

Engineering Low-code Modelling Environments with Dandelion

Francisco Martínez-Lasaca 
UGROUND
Madrid, Spain
francisco.martinezl01@
estudiante.uam.es

Pablo Díez 
UGROUND
Madrid, Spain
pdiez@uground.com

Esther Guerra 
Universidad Autónoma de Madrid
Madrid, Spain
esther.guerra@uam.es

Juan de Lara 
Universidad Autónoma de Madrid
Madrid, Spain
juan.delara@uam.es

Abstract—Low-code development platforms (LCDPs) are gaining momentum, since they enable less technical profiles (*citizen developers*) to participate in software development. LCDPs typically run on the cloud with zero-cost installation and often use graphical languages to describe the target applications. While building LCDPs manually is costly, the process can benefit from the automation that model-driven technologies offer.

To this end, we propose Dandelion, a cloud-based graphical-language workbench specifically designed to create LCDPs. Dandelion supports the definition of the abstract and concrete syntax of graphical languages, resulting in tailored cloud-based modelling environments. To cater for industrial needs, the tool can be connected with heterogeneous modelling technologies – including proprietary formats or standards like EMF – and can handle large models via persistence in Elasticsearch and a model pagination mechanism. To facilitate modelling by citizen developers, it features sensemaking strategies to ease understanding (meta-)models via interactive graphics and diagrams; and flexible modelling support to relax the meta-model/model conformance relation, thus tolerating inconsistencies. A video of the tool is available at <https://youtu.be/kQjdOgopGvI>.

Index Terms—low-code platforms, model-driven engineering, graphical languages, flexible modelling

I. INTRODUCTION

Low-code development platforms (LCDPs) provide mechanisms for automated software construction within specific domains. They are typically cloud-based environments, enabling the description of applications via forms and graphical languages. Being cloud-based, LCDPs require zero-installation cost, and they may also host the generated application, providing run-time management facilities. This ease-of-use, the target to specific domains, and the reliance on graphical languages open their use to so-called *citizen developers*, i.e., users with non-technical backgrounds [1].

Many companies offer LCDPs targeting business applications, including the main cloud vendors – like Google,¹ Microsoft,² and Amazon³ – and specialised companies, including Appian,⁴ Mendix,⁵ or OutSystems.⁶ Many domain-specific LCDPs have emerged for a plethora of domains, like chatbots (e.g., Dialogflow, Amazon Lex [2]), IoT (e.g., Node-RED⁷), or machine learning (e.g., Google’s AutoML⁸ or

RapidMiner⁹). Additionally, many companies use custom low-code platforms for their internal development processes, e.g., UGROUND¹⁰ [3]. However, building LCDPs is challenging because it requires creating dedicated cloud-based graphical modelling environments, integrating with diverse modelling technologies, and handling large-scale industrial models.

To tackle these challenges, we propose Dandelion, a cloud-based, graphical language workbench [4]. The tool uses Model-driven Engineering to automate some steps of the creation of industrial LCDPs. In particular, it supports the definition of cloud-based environments for graphical modelling languages, facilitates the integration of heterogeneous modelling technology, and eases understanding tasks via model sensemaking strategies (SMSs) [5]. Dandelion also supports modelling in the large thanks to a model pagination mechanism and persistence atop a cloud-native database. Finally, to facilitate citizen developers modelling and support diverse modelling scenarios, Dandelion allows adjusting the degree of inconsistency tolerance via flexible modelling techniques [6].

The scalability support of Dandelion and SMSs were previously presented at [4], [5]. Here we focus on the tooling aspect, introducing novel support for flexible modelling, including quick fixes and the definition of modelling phases.

Paper organisation. Section II examines related works, positioning Dandelion w.r.t. them. Section III outlines the tool’s architecture, describes its most salient features, and reports on performed evaluations. Finally, Section IV ends with the conclusions and prospects for future work.

II. RELATED WORK

Next, we revise and position Dandelion w.r.t. works on cloud-based graphical editors, heterogeneity support, scalability, flexibility, and visualisation techniques in modelling.

Cloud-based editors for graphical DSLs. Graphical modelling editors trace back to the 90s, motivated by the raise of graphical modelling notations (e.g., MetaEdit [7] or AToM³ [8]). Later, in the 2000s and 2010s, Eclipse-based desktop tools proliferated, including GMF [9], Eugenia [10], or Sirius [11]. Finally, advances in cloud computing in the late 2010s promoted the development of workbenches to

¹ <https://cloud.google.com/appsheets>

² <https://powerapps.microsoft.com>

³ <https://www.honeycode.aws>

⁴ <https://appian.com>

⁵ <https://www.mendix.com>

⁶ <https://www.outsystems.com>

⁷ <https://nodered.org>

⁸ <https://cloud.google.com/automl>

⁹ <https://rapidminer.com>

¹⁰ <https://www.uground.com>

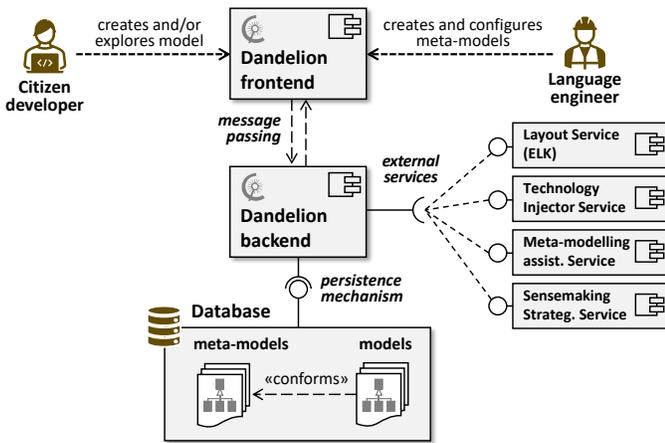


Fig. 1: Dandelion's working schema.

develop cloud-based editors, e.g., WebGME [12] or Sirius Web [13]. Language protocols for textual languages (LSP¹¹) also popularised, but their graphical counterparts (GLSP¹²) are still in evolution. The need to create custom cloud-based editors is especially relevant nowadays given the ubiquity of LCDPs. However, current workbenches for cloud editors [12], [13] are typically tied to a specific modelling technology, and Dandelion tries to fill that gap.

Modelling heterogeneity. Models can be described using a plethora of formats, hampering their interoperability. Some tools like EXTREMO [14] propose a common data model to represent models coming from any format. Other frameworks, like Epsilon, propose a model access and modification interface that can be implemented for any model format or source, e.g., EMF, Simulink, CSV files, or databases [15]. Dandelion employs a level-agnostic meta-model to harmonise different formats, and provides technology injector services to accommodate different technical spaces.

Scalability in modelling. Models have been predominantly persisted in files, as supported by the EMF ecosystem [16]. However, this approach has proven insufficient for large artefacts. Some file-based solutions include file fragmentation, partial loading, model decomposition, indexers, and caching, e.g., [17], [18]. Other alternatives propose database persistence (e.g., CDO [19], Teneo [20]) and dedicated model repositories, e.g., MDEForge [21]. Dandelion uses Elasticsearch,¹³ a cloud-native database, to persist models and employs a pagination mechanism to display them on demand.

Flexible modelling. Relaxing the level of consistency of models can ease different modelling scenarios such as model prototyping, evolution, or technology migration [6]. This extra flexibility is especially relevant for LCDPs, given their wide variety of target users. Some typically supported features include deferring or disabling typing, cardinality, or domain-specific checks. Moreover, checks can be combined in phases that can be promoted manually or automatically, and arranged

¹¹ <https://microsoft.github.io/language-server-protocol/>

¹² <https://www.eclipse.org/glsp/> ¹³ <https://www.elastic.co/elasticsearch>

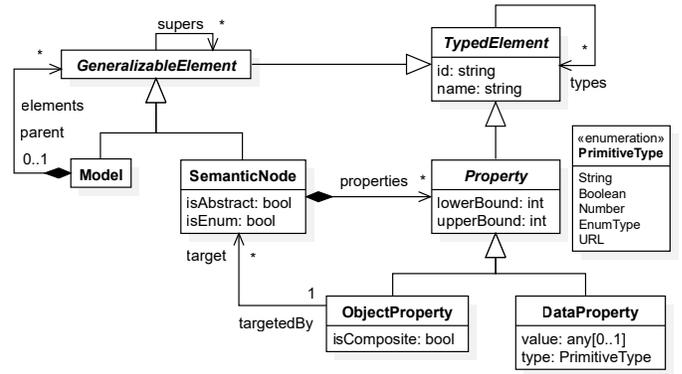


Fig. 2: Dandelion's harmonising meta-model.

to support different modelling styles, e.g., bottom-up or top-down. To our knowledge, Dandelion is the first cloud-based platform for graphical languages that supports configurable modelling phases.

Visualising and comprehending large models. The field of Visual Analytics has extensive literature on large graph visualisation. In contrast, visualisation techniques catered to models are scarce. Dandelion relies on the notion of *graph sensemaking* (the iterative process of understanding graph-formatted data to achieve a concrete goal) [22] to devise visualisation tailored for (meta-)models. To enable reusability, Dandelion's modelling sensemaking strategies are exposed as generic components that are instantiated via a binding mechanism [5].

III. DANDELION

Dandelion is a cloud-based workbench for graphical languages. Its architecture comprises a frontend and a backend components, as illustrated in Fig. 1. The frontend serves the tool's interface, where citizen developers and language engineers create and/or explore models, and create and configure meta-models (i.e., language definitions), respectively. The backend features a persistence mechanism responsible for managing (meta-)models, which are persisted in a database. Additionally, the backend integrates with multiple internal and external modelling services. The tool is available online.¹⁴

Next, we delve into the salient features of the tool.

A. Harmonising meta-model for heterogeneity support

Dandelion relies on models to describe application domains. However, a plethora of formats can be used to represent them, including EMF [16], RDF,¹⁵ or ad hoc industrial formats like UGROUND's ROSE [23]. In order to bridge heterogeneous data formats, Dandelion represents every (meta-)model using a *harmonising meta-model* (cf. Fig. 2).

This meta-model strikes a balance between simplicity and expressiveness to encompass a large number of modelling styles while achieving level-agnosticism. That is, models and meta-models are represented uniformly. Its building pieces

¹⁴ <https://miso.es/tools/Dandelion.html> ¹⁵ <https://www.w3.org/RDF/>

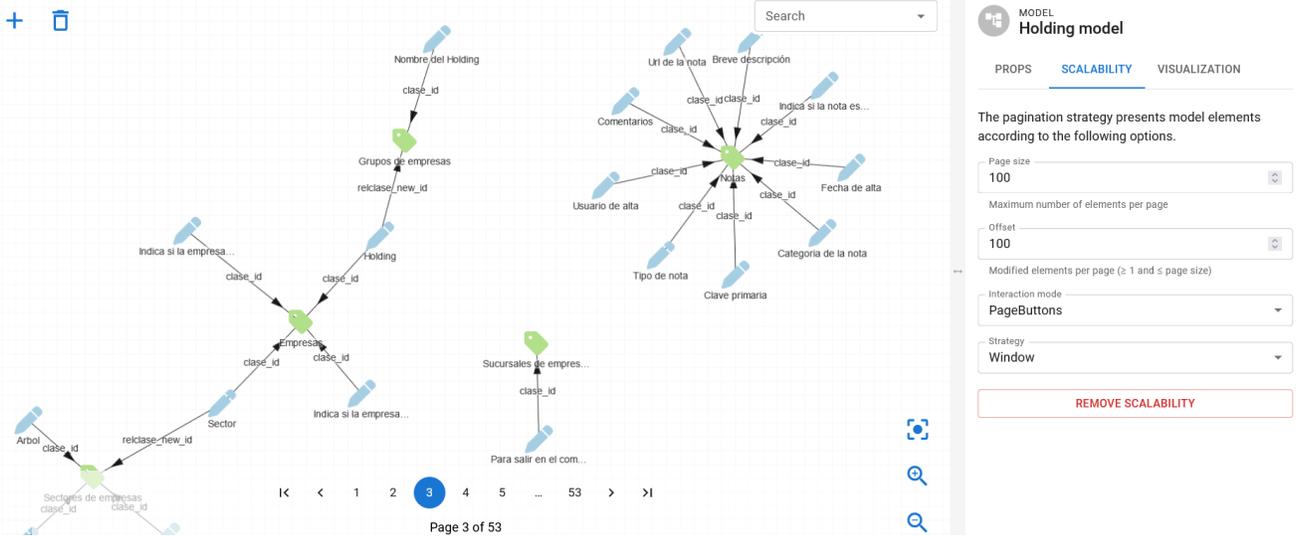


Fig. 3: A model loaded with pagination. The scalability tab allows configuring how to split the model into pages.

are Model and SemanticNode: Models encode both models and meta-models, while SemanticNodes represent meta-classes/objects and can define properties. In particular, a Property is either an ObjectProperty, to represent associations/links between meta-classes/objects, or a DataProperty for primitive attributes.

(Meta-)models conformant to this linguistic meta-model are persisted in Elasticsearch, a highly scalable database, enabling fast model retrieval. The specifics on the persistence mechanism are detailed in [4].

B. Scalability mechanism: pagination for large models

Scalability mechanisms help users navigate large models without overwhelming them with too much information. Dandelion supports pagination to split large models into pages that can be loaded on demand. The mechanism permits configuring page capacities (i.e., the maximum number of elements per page) and the loading strategy, among other parameters. Moreover, referenced elements that do not fit the current page are displayed as proxy nodes, whose location can be navigated to upon clicking. Fig. 3 shows a UGROUND model on holdings management loaded with pages of size 100.

C. Understanding large models with sensemaking strategies

Exploring artefacts (e.g., models) is essential in low-code platforms, as well as in common modelling tools. However, exploration techniques typically rely on graph-based visual metaphors, which do not scale well with large model sizes. For this reason, Dandelion introduces *model sensemaking strategies* (SMSs): purposeful visualisations that exploit alternative visual metaphors to distil insights from large models [5]. These visualisations can be instantiated by binding their “context” meta-model to the desired target (meta-)model with a structure-preserving mapping.

Fig. 4 illustrates the exploration of an industrial meta-model guided by two sensemaking strategies aimed at understand-

TABLE I: Checks for flexible modelling.

#	Check name	What it detects
C1	Cardinality of attributes	Attributes not respecting their multiplicity.
C2	Cardinality of references	References not respecting their multiplicity.
C3	Type of attribute values	Attr. values that do not respect the attr.’s type.
C4	Type of reference values	References pointing to objs. of the wrong type.
C5	Missing property	Objs. missing a prop. defined in the meta-model.
C6	Superfluous property	Objs. defining a prop. absent in the meta-model.
C7	Duplicate property	Objs. defining a prop. with the same name twice.
C8	Non-existent target	References pointing to non-existent objs.
C9	Instanced abstract class	Objs. that are instances of abstract classes.
C10	Untyped objects	Objs. that do not conform to a meta-class.
C11	Untyped properties	Props. that do not conform to a property.

ing the complexity of the meta-model. SMS 1a displays a metric: the degree of coupling between meta-classes, i.e., the proportion of edges connecting meta-classes if compared to a complete graph. On the other hand, SMS 1b summarises the complexity of meta-classes attributes via a histogram of the number of attributes per meta-class. The dashboard (1) accommodates multiple SMSs and integrates in the editor to enable exploring (meta-)models in multiple ways simultaneously.

D. Flexible modelling

In order to tolerate inconsistencies between models and their meta-models, flexible modelling allows relaxing the degree of conformance in a controlled manner. As a contribution of this paper, we have improved flexible modelling support twofold.

On the one hand, we have extended the catalogue of constraint checks (cf. Table I) and introduced quick fixes. Checks are evaluated on models and reported in the editor. Some checks are equipped with quick fixes, which can amend models and/or their meta-models to fix a check’s violation. In total, Dandelion supports 11 checks and 17 quick fixes.

On the other hand, we have introduced modelling phases to bundle checks with different severities: ignored, warning, or error. If marked as errors, the tool also impedes checks’

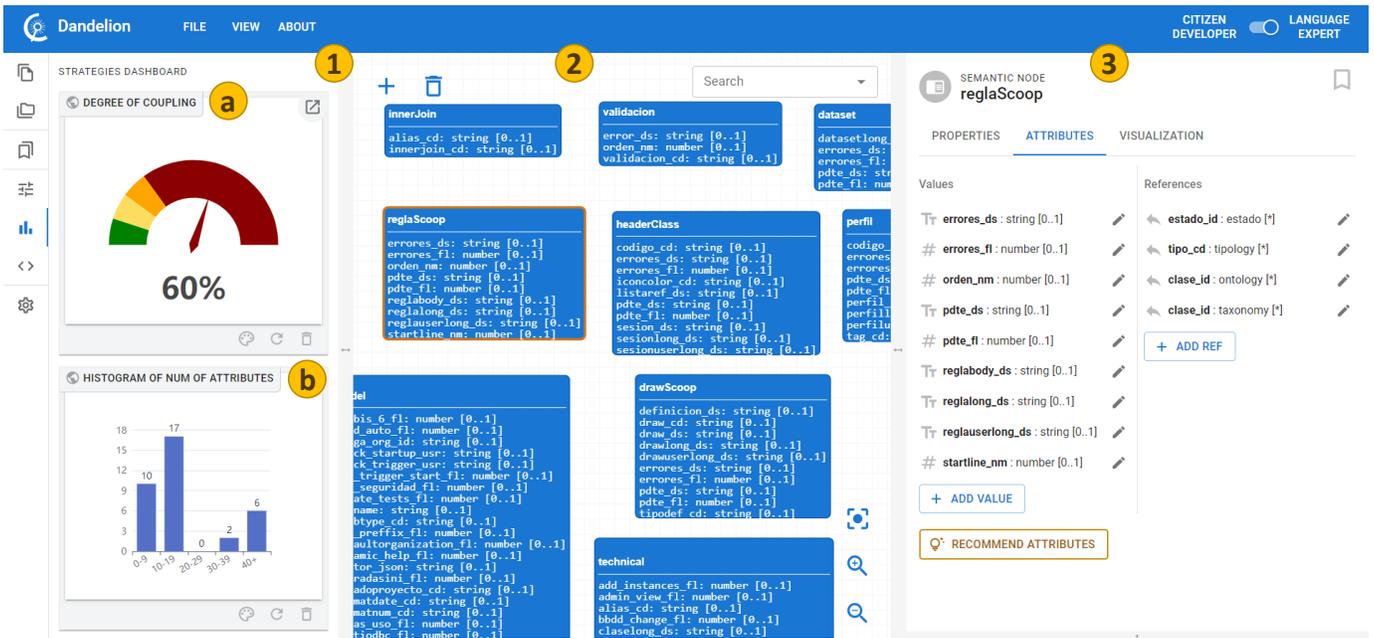


Fig. 4: Visualising an industrial meta-model in Dandelion. (1) Sensemaking strategies to understand the meta-model; (2) modelling canvas, where edges have been hidden for clarity; (3) selected meta-class editor.

violation in the editor. Furthermore, phases support three promotion strategies: free (can promote with errors or warnings), no errors (only warnings are tolerated), or no errors nor warnings (the most restrictive). In Dandelion, phases are first defined by the language engineer and then employed by citizen developers. Early phases in a modelling process usually offer more flexibility, tolerating inconsistencies. Then, subsequent phases are more strict (more checks are enabled), and the user can progress to further phases, obtaining a more precise model, via quick fixes.

Fig. 5 illustrates the citizen developer’s perspective on a network diagram model. Active phases determine which checks are active and their severity, and can be promoted or retraced according to the promotion strategy. Violations are reported in the Problems panel, and amendable ones suggest quick fixes. For example, in the figure, the Computer4 object is untyped (i.e., someone forgot to link it to the Computer meta-class), and the current phase reports ‘C10. Untyped objects’ violations as errors. The resulting error is reported, and the application of the three quick fixes available for this check is offered. Specifically, (i) creating a dedicated meta-class shaped after the object and retyping the object; (ii) retyping the object to an existing meta-class selected by the user; and (iii) relaxing the check’s severity. Please note that quick fixes can relax check’s severities and alter models, meta-models, or both.

E. Evaluating Dandelion

In this section, we report on evaluations performed on Dandelion for its different contributions.

a) *Handling very large models:* Handling scalability effectively is an open challenge in model-driven and low-code applications [24]. In [4], we evaluated Dandelion’s persistence

mechanism by exploring huge models (i.e., with 1,000,000+ entities) with pagination. In particular, the benchmarks show that large models can be loaded in pages containing up to 500 objects and 1000 edges. Pages are retrieved, layouted, and rendered in less than 3 seconds, thus providing a reactive user experience.

b) *Supporting new technologies:* Integrability is crucial in low-code industrial settings, as data is typically drawn from multiple heterogeneous sources. In [4], we proposed a process to integrate new technologies into Dandelion and applied it to support artefacts from UGROUND’s ROSE [23] modelling platform. The process relies on a schema and data injectors, loading meta-models and models, respectively. To ensure a sound integration, a conformance analysis step making use of flexible modelling is undergone in every iteration of the process. In the future, we intend to evaluate the impact of phases and fine-grained checks on this step with a user study.

c) *Comprehending models with sensemaking strategies:* Complex (meta-)models may become unwieldy and difficult to understand. In [5], we evaluated the feasibility of model sensemaking strategies to understand models, language usages, and mega-models in an industrial case study on UGROUND’s ecosystem. The results show that SMSs are effective in addressing complex sensemaking tasks to understand large models, and that they are highly reusable, especially in multi-level modelling scenarios.

IV. CONCLUSIONS AND FUTURE WORK

This paper has presented Dandelion, a cloud-based workbench to define editors for graphical languages that can be integrated within low-code development platforms. Dandelion

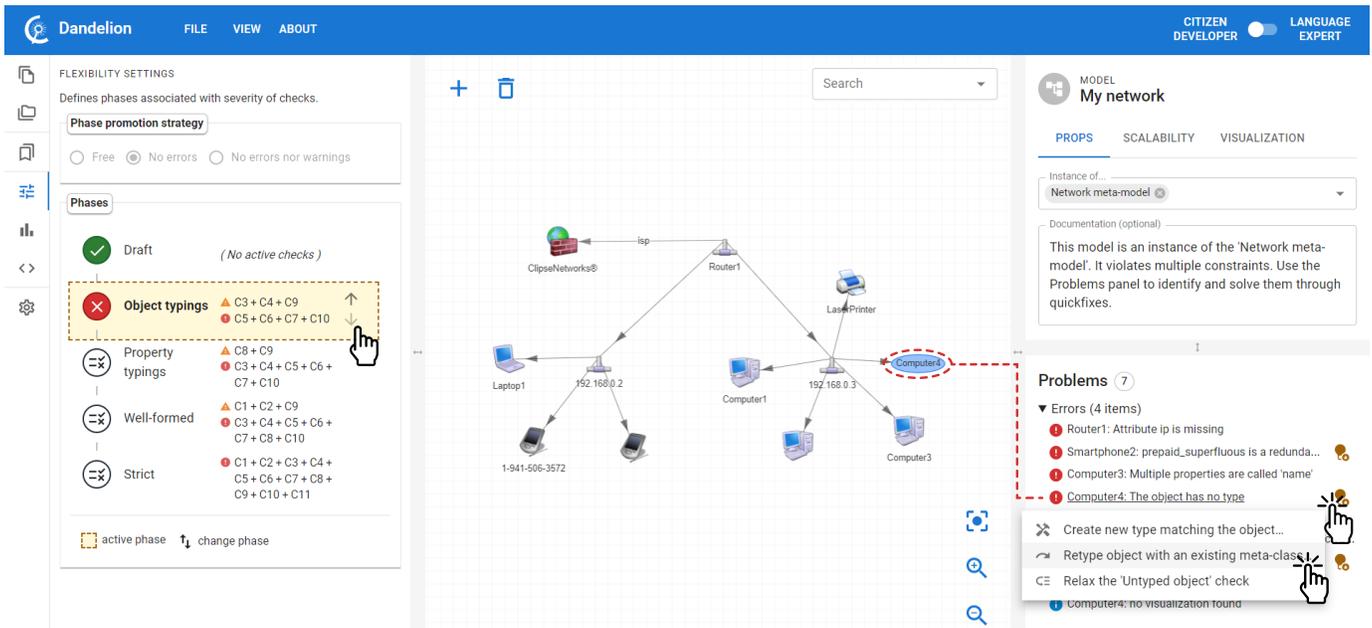


Fig. 5: Flexible modelling in Dandelion. Left: settings panel. Right: reported problems and suggested quick fixes.

supports heterogeneous models through a harmonising meta-model, and provides a pagination mechanism to handle large models. The tool especially considers the usage experience of citizen developers by supporting flexible modelling – together with quick fixes – and sensemaking strategies to facilitate model exploration.

We are currently extending the tool to integrate model management languages of the Epsilon family. This will permit using the validation language to specify meta-model integrity constraints, as well as to define code generators and model manipulation operations that can then be run on the cloud.

ACKNOWLEDGEMENTS

This project is funded by the EU Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie grant agreement No 813884 and the Spanish MICINN (PID2021-122270OB-I00, TED2021-129381B-C21).

REFERENCES

- [1] D. D. Ruscio, D. S. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, and M. Wimmer, “Low-code development and model-driven engineering: Two sides of the same coin?” *Softw. Syst. Model.*, vol. 21, no. 2, pp. 437–446, 2022.
- [2] S. Pérez-Soler, S. Juárez-Puerta, E. Guerra, and J. de Lara, “Choosing a chatbot development tool,” *IEEE Softw.*, vol. 38, no. 4, pp. 94–103, 2021.
- [3] A. Díez, N. Nguyen, F. Díez, and E. Chavarriga, “MDE for enterprise application systems,” in *MODELSWARD*. SciTePress, 2013, pp. 253–256.
- [4] F. Martínez-Lasaca, P. Díez, E. Guerra, and J. de Lara, “Dandelion: a scalable, cloud-based graphical language workbench for industrial low-code development,” *Journal of Comp. Langs.*, vol. 76, p. 101217, 2023.
- [5] —, “Model sensemaking strategies: Exploiting meta-model patterns to understand large models,” in *Proc. MoDELS*, 2023, pp. 1–12.
- [6] E. Guerra and J. de Lara, “On the quest for flexible modelling,” in *MoDELS*. ACM, 2018, pp. 23–33.
- [7] S. Kelly and J. Tolvanen, *Domain-Specific Modeling - Enabling Full Code Generation*. Wiley, 2008.

- [8] J. de Lara and H. Vangheluwe, “AToM³: A tool for multi-formalism and meta-modelling,” in *FASE*, ser. LNCS, vol. 2306. Springer, 2002, pp. 174–188.
- [9] The Eclipse Foundation, “Graphical Modeling Framework (GMF),” <https://www.eclipse.org/gmf-tooling/>, 2014.
- [10] D. S. Kolovos *et al.*, “Eugenia: Towards disciplined and automated development of GMF-based graphical model editors,” *SoSyM*, vol. 16, no. 1, pp. 229–255, 2017.
- [11] V. Vijović, M. Maksimović, and B. Perišić, “Sirius: A rapid development of DSM graphical editor,” in *INES*, 2014, pp. 233–238.
- [12] M. Maróti, T. Kecskés, R. Kereskényi, B. Broll, P. Völgyesi, L. Jurácz, T. Levendovszky, and Á. Lédeczi, “Next generation (meta)modeling: Web- and cloud-based collaborative tool infrastructure,” in *MPM@MoDELS*, vol. 1237. CEUR, 2014, pp. 41–60.
- [13] The Eclipse Foundation, “Sirius Web,” <https://www.eclipse.org/sirius/sirius-web.html>, 2023.
- [14] A. Mora-Segura, J. de Lara, P. Neubauer, and M. Wimmer, “Automated modelling assistance by integrating heterogeneous information sources,” *COMLAN*, vol. 53, no. September, pp. 90–120, 2018.
- [15] D. Kolovos, R. F. Paige, and F. A. Polack, “Eclipse development tools for Epsilon,” in *Eclipse Summit Europe*, vol. 20062, 2006, p. 200.
- [16] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework, 2nd edition*. Pearson Education, 2008.
- [17] K. Jahed *et al.*, “On the benefits of file-level modularity for EMF models,” *Softw. Syst. Model.*, vol. 20, no. 1, pp. 267–286, 2021.
- [18] R. Wei, D. S. Kolovos, A. García-Domínguez, K. Barpis, and R. F. Paige, “Partial loading of XMI models,” in *MoDELS*. ACM, 2016, pp. 329–339.
- [19] The Eclipse Foundation, “CDO. The model repository,” <https://www.eclipse.org/cdo/>, 2009.
- [20] —, “Teneo,” <https://wiki.eclipse.org/Teneo/>, 2010.
- [21] F. Basciani *et al.*, “MDEFForge: An extensible web-based modeling platform,” in *CloudMDE@MoDELS*, ser. CEUR Workshop Proceedings, vol. 1242, 2014, pp. 66–75.
- [22] R. Pienta *et al.*, “Scalable graph exploration and visualization: Sensemaking challenges and opportunities,” in *BIGCOMP*, 2015, pp. 271–278.
- [23] A. Díez, “Recursive ontology-based systems engineering,” Patent, 2015. [Online]. Available: <https://patents.google.com/patent/US9760345B2/en>
- [24] D. S. Kolovos, M. Tisi, J. Cabot, L. M. Rose, N. Matragkas, R. F. Paige, E. Guerra, J. S. Cuadrado, J. De Lara, I. Ráth, and D. Varró, “A research roadmap towards achieving scalability in model driven engineering,” in *Proceedings of the Workshop on Scalability in Model Driven Engineering - BigMDE '13*. ACM Press, 2013, pp. 1–10.