



GRAPHxx: DSML engineering for knowledge graph building and streamlining with GraphRAG

Andrei Chiș¹ · Ana-Maria Ghiran¹ · Robert Andrei Buchmann¹

Received: 29 March 2024 / Revised: 28 April 2025 / Accepted: 9 May 2025
© The Author(s) 2025

Abstract

Symbolic AI has been facing long-term adoption obstacles and a slow uptake caused by the limited availability of frictionless tooling—that can support not only knowledge engineers, but also novice users and educators, with designing and presenting graph exemplars that can be visually communicated, edited and ad hoc processed. There's still a shortage of tools that can democratize knowledge graph (KG) creation, something that is increasingly needed—firstly by educators trying to discuss examples with novices without stumbling on OWL jargon at the earliest step, but also for more recent integration cases such as (i) GraphRAG, where private KGs are called to augment or ensure factuality of large language model services; or (ii) in search engine optimization (SEO), where SEO practitioners lacking knowledge engineering background must embed Schema.org graph data into their Web content. Most visual KG tools are visualizers of KGs created by other means—either in OWL-centric ontology editors posing high expertise barriers, or converted from available serializations, or lifted from legacy data sources. The few actual KG editors are mostly OWL editors neglecting to support the creation of schemaless graph datasets as well as of flexible combinations of graph data and schema fragments. This paper reports on a Design Science effort adopting metamodeling means, traditionally employed for the engineering of domain-specific modeling languages, toward defining a KG development and integration method that facilitates both the visual design of KG exemplars and their operationalization. We aim to balance a diagrammatic look and feel with machine readability of the semantic content being produced—further streamlined in an architecting proposition for integration with LLM services and for the production of Schema.org graph snippets. The DSML was deployed and evaluated as a tool implemented on the ADOxx metamodeling platform, using RDF and LangChain as mediators that streamline the content toward triplestores and LLM services.

Keywords Knowledge graphs · Domain-specific modeling language · RDF · Retrieval augmented generation · LLM · ADOxx

1 Introduction

Although knowledge graphs (KG) are not a recent paradigm (see a rich background in [1]), their uptake has been

Communicated by Lola Burgueño, Davide Di Ruscio, and Dominik Bork.

✉ Andrei Chiș
andrei.chis@econ.ubbcluj.ro

Ana-Maria Ghiran
anamaria.ghiran@econ.ubbcluj.ro

Robert Andrei Buchmann
robert.buchmann@econ.ubbcluj.ro

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

generally slow due to a number of factors—from technological readiness and curricular coverage in Computer Science/Information Systems education, to limited availability of educational tools. In the absence of tooling easily adopted for didactic exemplars, students tend to perceive symbolic AI and knowledge representation as an academic niche of little practical readiness, failing to grasp analogies with more traditional data management approaches [2]. In an age when automation became so prevalent that we're discussing “citizen development” and “citizen automation” [3], there is an emerging need to also enable “citizen knowledge acquisition” which raises a global requirement for making available low friction tools for knowledge structuring, not too different from how spreadsheets democratized the collection and editing of tabular data. In this context, we use the term “democratization” to refer to lowering adoption barriers and stimulating technological uptake among non-technical users

(i.e., who lack in-depth training with knowledge engineering standards and formalisms). This desideratum is achieved in the automation field through low-code platforms, and we look for a similar approach to KG building—this requires artifacts, design propositions and knowledge modeling procedures that we build through the hereby presented Design Science research (DSR) [4] project.

By analogy with established interpretations of low code [5], we employ here the term “low-code knowledge graph building” for means that exempts the user from manually creating machine-readable graph serialization or from writing code that generates such serializations. Low-code KG building must remove usage friction, allowing users to quickly create KG structures without necessarily mastering description logics or the jargon of formal standards—ideally this should be similar to how data entry operators manage tabular datasets without extensive background about relational algebra or normalization. While we take for granted the data entry user experience for tabular data, there has been little work on enabling a similar experience for building KG exemplars, although graph-like knowledge visualization has always been around—as mind maps or other kinds of diagramming.

Our design proposition of enabling in the same tool visual communication, visual editing and ad hoc processing of KG exemplars should not be confused with KG visualization of content created by other means—an approach widely supported in both commercial and free tooling, e.g., GraphDB’s visual graph feature [6], isSemantic [7] or similar ones.

We identified three application cases raising requirements for this ability to quickly build KG fragments by circumventing formal and technical expertise as much as possible. The first two requirements are derived from direct interactions with educators and practitioners, whereas the last was identified as a trend in the literature:

1. Academics need to be able to **visually manage KG exemplars** for fast on-boarding of research partners or to stimulate the interest of novices (students), similarly to how tabular data can be interacted with on spreadsheets. This should minimize formalism jargon (in this case RDF/OWL), preparatory steps (as imposed by ontology-first tooling, e.g., Protégé [8]) or the need to switch tooling even for the simplest demonstration. This should be achieved by means that serve at the same time visual communication, interactive visual editing and some extent of ad-hoc demonstration for major use cases (semantic queries or transformations). The need also arises when incorporating examples in scientific publications or during workshopping;
2. Since the advent of Google’s Knowledge Graph, SEO (search engine optimization) specialists are incentivized to incorporate knowledge graph snippets into HTML

[9]. Google prioritizes for this the JSON-LD serialization format [10] to allow Web developers to work with a JSON-like syntax without learning knowledge representation or mastering the extensive RDF ecosystem. Still, the structure they need to create is an RDF graph serialized as JSON-LD [11] that employs Schema.org as a globally shared vocabulary recognized by search engines;

3. **GraphRAG is a family of Generative AI-KG hybridization** approaches (with several flavors advertised by technology vendors [12]) that leverage the relationship-dense and semantically rich content of KGs to improve large language model (LLM) answer quality and grounding in private structured data. LLM users are encouraged to adopt GraphRAG architectures [13, 14] to improve the factuality and contextual relevance of responses. This has possible ramifications toward Personal Knowledge Graphs [15] as a citizen-oriented approach to structured note taking that can also be leveraged by LLM agents.

The source of application cases and associated requirements (to be detailed in Sect. 2) are interviews with 13 KG educators and applied science researchers across 6 European universities who use heterogeneous toolkits to manage their KG content. We also interviewed two SEO practitioners from the local outsourcing industry who routinely use the Schema.org—Google Knowledge Graph approach for semantic search engine optimization. The interviews aimed to glean the pains (to be illustrated by quotes in Sect. 2) and preferred tooling on graph content management. Three streams of questioning looked for information pertaining to the following scenarios:

- How KGs are introduced to novices in terms of teaching content and visually-supported exemplification, especially as audiences arrive at KGs from very diverse needs and perspectives—graph databases, GraphRAG, SEO, and ontology engineering;
- How KGs are presented to funding bodies representatives, industry or research partners lacking KG expertise (i.e., in the context of research projects);
- How graph snippets are built in proof-of-concept and applied research demonstrators of low technological readiness, for technical reports, during co-creation workshopping or for various pragmatic purposes (e.g., SEO).

In terms of preferred tooling, the following were reported as prevalent: Protégé with associated visualizers (Ontograf,¹

¹ <https://protegewiki.stanford.edu/wiki/OntoGraf>

Ontoviz²), free on-line visualizers (isSemantic,³ RDF grapher⁴), commercial tools that also have visualizers (Ontotext GraphDB,⁵ even Neo4J—indirectly through the RDF ingestion plug-in⁶ for the sake of visualization), the Fuseki-Jena ecosystem with AWS graph explorer,⁷ WebVOWL⁸ and its deployment in Gra.fo.⁹ Not surprisingly, a major tool family for content creation are free-shape diagramming tools that are not RDF-specific—Powerpoint, Miro, Draw.io—and consequently provide no technological support besides presentation. *The work reported in this paper aims for the diagramming freedom of these, coupled with the technological specificity of the RDF/OWL tools.* Various shortcomings arising from the dominance of visualizers and ontology-first tooling will be detailed in the Evaluation section.

The SEO practitioners specifically appear to be disconnected from the entire RDF/OWL ecosystem despite using RDF graphs (in JSON-LD format). The industry adopts Google-centric tools like the Schema markup validator¹⁰ and the Rich Results testing tool.¹¹ Content is created by editing and manipulating JSON-LD snippets already available on Schema.org, while remaining agnostic of the graph management ecosystem and this also misses opportunities—like lifting or managing JSON-LD in a semantic graph database or exposing it to GraphRAG. None of the SEO practitioners were aware of such possibilities, assuming that JSON-LD snippets only contribute to Google’s Knowledge Graph instead of seeing them as general purpose KG fragments.

To support such stakeholders, this paper’s proposal is a graphical DSML (and its associated modeling method deployment) that obscures as much as possible the formal details of knowledge representation standards such as RDF/S and OWL, while flexibly producing diverse RDF patterns (graph data, schema fragments or combinations) and delivering them to external RDF-aware systems (triplestores, GraphRAG installations). We’ve been surprised by the shortage of tooling available for this—the RDF ecosystem being dominated by visualizers of graphs created by non-visual means [7], or ontology-first (OWL-centric) editors exposing the underlying logic perspective and the OWL jargon—that

must be acquired even for a “hello world” example. The low-code paradigm did inspire some recent efforts of enabling visual KG building, but the focus is typically on the TBOX layer [16] or limited to the presentation use case [17].

Our hereby proposed DSML supports the creation of flexible graph structures—that may contain either only facts, or data with some superficial schema elements (e.g., RDFS-plus), or corner case patterns (e.g., blank nodes, lists), but at the same time has a reasonable scalability to populate a fully fledged semantic graph database, along with the interoperability—enabled through LangChain [18]—required to expose graph fragments to a GraphRAG architecture.

The tradition of domain-specific modeling has understood this low friction knowledge capture requirement for a long time; highly usable diagramming tools are widely available for ER, UML, etc., supporting code generation or roundtrip engineering—approaches that evolved toward modern model-based digital twin propositions [19]. Our idea was inspired by Bee-Up tool’s ability to generate SQL statements from ER diagrams [20, 21]. However, ER diagrams are closer to OWL editors, focusing on the schema level rather than on populating data structures. With graph data, there’s also value in having instance-level data exemplars depicted as navigable, relationship-dense semantic networks—e.g., to discuss a blank node pattern or to provide visual support to novices trying to visually assess the navigation paths of a SPARQL query [22].

The initial DSR iteration of the tool was presented as a short paper in [23]. This paper extends that report by providing a detailed account of progress achieved with a richer metamodel, enhancing the user experience and broadening the tool’s functionality to address use cases pertaining to Semantic SEO and GraphRAG.

The paper is structured according to the Design Science methodological frame: Sect. 2 provides extensive information about the problem identification objectives definition. Sections 3 and 4 discuss design decisions and implementation, split according to the two design propositions of our treatment: the DSML and the architecture to enable its knowledge streamlining. Section 5 presents application scenarios. Section 6 discusses the evaluation strategy. Section 7 comments on related works, and the paper concludes with a SWOT analysis.

2 Problem statement and methodological framing

We aim for a Pareto-driven approach where 80% of the user needs are served by 20% of the RDF-based specifications captured in a graphical modeling method. Usage friction should be gradual and not an entry barrier—i.e., manifesting only as knowledgeable users advance toward

² <https://protegewiki.stanford.edu/wiki/OntoViz>

³ <https://issemantic.net/rdf-visualizer>

⁴ <https://www.ldf.fi/service/rdf-grapher>

⁵ <https://graphdb.ontotext.com/documentation/10.8/visualize-and-explore.html>

⁶ <https://neo4j.com/labs/neosemantics/>

⁷ <https://www.bobdc.com/blog/graphexplorerandfuseki/>

⁸ <https://github.com/VisualDataWeb/WebVOWL>

⁹ <https://gra.fo/>

¹⁰ <https://validator.schema.org/>

¹¹ <https://search.google.com/test/rich-results>

corner cases and advanced patterns. This has turned our effort into a layered approach that naturally fits the iterative nature of both (a) the DSR cycle—as a generic artifact building approach [4] and of (b) the Agile Modeling Method Engineering framework (AMME)—as a specific methodology for DSML engineering with diagrammatic tool deployment [24].

Being experienced with modeling tool implementations within the ADOxx ecosystem [25], we employed this meta-modeling platform (comparable to alternatives such as Sirius [26], MetaEdit [27] or AOAME [28]) because of its long-term viability in producing both commercial tool and in supporting exploratory research on DSMLs—typically within the OMILAB modeling community [29]. The platform’s meta-metamodel, metalanguage, scripting capabilities and “low-code” features have been analyzed over the years, see [30, 31], making it a well-established technological choice.

Relative to the “Purpose-Specificity framework” in [32], our proposed DSML is a *technology-specific modeling language* because it only provides support for semantic graph databases—for now RDF, and has a rather narrow set of purposes—to visually build graphs that can also be streamlined (using RDF as a mediator format) in knowledge flows involving LLM services. At the same time, it also aims to support visual demonstration when it is more important to obtain a KG exemplar by diagramming means, than to establish the formal grounds of knowledge representation and standard specifications.

Tool requirements have been accumulated iteratively—we started from our own experience as KG educators working on the gradual improvement of a KG bachelor course over the last 15 years—a journey we reported in ER’s symposium of conceptual modeling education [2]. The journey allowed us to exchange experiences with other KG educators and weakly structured interviews were used to collect pain points about the KG editing tooling and the learning objectives affordances of available tooling. Below are samples from such interviews with educators from both Information Systems and Computer Science programs:

- Sample statement from educators: *Sometimes we simply want to define ad hoc nodes and edges regardless of serialization and without stumbling at every step on the need to explain OWL constructs. We’re also stressing the schema-free nature of linked data as opposed to the schema-first approach of relational databases, and this gets contradicted when students face tools that force them to take an ontology-first approach;*
- Sample statement from educators: *I will explain later prefixes, URIs, domains, ranges and axioms. First, I need to convince novices that, whatever table queries they are familiar with, they can be easily adapted to the graph mindset by means of visual patterns. They will have to deal with the syntactic technicalities later when they create a*

fully fledged semester project. When we used to start with description logics, we never arrived at a running example implemented with basic Web development frameworks;

- Similar interviews have taken place with SEO practitioners in our region, dominated by outsourcing services in Web development and SEO, with an eloquent sample selected here: *I would like to create the JSON-LD snippets without learning RDF, but it is easier to structure my data as a network of associations than directly writing JSON-LD. So I am looking for a middle ground where I draw my graph and obtain JSON-LD out of my drawing.*

Lastly, the GraphRAG scenario was adopted from recent technical propositions available in the literature [33] or made by KG platform vendors [12], as this is maturing into a key hybrid AI approach. This component is intended to support our future project-based experimentation with LLM factual improvements and does not have yet concrete stakeholders outside our workspace. Therefore, part of this work is also an architecting approach following structured prompting strategies recommended by OpenAI [34] to streamline the diagrammatic designs within a GraphRAG knowledge flow, using RDF as a mediator/interchange format.

Currently, KG building is done by semantic technology experts in specialized tools that first require familiarity with standards such as OWL and RDF, if not with underlying formal logics. Visual editing tools focus on the TBOX level, with a few exceptions. The lack of tooling with low entry barriers also gave opportunity for LLMs to be advocated as stochastic KG building tools, e.g., through the KG “completion” task [35].

From stakeholder interactions, we distilled granular functional and usability requirements that drive this project:

R1. A drag-and-drop user experience must be facilitated for connecting nodes with arrows; mind mapping and graphical note-taking tools have established this as a popular practice of structuring ideas, and was the originating motivation for repurposing here the practice of DSML engineering;

R2. Flexible content mixing: the graph exemplars to be created may be of heterogeneous nature—i.e., to contain only data points, only networks of associations between identifiers (URIs), only schema/ontology constructs or any convenient combinations of these, with good visual discrimination and partitioning possibilities to avoid visual cluttering—i.e., supporting both the separation of TBOX and ABOX in hyperlinked diagrams, as well as their combined connected visualization;

R3. From an educational perspective, the tool must convey the **message that semantic graph databases are not schema-first**, but rely on a looser and decoupled notion of schema—i.e., the schema can be postponed, implicit (as in Neo4J), created by someone else, linked to data points at a

later time or even entirely neglected (data can be queried in the absence of an explicit ontology);

R4. Dynamic notation should be agile to avoid visual cluttering, providing mechanisms—well supported by meta-modeling platforms—for switching between different visual loads: a default style with minimalist visual symbols and the possibility for users to add custom icons for semantic transparency; hiding or showing attributes (in the diagram or as tabular annotations); hiding or showing types/classes in the same diagram with their instances, or linked from a separate “schema” diagram; visual identification by flexible means (prefixed URIs or local identifiers/labels);

R5. RDF/OWL jargon should be avoided wherever possible, users must think in terms of relationships between things, with data attributes attached to those things and occasional visual grouping (for RDF patterns such as blank nodes, containers/lists). Since we plan to also support in future versions LPG (labeled property graphs) [36], the tool itself avoids RDF-specific terminology in the first-class modeling constructs, but also allows it to be selected by knowledgeable users who want to designate some constructs as standard RDFS/OWL terms;

R6. Serialization must be provided in convenient formats. The semantic SEO scenario requires generation of JSON-LD that can be copy-pasted into HTML code; the general case requires a well-established format that can be uploaded to a triplestore or exposed to LLM services (Turtle/TriG is the current choice);

R7. Interoperability with triplestores and LLM services must expose graphs to KG-LLM hybridizations such as GraphRAG configurations and natural language querying. This implies an architecting and functional requirement that in the current implementation is addressed using Ontotext’s GraphDB and OpenAI models, mediated by Python and Langchain scripts triggered from ADOxx’s internal scripting and user events.

The tool does not aim to substitute a fully-fledged KG platform or semantic graph server—i.e., storage, reasoning, indexing, SPARQL and general KG management are delegated to Ontotext’s GraphDB through its standards-based APIs. Thus, the tool is a model-driven editing layer over a standards-compliant KG platform, with some knowledge flow streamlining capabilities.

To conclude this section, we employ the Design Science problem template recommended by [37] to articulate the objective and nature of the contribution:

Improve KG development support (context)

...by treating it with a DSML deployed in a diagrammatic tool (GRAPHxx)(artifact)

...to satisfy a need for flexible graph editing while obscuring formalism, avoiding a schema-first approach and ensuring knowledge flow streamlining to external RDF-based systems (requirements)

...in order to facilitate low friction creation and operationalization of KG exemplars for educators, semantic SEO and GraphRAG (goals)

3 DSML design decisions and implementation details

Methodologically, DSR helped organize the overall research project, as it prescribes an iterative engineering-evaluation cycle for information systems artifacts. For the more specific tasks of DSML engineering and tool development, we resort to the framework of Agile Modeling Method Engineering (AMME) which is commonly employed in tandem with the ADOxx metamodeling platform [25]. ADOxx supports the definition and scripting of modeling method building—concrete and abstract syntax, semantics, modeling procedure support/documentation, model-driven mechanisms and interoperability. It has been traditionally used to produce modeling tools for BPMN, UML, ER, Petri Nets and others [21], a practice we are repurposing here for KG building and serialization, with semantic transparency achieved by allowing end-users to add secondary notation [38] on top of default minimalist graphics. This is a tradeoff of Moody’s principles [39] of graphical economy and semantic transparency by giving both options to the user. ADOxx also allows visual diagramming to be complemented by structured annotations and hyperlinking, to obscure details when visual cluttering occurs, thus helping satisfy requirement R2 stated in the previous section. The built-in scripting language of ADOxx enabled the transformation of diagrammatic content and the interoperability with external systems.

3.1 DSML overview

The language metamodel (Fig. 1) was tailored as a conceptual bridge between the formal RDF semantics and the DSML visual structures. It supports the user who needs to create networks of associations based on a basic understanding of several traditional constructs—things of different types, relations to other things, attributes holding data values. The abstract classes: *GRAPHxxNode*, *SchemaNode* and *ContentNode*, act as superclasses governing distinct model types for the TBOX and ABOX levels, although the tool allows them to also be mixed in the same diagram or to be hyperlinked and navigated between different diagrams.

The TBOX model type is called *Graph Schema* (which may include data points if needed), the ABOX model type is called *Graph Content* (which may include schema elements if needed). Concepts are shared between the two thus avoiding a rigid separation—the schema model type is assumed to use predominantly the *ThingType* and the *Property*, alongside the *IsOfType* (a trivialized label for *rdf:type*), the *From* (*rdfs:domain*) and the *To* (*rdfs:range*) relation classes. In *Graph Schema* there is also the class *IsAKindOf*, which acts as a trivialized version for *rdfs:subClassOf* or *rdfs:subPropertyOf*, depending on the modeling elements it connects (*ThingType* to *ThingType* or *Property* to *Property*). In the *Graph Content* diagram, we have the *ThingIdentifier* (individual), *Data* (literal), *HelperNode* (for blank nodes), alongside *RelatedTo*, *HasAttribute* and *IsOfType* relationships. We also added a *Container* that is used to store collections, ordered or unordered. The bridge between schema and content diagrams (i.e., *rdf:type* triples) can be flexibly set either as hyperlinks between nodes in the two diagram types, or as visual arrows within the same diagram, with the option of converting between the two (e.g., hyperlinks generated out of visual arrows and vice versa). We included the *isTheSame* dedicated link to specify *owl:sameAs*, *owl:equivalentClass* or *owl:equivalentProperty* equivalences if needed, according to the nature of the linked elements.

The trivialized concept naming conventions are detached from RDF jargon to satisfy requirement **R5**, potentially allowing in the future alternative graph data constructs (e.g., Neo4J's labeled property graph model uses its own jargon). We decided to add less abstract classes because ADOxx as a metamodeling environment allows us to enforce relation constraints on user events. We aim for non-technical users to grasp the modeling language as if it were a sort of mind mapping approach that gradually reveals its alignment to RDF, when the user decides to pick standard terms from OWL/RDFS (see the recurring attribute in the metamodel) for the otherwise generic visual constructs. If they are not used, the export serialization defaults to generic RDF constructs.

3.2 Modeling concepts and their annotation attributes

A series of annotation attributes are accessible to all modeling concepts as seen in Fig. 1. The user can define separate prefixes for each diagrammatic element and they will accumulate in a list for subsequent reuse. Modelers can show or hide prefixes on the modeling canvas, the toggling being implemented as a diagram-level attribute shown in Fig. 2. Concealing prefixes during the design phase can reduce visual complexity.

Table 1 provides a list of the metamodel concepts in correspondence with their graphical symbols (to be used in subsequent figures) and their RDF-specific interpretations.

The *ThingType* and *Property* classes have attributes (i.e., ADOxx annotation attributes) for defining a *Prefix* and a *Local identifier*. These two attributes form an URI, which is also available as an attribute for direct typing and as a drop-down list of standard OWL/RDF terms. Moreover, an AdoScript mechanism offers for selection the applicable prefix from a list previously set at model level—this mitigates the risk of incorrect prefix reuse.

The ADOxx annotation attributes are edited in forms shown in Fig. 3.

A built-in mechanism generates a *Local identifier* by collapsing the user-written text label. *Multi-Language Labels* can also be used to create language-tagged literals.

The annotation attribute *IsOfType* generates an inter-model link between an instance and a type available in different diagrams. It can also take the form (or be converted into) an arrow, if both elements are depicted/connected in the same diagram. A similar switch between visual arrows and tabular editing is allowed for data properties, as shown in Fig. 4 (right). By giving modelers alternatives to express relationships/properties the tool provides a way to conveniently hide or reveal details.

The combination of *File selection* and *Load image* attributes allows users to achieve semantic transparency (in the sense of Moody's theory [39]) by attaching their own domain-specific meaningful visual notation, instead of the graphically economical defaults.

The *Data* nodes contain the literal value and the type of data picked from a list of predefined XML Schema entries or linked to custom types available in a schema diagram.

The *HelperNode* supports the same treatment as URI identified nodes, except that it has only a local identifier (blank node) instead of a URI.

RelatedTo and *HasAttributes* connectors act as RDF predicates either using user provided URIs or named by controlled terms picked from OWL/RDFS, for users knowledgeable about the respective standards and jargon.

At schema level *Property* elements can have their domains and ranges defined via the *From/To* visual connectors—e.g.,

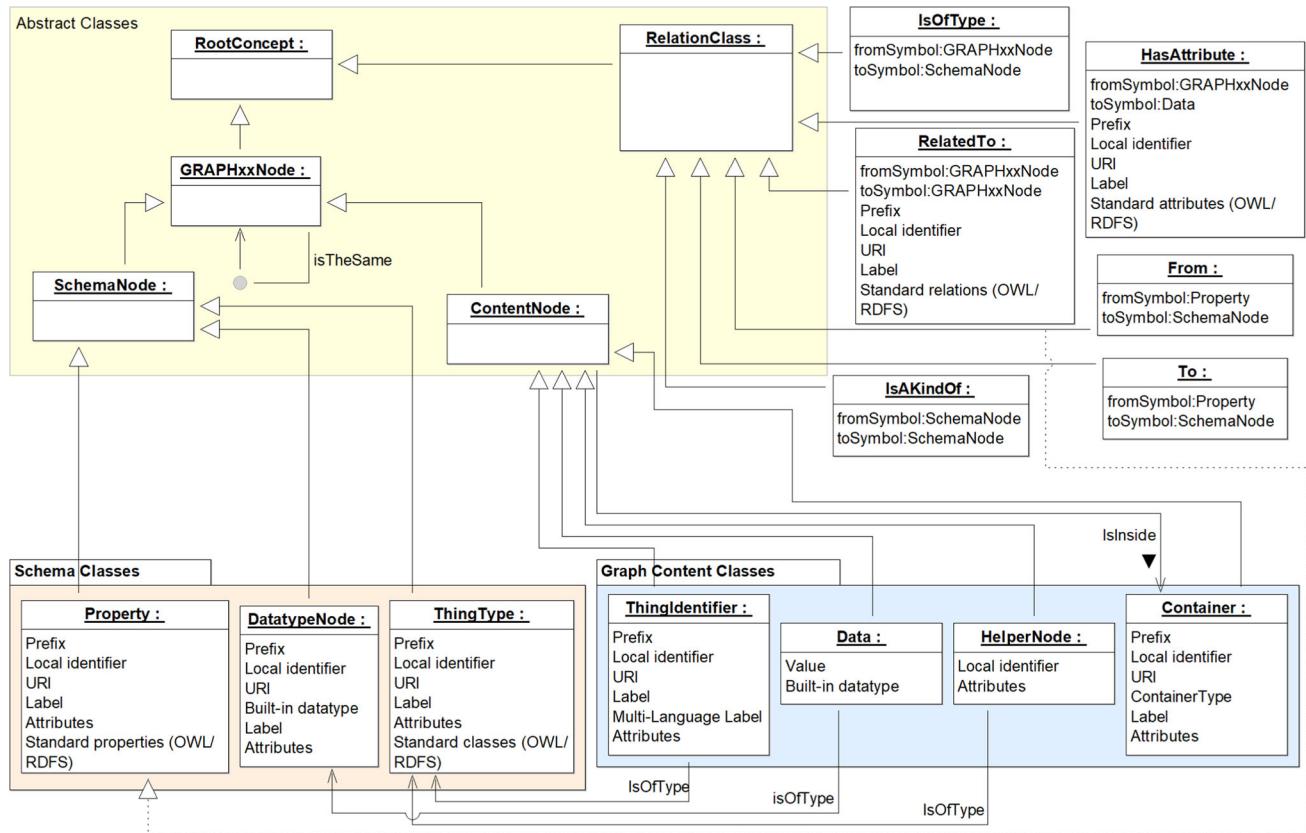


Fig. 1 Metamodel of the DSML of GRAPHxx

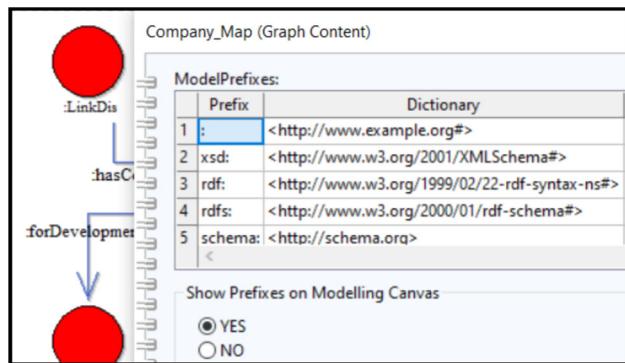


Fig. 2 Model level attributes

worksFor arrow should be drawn from a person to an organization. Taxonomies of classes or properties can be created with *isAKindOf* arrows (which will produce subclass or subproperty relationships in the serialization). Sameness can be declared for all types of constructs—between instances, types, and properties—scattered across different diagrams.

3.3 Visualization and usability mechanisms

Requirements **R1-R4** formulated in Sect. 2 are addressed by scripted user interaction mechanisms as discussed in relation to the metamodel attributes that enable them:

- **Identity management** refers to means of accumulating prefixes on diagram level, and reusing them across elements present in the diagram, thus preventing mistyping caused by their repetition. URIs are built from such prefixes and local identifiers—the latter can be either manually entered or collapsed from the free text label provided by the user. If the user neglects to tackle URIs, they are generated from the internal identifiers used by the underlying ADOxx platform;
- **Switching between tabular and diagramming views** (see Fig. 4) supports the user when deciding/converting between ABOX-TBOX bridging (*rdf:type*) relationships to be displayed either as hyperlinks or as visual connectors within the same diagram. A similar switch is enabled between data properties shown as attributes in a table or visual arrows around a node;
- **Container management** refers to the *Container* construct from the metamodel, which can be serialized either

Table 1 Modeling concepts of GRAPHxx

Modeling concept (nodes)	Graphic notation	Description
ThingType		RDF classes, can be present either in schemas or content diagrams
Property		RDF properties, when defined on the schema/ontology level
DatatypeNode		RDF datatypes, when defined on the schema/ontology level (as domains/ranges)
IsOfType		rdf:type
RelatedTo		Object properties, when used on instance level
From		The domain of RDF properties, used in schema graphs
To		The range of RDF properties, used in schema graphs
IsAKindOf		A visually unified version of rdfs:subClassOf or rdfs:subPropertyOf, interpretation depending on the modeling elements it is linking
ThingIdentifier		Individual resources, identified by an URI
HelperNode		RDF blank nodes, typically employed for grouping or n-ary patterns
Data		RDF literals
Container		RDF collections or containers, of possibly ordered elements. Order is inferred from visual position of elements on X or Y axis
HasAttribute		Data properties, as used on instance level

^a<https://search.google.com/test/rich-results>

as a group of unordered elements (grouped through rdfs:member) or as an ordered list. The order is determined from the visual positioning chosen by the user, along the X or Y axis. In addition, the container can be broken apart by reconnecting internal nodes to the subject where the container was originally connected to; or, the container can be generated around a selection of individual nodes. The appropriate RDF pattern is applied at the moment of serialization;

- **Switching between node types:** turning “things into strings”, i.e., reversing the Google KG principle of “things, not strings” [40] makes sense from SEO perspective when the Schema.org snippets (to be incorporated in HTML

pages) are intended to have certain textual terminal nodes instead of entity identifiers—when it is more important for those texts to be searchable rather than to be linked or navigated;

- **Dynamic notation** refers to balancing Moody’s [39] principles of graphical economy (minimalist default notation, distinguished mostly by color) and semantic transparency (users can individually replace the default node shapes with meaningful domain-specific graphics of their preference).

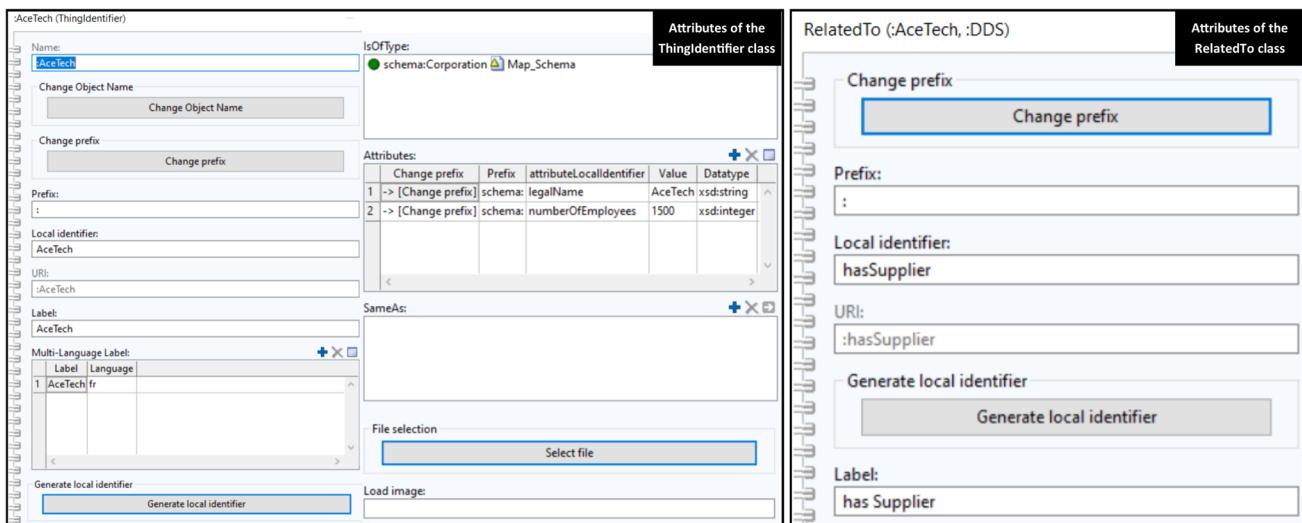


Fig. 3 Attributes of ThingIdentifier (left) and attributes of RelatedTo (right)

4 Architecture proposal for knowledge streaming

The answer to requirements **R6** and **R7** moves away from language engineering toward an architecting proposition for knowledge streamlining. This relates on a theoretical level to the Knowledge Management discipline where the Nonaka model of knowledge conversion was introduced to categorize the types of knowledge flows observed in organizations [41]. Prior proposals have discussed the role that graphical modeling can play as a means of knowledge externalization or retrieval [42], an idea that evolves in the current LLM age to a need for revisiting Nonaka’s theory, by involving AI services as “knowing subjects” in the traditional knowledge transfer modes.

The hereby introduced architecting proposal instantiates this idea by streamlining the diverse knowledge modes involved—diagrammatic, standard-formal (RDF) and natural language (LLM content). Model-driven engineering traditionally used model transformations (e.g., platform-independent to platform-specific) to generate code or automate behavior, and we repurpose this tradition to produce RDF graphs that can be ingested by RDF-reliant systems. Currently, integration is achieved by uploading graphs to the triplestore where queries and reasoning are executed, or by applying a GraphRAG prompting strategy where graph fragments are delivered to an OpenAI language model to enable natural language querying over the diagrams drawn in GRAPHxx. Figure 5 depicts the integration channels of this architecture, thus deploying a **model-driven GraphRAG experimentation** environment and testbed, that we plan to use in future to evaluate different LLMs and graph-enhanced prompting strategies.

The ingredients implemented to deploy this architecture are:

TriG/Turtle/JSON-LD serialization (Fig. 6) repurposes model-driven code generation, while also ensuring the jargon translation between the visual patterns, RDF(S)/OWL and the designated target formats. It collects all diagrammatic elements, inter-model hyperlinks (that integrate disparate ABOX and TBOX elements) and annotation attributes to build RDF files readily available for a triplestore. SEO practitioners can pick the JSON-LD serialization to incorporate in Web pages graph content for search engine optimization purposes. The serializations can either be manually picked for subsequent use or delivered as HTTP payloads to the other architectural nodes visible in Fig. 5.

Triplestore interoperability: uploading the serialized graphs to an Ontotext GraphDB server directly from the modeling environment helps users to easily shift between visually developing KGs and operationalizing them on a dedicated KG server. This feature is implemented through the SPARQL HTTP protocol [22] handled by a Python mediator script triggered from the modeling environment;

GraphRAG, as an architectural pattern with different flavors [12], facilitates natural language queries over the user’s diagrammatic content. In our deployment it is achieved in a model-driven manner through the aforementioned serialization and interoperability mechanism, extended with a scripted orchestration that leverages interfaces made available by GraphDB and LangChain [18], marked as numbered “steps” in Fig. 5. The LLM service can be changed via LangChain, making the modeling tool less dependent on a specific service. Again, Python acts as a mediator, receiving diagrammatic content from AdoScripts and orchestrating the GraphRAG pattern steps:

The figure consists of four panels:

- Top Left:** A graph visualization titled "Relationships between a legal entity and related data attributes graphically". It shows nodes like "541106", "LegalEntity", "Acetech", "Some Street, No. 1", "USD", "Design", and "Testing". Relationships are labeled with properties such as ":fulfillsContract", ":ha", ":schema:address", ":schema:legalName", ":schema:numberOfEmployees", and ":forDevelopment".
- Top Right:** A table titled "Acetech (ThingIdentifier) - Attributes" showing data attributes for the Acetech node. The table has columns: Change prefix, Prefix, attributeLocalIdentifier, Value, and Datatype. Rows include:

Change prefix	Prefix	attributeLocalIdentifier	Value	Datatype
1 -> [Change prefix]		schema:legalName	AceTech	xsd:string
2 -> [Change prefix]		schema:numberOfEmployees	1500	xsd:integer
3 -> [Change prefix]		schema:address	Some Street, No. 1	xsd:string
- Bottom Left:** A screenshot of a modelling tool's context menu for a selected node. The menu includes options like "Table", "DName...", "Cut", "Copy", "Delete", "Position/Size...", "Layer", "Lock object access", "Object references...", "New relation", and "Generate types".
- Bottom Right:** A code extract titled "Code extract from the process of parsing semantic links and generating the relationships". The code is written in AdoScript and handles operations like GET_INTERREF, IS_OPENED, and SETL commands for modelIdToEnter and typeNodeAttributes.

Fig. 4 Dual view of how relationships can be handled either visually or in a tabular format (upper side) and an AdoScript extract for class node generation (bottom)

Step 1 The repository name and the natural language user query are gathered.

Step 2 The natural language query is sent to OpenAI's *gpt-4o* model, in order to extract the subject of the query. This step leverages the LLMChain class from LangChain to send a query to the designated LLM. For obtaining the subject of the user query, a templated prompt was developed through prompt engineering: *Identify the subject of the prompt {user_input_placeholder}*. When providing the response write strictly the subject without other comments

or details. For example, if the prompt is: *Give me information about Jane Doe*, the returned result should be *Jane Doe*;

Step 3 After the subject is identified, a SPARQL query is sent to GraphDB to extract the subgraph of entities connected around the subject of the user query - a SPARQL CONSTRUCT query gathers all entities and relations up to n-th-degree neighbors, with a conveniently picked distance to obtain a “relational context” by navigating radial relationships around the subject node (a graph interpretation of “context” adopted by KG practitioners [43], in contrast to the token “context” typically mentioned for LLMs);

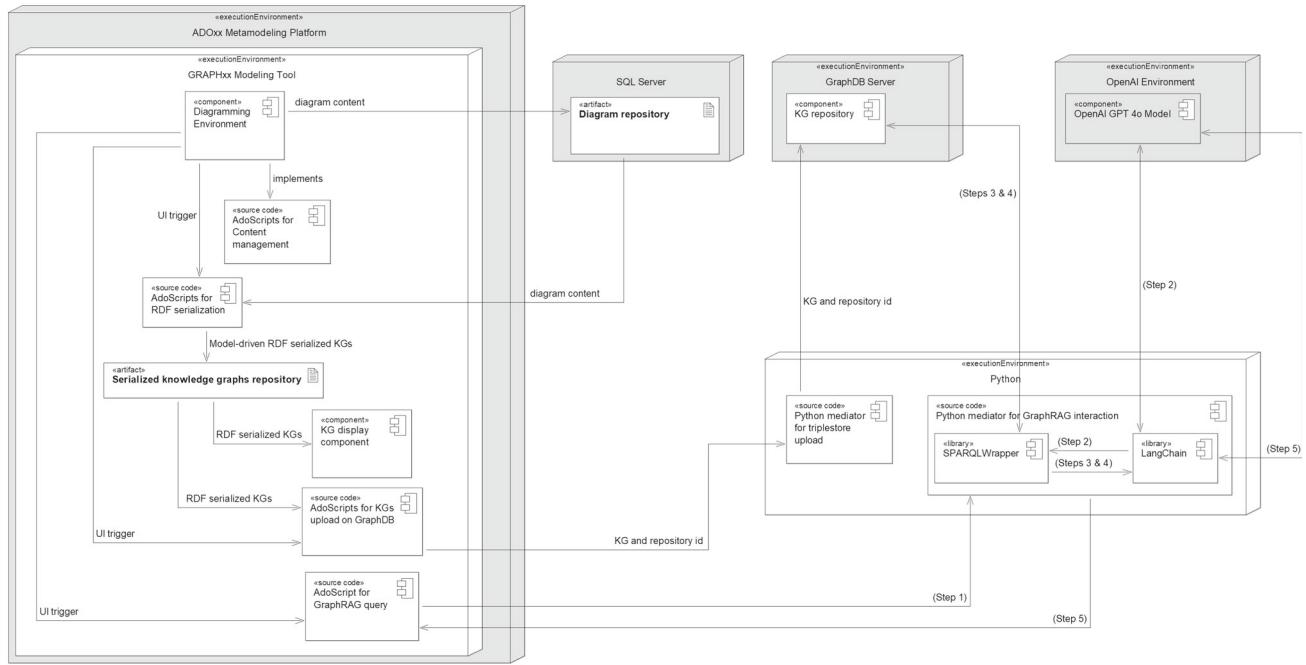
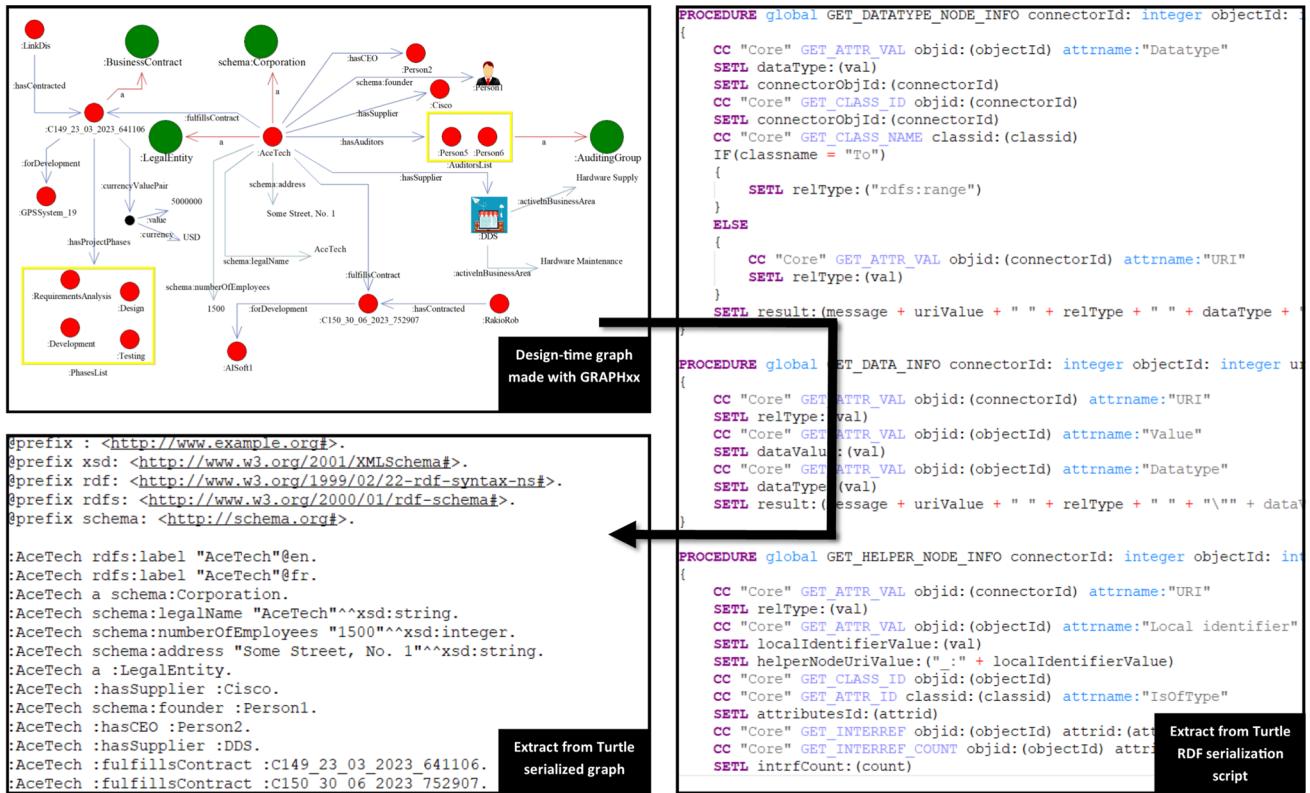
**Fig. 5** Architectural view on integration with RDF-reliant external systems**Fig. 6** Extract from the script converting diagram elements into RDF serialization

Fig. 7 Excerpt from the Python script for LLM integration via LangChain

```

sparql.setQuery(queryWithsubjectV2)
sparql.setReturnFormat(TURTLE)
graph_context_as_Turtle = sparql.queryAndConvert()
g.parse(graph_context_as_Turtle, format="turtle")
entities = [str(s) for s in g.subjects()]

if not entities:
    sparql.setQuery(query)
    sparql.setReturnFormat(TURTLE)
    graph_context_as_Turtle = sparql.queryAndConvert()
    print(graph_context_as_Turtle)

llm = ChatOpenAI(model='gpt-4-1106-preview', temperature=0)

rdfTemplate = PromptTemplate(
    input_variables = ["user_input_placeholder", "graph_context_as_Turtle"],
    template = "{user_input_placeholder} by extracting information from the RDF data in {graph_context_as_Turtle}"
)

rdfChain = LLMChain(llm = llm, prompt = rdfTemplate, verbose = True)
rdfResponse = rdfChain.run(user_input_placeholder = user_input_placeholder, graph_context_as_Turtle = graph_context_as_Turtle)

print(rdfResponse)
writeFile.write(rdfResponse)

readFile.close()
writeFile.close()

```

Step 4 In case this query didn't return anything, a second one is sent for gathering not only a subgraph but the whole knowledge graph. The results obtained either from the first SPARQL query or from the second one are sent as context to the second query executed through LangChain to the LLM;

Step 5 The prompt for the main LLM query is composed from the obtained information: *{user_input_placeholder}* by extracting information from the RDF data in *{graph_context_as_Turtle}*. The response is forwarded back to the modeling environment by the receiving Ado-Script (Fig. 7).

5 Running examples

5.1 The GraphRAG scenario

In Fig. 8 we showcase an example of mapping a business ecosystem, with the ABOX and TBOX separated into distinct diagrams. This sort of content captures private knowledge, which hasn't been accessible to LLMs for training—the traditional benefit motivating GraphRAG [13].

The scenario captures a network of associations pertaining to business contracts of a tech company in relation to business partners and other stakeholders (auditors). Some of the aspects discussed in the language design are visible—the possibility to assign types in the same diagram or from a distinct schema diagram, dynamic notation for improved semantic transparency, container-based grouping or ordering, blank node-based grouping.

Natural language questions will benefit from LLM grounding on this assumed-to-be private knowledge representation snippet. For instance, the question *Which companies would be affected in case AceTech goes bankrupt?* receives the following answer:

Based on the provided RDF data, here are the companies that have a business relationship with AceTech: D&D Solutions (DDS) is a supplier to AceTech.

Cisco Systems is a supplier to AceTech.

AceTech has been contracted by Linkstar Distribution (LinkDis) for the development of a GPS tracking system.

AceTech has been contracted by Rakio Robotics (RakioRob) for the development of AI software.

If AceTech goes bankrupt, the following companies would be affected: D&D Solutions (DDS): As a supplier to AceTech, DDS may face financial losses due to unpaid invoices or loss of future business.

Cisco Systems: Similar to DDS, Cisco may also face financial losses due to unpaid invoices or loss of future business.

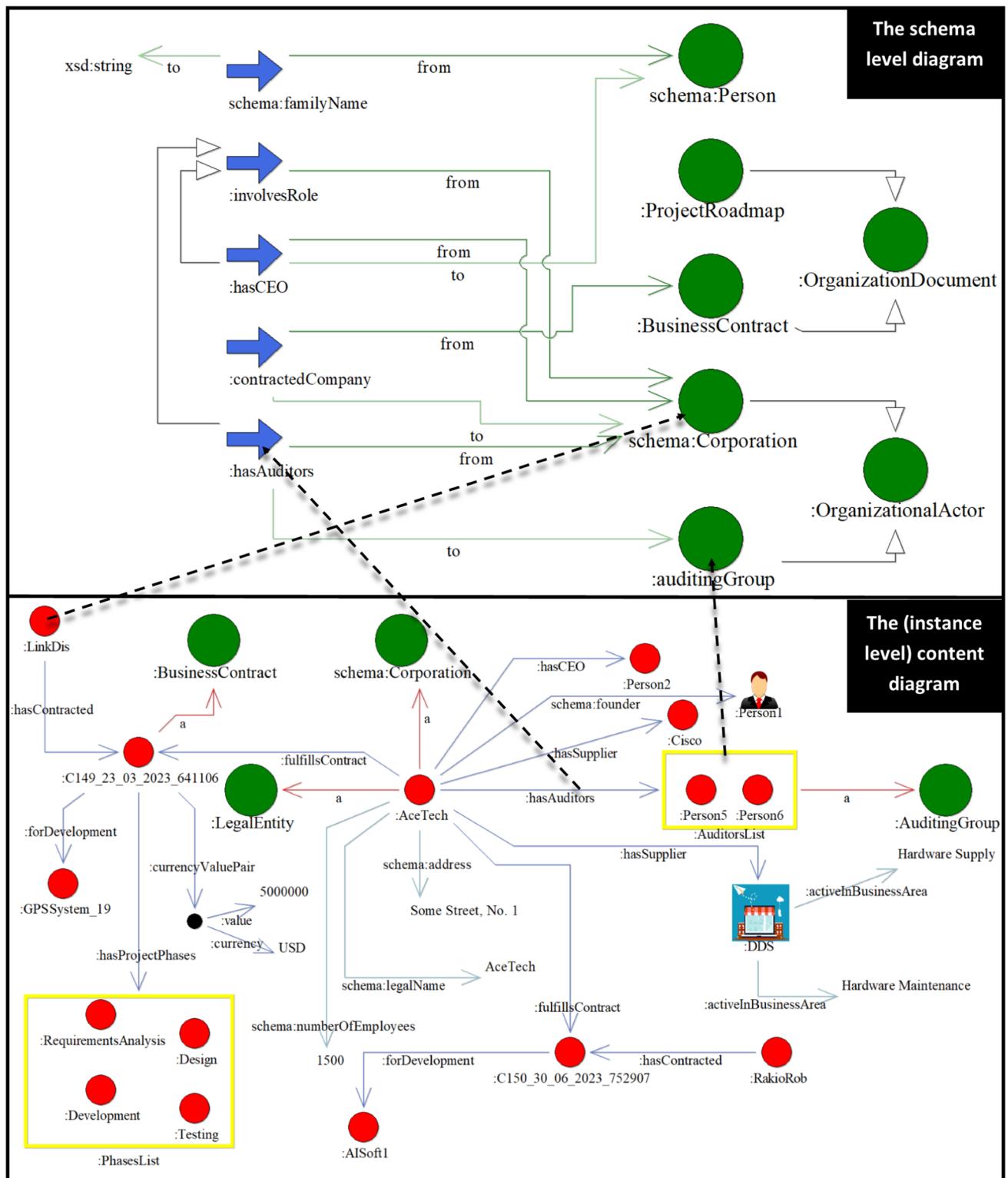
Linkstar Distribution (LinkDis): If AceTech is unable to complete the development of the GPS tracking system, LinkDis may need to find another contractor or face delays in their project.

Rakio Robotics (RakioRob): If AceTech is unable to complete the development of the AI software, RakioRob may also need to find another contractor or face delays in their project.

Additionally, any other companies or individuals with undisclosed business relationships with AceTech would also be potentially affected by its bankruptcy.

5.1.1 The semantic SEO scenario

In semantic SEO, graph snippets must be embedded in Web pages after being serialized in the JSON-LD (“JSON for Linked Data”) format. The tool allows the SEO practitioner to design the graph snippet by circumventing the error-prone JSON-LD serialization and also to decide on the nature of the terminal nodes of such a graph snippet—often, JSON-LD graphs end in entity names rather than URIs, if the SEO specialist does not intend for the graph to be open-endedly

**Fig. 8** KG exemplar split across two linked diagrams in GRAPHxx

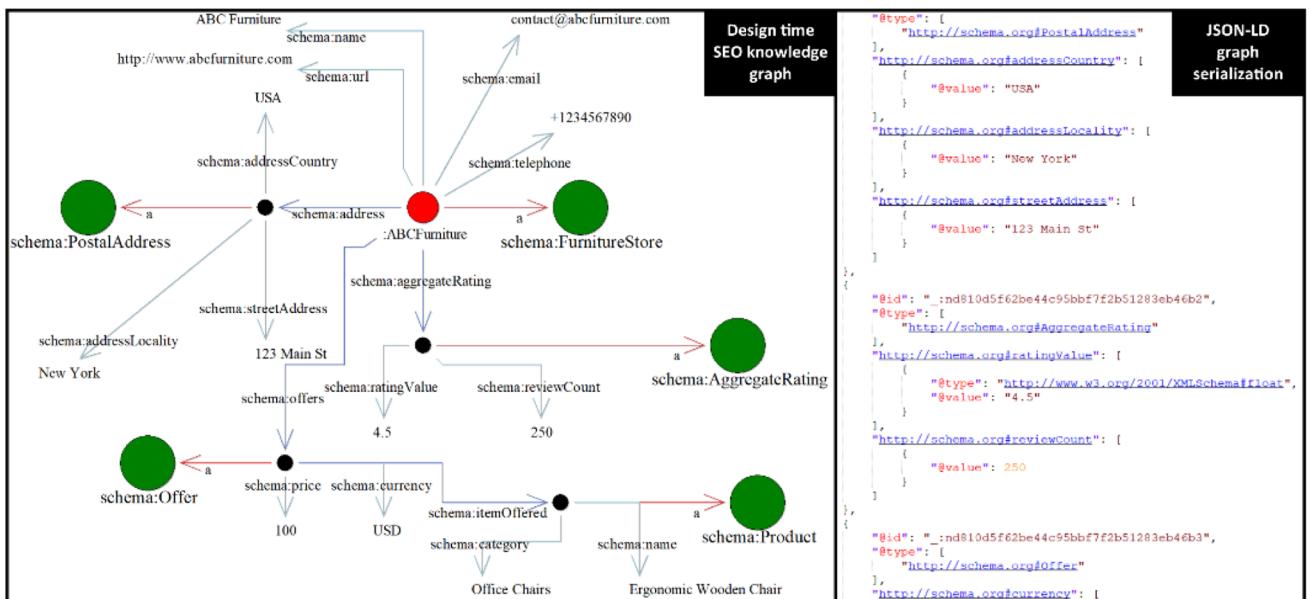


Fig. 9 Visual design of a graph snippet for SEO (left) and an excerpt from its JSON-LD serialization (right)

linkable to other graphs/resources through its terminal entities. A “content management” mechanism for URI-to-text conversion facilitates this manual “pruning” without touching the background data format, or the terminal nodes can be directly defined as terminal text elements in the model. The JSON-LD graphs thus designed are ready for both the (a) typical use case of being embedded in Web pages (communicated to Web developers in the serialized form, see Fig. 9) and (b) the less typical use cases—at least for SEO practitioners unaware of the RDF ecosystem—of uploading the JSON-LD snippet to a triplestore or exposing it the GraphRAG mechanism.

6 Evaluation approaches

The tool (as ADOxx source) and associated resources used in evaluations (content and scripts for generating synthetic content, e.g., in scalability assessments) are being maintained at <https://github.com/andrechis97/GRAPHxx>.

6.1 User-oriented evaluation

A key goal of this work is to minimize the friction required to build KG fragments for the targeted application cases. For this, we set up an evaluation protocol for user subjects that comprises several tasks pertaining to the creation of the same graph by different means: (a) modeling in GRAPHxx; (b) free diagramming with Powerpoint shapes; (c) directly typing a serialization (as required by the dominant visualization-only

tools); (d) creating the graph in the most popular tool among interviewed stakeholders, Protégé.

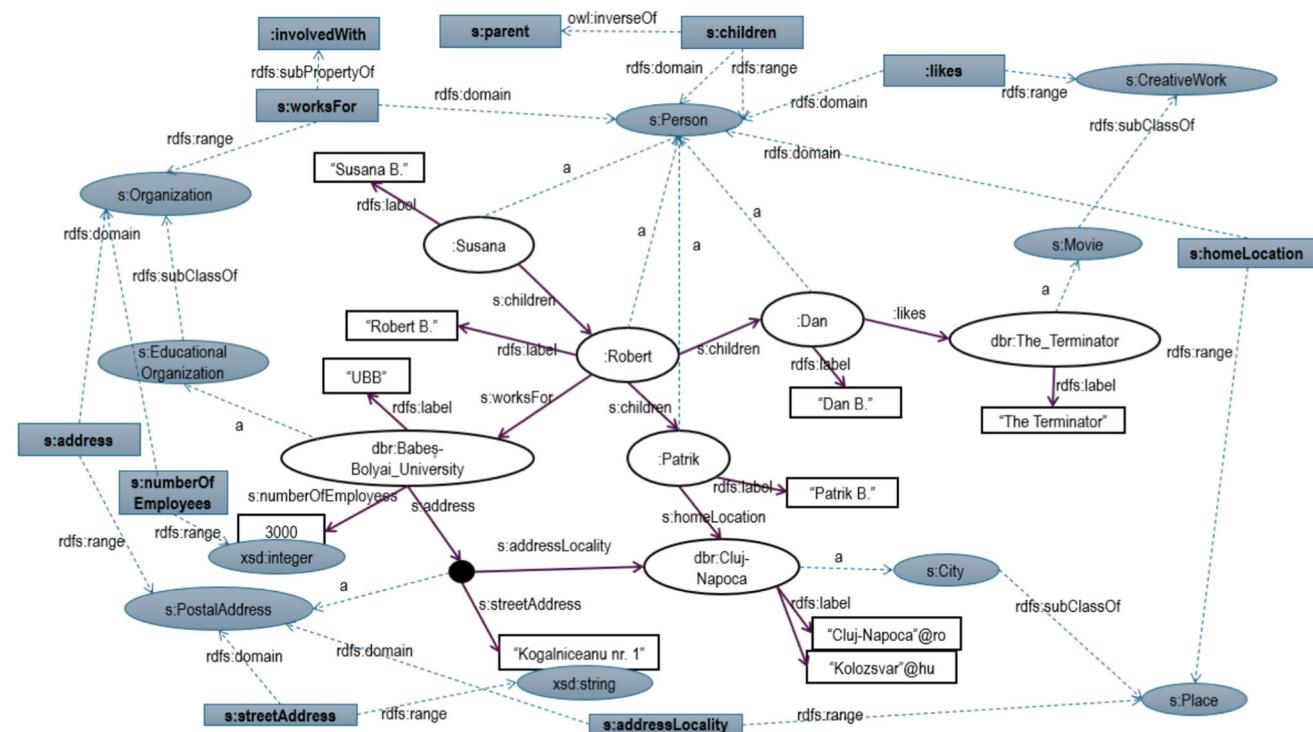
We are first measuring the time taken to create the same graph by these different means. Additionally, an *effort metric* is applied by accumulating scores of granular front-end actions during the graph creation: *simple clicks* (weight 1) for selecting elements, positions, context menu access; *control keystrokes*—copy, paste, delete (also weight 1) for multiplicating/deleting shapes; *dragging* (weight 1.5) for repositioning or resizing; *typing into fields/boxes* (weight 2).

A qualitative assessment also looks for mistakes, while giving the human subjects the possibility to give verbal feedback or to explain mistakes and confusions.

We chose Powerpoint because of general familiarity and because it provides a free-shape diagramming environment that resembles the structural freedom and diagramming experience we also intend to offer. Powerpoint and free diagramming tooling of similar usage experience are actively used by the stakeholders that raised our motivating method requirements, when they need to showcase examples before approaching any specific RDF tooling, and before delving into serialization formats (as expected by any visualizer).

The graph exemplars used in these evaluation tasks are designed to comprise a diversity of patterns commonly found in KGs. An example is given in Fig. 10 as Powerpoint representation, accompanied by equivalent narrative and Turtle representations. It reflects a mix of aspects that shift the traditional OWL focus to graph and triple patterns, to legal combinations of node and edge types:

- It contains both ABOX-level (facts and data) and TBOX (schema-level) explicit statements; it contains attributes

**Data content:**

Susana is a person who has the child Robert.
 Robert is also a person who has the children Dan and Patrik.
 Robert works for Babes-Bolyai University which is an educational organization.
 Dan likes The Terminator which is a movie.
 Patrik's home location is in Cluj Napoca, which is a city.
 The number of employees in Babes-Bolyai University of 3000; the university's postal address in Cluj Napoca is at the street address M. Kogalniceanu nr. 1.
 All entities should have both an identifier and a text label (they can be similar); one should have bi-lingual labels (e.g. Cluj Napoca known both as "Cluj Napoca" in Ro and as "Kolozsvár" in Hu)

Schema level:

An educational organization is a kind of organization.
 A movie is a kind of Creative work.
 A city is a kind of place.
 The working relationship connects a person to an organization.
 Having a children connects a person to another person.
 Having a home location connects a person to a place.
 Liking something connects a person to the creative work they like.
 Having an address connects an organization to a postal address.
 Having an address locality connects a postal address to a place.
 Having a street address connects a postal address to a text.
 Having a number of employees connects an organization to a number/integer.
 The working relationships is a kind of involvement relationship (between a person and an organization).
 Having a children is the inverse of having a parent.

Turtle representation

```

@prefix : <http://example.org#>.
@prefix s: <http://schema.org/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix dbr: <http://dbpedia.org/resource/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

:Susana a s:Person; rdfs:label "Susana B."; :children :Robert.
:Robert a s:Person; rdfs:label "Robert B.";
  :worksFor dbr:Babes-Bolyai_University; :children :Dan, :Patrik.
:Dan a s:Person; rdfs:label "Dan B."; :likes dbr:The_Terminator.
dbr:The_Terminator a s:Movie; rdfs:label "The Terminator".

:Patrik a s:Person; rdfs:label "Patrik B."; :homeLocation dbr:Cluj-Napoca.
dbr:Babes-Bolyai_University a s:EducationalOrganization; rdfs:label "UBB";
  :numberOfEmployees 3000 ;
  :address [a s:PostalAddress;
    :streetAddress "Kogalniceanu nr. 1";
    :addressLocality dbr:Cluj-Napoca].
dbr:Cluj-Napoca a s:City; rdfs:label "Cluj-Napoca"@ro,
  "Kolozsvár"@hu.

:s:Place a s:Place.

:s:Organization rdfs:subClassOf s:Organization.
:s:Movie rdfs:subClassOf s:CreativeWork.
:s:City rdfs:subClassOf s:Place.
:s:PostalAddress rdfs:subPropertyOf :involvedWith.
:s:parent owl:inverseOf s:parent.
:s:children rdfs:domain s:Person; rdfs:range s:Person.
:s:homeLocation rdfs:domain s:Person; rdfs:range s:Place.
:s:CreativeWork rdfs:domain s:Person; rdfs:range s:CreativeWork.
:s:address rdfs:domain s:Organization; rdfs:range s:PostalAddress.
:s:addressLocality rdfs:domain s:PostalAddress; rdfs:range s:Place.
:s:streetAddress rdfs:domain s:PostalAddress; rdfs:range xsd:string.
:s:numberOfEmployees rdfs:domain s:Organization; rdfs:range xsd:integer.
:s:worksFor rdfs:subPropertyOf :involvedWith.
:s:children owl:inverseOf s:parent.

```

Fig. 10 Powerpoint representation of a graph example used in evaluation tasks (top), its narrative equivalent (bottom-left) and Turtle equivalent (bottom-right)

- (i.e., data properties) and relationships (i.e., object properties);
- The TBOX contains both taxonomical axioms (subproperty, subclass) and schema-oriented axioms (domain, range);
 - It contains multi-lingual labels and literals of different data types;
 - It uses resources identified within diverse namespaces;
 - It contains relationships of higher arity relying on blank nodes.

Namespaces are not relevant to all the tasks, since the observed efforts do not include the browsing of vocabularies for controlled terminology. Wherever namespace switching is relevant, the prefixed terms are given to the user subject to eliminate the effort of terminological choices.

Prior to the evaluation tasks, a balancing of familiarity must be achieved with respect to front end and look and feel, as Powerpoint is much more known than ADOxx-based user interfaces. Therefore, a preparation phase makes the user subjects familiar with the GRAPHxx environment, the DSML terminology and the general modeling procedure (how to create a link, how to access the sheet of annotation attributes). This is achieved through minimalist exercises where each language construct is instantiated, connected and annotated, accompanied by instructions of the following manner to ensure that all menu options and annotation sheets can be easily identified during evaluation tasks:

For defining all namespaces you need to access the Model attributes option.

For making schema-level elements visible you need to use the Graph Content model type in extended mode.

Labels and data can be attached to things either through visual connectors or by filling the table accessible through double-clicking an element.

After this preparation, the evaluation tasks are deployed in the following order:

Task 1: Creation of the graph exemplar in GRAPHxx, from input given solely in natural language (Fig. 10, bottom-left). Namespaces are not imposed, as priority is given to the general structuring of the information as a graph. Total time and mistakes are assessed.

Task 2: Creation of the graph exemplar in Powerpoint, from input given as the picture of the desired outcome (Fig. 10, top), applying the exact namespaces included there. Total time and the effort metrics based on front-end interaction weights are assessed. Formatting effort is excluded, as the visual shapes are templated in advance, to be reused similarly to a DSML;

Task 3: Creation of the graph exemplar in GRAPHxx, from input given as a picture of the same Powerpoint representation as in the previous task, again applying the exact namespaces, to involve the prefix switching effort. The corresponding total time and effort metrics are assessed accordingly. The comparison between Tasks 2 and 3 focuses solely on visual editing effort and friction, to identify any opportunities to minimize them—see at the end of this subsection backlog tasks arising from this analysis;

Task 4: Creation of the graph exemplar by direct manual typing the Turtle equivalent of the visual Powerpoint exemplar. Total time and mistakes are assessed—both syntactic and structural, to highlight the guard-railing value of using of DSML that prevents mistakes through mechanisms enabled by the underlying ADOxx platform—constraints derived from the metamodel configuration or explicitly scripted;

Task 5: Creation of the graph exemplar by manually typing the JSON-LD equivalent of the visual Powerpoint exemplar. Direct conversion from Turtle is not allowed, but any manipulation and editing of snippets available in the Schema.org website is allowed, since that is also the general practice of SEO experts who are solely interested in the SEO use case and not in mastering the RDF ecosystem. Again, total time and mistakes are assessed.

Task 6: Creation of the same graph exemplar in the form-based interface of Protégé 5.6.4, from the same visual Powerpoint example, again applying the exact namespaces to involve the prefix management effort.

In the most recent deployment of this evaluation protocol, it was applied on 7 educators—4 having first-hand experience with semantic RDF graphs and 3 having no prior experience with RDF but being familiar with both mind mapping and formal diagramming (BPMN, ER). Tasks 4–6 could only be performed by the first category, as they required to be knowledgeable about the Turtle and JSON-LD formats. Results are averaged in Table 2.

This kind of evaluation is performed iteratively in line with the Design Science cycle, to reflect on the prior iteration and to inform the subsequent implementation tasks. We exemplify past findings that turned into current design decisions, as well as current findings added to the backlog as next iteration implementation tasks:

• **Past findings turned into current design decisions:**

Ensuring that the naming of the DSML concepts avoids as much as possible the RDFS/OWL jargon, while the serialization produces machine-readable RDF, by repurposing the platform-independent-to-platform-dependent approach of model-driven engineering. This is reflected in the last version of the metamodel as depicted in this paper, based on interactions with stakeholders. For example, the taxonomical relationship *isAKindOf* evolved away from *rdfs:subClassOf* through several intermediate

Table 2 Evaluation findings summary

Task	Average time (seconds)	Average effort (the effort metric)	Average mistake count	Insights
1 Knowledgeable about RDF: 1852 Not knowledgeable about RDF: 2613	Knowledgeable about RDF: 358 Not knowledgeable about RDF: 435	Knowledgeable about RDF: 358 Not knowledgeable about RDF: 6.33	Knowledgeable about RDF: 1.25 Not knowledgeable about RDF: 6.33	There seems to be a gap given by knowledge structuring expertise (mistakes on how to use blank nodes, how to express some axioms, delays with consulting the modeling procedure and some retries—deletion and recreation of elements due to lack of confidence). However, the avoidance of RDFS/OWL jargon allowed all user subjects to focus on expressing navigable relations that mimic the natural language grammar of the original statements
2 Knowledgeable about RDF: 2123 Not knowledgeable about RDF: 2401	Knowledgeable about RDF: 484 Not knowledgeable about RDF: 498	Knowledgeable about RDF: 0 Not knowledgeable about RDF: 0	Having all the constructs visually defined for reproduction closed some of the experience gap. In terms of look and feel, both categories relied on their Powerpoint familiarity. Here we are interested in the effort difference between GRAPHxx and Powerpoint—the latter being similar to free-shape diagramming environments like yED [44] and draw.io	
3 Knowledgeable about RDF: 1112 Not knowledgeable about RDF: 1404	Knowledgeable about RDF: 370 Not knowledgeable about RDF: 442	Knowledgeable about RDF: 0 Not knowledgeable about RDF: 1.33	Diagramming canvas reduces the effort of labeling, sizing, positioning and connecting compared to the free-shape diagramming experience of Task 2. See below the table additional qualitative feedback insights for further improvement tasks to further widen the gap against the Powerpoint/free-shape diagramming friction	
4 Knowledgeable users only: 1034	Knowledgeable users only: 2.5	-	Direct typing in a text editor for the knowledgeable subjects took slightly shorter than diagramming. However, metamodel-level constraints prevent not only syntactic errors (which are also spotted by the typical code editors) but also structural ones—i.e., illegal triple patterns (e.g., wrong direction of rdf:type between an instance and a type, triples having a literal as a subject, subclassing between things that are not types). Diagramming also makes the knowledge structure visually navigable, in comparison with the more technical Turtle view	

Table 2 (continued)

Task	Average time (seconds)	Average effort (the effort metric)	Average mistake count	Insights
5	Knowledgeable users only: 2933	-	Knowledgeable users only: 6	Again, metamodel-based constraints prevent both syntactic and structural errors, this being a much more challenging format for manual text-based manipulation even for those familiar with the format. JSON-LD being also less human-readable, the diagram is a visual aid for the structure to be designed and communicated in a SEO scope. A shortcoming of GRAPHxx is the lack of terminological validation with respect to the Schema.org vocabulary, which is available in the standard Google validator familiar to SEO practitioners. It remains to be decided whether GRAPHxx should also replace the Google validator ^a , although that tool will never be circumvented by a practitioner as a last validation authority—moreover, the validation recommended by Google/Schema.org is evolving (much faster than standards like OWL/RDFS).
6	Knowledgeable users only: 1468	Knowledgeable users only: 350	Knowledgeable users only: 0	Slightly higher time in Protege compared to Task 2 but also slightly lower effort metric, with some overhead caused by (a) the need to first create all properties into the object property and data property hierarchies and (b) weak support for defining blank node patterns. Dragging effort is generally unnecessary due to the form-based interface (this lowered the effort metric value), whereas frequent tab/window switching and the aforementioned overhead of OWL-first definitions added to the time

variants including *subclass*, *subtype*, *is-a*, *generalizedBy*, *hasMoreGeneralVariant* and others, currently also covering *rdfs:subPropertyOf*—to be detected at serialization time from how it was used in the diagram. The OWL/RDFS jargon is mostly hidden (except for the prefix/namespace notion) in the sense that users will not find them as first-class modeling citizens, nor as presentation cues, although at serialization time all constructs will default to some standards RDF terms. At the same time, the more knowledgeable users can pick OWL/RDFS constructs from drop-downs present in the DSML constructs (see the „Standard OWL/RDFS” attribute in the metamodel).

- Ensuring that the partitioning of the metamodel is not too rigid—to allow, but not enforce, the separation between the TBOX and ABOX levels. Especially with educators, there is a key requirement to be flexible about the nature of the KG fragment that can be quickly showcased—sometimes a small schema with some instances exemplified on it, sometimes a few connected instances with a schema fragment, sometimes complete separation and navigation between the different levels (through hyperlinks parsed at serialization time). Moving away from a rigid earlier implementation, now the separation into distinct model types, as shown by the metamodel, is „softer”—each model type exposes its concepts to an “extended mode” of the other model type, a concept partitioning mechanism available in the ADOxx metamodeling platform to allow layered DSMLs.

Current findings to be turned into future development

tasks: A major source of usage friction remains the occasional need to multiply/reuse relevant graph data patterns: *reuse of the same property for the same node* (e.g., subclassing in a taxonomy), *replication of a triple* (for subsequent editing of some its elements), *replication of a blank node pattern* (i.e., as a reusable a data structure or n-ary pattern), and *replication of ABOX-TBOX bridge patterns* (e.g., a class together with all its instances). Currently, the tool employs the default copy-pasting approach to multiplication through a granular mechanism inherited from the ADOxx metamodeling platform, not very different from how shapes are copy-pasted in Powerpoint. By leveraging the AdoScript’s event-based capabilities, single-click selection options (in context menu) are being implemented for: *selecting the full triple of the current connector*—a minor drop in an effort of high recurrence; *selecting all triples involving the current blank node*—the drop in effort varies with the number of nodes involved in the data structure expressed by the blank node; *all instances of the current type*—a major drop in effort when instances are hyperlinked across different diagrams.

Such multi-item selection options will objectively improve the effort metric compared to both form-based interfaces, e.g., Protégé and free diagramming.

6.2 Requirements and feature-oriented evaluation

Regarding the tool feature assessment, requirements R1–R7 have been referenced throughout this paper by the specific mechanisms implemented for each of them—**R5** in Sect. 3.1, **R1-4** in Sect. 3.3, **R6-7** in Sect. 4.

Alternative tools have been surveyed for their feature support and adequacy in relation to these requirements. We only checked open use free tools and not commercial solutions, since GRAPHxx is also intended to be made available through an open community, as it is a project of the OMI-LAB¹² Community of Practice [45]. Our findings regarding the surveyed tools are as follows:

- A family of **tools** (including Bee-Up [20] and BPMN2KG [46]) **convert diagrams** into KGs structured according to a vocabulary derived from a metamodel that was pre-designed for a target application domain and specificity: BPMN and EPC for business process descriptions, DMN for decision requirements descriptions, and behavioral models such as Petri Nets. An exception is [47] which works on the generic abstraction level of UML class diagrams, however aiming for the LPG graph data model of Neo4J, not targeting use cases such as SEO, nor direct involvement in GraphRAG pipelines;
- Another family of tools are the **OWL ontology editors**—including Protégé [8], OWLGrEd¹³ (UML-based interaction) [48] and Gra.fo [17] (visual interaction based on WebVOWL [49]). Being OWL-centric, they take a schema-first approach and most of them expose upfront the OWL jargon/notions even to perform the most basic editing. An exception is Gra.fo, which recognized the requirement of pushing standard-based jargon in the background whenever possible. Ultimately, these are all ontology editors, whereas our goal is to have a flexible tool that can be used as a data graph editor, a graph schema editor or any convenient combinations of both. None of the referenced tools have in scope our architecting proposition for knowledge streamlining, which becomes a prominent requirement in graph-based AI systems;
- A loosely related family of tools are the **RDF visualizers** which provide a visual exploring experience, but they rely on the partial or complete creation of the graph by non-graphical means. These include Protégé plug-ins (OntoViz, OntoGraf), isSemantic [7], RDFGrapher [50]

¹² <http://omilab.org>

¹³ At the time of writing appears to be discontinued.

with many visualizers being present as “nice to have” features in commercial tools (e.g., Ontotext’s Visual Graph [51]);

- A large selection of **generic graph building** SDKs and libraries are widely available (yED [44] as a tool, or VisJS [52] as a programming library), however not targeting specifically semantic graph standards but rather generic graph formats (GraphML, Graphviz) that preserve layout and visual appearance rather than a graph data model and semantic patterns. Needless to say, they do not pursue goals related to the exploitation of diagrammatic knowledge structures in SEO or AI contexts;
- We mentioned the previous family because, despite their genericity, they are **enablers for further customization** (on graphical and labeling level) that can bring them closer to our scope—a relevant RDF/OWL-oriented customization is Graffo [53] which defined visual editing palettes for yED and draw.io that provides the frictionless diagramming freedom that we also aim for. The scope is, however, limited to presentation and does not extend to RDF production and streamlining. In terms of content management, they do not exploit hyperlinking and traceable modularization across multiple diagrams while preserving the graph integrity.

The use of a DSML converges the diagramming capabilities and learned lessons from graphical modeling languages, the metamodel-based management and constraining of the produced content, and the model transformations and interchange necessary to preserve structure and semantics, not only layout information, when interoperating with external systems.

In OWL-first editors, instances and instance-level facts cannot be created unless a governing, even rudimentary, ontology fragment is first created—which propagates to the need to explain OWL/RDFS constructs (at the very least, an introductory lecture on: what are taxonomies and their reasoning rationale, how is an object property different from a data property, and to master the associated jargon) before the simplest example can be articulated. Instead, we aim to follow the „implicit schema” principle that ensured the popularity of commercial graph databases (e.g., Neo4J) because it allowed users to assume availability of data detached from the data model and to pragmatically deal with instance-level graph structures and their navigability. The schema concerns are stepping in later, as a semantic enrichment layer introduced to expand the querying and reasoning capabilities, not as a prerequisite for the most basic presentation and ingestion.

6.3 Competency questions

Rudimentary inferences and queries can be emulated directly in the graph-oriented AQL query language of ADOxx [54], which automatically adapts to the deployed metamodel. Thus, AQL can be seen as a subset of SPARQL, particularly supporting property paths that may include both visual connectors and inter-model hyperlinks to ensure traceability across many inter-connected diagrams. The results of a query that emulates the rdfs:subClassOf type inferencing are shown in Fig. 11.

6.4 Scalability evaluation

In our effort to assess GRAPHxx’s adequacy we also performed a quantitative evaluation of the performance of the modeling tool when dealing with complex graph content. The reason was a legitimate concern that the overhead of using a metamodeling platform, instead of implementing from scratch, might impose limitations on the creation of reasonably realistic graph datasets and ontologies, beyond the minimalist snippets of didactic value that were the starting motivation.

We used ADOxx’s scripting language to automatically populate diagrams in GRAPHxx with synthetic content. Although the tool is not intended to visualize real graphs of millions of triples, we aim to use it for reasonably large exemplars in the range of thousands.

The initial experiment involved creating 5000 ThingIdentifiers without relationships or attributes beyond Local identifier, URI, label and a prefix. This tested GRAPHxx’s ability to manage diagrams with an extensive number of nodes. Our findings, illustrated in Fig. 12 (on the left), indicated that the tool can efficiently handle up to 5000 modeling objects per diagram. Notably, there was a significant increase in diagram population times beyond 3000 objects, while the RDF export time grew more gradually.

A second test involved generating up to 1500 ThingIdentifier nodes per diagram, each connected to two RelatedTo connectors. This setup, totaling a maximum of 4500 concepts per diagram, also included only prefixes, Local identifiers, URIs and labels for ThingIdentifier nodes and RelatedTo connectors. As shown in Fig. 12 (in the middle), a steady rise was observed in the time required for diagram population beyond 1000 nodes (and 2000 connectors), taking 1680.094 s for diagram creation and 1.312 s for RDF export for a diagram with 1500 nodes and 3000 connectors. Similar to the first experiment, the time for RDF serialization increased slower than the time for diagram population.

In our third experiment, we generated up to 1500 ThingIdentifier nodes, each enhanced with 30 additional attributes, adding to the existing Local identifier, URI, label and prefix. This is depicted in Fig. 12 (on the right), where the pattern

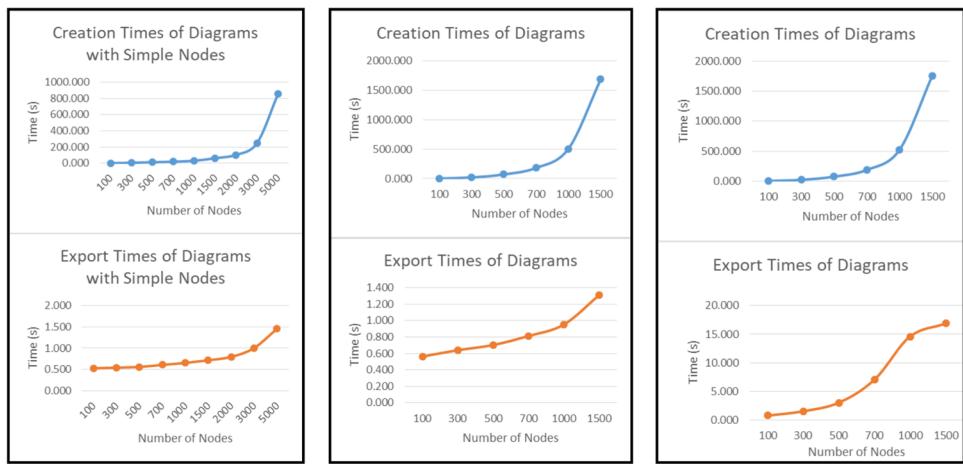
Fig. 11 The AQL query (on the left) and its results (on the right)

Query: Show all the types of the instance with URI “`AceTech`” – both the direct types attached by a visual connector or by a hyperlink.

Code: (`<"ThingIdentifier">[?"Local identifier"="AceTech"]->"IsOfType"`) OR (`<"ThingIdentifier">[?"Local identifier"="AceTech"] --> "IsOfType"`)

	Model	Name
1. Company Map		:legalEntity
Map Schema		schema:Company

Fig. 12 Quantitative evaluation results in terms of creation and export times



mirrors the second experiment, but with a notable increase in the time needed to export diagrams as RDF graphs. This increase is attributed to the growth from 4 to 34 attributes per node, leading to an 11-fold expansion in the number of generated statements. Despite this, the time increase was only 3%, indicating favorable scalability.

The system configuration for this evaluation task was: RAM Memory: 16 GB;

Processor: Intel Core i7-8750H @ 2.20 GHz (12 CPUs); Operating System: Windows 10 Home 64-bit; ADOxx version 1.5.

6.5 Validity limitations

The ongoing engineering-evaluation cycle prescribed by Design Science expands the scope of the evaluation tasks with every iteration, gradually adding dimensions of model quality as outlined by quality frameworks [55]. With respect to the evaluation outcomes hereby reported, we highlight several limitations with respect to validity:

- The human subjects involved in the modeling tasks (Sect. 6.1) had either diagramming experience or RDF experience. We did not assess a complete learning curve yet—i.e., for users completely lacking any experience—which would require a longitudinal tracking of student evolution going through a knowledge engineering and management course and using the tool in parallel with more traditional ones. Such an assessment would also need to distinguish educational factors, not only the tool or DSML quality. We do plan to involve the tool in our own

semantic technology courses to have a testbed closer to realistic adoption scenarios, but that remains to be reported by purely empirical longitudinal studies;

- In the same vein, industry-level workshops are planned for practitioners of weakly structured knowledge capture methods (design thinking, mind mapping), to identify the skill gaps and requirements toward attaining the desired status of “citizen-oriented KG development”—for this, the tool needs to reach a higher Technological Readiness Level to be easily shared and deployed for a large user base;
- A full benchmarking capability requires a large diversity of KG exemplars of varying categories of complexity, provided as input for effort measurements. In our assessments, only a few examples recombining the patterns captured in Fig. 10 were used. The literature also suffers from a shortage of such diagramming task specifications, as most KG benchmarking are concerned with radically different aspects (e.g., performance of RDF engines, question answering), and not really with the RDF diagramming experience—we take it as a future goal to develop such a benchmarking corpus on which RDF editing effort can be assessed regardless of the interface type;
- While the effort metric is for now adequate, computed from measurable components from weights assigned to user interface actions (decomposed as described in Sect. 6.1), a more precise “mistake” metric can also be developed—not just to count, but also distinguish different classes of mistakes (syntactic, semantic, tool-induced) correlated with user knowledge and experience.

In other validity aspects, we've already refined the evaluation task specifications to improve precision and objectivity: (i) averaging the effort over multiple repetitions of the task (although this incorporates some improvement due to increasing familiarity, it is similarly applicable to any comparable tools); (ii) devising RDF content exemplars that involve diverse patterns, not only the basic “array of triples” view that recent data-centric research tends to take on KGs; (iii) reporting design decisions in ways that are generalizable by transfer to alternative metamodeling platforms; (iv) increasing Technological Readiness Level by prototyping on platforms that are viable for commercial projects (ADOxx for the modeling tool, an enterprise license of GraphDB as triplestore)—however, the project remains in the scope of academic experimentation to be promoted through conference tutorials and open-source communities.

7 Related works

A growing interest is manifesting in bringing KGs to low friction visual environments, although the field is still dominated by visualizers of content created by other means.

Low-code development platforms become a focal point within educational institutions. Charles et al. [56] analyzed teaching implications highlighting the potential of such platforms to democratize system development by people with limited coding skills but with extensive domain knowledge. Our work is particularly meaningful as a possible tool for domain knowledge capture through a “low-code” approach, lowering the entry barrier for capturing formal knowledge structures needed by knowledge-based systems.

Daniel et al. [57] proposed the UMLtoGraphDB framework which can convert conceptual schemas expressed as UML diagrams to graph equivalents. The paper of Bachhofner et al. [46] reports on the development of BPMN2KG, which is a transformation tool that serializes BPMN models into query-able RDF graphs.

Within the ontology engineering domain, many commercial tools are available to facilitate the development of ontologies and taxonomies, showing an interest in user-friendly low friction interfaces for creating the knowledge content, e.g., TopBraid Composer [58], Gruff for AllegroGraph [59], PoolParty [60]—we do not intend to compete with commercial platforms but to provide a reusable metamodeling design and architecting proposition that can also be incorporated in such platforms.

The work in [61] introduced GRAPHOL, which acts as a visual notation for defining OWL 2 ontologies—GRAPHOL presents similarities with UML and maps with description logics (DL). It is supported by Eddy [62], which provides the design environment for ontologies. Chowlk [63] proposes a converter for transforming UML diagrams developed on

diagrams.net into OWL ontologies, subsequently generating associated OWL code. Graffoo, presented in [64], used in conjunction with yEd [44], also provides means for designing OWL ontologies in a graphical environment, making it a suitable approach for presentations that do not need to delve into technical details regarding Turtle or Manchester serializations.

GRAPHxx takes the DSML and model-driven engineering route to empower users to define schemas and graphs directly on a visual canvas, proposing a step in the direction of “citizen KG development” and an approach to streamlining citizen or personal knowledge for GraphRAG installations.

Ontological modeling and analysis in the context of systems engineering was explored through the openCAESAR project [65], which contributed to the development of OML (Ontological Modeling Language) [66, 67]—a textual grammar language to create and manage ontologies for systems’ descriptions.

OntoCASE4G-OWL [68] is an Eclipse-based prototype modeling and editing KGs in a graphical way relying on G-OWL [69], a visual language that also aligns to the OWL-centric dominant approach although it provides important considerations on the DSML-OWL convergence especially through the lens of Moody’s theory [39] on visual notations. The tool employed EMF (Eclipse Modeling Framework) [70] and the Sirius Framework [26] for the metamodeling and visual syntax deployment as an Eclipse plug-in.

The work presented in [71] describes SURGE, a framework designed to generate relevant knowledge by retrieving appropriate subgraphs and utilizing them as query context when providing responses. Then, contrastive learning is utilized to ensure that the generated content is related to the extracted subgraph. Our approach also extracts subgraphs in the RAG process, but GRAPHxx is more focused on how to capture and streamline knowledge structures rather than on implementing the RAG process itself.

We also highlight the potential convergence of this work with the emerging interest for Model-based Digital Twins, particularly in the context of the MODELS conference. Digital Twins also require flexible means for capturing representations of real-world elements and their interactions, from stakeholders lacking technology-specific expertise. Authors in [19] pointed to the lack of systematic methods to facilitate the modeling of Digital Twins and the requirement to consider the integration of these models with the observed cyber-physical world. Our approach takes a subsequent step in that direction by enabling a knowledge flow streamlining several modalities—from visual, through formal, to LLMs.

In terms of evaluation strategies, the literature reports many frameworks for DSML evaluation, from those with general applicability (although often discussed for a specific family of models, like SEQUAL for process models in [55]) to more specific ones (e.g., for multi-agent systems modeling

[72–74]). The evaluation strategy we reported in Sect. 6 is driven by pragmatic needs, concerns related to user adoption and technological integration: (i) feature availability based on tool requirements (that inform not only the architectural deployment, but also some of the language design decisions); (ii) technical quality and scalability, to allow the tool to be used for more than minimalist toy examples; and (iii) user experience to bring a measurable benefit against traditional ways of building of RDF graphs (graphical-only or form-based). The reported evaluation tasks can be mapped on both the step-wise protocols (from scoping to DSML refinement) and the criteria established in the mentioned quality frameworks (usability, expressiveness) including concerns related to the model-driven environment and the so-called execution dimension (scalability, GraphRAG integration).

8 SWOT conclusions and future work

KGs rely on a flavor of conceptual modeling that places focus on machine-readability and reasoning rather than visual communication, but nothing says that the latter should be excluded. Visual tools may prove in the long run to be an enabler for a citizen KG approach, e.g., a key requirement for the personal KG vision of, for instance, the Solid project [75] or the Obsidian project [76].

This paper contributes a design and architecting proposition (in the spirit of DSR) of integrating model-driven engineering with LLMs using RDF serializations as a mediator. This is achieved by the engineering of a DSML deployed as an ADOxx tool for visually designing and managing semantic graphs, extended with features acting as enablers for GraphRAG or SEO-oriented structured data markup. Below, a SWOT analysis of the presented approach is provided as a concluding summary.

Strengths: The language engineering and architecting proposal are tightly interwoven as the proposal introduces a DSML for capturing knowledge structures for both the human benefit (as a visual communication and productivity tool) and for the benefit of external systems that must ingest such knowledge structures—such concerns are traditionally supported by separate toolkits. The proposed artifact can democratize KG building as a design activity, with minimal awareness of RDF-specific jargon. It repurposes the look and feel of diagrammatic conceptual modeling successfully employed in the past for UML, BPMN, etc., while supporting GraphRAG and Semantic SEO by employing RDF as a mediator format.

Weaknesses: The tool doesn't offer yet support for LPG—although it obscures (most) RDF jargon, it is still fundamentally aligned to the RDF constructs and only interoperates with the Ontotext GraphDB triplestore, for storage and for forwarding natural language queries to LLM services.

Opportunities inform future work iterations: abstracting away from RDF to equally support Neo4J's LPG approach and adding corresponding interoperability; further linking to other modeling grammars available for ADOxx, the space of possibilities being the extensive catalogue of modeling tools in the OMILAB book series [29]. The architecting proposition and mechanisms for streamlining the diagrammatic designs establish an experimentation platform for future work on evaluating comparable alternatives, in terms of the choice of LLMs, prompting strategies and subgraph patterns exposed as RAG context. Finally, the KG completion task—recently advocated as LLM-based stochastic means of building knowledge graphs [35]—could generate visual content into GRAPHxx, to be visually refined and cleaned up, thus leading to a KG-specific form of roundtrip engineering (currently the LLM is only at the receiving end of the model-based knowledge transfer).

Threats: The evolution of graph database-related formats can take new turns. Currently, RDF and LPG are the most widely used but we can't exclude the possibility of having a new standard in future that would require redesigning GRAPHxx—the recent proposal of GQL [77] is such a candidate and has the potential to inform us about higher KG representation and management abstractions.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G.D., Gutierrez, C., Zimmermann, A.: Knowledge graphs. ACM Comput. Surv. Comput. Surv. **54**(4), 1–37 (2021)
- Buchmann, R., Ghiran, A.M.: Teaching knowledge graphs: A journey from logic to Web development and semantics-driven engineering. In: The SCME23: 7th symposium on conceptual modeling education @ ER 2023, vol. 3618, CEUR-WS (2023). https://ceur-ws.org/Vol-3618/scme_paper_4.pdf.
- Herm, L.V., Janiesch, C., Helm, A., Imgrund, F., Hofmann, A., Winkelmann, A.: A framework for implementing robotic process automation projects. Inform. Syst. e-Bus. Manag. **21**(1), 1–35 (2023). <https://doi.org/10.1007/s10257-022-00553-8>
- Wieringa, R.J.: Design science methodology for information systems and software engineering. Springer (2014). <https://doi.org/10.1007/978-3-662-43839-8>

5. Bock, A.C., Frank, U.: Low-Code Platform. Business & Information Systems Engineering 63, 733–740. Springer (2021). <https://doi.org/10.1007/s12599-021-00726-8>.
6. Ontotext, GraphDB—Create custom graph view your RDF Data, last Accessed 2025/01/17. <https://graphdb.ontotext.com/documentation/10.0/devhub/custom-graph-views.html?highlight=visual%20graph>
7. isSemantic RDF graph visualizer, <https://issemantic.net/rdf-visualizer>, last accessed 2025/01/17.
8. Protégé Ontology Editor, <https://protege.stanford.edu/>, last accessed 2025/01/17.
9. Ignite Visibility, JSON-LD SEO: Why JSON-LD schema is crucial for SEO, <https://ignitevisibility.com/everything-to-know-about-json-ld-for-seo/>. Accessed 17 Jan 2025
10. Google Search Central, Introduction to structured data markup in google search, <https://developers.google.com/search/docs/appearance/structured-data/intro-structured-data> (2025). Accessed 12 Jan 2025
11. W3C Recommendation. JSON-LD 1.1, <https://www.w3.org/TR/json-ld11/>, Accessed 17 Jan 2025
12. Ontotext, What is graph RAG?, <https://www.ontotext.com/knowledgehub/fundamentals/what-is-graph-rag/>, Accessed 17 Jan 2025.
13. Microsoft Corporation, GraphRAG: Unlocking LLM discovery on narrative private data, <https://www.microsoft.com/en-us/research/blog/graphrag-unlocking-lm-discovery-on-narrative-private-data/>, Accessed 17 Jan 2025.
14. LlamaIndex, LlamaIndex documentation, https://docs.llamaindex.ai/en/stable/examples/query_engine/knowledge_graph_rag_query_engine/, Accessed 17 Jan 2025
15. Velitchkov, I., Anadiotis, G.: Personal knowledge graphs: connected thinking to boost productivity, creativity and discovery. Exapt press (2023)
16. Lohmann, S., Negru, S., Haag, F., Ertl, T.: Visualizing ontologies with VOWL. Semant. Web J. 7(4), 339–419 (2016). <https://doi.org/10.3233/SW-150200>
17. Gra.fo Homepage, <https://gra.fo/>, Accessed 17 Jan 2025
18. LangChain Inc, LangChain documentation, https://python.langchain.com/docs/get_started/introduction, Accessed 17 Jan 2025
19. Kirchhof, J.C., Michael, J., Rumpe, B., Varga, S., Wortmann, A.: Model-driven digital twin construction: synthesizing the integration of cyber-physical systems with their information systems. In: Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems, Association for Computing Machinery, pp. 90–101 (2020). <https://doi.org/10.1145/3365438.3410941>.
20. OMILAB, The bee-up modeling tool, <https://bee-up.omilab.org/activities/bee-up/>, Accessed 17 Jan 2025
21. Karagiannis, D., Buchmann, R.A., Burzynski, P., Reimer, U., Walch, M.: Fundamental conceptual modeling languages in OMILAB. In: Karagiannis, D., Mayr, H.C., Mylopoulos, J. (eds.) Domain-specific conceptual modeling, pp. 3–30. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-39417-6_1
22. W3C Recommendation. SPARQL query language for RDF, <https://www.w3.org/TR/rdf-sparql-query/>, Accessed 17 Jan 2025
23. Chiş, A., Buchmann, R., Ghiran, A.M.: Towards a modeling method for low-code knowledge graph building. In: PoEM & EDEWC—companion 2023, vol. 3645, CEUR-WS (2024). <https://ceur-ws.org/Vol-3645/forum4.pdf>.
24. Karagiannis, D.: Agile modeling method engineering. In: Proceedings of the 19th Panhellenic conference on informatics, association for computing machinery (2015), pp. 5–10. <https://doi.org/10.1145/2801948.280204>.
25. The ADOxx Metamodeling platform, <https://www.adoxx.org/live/home> (2025), Accessed 2025/01/17.
26. Viyović, V., Maksimović, M., Perisić, B.: Sirius: A rapid development of DSM graphical editor. In: IEEE 18th international conference on intelligent engineering systems INES 2014, IEEE, pp. 233–238 (2014). <https://doi.org/10.1109/INES.2014.6909375>.
27. MetaCase, MetaEdit+, <https://www.metacase.com/products.html> (2025), Accessed 12 Jan 2025
28. Laurenzi, E.: An Agile and ontology-based meta-modelling approach for the design and maintenance of enterprise knowledge graph schemas. Enterprise modelling & information systems architectures (EMISAJ). German Inform Soc (2024). <https://doi.org/10.18417/emisa.19.6>
29. Karagiannis, D., Mayr, H.C., Mylopoulos, J. (eds.): Domain-specific conceptual modeling. Springer International Publishing, Cham (2016). <https://doi.org/10.1007/978-3-319-39417-6>
30. Fill, H.-G., Redmond, T., Karagiannis, D.: Formalizing meta models with FDMM: the ADOxx case. In: Cordeiro, J., Maciaszek, L.A., Filipe, J. (eds.) Enterprise information systems: 14th international conference, ICEIS 2012 Wroclaw, Poland, June 28–July 1, 2012 Revised Selected Papers, pp. 429–451. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40654-6_26
31. Karagiannis, D.: ADOSCRIP: A DSL for developing conceptual modeling methods. In: informing possible future worlds. Essays in Honour of Ulrich Frank, pp. 287–308. Logos Verlag Berlin (2024)
32. Buchmann, R.A.: The purpose-specificity framework for domain-specific conceptual modeling. In: Karagiannis, D., Lee, M., Hinkelmann, K., Utz, W. (eds.) Domain-specific conceptual modeling: concepts, methods and ADOxx tools, pp. 67–92. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-93547-4_4
33. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, H.: Retrieval-augmented generation for large language models: a survey. arXiv preprint [arXiv:2312.10997](https://arxiv.org/abs/2312.10997). ArXiv (2023). <https://doi.org/10.48550/arXiv.2312.10997>.
34. OpenAI platform Docs, Prompt engineering: enhance results with prompt engineering strategies, <https://platform.openai.com/docs/guides/prompt-engineering#tactic-use-delimiters-to-clearly-indicate-distinct-parts-of-the-input1> (2025), Accessed 12 Jan 2025
35. Iga, V.I.R., Silaghi, G.C.: Assessing LLMs suitability for knowledge graph completion. In: Besold, T.R., Garcez, A.D., Jimenez-Ruiz, E., Confalonieri, R., Madhyastha, P., Wagner, B. (eds.) Neural-symbolic learning and reasoning: 18th international conference, NeSy 2024, Barcelona, Spain, September 9–12, 2024, Proceedings, Part II, pp. 277–290. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-71170-1_22
36. Angles, R.: The property graph database model. In: Alberto Mendelzon Workshop on Foundations of Data Management, CEUR-WS, vol. 2100, paper 26 (2018). URL: <https://ceur-ws.org/Vol-2100/paper26.pdf>.
37. Wieringa, R.J.: Design science research methods, <https://wwwhome.ewi.utwente.nl/~roelw/DSM90minutes.pdf>, last Accessed 17 Jan 2025.
38. Ghiran, A.M., Buchmann, R., Karagiannis, D.: Towards a framework of techniques for enabling semantics-driven secondary notation in conceptual models. In: Proceedings of RCIS 2018, pp. 1–6. IEEE CS (2018). <https://doi.org/10.1109/RCIS.2018.8406684>.
39. Moody, D.L.: The “Physics” of notations: towards a scientific basis for constructing visual notations in software engineering. In: IEEE transactions on software engineering. IEEE computer society (2009), 35(5), 756–779. <https://doi.org/10.1109/TSE.2009.67>.
40. Google, Introducing the knowledge graph: things, not strings, <https://blog.google/products/search/introducing-knowledge-graph-things-not/> (2012), Accessed 12 Jan 2025.

41. Nonaka, I., von Krogh, G.: Perspective—tacit knowledge and knowledge conversion: controversy and advancement in organizational knowledge creation theory. *Organ. Sci.* **20**(3), 635–652 (2009). <https://doi.org/10.1287/orsc.1080.0412>
42. Karagiannis, D., Buchmann, R., Walch, M.: How can diagrammatic conceptual modelling support knowledge management?. In: Proceedings of ECIS 2017, Guimarães, Portugal, June 5–10, pp. 1568–1583. AIS eLibrary (2017). https://aisel.aisnet.org/ecis2017_rp/101.
43. Cagle, K.: RAG, Context and knowledge graphs, the cagle report LinkedIn, <https://www.linkedin.com/pulse/rag-context-knowledge-graphs-kurt-cagle-smvmc/> (2024), last Accessed 12 Jan 2025
44. yEd graph editor, <https://www.yworks.com/products/yed#>, Accessed 17 Jan 2025
45. The OMILAB Community: development of conceptual models and realization of modelling tools within the ADOxx meta-modelling environment: a living paper. In: Karagiannis, D., Lee, M., Hinkelmann, K., Utz, W. (eds.) Domain-specific conceptual modeling. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-93547-4_2
46. Bachhofner, S., Kiesling, E., Revoredo, K., Waibel, P., Polleres, A.: Automated process knowledge graph construction from BPMN models. In: Strauss, C., Cuzzocrea, A., Gabriele Kotsis, A., Tjoa, M., Khalil, I. (eds.) Database and expert systems applications: 33rd international conference, DEXA 2022, Vienna, Austria, August 22–24, 2022, Proceedings, Part I, pp. 32–47. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-12423-5_3
47. León, A., Santos, M.Y., García, A., Casamayor, J.C., Pastor, O.: Model-to-model transformation. *Bus. Inform. Syst. Eng.* **66**, 85–110 (2024). <https://doi.org/10.1007/s12599-023-00824-9>
48. OWLGrEd, <http://owlgred.lumiil.lv/>, last Accessed 17 Jan 2025
49. VOWL: visual notation for OWL ontologies, <http://vowl.visualdataweb.org/v2/>, Accessed 17 Jan 2025
50. RDF Grapher, <https://www.ldf.fi/service/rdf-grapher> (2025), Accessed 17 Jan 2025
51. Ontotext, The power of visualization: GraphDB now enables custom graph views, <https://www.ontotext.com/blog/power-visualization-graphdb/> (2024), Accessed 17 Jan 2025.
52. Vis.js library, www.visjs.org (2025), Accessed 17 Jan 2025.
53. Graffoo specification, <https://esepuntato.it/graffoo/specification/> (2013), Accessed 17 Jan 2025
54. The AQL query language, <https://www.adoxx.org/live/adox-query-language-aql> (2025), Accessed 17 Jan 2025
55. Krogstie, J., Sindre, G., Jørgensen, H.: Process models representing knowledge for action: a revised quality framework. *Eur. J. Inf. Syst.* **15**, 91–102 (2006). https://doi.org/10.1007/978-3-319-39417-6_1
56. Charles, J., Michael, J., Netz, L., Rumpe, B.: Teaching model-driven low-code development platforms. In: proceedings of the ACM/IEEE 27th international conference on model driven engineering languages and systems, Association for computing machinery (2024), pp. 570–577. <https://doi.org/10.1145/3652620.3687778>.
57. Daniel, G., Sunyé, G., Cabot, J.: UMLtoGraphDB: Mapping conceptual schemas to graph databases. In: Comyn-Wattiau, I., Tanaka, K., Song, I.-Y., Yamamoto, S., Saeki, M. (eds.) Conceptual Modeling, pp. 430–444. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-46397-1_33
58. TopBraid EDG official website, <https://www.topquadrant.com/release-notes/7-1>, Accessed 17 Jan 2025.
59. Gruff Graph Visualizer Homepage, <https://allegrograph.com/products/gruff/>, Accessed 17 Jan 2025
60. PoolParty semantic suite official website, <https://www.poolparty.biz/product-overview>, Accessed 17 Jan 2025
61. Lembo, D., Santarelli, V., Savo, D.F., De Giacomo, G.: Graphol: a graphical language for ontology modeling equivalent to OWL 2. *Future Int.* **14**(3), 78 (2022). <https://doi.org/10.3390/fi14030078>
62. Eddy graphical editor, <https://github.com/obdasyystems/eddy>, Accessed 17 Jan 2025
63. Chávez-Feria, S., García-Castro, R., Poveda-Villalón, M.: Chowlk: from UML-based ontology conceptualizations to OWL. In: Groth, P., Vidal, M.-E., Suchanek, F., Szekley, P., Kapanipathi, P., Pesquita, C., Skaf-Molli, H., Tamper, M. (eds.) The semantic web: 19th international conference, ESWC 2022, Heronissos, Crete, Greece, May 29–June 2, 2022, Proceedings, pp. 338–352. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-06981-9_20
64. Falco, R., Gangemi, A., Peroni, S., Shotton, D., Vitali, F.: Modelling OWL ontologies with graffoo. In: Presutti, V., Blomqvist, E., Troncy, R., Sack, H., Papadakis, I., Tordai, A. (eds.) The semantic web: ESWC 2014 satellite events: ESWC 2014 satellite events, Anissaras, Crete, Greece, May 25–29, 2014, Revised Selected Papers, pp. 320–325. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-11955-7_42
65. OpenCAESAR project, <https://www.opencaesar.io/>, Accessed 17 Jan 2025
66. Wagner, D.A., Chodas, M., Elaasar, M., Jenkins, J.S., Rouquette, N.: Ontological metamodeling and analysis using openCAESAR. In: Madni, A.M., Augustine, N., Sievers, M. (eds.) Handbook of model-based systems engineering, pp. 925–954. Springer International Publishing, Cham (2023). https://doi.org/10.1007/978-3-030-93582-5_78
67. Ontological Modeling Language v2 specification, <https://www.opencaesar.io/oml/>, last Accessed 17 Jan 2025
68. Héon, M., Nkambou, R., Gaha, M.: 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). <https://ceur-ws.org/Vol-1690/paper52.pdf>.
69. Héon, M., Nkambou, R., Langheit, C.: Towards G-OWL: A graphical, polymorphic and typed syntax for building formal OWL2 ontologies. In: Proceedings of the 25th international conference companion on the world wide web, association for computing machinery, pp. 39–49 (2016). <https://doi.org/10.1145/2872518.2889377>
70. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse modeling framework. In: Pearson education (2009)
71. Kang, M., Kwak, J.M., Baek, J., Hwang, S.J.: Knowledge graph-augmented language models for knowledge-grounded dialogue generation. ArXiv (2023). <https://doi.org/10.48550/arXiv.2305.18846>
72. Alaca, O.M., Tezel, B.T., Challenger, M., Goulão, M., Amaral, V., Kar-das, G.: AgentDSM-Eval: A framework for the evaluation of domain-specific modeling languages for multi-agent systems. In: Computer standards & interfaces **76**, 103513. Elsevier (2021). <https://doi.org/10.1016/j.csi.2021.103513>.
73. Kahlaoui, A., Abran, A., Lefebvre, E.: DSML success factors and their assessment criteria. *METRICS News* **13**(1), 43–51 (2008)
74. Challenger, M., Kardas, G., Tekinerdogan, B.: A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. *Softw. Quality J.* **24**(3), 755–795 (2015). <https://doi.org/10.1007/s11219-015-9291-5>
75. Inrupt, Solid, <https://www.inrupt.com/solid>, Accessed 17 Jan 2025
76. Obsidian, <https://obsidian.md/>, Accessed 17 Jan 2025
77. The GQL Manifesto, <https://gql.today/>, Accessed 17 Jan 2025



Andrei Chiș is a PhD student in the Department of Business Information Systems and a research member of the OMILAB Laboratory at Babeș-Bolyai University, Romania (<https://econ.ubbcluj.ro/omilab/index.php>). His research interests lie at the intersection of Conceptual Modeling, Semantic Web Technologies and Generative Artificial Intelligence, with a particular focus on the application of Knowledge Graphs in organizational settings. He is currently engaged in developing a

suite of conceptual modeling tools designed to function as endpoints for organizational knowledge acquisition and representation. These tools integrate management theories, knowledge graphs, model-driven engineering (MDE) principles and generative AI techniques to support decision-making and knowledge reuse within enterprises. His work contributes to the broader goal of enhancing the expressiveness, automation and adaptability of model-driven software engineering methodologies in various academic and industrial contexts.



Ana-Maria Ghiran is a Lecturer at Business Information Systems Department and a member of the OMILAB Laboratory at Babeș-Bolyai University, Romania (<https://econ.ubbcluj.ro/omilab/index.php>). She has research experience with Semantic Web technologies, as she applied them in her doctoral thesis developed at Babeș-Bolyai University, in the field of IT Infrastructure Auditing. Currently, she is investigating the Conceptual Modeling paradigm and applications of

semantic technology in model-driven software engineering. She is involved as a Program Committee member for numerous international conferences in the areas of Information Systems, Knowledge Engineering and Conceptual Modeling, such as KSEM, BIR, ISD, PoEM, RCIS. Furthermore, she was engaged in several editorial roles with various publishers: Springer, CEUR-WS and CSIMQ Journal. In 2024, she received a visiting fellowship at University Sorbonne Paris Nord and she became interested in Large Language Models and their application in Conceptual Modeling in conjunction with Knowledge Graphs.



Robert Andrei Buchmann occupies a Professor position in Business Information Systems and manages the OMILAB laboratory at Babeș-Bolyai University, Romania, focusing on Knowledge Graphs for Business Information Systems (<https://econ.ubbcluj.ro/omilab/index.php>). He is the Scientific Director of the Business Informatics Research Center, where his team is investigating opportunities of interplay between Knowledge Graphs, Large Language Models and

Enterprise Modeling. During 2012–2015, he occupied a postdoctoral research position at University of Vienna, specializing in semantic graph databases and the engineering of domain-specific modeling languages and tools. He's been involved in various collaborations within the OMILAB network of laboratories, including the NEMO summer school series on Enterprise Modeling (<https://nemo.omilab.org/>, since 2014) and several thematic workshops on Knowledge Graphs or Knowledge Engineering (e.g., KG4SDSE at CAiSE since 2023). He has chaired during recent years the program of numerous conferences in the areas of Information Systems and Knowledge Engineering: KSEM 2023 and 2014 (with Springer), BIR 2020–2021 (with Springer), ISD 2022 (with AIS and Springer), PoEM 2018 (with Springer), the Enterprise Modeling track at ECIS 2020–2023 (with AIS). He is currently a member of IFIP (WG 8.1, the working group on Design and Evaluation of Information Systems) and IAOA (International Assoc. on Ontology and its Applications).