Democratizing knowledge graph engineering through domain-specific visual modeling: A tool paper

Andrei Chiş[1], Ana-Maria Ghiran[1]

[1]OMiLAB, Faculty of Economics and Business Administration, Babeş-Bolyai University, Cluj-Napoca, Romania

Abstract

Knowledge Graphs are an established means of structuring personal or enterprise knowledge. Recent advances in Generative AI make Knowledge Graphs even more relevant for knowledge management as semantic structures can be operationalized by LLMs and Agents. Both educational and practitioner communities highlight a lack of flexible diagramming environments for Knowledge Graphs, hindering the adoption to full potential. Existing means are either too technical for novice/business users or they are primarily designed for ontology management, with limited support for instance level or schemaless modeling. This tool paper promotes GRAPHxx, a visual Knowledge Graph building tool built on the ADOxx platform. The tool can be used for both graph schema development and instance-level Knowledge Graph creation, by means of a diagrammatic modeling experience. A Domain-Specific Modeling Language (DSML) was tailored for the RDF specificity, while obscuring RDF-specific jargon behind intuitive diagrammatic constructs aimed to support novice / citizen users.

Keywords

Metamodeling, ADOxx, Visual Knowledge Graph Building, RDF, Teaching Knowledge Graphs

1. Introduction

Knowledge graphs (KGs) provide a structured representation of entities, attributes, and relationships, enabling information to be organized as semantically rich and interconnected networks rather than isolated records. Although the paradigm has a long history [1], its broader adoption remains limited by the technical complexity of standards such as RDF, RDFS, and OWL, the dominance of ontology-first development environments, and the shortage of accessible tools for creating small scale knowledge graph exemplars by intuitive means. This is especially problematic in educational settings and research-oriented demos, Semantic SEO (search engine optimization) based on structured data markup, and GraphRAG [2] context management, where users may need to rapidly capture, communicate, exemplify and operationalize graph-based knowledge without mastering description logics, formal terminology, or even serialization formats. Consequently, there is a growing need for low-code, citizen-oriented knowledge graph building approaches that reduce these barriers and enable a form of knowledge acquisition comparable to the way spreadsheets democratized the management of tabular data.

In order to address these shortcomings, a domain-specific metamodeling approach was taken in developing GRAPHxx, the environment presented in this tool paper. GRAPHxx hides much of the underlying RDF/RDFS complexity while allowing users to model either schema, or instance data, or both, including auxiliary constructs like blank nodes and RDF lists. Model-driven scripts serialize the resulting diagrams as RDF, and transfer them to triplestores or GraphRAG environments; roundtrip deserialization is also supported.

A full research paper on the development and evaluation of GRAPHxx as a Design Science project was reported in [3]. We hereby focus on use and operationalization in more diverse use cases, including newly developed features such as the roundtrip made possible by diagram generation based on RDF import. Therefore we present this paper as a tool demonstrator and usage scenarios for users with different levels of semantic technology expertise, while we refer for engineering and empirical evaluation aspects to the aforementioned paper. Through a series of representative scenarios, the paper demonstrates how

Knowledge Graphs and Model-driven Systems Engineering (KGMDSE), in ISWC 2026 Workshops Joint Proceedings, October 25–26, 2026, Bari, Italy

andrei.chis@econ.ubbcluj.ro (A. Chiş); anamaria.ghiran@econ.ubbcluj.ro (A. Ghiran)



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

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

published 2026-11-05

CEUR-WS.org/Vol-4313/kgmdse-paper2.pdf

GRAPHxx can support enterprise Knowledge Graph engineering, Semantic SEO, personal Knowledge Graph construction, and educational exemplar creation.

The remainder of the paper is structured as follows: Section 2 positions and motivates the GRAPHxx modeling tool against existing approaches for Knowledge Graph engineering. Section 3 presents representative demonstration scenarios. Section 4 further adds details on the diagrammatic usage experience as customized in a metamodeling environment, Section 5 briefly touches on evaluation. Finally, section 6 discusses the current limitations and outlines future directions for the tool evolution.

2. Background and motivation

There is a tendency to advocate Large Language Models (LLMs) as critical means for KG construction to circumvent problematic lack of expertise and low-friction tooling. However, KGs have been built through specialized tooling for a long time before this, as it is equally important to have means for prescriptive (rather than derivative) KG designs, for the ability to communicate, inspect and edit such designs without necessarily involving the token-driven ecosystems of LLMs. Shortage of talent and expertise has been linked to lack of learner-centric resources and visual editors [4], KG tooling being dominated by visualizers or ontology-first toolkits. Preoccupations with curating learning resources for KG management are rather recent [5], with the literature regularly reporting efforts of in-house "professor-ware" development [6]. Surveys are pointing to numerous educational obstacles in the general KG tool and courseware ecosystem [7].

Compared to traditional KG development approaches, GRAPHxx aims to occupy a middle ground between ontology engineering environments, RDF visual mapping, and diagram-to-KG-to-LLM streamlining solutions. Ontology editors such as Protégé [8], OWLGrEd [9] or OntoCASE4G-OWL [10] primarily adopt an OWL-centric and schema-first perspective, requiring users to understand ontology engineering or OWL jargon before creating the simplest triple. Graphol [11] also emphasizes formal ontology specification through a graphical language with description logic semantics and expressiveness equivalent to OWL 2. GRAPHxx [3] places emphasis on the actual graph-based modeling experience derived from the tradition of model-driven engineering and DSMLs (domain-specific modeling languages). KG teaching relies on RDF visualizers to illustrate structure and key constructs, see for example isSemantic [12], RDF Grapher [13], and Graphwise's GraphDB [14] visual exploration facilities, however they generally operate on KGs created through other means - which implies that the user already has some level of syntactic and structuring expertise. Conversely, visual mapping tools such as Miro or PowerPoint provide flexible diagramming experience but are agnostic of any KG technological specificity, serving primarily communication and interaction purposes. Recently some model-driven approaches, such as UMLtoGraphDB [15] or BPMN2KG [16], proposed the transformation of diagrammatic models into graph structures but proposals are typically bound to a prechosen modeling language. GRAPHxx combines the visual flexibility of diagramming tools with metamodel-based constraints, semantic annotations, RDF serialization, and interoperability with external triplestores and GraphRAG/LLM services in a manner that repurposes traditional model-driven engineering. It can be used to create and edit instance data, schema elements, or flexible combinations of both layers within the same diagramming environment that employs model-driven transformations to produce technology-specific (RDF 1.1) content.

Existing toolkits often force users to pick between expertise-intensive ontology editors or visually illustrative tools lacking actual integration with KG ecosystems. GRAPHxx addresses this gap through a domain-specific modeling language that hides much of the RDF terminology behind intuitive concepts such as things, relationships, attributes, types, and containers, while preserving standard compliance at serialization/deserialization time and offering interoperability with triplestores and OpenAI services. Its novelty lies not only in enabling schema-optional or mixed schema-instance editing of KG exemplars (including patterns such as blank nodes and RDF collections), but also in treating the visual model as the starting point of a broader knowledge pipeline integrating visual user-centricity with triplestores and LLMs.

GRAPHxx was implemented in the ADOxx metamodeling platform [17], made available by the OMiLAB digital innovation ecosystem [18]. It enables users to work within a single visual modeling environment while shifting between semi-formal knowledge capture and more structured semantic modeling, depending on the user's preference and background. This is an important feature as targeted users of the tool have various levels of expertise and different purposes: A knowledge engineer might require working predominantly with schema elements whereas an educator may wish to introduce RDF concepts gradually, emphasizing layers of complexity or graph patterns; a SEO specialist may primarily be interested in visualizing the JSON-LD graph to be turned into structured data markup for search engines; a citizen user may simply need an intuitive mind-mapping environment for organizing interconnected networks of associations in a way that can interoperate with triplestores and LLMs. The scenarios presented in this tool paper consequently aim to demonstrate this practical versatility.

3. Use cases

The demonstrator perspective adopted in this paper is illustrated through four representative use cases, targeting different categories of users, as depicted in the use case diagram shown by Figure 1.

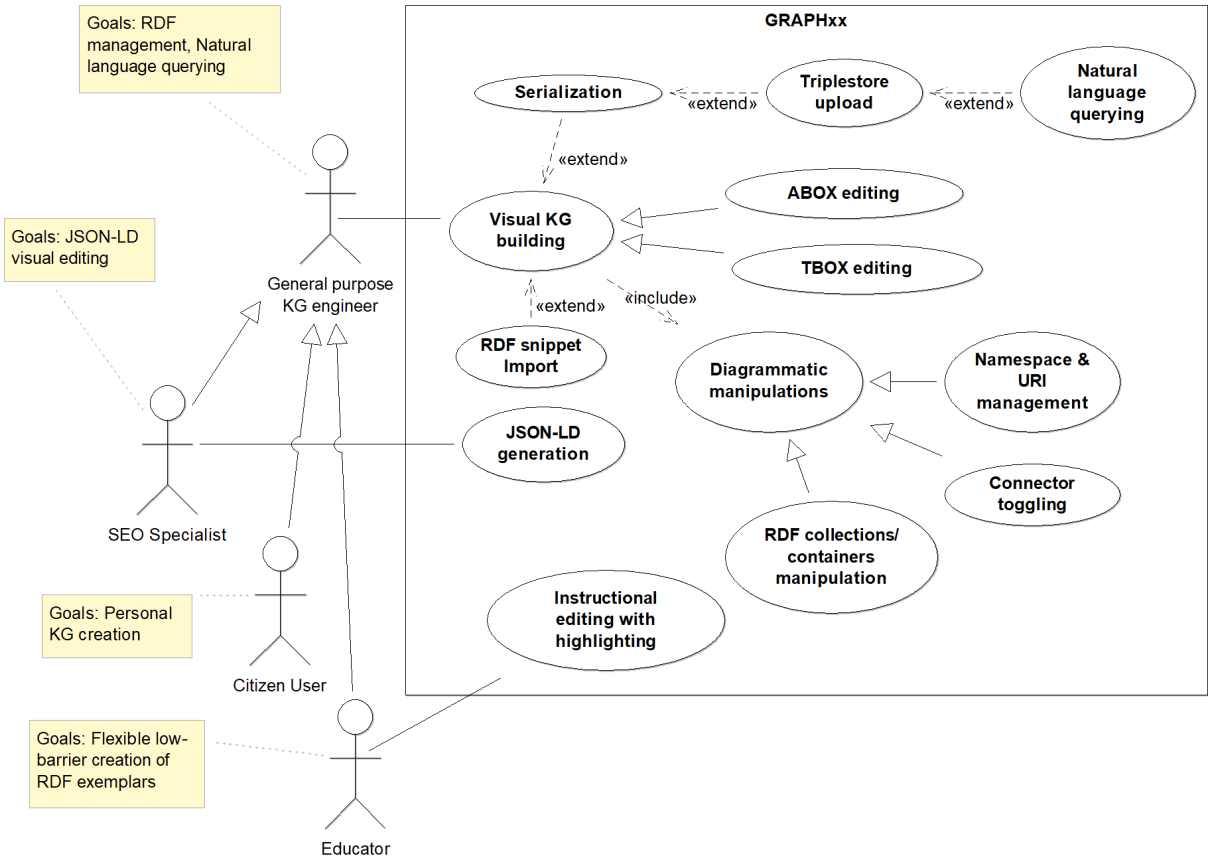


Figure 1: Use Case diagram for GRAPHxx

Primarily, the tool aims to become a visual KG building environment for general purpose KG engineers who need to describe enterprise knowledge in a structured semantic form compliant with the RDF (1.1 for now) abstract syntax. Secondly, SEO Specialists are supported with diagrammatic editing and model-driven JSON-LD snippets generation. Thirdly, citizen users can create personal Knowledge Graphs using intuitive visual constructs as GRAPHxx minimizes exposure to RDF-specific terminology and lowers the entry barrier to an experience closer to mind mapping. Armed with a "learning by doing" mindset, any user can build its own Knowledge Graph, supported by a drag and drop experience and, in most parts, avoiding direct interaction with RDF terminology. Finally, Educators can use the modeling

tool to build live examples in front of students or learners, supported by a functionality that highlights certain constructs or modeling mistakes.

3.1. Knowledge engineers

Knowledge engineers and practitioners of KG-based knowledge management represent the main class of general purpose users for GRAPHxx. Their responsibility is to capture, structure and maintain enterprise knowledge in KG form. Rather than direct manipulation of RDF triples or ontology serialization syntax, GRAPHxx supports them to visually construct either schema level, instance level or combinations of them within the same modeling environment that places the diagramming experience in front of formalism.

An exemplar is illustrated in Figure 2, where an enterprise KG inspired from a medical use case describes relations between patients, clinical encounters, diagnostics, or symptoms. Schema-level concepts combine terms from standard terminologies (e.g. owl:DatatypeProperty) with own vocabulary terms (e.g. brh:DiseaseEpisode, brh:ClinicalEncounter) to define the conceptual structure of the domain. Individual entities such as patients, clinical encounters or disease episodes are described in the instance-level diagram. This separation between schema and instance models allows enterprise knowledge to evolve/be maintained independently by multiple knowledge engineers and business stakeholders. Business users can approve the modeled relationships through intuitive diagrams (no need for RDF knowledge) while KG engineers benefit from automatic RDF serialization, namespace management, URI generation and direct upload to semantic repositories. The example shows inter-diagram links that become rdf:type relations, domains and ranges for properties (at schema level, on the left side) and visual aids that become RDF lists or blank nodes (at instance level, on the right side).

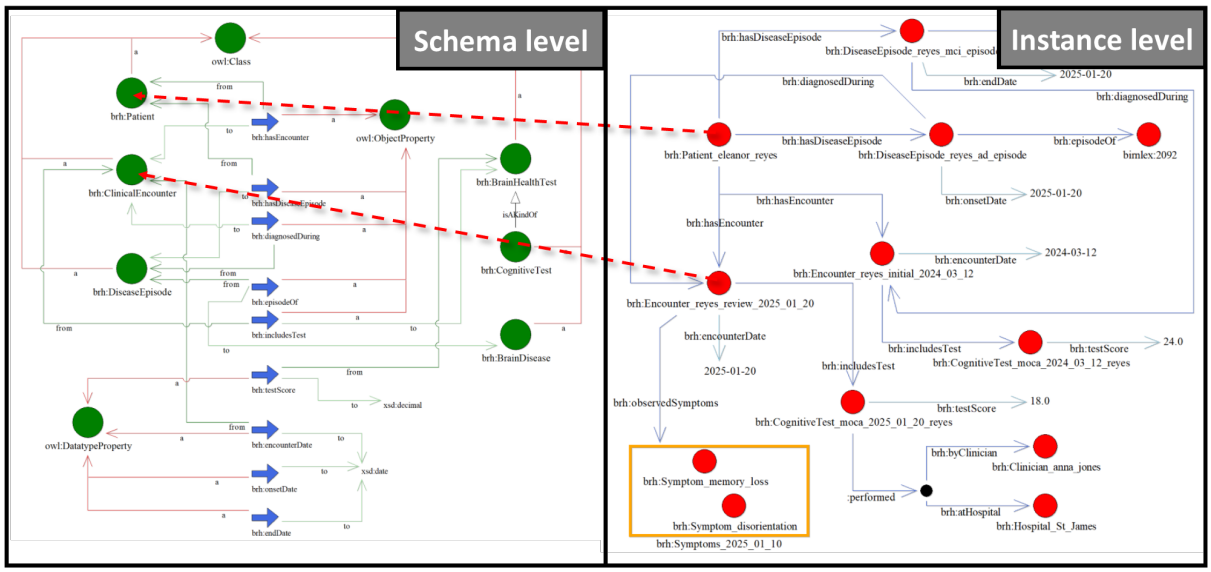


Figure 2: Schema and instance level diagrams

The resulting KG is serialized as RDF and uploaded directly to a GraphDB repository straight from the modeling environment. As Graphwise's GraphDB enables LLM-based querying capabilities, analysts can further exploit this content in a GraphRAG approach without writing SPARQL queries to interact with the diagrammatic content. Thus GRAPHxx serves as an end-to-end environment for knowledge engineers to operationalize key steps in the enterprise KG lifecycle, allowing KG interaction at different levels of technical expertise.

3.2. SEO specialists

SEO specialists require mechanisms for producing structured data markup snippets that can be incorporated in their webpages. Their objective is not necessarily to engineer a complete domain ontology but to structure knowledge objects for Web consumption (search engines, typically). Simultaneously, they would like to minimize the effort in creating and maintaining metadata. GRAPHxx supports this process by adding JSON-LD generation out of diagrammatic designs, allowing semantic annotations to be applied through high level conceptual models rather than custom scripts.

For example (Figure 3), SEO specialists can visually model the products, services, events and organizational information presented on a company's website. The visual design can be automatically exported as JSON-LD for integration as structured data markup into the website's HTML code. Structure data SEO often require users to think in terms of entities and relationships they want to express via the Schema.org terminology, so the tool offers an alternative to manually editing JSON-LD snippets from the Schema.org website[1].

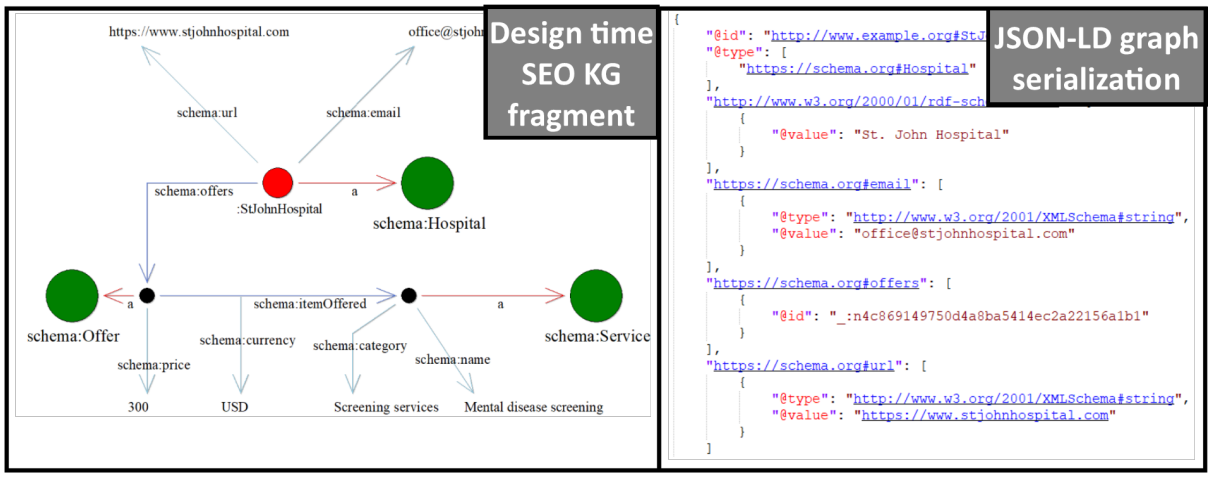


Figure 3: SEO knowledge graph example

3.3. Citizen users

Citizen users who are supposed to potentially organize personal information as semantic descriptions (e.g. the Solid project[2]) typically lack expertise in semantic technologies. They might want to construct a Knowledge Graph for a personal project, a specific area of interest or in communicating an idea to other users or to AI agents. GRAPHxx supports this by abstracting RDF-specific concepts behind intuitive visual constructs.

For example, in Figure 4, a user incrementally builds a KG of books, projects, travel experiences and personal notes. By organizing information through explicit entities and relationships there's a gradual development of a structured personal knowledge base that facilitates long-term organization of personal information. In contrast to conventional note-taking approaches, GRAPHxx encourages users to capture relationships between concepts and to edit data properties which makes the expressed knowledge evolve as a slightly more rigid mind map, where RDF(S) intervenes to support power users.

3.4. Educators

In a similar light, instead of introducing learners directly to RDF syntax or ontology editors, educators can gradually reveal KG concepts, patterns and examples through intuitive diagrammatic constructs - the first triple, the first bucket of data properties annotating an individual, before any considerations on

[1] schema.org
[2] https://solidproject.org/

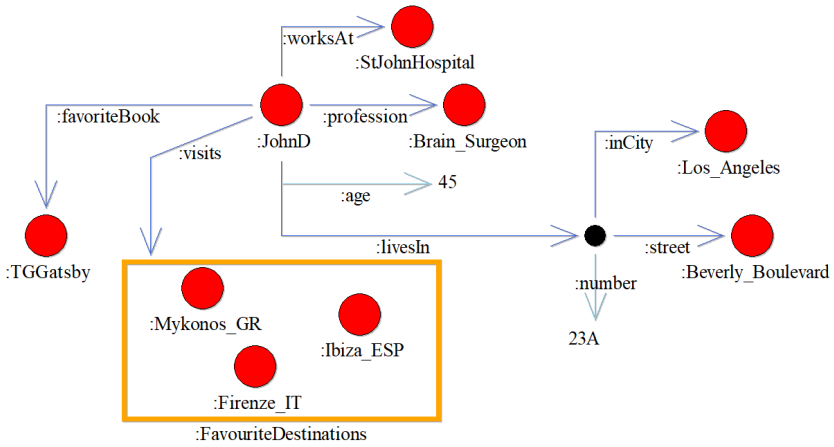


Figure 4: Personal KG example

OWL or RDF jargon are involved (these can be gradually revealed around minimalist examples). The introductory lecture can start by exemplifying the minimum KG with two nodes and a relationship followed by the incremental addition of other patterns, with graphical highlighting of discussed aspects. Students need to focus on the organization of knowledge, to grasp its value and importance before developing expertise on formalisms and encoding syntaxes.

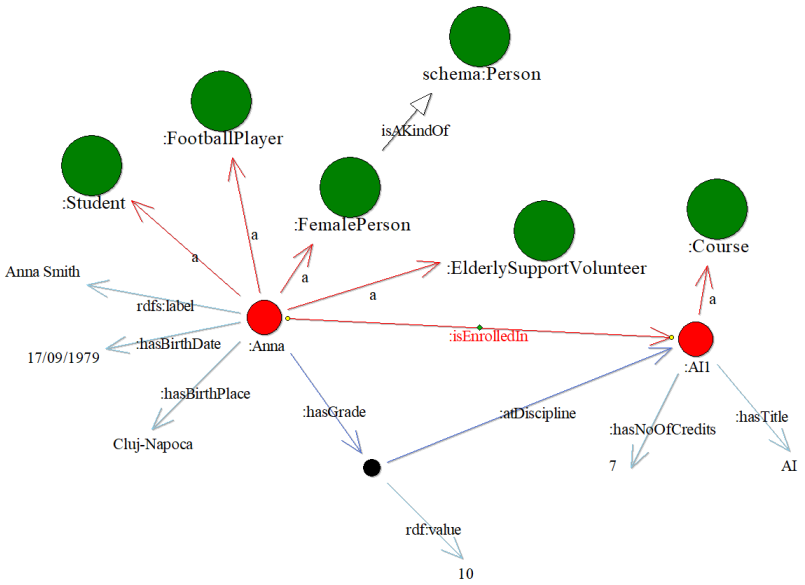


Figure 5: Educational KG example

As suggested in education-focused experience reports [4], educators are encouraged to start from parallels between the more established relational databases and graph-based knowledge representation. The graph illustrated in figure 5 aims to support such a discussion of emulating JOINs between database records with equivalent graph snippets connected directly through object properties. The example is gradually pushed from the initial minimalist data records towards graph-based integration - e.g. multiple types assigned to :Anna, enrichment by subclassing, a blank node for n-ary connections enriched by a direct inferred shortcut relationship (:isEnrolledIn), support for discussing different kinds of nodes (types, individuals, literals) or connections (data properties, object properties, isA and so on) and finally introducing elements of the RDF jargon to achieves standardized descriptions (rdfs:label, rdf:value, subclass assertions). While it is not in the scope of this tool paper to report teaching experience, we only mention that courseware design around this gradual revealing of KG aspects appears to connect

to novices more efficiently than starting from background on description logics and OWL before the first triple is even created. Furthermore, it is followed by a natural step of immediately showing graph queries in a manner that mimicks how relational databases have been taught for a long time to absolute beginners, before delving into relational algebra and normalization.

By first engaging with the visual representation and only subsequently revealing aspects of the RDF specification, students also gain a more intuitive understanding of technology-agnostic graph abstraction. Integrated validation mechanisms further support the learning process by highlighting some incorrect modeling decisions (e.g. triples starting with literal values) or elements that receive emphasis (e.g. CONSTRUCT-inferred connectors). By following this strategy, GRAPHxx aims to provide an educational environment that bridges conceptual understanding and formal semantic modeling - for now producing RDF 1.1 but planning to expand to more diverse graph standards in future versions.

4. General modeling experience

4.1. Visual modeling

GRAPHxx supports three complementary modeling modes: schema diagrams, instance diagrams, and mixed diagrams. The distinction provides flexibility in the presented scenarios, i.e. if they require to separate concerns on the schema and instance levels or not. For example, the first scenario depicted in Figure 2 differentiates them to accommodate distinct viewpoints and abstraction layers for different stakeholders. Therefore the diagrammatic modeling language is partitioned in 2 viewpoints, corresponding to the two main modeling modes:

(a) *Schema level* – contains modeling concepts for defining Knowledge Graphs schemas with different classes, properties and constraints such as domain/range. The typical RDF jargon is avoided in order to make the tool suitable for citizen users while remaining suitable for experienced KGs developers or knowledge engineers. For example, adding domain/range constraints is done by using dedicated connectors: *From* for domain and *To* for range. During the serialization process, they are transformed into the typical RDF constructs. A similar mechanism is also applied for subclassing, or for defining types.

(b) *Instance level* – contains modeling concepts for describing facts, instance-level triples or data properties; Instance level concepts can be linked to their types from the Schema level through inter-model hyperlinks (*Interrefs* in ADOxx terminology). The use of RDF-specific patterns like blank nodes or lists is supported by dedicated visual constructs that are perceived as grouping features on visual level. It is also possible to use schema level concepts at instance level, because the tool aims to provide modelers flexibility for designing visual KGs in any way they want.

The engineering components of the modeling tool are shown in Figure 6, detailing the underlying technologies and connectivity.

4.2. From visual models to RDF

Several mechanisms are integrated by leveraging the ADOxx model-driven scripting capabilities [19]:

Serialization serves the critical requirement of producing actual RDF content from the diagrammatic designs done. Custom diagram parsing scripts were preferred over ADOxx native serialization capabilities [20] for improved streamlining and interoperability. We also want to obtain clean graph representations (which are important in the GraphRAG case), that exclude irrelevant information that has more to do with presentation (layout, colors). The serialization algorithms cover multiple formats: Turtle, TriG and JSON-LD, the latter being particularly relevant for SEO practitioners.

Triplestore integration and *LLM-based natural language querying* are accomplished through a streamlining of AdoScripts and Python scripts that interact with the required external services. In the current version GRAPHxx provides integration with Graphwise GraphDB, allowing serialized Knowledge Graphs to be uploaded to a selected repository without leaving the modeling environment. GraphRAG

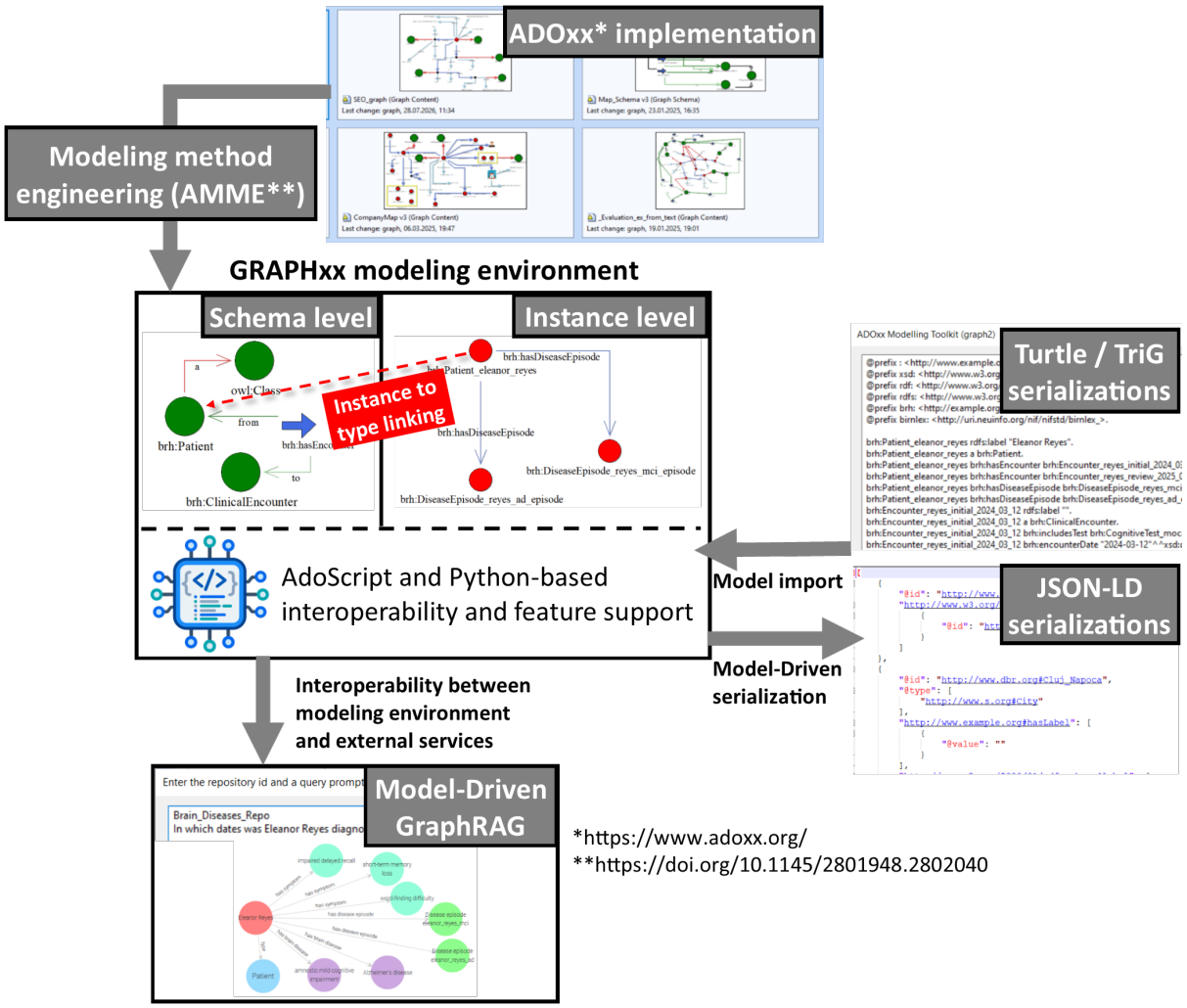


Figure 6: Engineering components of the modeling tool

workflows are supported because natural language questions can be submitted from the modeling environment. These are forwarded to an LLM which responds based on the Knowledge Graphs pushed into selected repositories based on a variety of subgraph context extraction options.

For element identifiers GRAPHxx applies a default unique URI generation from labels and provides mechanisms for namespace management at model level. ThingIdentifier constructs (Figure 7) provide the mechanism that composes and associates identifiers to visual elements.

The Name is the internal ADOxx identifier at diagram level, preventing two visual elements with the same name in the same diagram. In the case of GRAPHxx, we also leverage such names as a form of uniqueness constraint. Concretely, modelers can choose to produce the URI from the Name attribute, thereby ensuring that two elements with the same URI cannot be added to the same diagram.

For prefix management, the tool provides a functionality through which prefixes can be defined at model level and are subsequently presented to users for reuse, rather than relying on manual typing more prone to errors.

Local identifiers can be manually specified by users or automatically generated by the modeling tool based on the visual labels. Multilingual labels are also supported.

The *IsOfType* attribute is a trivialized version for *rdf:type* or its short form *a*, allowing modelers to set instance-to-class hyperlinks across different diagrams representing instances and schemas.

The *Attributes* table can be used for adding RDF properties for the current entity in a tabular form rather than drawing them as diagram connectors. This approach offers visual flexibility (e.g. to obscure data properties, to toggle between showing and not showing them in the visual graph). Syncing

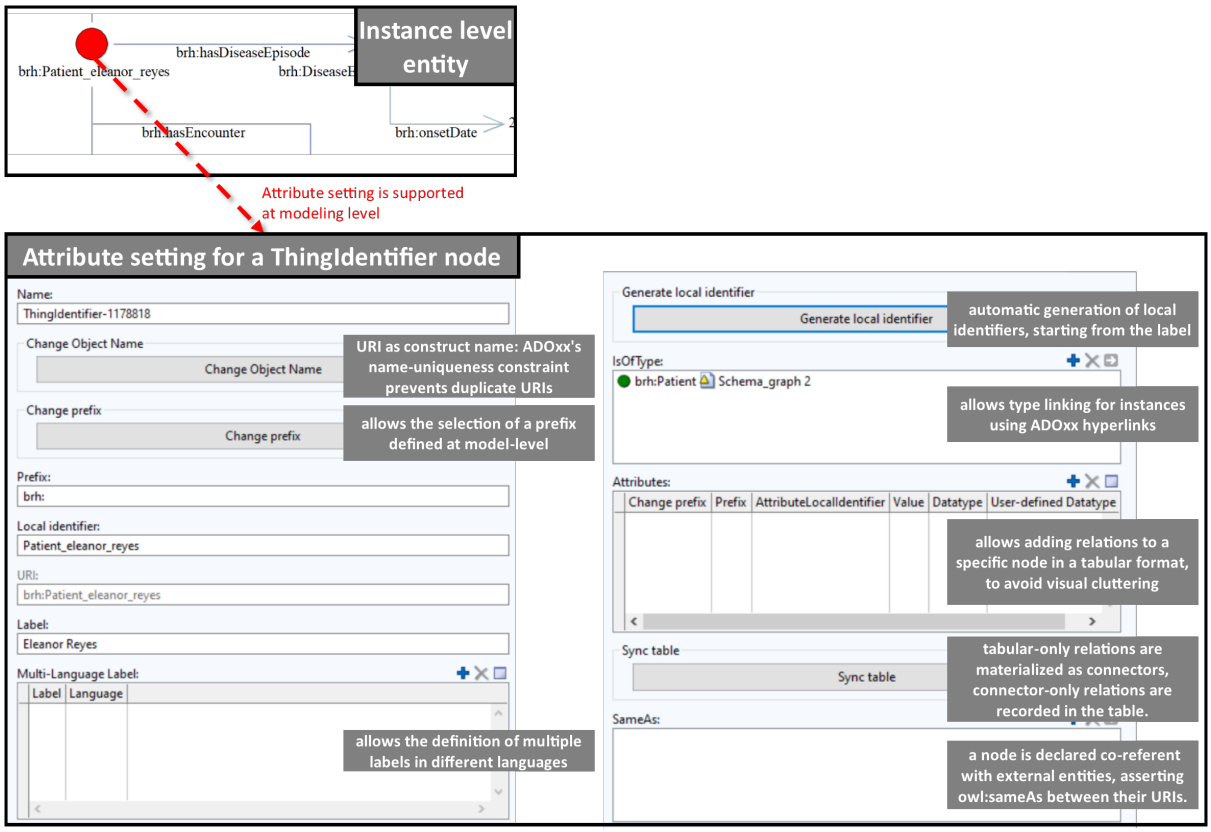


Figure 7: Attributes of a ThingIdentifier modeling construct

mechanisms transform such tabular property lists into visual connectors and vise-versa.

The *SameAs* attribute is a trivialized version of *owl:sameAs* and it was implemented to declare equivalence between elements from different diagrams, thus it also acts as a hyperlink between subgraphs pertaining to the same resource.

Key mechanisms for correctness or usability support are suggested in Figure 8.

4.3. Managing and balancing the visual experience

A graphical model created on GRAPHxx is more than a simple collection of nodes and binary relationships - grouping achieved by blank nodes, *containers* and collections of related elements can also be defined. The user interacts primarily with the DSML concepts and connectors, while the tool applies specific transformation rules to generate in the background correct RDF patterns.

These mechanisms include the conversion of groups of visual elements into RDF lists, or viceversa. For instance, the modeler chooses which connectors should be deleted, what will be the URI of the generated container, what relation will be used to link the list head instead of individual connectors.

The tool also includes mechanisms for showing or hiding connectors and types according to specific needs and the risk of visual cluttering - e.g. modelers can choose whether to depict types and relations through visual connectors in instance diagrams or whether these should be kept solely in schema level diagrams and connected through hyperlinks.

Connector multiplication helps with reusing the same RDF property numerous times, while minimizing the UI interaction (drag and dropping) effort. Such usability aspects are designed to lower the typical RDF graph editing (repetitive) effort on diagrammatic interaction level.

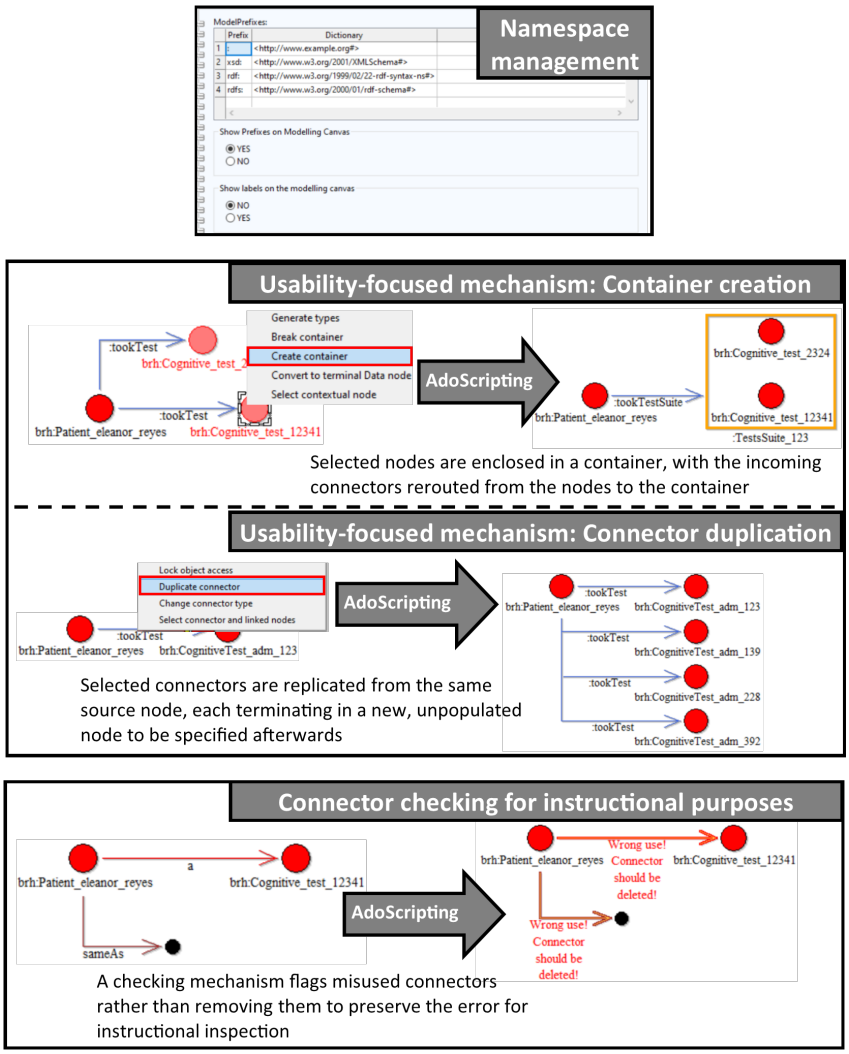


Figure 8: Modeling tool mechanisms

4.4. Roundtrip engineering of knowledge graphs

The capabilities described above can be combined in an integrated workflow: a user can create concepts and instances, define relationships and attributes, assign identifiers and namespaces, obtain the serialized representation, upload the Knowledge Graph to a triplestore, and querying the graph in natural language, thanks to the GraphRAG pipeline enabled by the tool.

Compared to the earlier reported iteration [3], now full roundtrip is supported by allowing the tool to also import external graphs and populate the visual canvas as suggested by Figure 9. KG fragments in Turtle or TriG formats can be provided either as pasted RDF snippets or as imported files. The imported RDF is first deterministically parsed into an intermediate GRAPHxx-specific representation containing nodes, edges, attributes, and prefix information, which is subsequently interpreted by AdoScript to instantiate the corresponding editable diagrammatic constructs. To reduce visual clutter, RDF data properties are by default generated as tabular attributes of their subject nodes rather than visual elements, then synching mechanisms can expand the visualization.

5. Evaluation aspects

A detailed evaluation protocol, balancing user experience, tool comparisons and technical aspects has been detailed in [3], we only highlight here some scalability considerations. Going beyond the

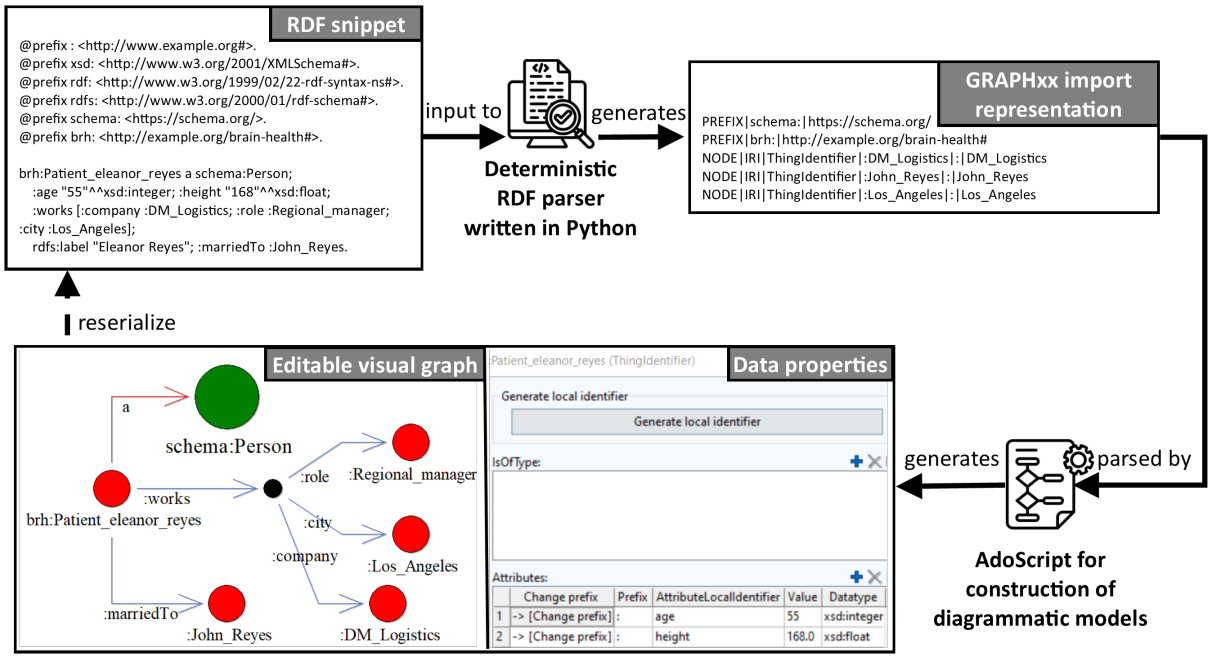


Figure 9: Roundtrip engineering of knowledge graphs with GRAPHxx

educational purpose of designing minimalist exemplars, we scripted the automatic generation of larger graphs (up to 1500 nodes) with quantitative outcomes shown in Table 1. The experiment took place on a personal laptop running Windows 10 Home 64-bit, with 16GB of RAM and Inter Core i7-8750H @ 2.20 GHz CPU. The used version of ADOxx is 1.5.

Practical limitations are noticeable regarding the import of industry-grade large graphs, which is reliant on the canvas-populating engine of the underlying metamodeling platform. Diagram serialization performs significantly better. Content fragmentation/consolidation mechanisms across multiple diagrams pertaining to the same graph will be considered for future iterations.

No. of nodes	No. of triples	Diagram generation time (s)	Diagram serialization time (s)
100	200	1.84	0.06
300	600	19.59	0.17
500	1000	75.47	0.30
700	1400	184.70	0.42
1000	2000	517.30	0.63
1500	3000	1707.44	0.94

Table 1
Automatic diagram generation and serialization times

Competency evaluations can be performed at DSML level through the underlying ADOxx AQL querying of visual connectors and inter-model hyperlinks, which is a general purpose model-query language leveraging the graph-like nature of the ADOxx meta-metamodel and allowing to emulate basic SPARQL features. An example of an AQL query designed for a medical Knowledge Graph use case, is shown in Figure 10. AQL queries can be particularly useful to business users using a query builder (to generate AQL syntax) in order to check for certain diagrammatic patterns following a mind-mapping inspection approach, rather than directly running SPARQL on an RDF repository.

As the tool has not yet reached full maturity, it is currently distributed as an ADOxx library that must be imported to access its full functionality, available at: https://github.com/andreichis97/GRAPHxx.A standalone installation kit is currently a work in progress.

Query: Display all the tests which were done for patient Eleanor Reyes.

Code:
```
((<"ThingIdentifier">[?"Label" like
"Eleanor Reyes"] -> "RelatedTo") ->
"RelatedTo")[?"URI" like
"brh:CognitiveTest*"]
```

	Label
1. Instances_graph	
ThingIdentifier-1178841	Cognitive Test MOCA 03.12.2024
ThingIdentifier-1178850	Cognitive Test MOCA 20.01.2025

Figure 10: AQL query (on the left) & its results (on the right)

6. Limitations and outlook

As reported in the paper, GRAPHxx offers a flexible visual KG building experience, with a suite of functionalities to streamline the designed content to external KG-based services or storage. Personal or Domain-Specific Knowledge can be captured and structured in a diagrammatic manner, while LLM-querying pipelines can be accessed directly from the modeling environment. Structured data markup SEO and educational presentations are supported as practitioner-specific use cases that have been traditionally lacking in support.

One key limitation of GRAPHxx is its exclusive support for RDF 1.1 Knowledge Graphs, without support for LPG or new RDF 1.2 constructs e.g. reification. RDF 1.2 and visual reification management is the priority for the next implementation iteration. Moreover, at the time being the modeling tool interoperates only with Graphwise GraphDB triplestores and large language models from the OpenAI family.

As part of the AI-readiness orientation of the tool, further work will also target AI assistance at both the input and output ends of GraphXX - i.e. as modeling companion to make suggestions, and as model consumers expecting granular graph-structured context for LLMs.

Acknowledgments

This research is supported by the project "Romanian Hub for Artificial Intelligence - HRIA", Smart Growth, Digitization and Financial Instruments Program, 2021-2017, MySMIS no. 351416.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT for grammar and spelling check and to reformulate for improving clarity. The authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. De Melo, C. Gutierrez, S. Kirrane, J. E. Labra Gayo, R. Navigli, S. Neumaier, A.-C. Ngonga Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, A. Zimmermann, Knowledge graphs, ACM Computing Surveys 54 (2021) 1–37. doi:10.1145/3447772.

[2] B. Peng, Y. Zhu, Y. Liu, X. Bo, H. Shi, C. Hong, Y. Zhang, S. Tang, Graph retrieval-augmented generation: A survey, ACM Transactions on Information Systems 44 (2025) 1–52. doi:10.1145/3777378.

[3] A. Chiş, A.-M. Ghiran, R. A. Buchmann, GRAPHxx: DSML engineering for knowledge graph building and streamlining with GraphRAG, Software and Systems Modeling Journal (SoSyM) 25 (2026) 109–134. doi:10.1007/s10270-025-01297-y.

[4] R. A. Buchmann, A.-M. Ghiran, Teaching knowledge graphs: A journey from logic to web development and semantics-driven engineering, in: The SCME23: 7th Symposium on Conceptual

Modeling Education @ The 42nd International Conference on Conceptual Modeling (ER 2023), volume 3618, CEUR-WS, Lisbon, Portugal, 2023. URL: https://ceur-ws.org/Vol-3618/scme_paper_4.pdf.

[5] E. Ilkou, H. Abu-Rasheed, D. Chaves-Fraga, E. Engelbrecht, E. Jiménez-Ruiz, J. E. Labra-Gayo, Teaching knowledge graph for knowledge graphs education, Semantic Web, 0 (0) (2024) 1–27.

[6] C. M. Keet, In-house developed tools for ontology engineering education (2023).

[7] B. Abu-Salih, S. Alotaibi, A systematic literature review of knowledge graph construction and application in education, Heliyon 10 (2024).

[8] Stanford University, Protégé ontology editor, 2026. URL: https://protege.stanford.edu/, n.d. Last accessed 17.07.2026.

[9] University of Latvia, Institute of Mathematics and Computer Science, OWLGrEd, 2026. URL: http://owlgred.lumii.lv/, n.d. Last accessed 27.07.2026.

[10] M. Héon, R. Nkambou, M. Gaha, OntoCASE4G-OWL: Towards a modeling software tool for G-OWL a visual syntax for RDF/RDFS/OWL2, in: International Workshop on the Semantic Web, CEUR-WS, 2016. URL: https://ceur-ws.org/Vol-1690/paper52.pdf.

[11] D. Lembo, V. Santarelli, D. Fabio Savo, G. De Giacomo, Graphol : A graphical language for ontology modeling equivalent to owl 2, Future Internet 14 (2022) 1–29. doi:10.3390/fi14030078.

[12] isSemantic.net, issemantic RDF graph visualizer, 2026. URL: https://issemantic.net/rdf-visualizer, n.d. Last accessed 25.07.2026.

[13] RDF grapher, 2026. URL: https://www.ldf.fi/service/rdf-grapher, n.d. Last accessed 29.07.2026.

[14] Graphwise, GraphDB, 2026. URL: https://graphwise.ai/components/graphdb/, n.d. Last accessed 31.07.2026.

[15] G. Daniel, G. Sunyé, J. Cabot, UMLtoGraphDB: Mapping conceptual schemas to graph databases, in: Proceedings of the 35th International Conference on Conceptual Modeling (ER), Conceptual Modeling, Springer International Publishing, Gifu, Japan, 2016, pp. 430–444. doi:10.1007/978-3-319-46397-1_33.

[16] S. Bachhofner, E. Kiesling, K. Revoredo, P. Waibel, A. Polleres, Automated process knowledge graph construction from BPMN models, in: Proceedings of the 33rd International Conference on Database and Expert Systems Applications, Springer International Publishing, Vienna, Austria, 2022, pp. 32–47. doi:10.1007/978-3-031-12423-5_3.

[17] BOC GmbH, The ADOxx metamodeling platform, 2026. URL: https://www.adoxx.org, n.d. Last accessed 28.07.2026.

[18] D. Karagiannis, R. A. Buchmann, W. Utz, The OMiLAB digital innovation environment: Agile conceptual models to bridge business value with digital and physical twins for product-service systems development, Computers in Industry 138 (2022) 103631. doi:10.1016/j.compind.2022.103631.

[19] OMILAB NPO, AdoScript, 2026. URL: https://www.omilab.org/adoscript/, n.d. Last accessed 29.07.2026.

[20] OMiLAB NPO, RDF transformation for ADOxx, 2026. URL: https://code.omilab.org/resources/adoxx-modules/rdf-transformation/-/tree/master, n.d. Last accessed 31.07.2026.