances. Therefore, changes made in the abstract syntax should accurately reflect in the model editor, and any model instance being edited when metamodel changes occur should be automatically migrated (as long as the changes are manageable, i.e., breaking resolvable [7]). Like this, metamodel development can become an iterative process in which metamodels and models can be developed interleaved to reflect domain needs and consistently test the metamodel expressiveness. This requirement, if fulfilled, facilitates practical domain development through a continuous and iterative metamodel development, validation, maintenance, and continuous alignment between the elicited concepts and their formalization.

III. IMPLEMENTING *jjodel*

jjodel realizes the modeling environment and its user interfaces utilizing cutting-edge technologies. *jjodel* greatly benefits from the smooth integration and synergy between Typescript⁴, React⁵, and JSX⁶.

The reflective architecture requirement (see Sect. II-B) has been satisfied with the support of Redux⁷, a predictable state container for JavaScript applications, commonly used within React for building user interfaces. Redux stores and manages the complete *jjodel* object model (JJOM), including the object representation of metamodels, models, syntactical viewpoints. Because Redux maintains and updates state in response to user actions, developers can characterize UI as a function of application state. The combination of OCL.js⁸ and JSX permits modelers to define the specific syntax of a modeling notation (see Sect. II-C). Specifically, an OCL expression selects model elements to be rendered by the associated JSX template. Navigational expressions permit to traverse and query the modeling artifacts and give them the desired syntactical representation. Conditional formatting can be realized differently, e.g., using JSX conditionals or defining different views with appropriate OCL expressions. It is worth noting that the JSX navigational expressions have access to all the information of a modeling artifact, including layout attributes, such as position, size, etc.

jjodel uses a client-server reactive and real-time architecture for collaborative modeling. A central server as the authoritative

source of truth for the shared model enables real-time collaboration between multiple clients. This architecture allows users to simultaneously change the model and receive real-time updates from other participants (see Sect. II-D). Firebase⁹, a mobile and web application development platform, implements the architecture with high-quality services like cloud messaging, real-time database, authentication, hosting, and more. Any collection or document data changes are immediately detected and sent to the client. The listener receives updates as a snapshot object. The following subsections describe how each requirement defined in Sect. II has been implemented.

*A. How *jjodel* minimizes the accidental complexity*

Accidental complexity is system complexity caused by design and implementation decisions and not needed to solve the challenge. Bad interfaces, redundant functionality, and complex code are examples. Reducing modeling tool complexity is a multifaceted problem involving technologies and socio-technical issues. Practical and usable solutions require user-centered design. Addressing such a complex issue demands extensively rethinking the tools' core components, while overlooking such difficulties may result in tools that are inadequate as demonstrators and impair cognitive integration into academic courses.

Figure 1 shows *jjodel* in action on a typical metamodeling scenario. *jjodel* tries to respond to such a challenge by relying on cloud-based infrastructure. The main objective of such an architectural design is to reduce the accidental complexity of the modeling tool by providing the platform with zero setup possibilities and no maintenance and simplifying the overall workflow by removing those platform-related steps like the regeneration of the editors. Moreover, it facilitates communication and artifact sharing by means of public links. Dynamic content, user customizations, and persistence, and internal storage is supported by Redux and the Typescript assures the proper gluing among the components. User assistance is provided by MEMOREC [11], a recently developed recommender system that performs information filtering to predict the preferences with related ratings that a user would give to a particular model element, be a package, class, or structural feature while formalizing a metamodel. Figure 3 shows how MEMOREC has been integrated in *jjodel*; in this case, the recommender system suggests a list of possible structural features with associated ranking that the user may want to consider based on a public dataset of metamodels. Finally, *jjodel* incorporates XMI export/import functionalities to ensure interoperability with established EMF modeling environments.

*B. How *jjodel* implements reflective architecture*

Modeling environments are typically based on metamodels. Whenever a metamodel is modified, related artifacts might no longer be valid, giving rise to the so-called co-evolution problem. Reflective systems have the capability to self-adapt to a changing environment. Therefore, a reflective modeling platform should be able to adapt to metamodel changes without

⁴<https://www.typescriptlang.org/>

⁵<https://react.dev/>

⁶https://www.w3schools.com/react/react_jsx.asp

⁷<https://redux.js.org/>

⁸<https://ocl.stekoe.de/>

⁹<https://firebase.google.com/>

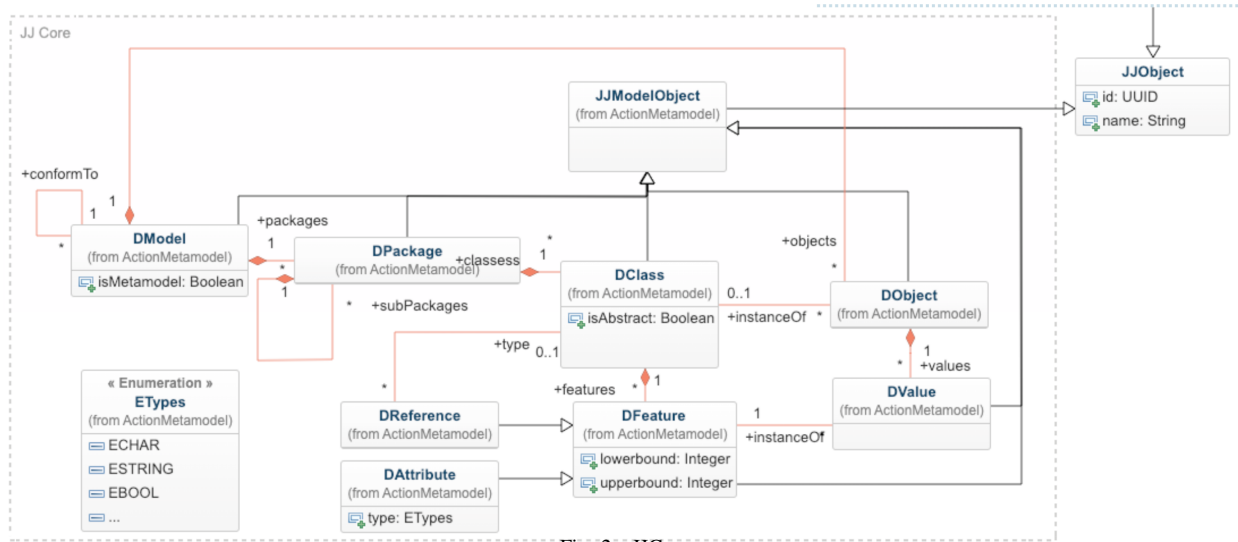


Fig. 2. JJCore

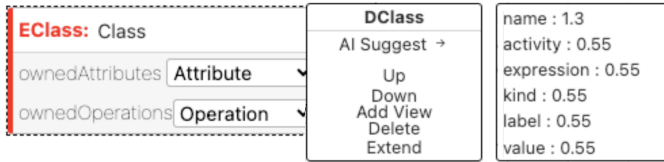


Fig. 3. *jjodel* model assistant

requiring the designer to manually adjust dependent artifacts or perform costly generation and compilation stages. For example, if a designer changes a metamodel while editing a conforming model, the model editor should consistently reflect such changes without discarding the model being edited that is also adapted, as long as the changes are manageable (non-breaking or breaking resolvable [7]). Such a characteristic satisfies the continuous co-evolution requirement in Sect. II-E.

jjodel realizes such behavior thanks to the JJCore object model, a simplification of which is illustrated in Fig. 2. The object model keeps track of all dependencies and contains a representation of the EMF/Ecore meta-metamodel. More in detail, the DModel captures a metamodel or model (in this case, the reference conformTo refers to the reference metamodel); whereas DPackage, DClass, DFeature, DReference, and DAttribute are the EMF concepts. Models are expressed by a set of model element DObject instances of DClass. A DObject contains DValues that are instances of DFeatures, i.e., DReferences and DAttributes. *jjodel* store models and metamodels relying on the abovementioned concepts with the support of Redux. It has been used to maintain a central state, enabling more efficient and predictable data flow control within the system. The *jjodel* web environment allows modelers to define metamodels (see Fig. 1), models (see Fig. 6), and concrete syntaxes (see Fig. 5).

C. How *jjodel* enhanced model syntax specification

A *jjodel* project consists of a metamodel, a set of models, and a specification of the concrete syntax, called syntactic

cal viewpoint. The latter is administered by DViewPoints and consists of multiple layered DViews. Each DViews determines its visual representation with a JSX template, layout constraints, and an OCL Expression that constraints its execution. A DViews may represent the model in its entirety or in part and connects a modeling element to a maximum of one DViewPoint. A DViews is a syntactic rule that consists of an OCL predicate and JSX template. Each of such rules specifies the concrete syntax of one or more metaclasses satisfying the OCL predicate by supplying the JSX template utilized by the renderer graph. Figure 5 shows an exemplification of using an OCL simple guard together with JSX template to define a custom concrete syntax of a DClass instance. Figure 6 shows two different views of a Point instance of Class metaclass.

D. How *jjodel* supports collaborative modeling

The Client-Server Reactive Real-Time Architecture (CSRRTA) [12] is a popular approach in collaborative modeling environments, allowing multiple users to work together on a shared model simultaneously. CSRRTA leverages a central server as the authoritative source of truth for the model, enabling real-time collaboration and immediate updates on concurrent changes made by participants. At the core of this architecture is the server, responsible for storing the model data, handling concurrent modifications, and broadcasting updates to all connected clients. This ensures a consistent and up-to-date view of the model for all users. Firebase SDK implements this idea. *jjodel* clients register listeners on Firestore data to update when changes occur. Firebase instantly detects and sends database changes to clients. The listener updates the application's user interface with a snapshot object.

The server lets clients modify the shared model, retrieve its state, or subscribe to real-time updates. Clients send modification requests to the server, which applies the change and broadcasts it to all connected clients, ensuring everyone sees the latest model. This architecture does well in situations

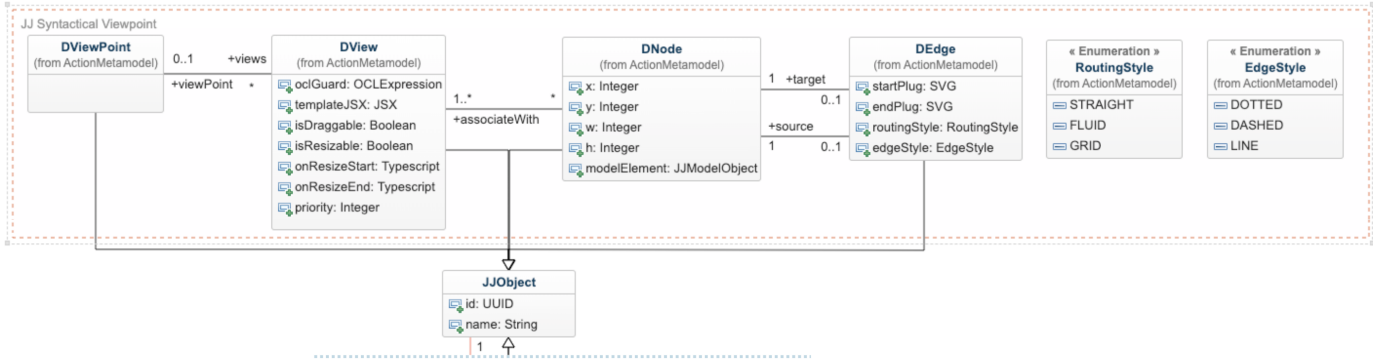


Fig. 4. Syntax notation

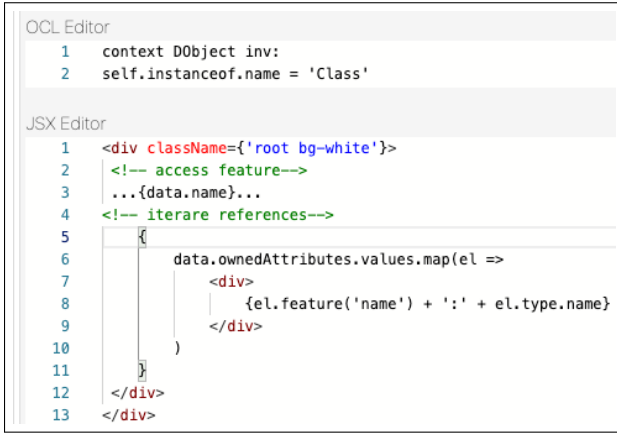


Fig. 5. A view definition

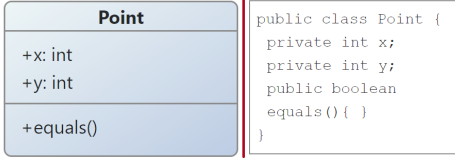


Fig. 6. Two different views of a Class instance

that require immediate feedback and response. *jjodel* uses *rooms* and *room tokens* for shared modeling. Each room in our Firestore database stores and synchronizes collaborative model actions. The room receives local user actions. Clients in the same room will receive and execute the action in sync. *jjodel* also promotes presence. To reduce conflicts and divergence, *jjodel* lets participants see who is working on which model fragment. A short video is available at: <https://youtu.be/VJzoO4sUOEw>.

IV. CONCLUSIONS

Existing model-driven platforms are complicated, decades-long systems. EMF has one of the richest software ecosystems due to generations of modelers in education, research, and industry. Low-code success shows that model-driven tool makers might learn from its potential to pamper users. Due to factors such as increased tool utilization in academic courses, we believe *jjodel* can address some of the concerns in this paper. In addition, geometric syntax can bridge the gap

between model editors and technical editors, broadening the applicability scope of model-driven techniques. Cloud-based infrastructures can also decrease unintentional complexity, and reflective techniques can aid in change management. Better edge management, scalability evaluation, and comparison with Atom3 and Gentlemen are planned soon.

ACKNOWLEDGMENT

We are grateful to Bran Selic for his insightful comments on a draught of this paper.

REFERENCES

- [1] D. C. Schmidt, "Model-driven engineering," *Computer-IEEE Computer Society*, vol. 39, no. 2, p. 25, 2006.
- [2] X. He, P. Avgeriou, P. Liang, and Z. Li, "Technical debt in MDE: a case study on GMF/EMF-based projects," in *Procs. MODELS*. ACM, 2016, pp. 162–172.
- [3] D. D. Ruscio, A. D. Salle, L. Iovino, and A. Pierantonio, "A modeling assistant to manage technical debt in coupled evolution," *Inf. Softw. Technol.*, vol. 156, p. 107146, 2023.
- [4] C. Atkinson and T. Kühne, "Reducing accidental complexity in domain models," *Software & Systems Modeling*, vol. 7, pp. 345–359, 2008.
- [5] D. Di Vincenzo, J. Di Rocco, D. Di Ruscio, and A. Pierantonio, "Enhancing syntax expressiveness in domain-specific modelling," in *Procs. MLE Workshop at MODELS*, 2021, pp. 586–594.
- [6] A. Bucchiarone, F. Ciccozzi, L. Lambers, A. Pierantonio, M. Tichy, M. Tisi, A. Wortmann, and V. Zaytsev, "What is the future of modeling?" *IEEE software*, vol. 38, no. 2, pp. 119–127, 2021.
- [7] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio, "Automating co-evolution in model-driven engineering," in *Procs. EDOC 2008*, 2008, pp. 222–231.
- [8] J. d. Lara and H. Vangheluwe, "Atom 3: A tool for multi-formalism and meta-modelling," in *Fundamental Approaches to Software Engineering: 5th International Conference, FASE 2002 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8–12, 2002 Proceedings 5*. Springer, 2002, pp. 174–188.
- [9] A. Ducoin and E. Syriani, "Graphical projectional editing in gentleman," in *Procs. Tool Demonstrations at MODELS*, 2022, pp. 46–50.
- [10] D. Di Ruscio, R. Lämmel, and A. Pierantonio, "Automated co-evolution of GMF editor models," in *Procs. SLE 2010*. Springer, pp. 143–162.
- [11] J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. T. Nguyen, and A. Pierantonio, "MemoRec: a recommender system for assisting modelers in specifying metamodels," *Software and Systems Modeling*, vol. 22, no. 1, pp. 203–223, 2023.
- [12] A. Debski, B. Szczepanik, M. Malawski, S. Spahr, and D. Muthig, "A scalable, reactive architecture for cloud applications," *IEEE Software*, vol. 35, no. 2, pp. 62–71, 2018.