Model-Independent Design of Knowledge Graphs

Luigi Bellomarini[1], Andrea Gentili[1], Eleonora Laurenza[1] and Emanuel Sallinger[2,3]

[1]*Bank of Italy (Italy)*
[2]*Technische Universität Wien (Austria)*
[3]*University of Oxford (UK)*

Abstract

Knowledge Graphs (KGs) can be seen as knowledge bases combining an extensional component, a database of facts, typically a property graph, and an intensional component, a formal specification of the available business experience, to derive new knowledge from those facts, often as new nodes and edges.

Capitalizing on our experience in KGs and model management for the rollout of financial KGs for the Central Bank of Italy, in this work we present KGModel, a model-independent design framework for KGs. The framework adopts a meta-level approach: the data engineer visually designs the extensional component of the KG at a conceptual level and augments it with intensional specifications in MetaLog, a new logical model-independent language. This high-level specification of the KG is then translated into enforceable schema definitions for the target database and executable logical rules for a target reasoner.

Our framework offers (i) a model-independent visual modeling language; (ii) MetaLog, a new language of the Datalog+/- family for the intensional component; (iii) new complementary software tools for the translation of meta-level specifications into their executable versions. We present the main ideas behind KGModel and show the suitability of the framework for real-world scenarios.

This work is a short version of an EDBT 2022 paper.

Keywords

knowledge graphs, Datalog, conceptual design, data modeling, schema and data translation

1. Introduction

Knowledge Graphs (KGs) can be seen as models for knowledge representation and reasoning, that combine an *extensional component*—a database of facts, typically a property graph (PG) [1]—and an *intensional component*, a formal specification of business experience, to derive new knowledge from those facts, often as new nodes and edges [2, 3]. Capitalizing on our experience in the construction of large KGs, especially in the financial and economic realms [4, 5, 6, 7], we observe that the need for a **KG design methodology** is clearly emerging.

Such a methodology: ❶ Should provide *conceptual data models*, enabling a simple, non-technical, high-level, visual representation of the domain. ❷ Should be *implementation-independent*, i.e., it should be possible to deploy the extensional component into any Graph Database Management System, relational, triple-store system, etc., and it should be possible to express the intensional components regardless of the target systems. ❸ Should provide a set of constructs

15th Alberto Mendelzon International Workshop on Foundations of Data Management, May 22–26, Santiago, Chile
luigi.bellomarini@bancaditalia.it (L. Bellomarini); andrea.gentili@bancaditalia.it (A. Gentili); eleonora.laurenza@bancaditalia.it (E. Laurenza); sallinger@dbai.tuwien.ac.at (E. Sallinger)
0000-0001-6863-0162 (L. Bellomarini); 0000-0001-7441-129X (E. Sallinger)

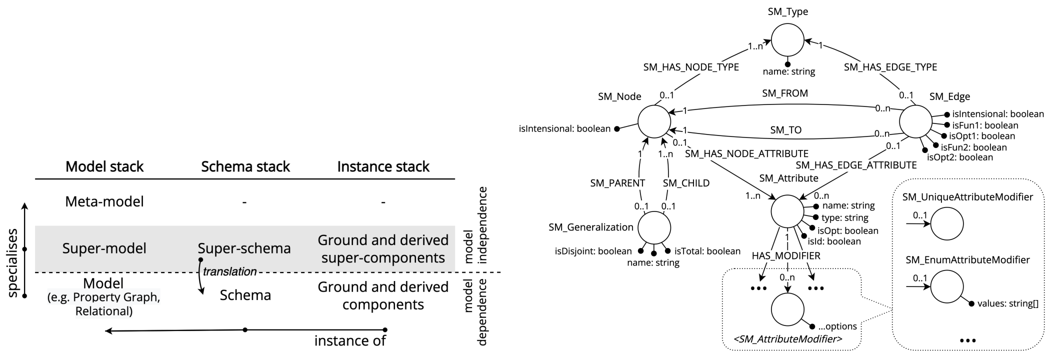

Figure 1: The KGModel stack (on the left) and the super-model dictionary (right).

to define *graph schemas.* ❹ Should allow encoding the *intensional component specification* with a *reasoning language* that is graph-ergonomic and expressive enough to handle KGs. Reflecting the UC2RPQs literature stream [8], navigational expressions should be intuitively supported by the syntax. Reasoning to the extent of tractable description logic should be feasible (e.g., $DL\text{-}Lite_R$). In other terms, the language should be expressive enough to cover any SPARQL query over RDF datasets, under the entailment regime of OWL 2 QL [9]. ❺ Should adopt a *model-driven approach* [10]: an enforceable graph schema and an executable version of the intensional components should directly derive from a high-level conceptual representation of the domain given by the data engineer.

To the best of our knowledge, there is no comprehensive methodology for KG design and none of the existing approaches satisfies the illustrated desiderata.

Contribution. In this short version of a recent EDBT paper [11], we present KGModel, a *model-independent framework for Knowledge Graphs design*, comprising a *methodology* and a set of *support tools.* The framework is described in Section 2 and some insights on the designer's perspective are given in Section 3. For space reasons, for a detailed presentation, including a discussion of all the KGModel components, the patterns of the design methodology, and the analysis of related literature, the reader is referred to the long version of the paper.

2. The KGModel Framework

KGModel adopts a layered approach to data representation, in Figure 1 (left-hand side). It is organized into three stacks of representations: *model*, *schema*, and *instance*, where each level contains a set of *constructs* that specialize (or are specialized by) the constructs of the level above (below). The instance stack instantiates the schema stack, which instantiates the model.

In the *model stack*, we adopt the idea of a *super-model* grouping the *super-constructs* that can be used to define different *Knowledge Graph* models, which are all specializations of the super-model. Examples are the models of Neo4J PG, Amazon Neptune, OrientDB, or even non-graph-like models. At the highest level, a ***meta-model*** contains the foundational *meta-constructs*, namely, MM_Entity (an abstract entity of the domain), MM_Link (a connection between entities), as well as their properties. The ***super-model***, visualized in Figure 1, contains *super-constructs* that specialize those of the meta-model and subsumes, i.e., generalizes, any possible KG model. Examples of super-constructs are SM_Node, SM_Edge. The various ***models*** comprise *constructs* that specialize the super-constructs for a specific use. Example of PG constructs are Node, Relationship, and Label, instantiating SM_Node, SM_Relationship, and SM_Type.

In the *schema stack*, *schemas* capture the type of specific nodes, edges, and properties of a given domain of interest, in the same sense that a relational database schema is an instance of the relational model. As the super-model generalizes every model, a schema can be expressed in either a model-dependent way, as an instance of a model, or in a model-independent way, as an instance of the super-model, in which case we call it *super-schema.* A super-schema S_1 can be cast into a schema S_2 of a model M by a specific set of translation rules, namely, *mappings*, which apply the needed simplifications, when the case eliminating constructs of the super-model that are not supported by the specific target model, and finally instantiate the super-constructs into M constructs, accordingly. We define the mappings as MetaLog rules. MetaLog is our new variant of the Vadalog language [12] for graphs. Vadalog extends Datalog with existential quantification and other useful features, while introducing mild syntactic restrictions to guarantee decidability and tractability of the reasoning task. Vadalog reasoning programs can be processed by the Vadalog System, a state-of-the-art reasoner. MetaLog inherits the Vadalog (and thus Datalog) semantics and expressive power, enriching its syntax with the possibility to use pattern-matching graph exploration primitives. It is model-independent, as it operates at meta-level, it is expressive and efficient enough to support ontological reasoning, it is model aware and ergonomic, as incorporates syntactic elements to exploit schema information.

The *instance stack* represents the **extensional component**, i.e., both the ground data and those derived by materializing the intensional component.

The Design Approach. With KGModel, we offer the data engineer a model-driven design approach to KGs. The data engineer is provided with a *conceptual visual modeling language*, named *Graph Schema Language* (GSL) to design a graph schema as a super-schema. A GSL diagram defines an instance of the super-model, with *visual graphemes* denoting instances of the super-constructs. The business knowledge is encoded by the data engineer in the intensional component, with MetaLog programs acting on the super-model constructs.

To deploy the designed schemas into the target systems, KGModel translates the super-schemas provided by the engineer into instances of the target models by applying the translation mappings. Schemas then contain all the information needed to be deployed and enforced, with different methods, depending on the target systems: for relational systems, for instance, they can be rendered as DDL statements, which include the respective constraints such as keys, foreign keys, domain constraints, and so on; for RDF stores, schemas can be rendered as RDF-S (RDF Schema) documents, to be validated by dedicated tools; for schema-less systems, schemas can be enforced with ad-hoc methodologies [13].

The Tools. Our framework incorporates (a) *Graph Dictionaries*: A set of graph databases to store the instances of the super-model and of the models. (b) *Knowledge Graph Schema Environment*: A tool to graphically design GSL schemas and store them in the super-model dictionary. (c) *MetaLog to* Vadalog *Translator (MTV)*: A compiler to generate Vadalog programs from MetaLog code. (d) *Super-Schema to Schema Translator (SSST)*: A module that takes as input a super-schema S, a super-model-level intensional component Σ expressed as MetaLog rules, a MetaLog mapping $\mathcal{M}(M)$ for the translation of a super-schema into a schema of the target model M, and generates: (i) the instance S' of M, i.e., the target schema; (ii) a new version of the intensional component that can be applied to S' instances. SSST uses MTV to compile and run MetaLog.

The Language. MetaLog combines *Warded Datalog*$^{\pm}$ [14], at the core of Vadalog, and graph pattern matching. A MetaLog program is a set Σ of existential rules $\boldsymbol{\varphi}(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \boldsymbol{\psi}(\bar{x}, \bar{z})$, where $\bar{x}$, $\bar{y}$, and $\bar{z}$ are tuples of variables, $\boldsymbol{\varphi}$ is a conjunction of atoms denoting *nodes*, *path patterns*, *conditions*, and *expressions* and $\boldsymbol{\psi}$ is a conjunction of *node atoms* and *path patterns*. A path pattern xRy individuates all the pairs of nodes $\langle x, y\rangle$ connected by a semi-path that conforms to the regular language $L(R)$ defined by R. The semantics of MetaLog descends from the Vadalog one. Given a graph G, for each fact of $\varphi(\bar{t}, \bar{t}')$, that is, a conjunction of G paths, there exists a tuple $\bar{t}''$ of constants and new symbols to satisfy existential quantification, such that the paths $\psi(\bar{t}, \bar{t}'')$ are also in G. Given a set Σ of MetaLog rules, the chase alters G by adding new paths, until $\Sigma(G)$ satisfies all of them.

3. The Designer's Perspective

We used KGModel to design the *Company KG* of the Bank of Italy. Let us try to narratively simulate a small fragment of that modeling journey. The domain revolves around the notions of physical persons, i.e., individuals, or legal persons. These two entities share many features, but participate in different relationships.
💡*I will capture the structure by introducing distinct SM_Nodes for persons, i.e., PhysicalPerson and LegalPerson, with a distinct set of SM_Attributes.*
💡*I will introduce an SM_Generalization, where a Person generalizes and collects the common features of PhysicalPerson and LegalPerson.*

A person can withhold stakes in a company capital, and multiple persons may have different rights upon the same portion of company capital.
💡*I will introduce a Share SM_Node and the HOLDS-BELONGS_TO SM_Edges decoupling owner-owned SM_Nodes so that multiple Persons can HOLD a Share each with right and percentage.*

Figure 2 shows a portion of the designed KG. The entities of the extensional component are represented by solid lines. The following program gives an idea of how MetaLog captures the intensional knowledge (dashed lines). Clearly, for space reasons, here we show only a minimal fragment of the KG, which actually has tens of entities and many MetaLog snippets.

$$(x : \mathit{Business}) \rightarrow \exists c\ (x)[c : \mathit{CONTROLS}](x) \tag{1}$$

$$\begin{aligned}(x : \mathit{Business})[: \mathit{CONTROLS}](z : \mathit{Business})\\ [: \mathit{OWNS}; \mathit{percentage} : w](y : \mathit{Business}),\\ v = \mathit{sum}(w, \langle z\rangle), v > 0.5 \rightarrow \exists c\ (x)[c : \mathit{CONTROLS}](y)\end{aligned} \tag{2}$$

A business x controls a business y, if: (i) x controls itself; or, (ii) the sum of the shares w of y owned by companies z (possibly including x), over all companies z controlled by x, is above the 50% threshold.

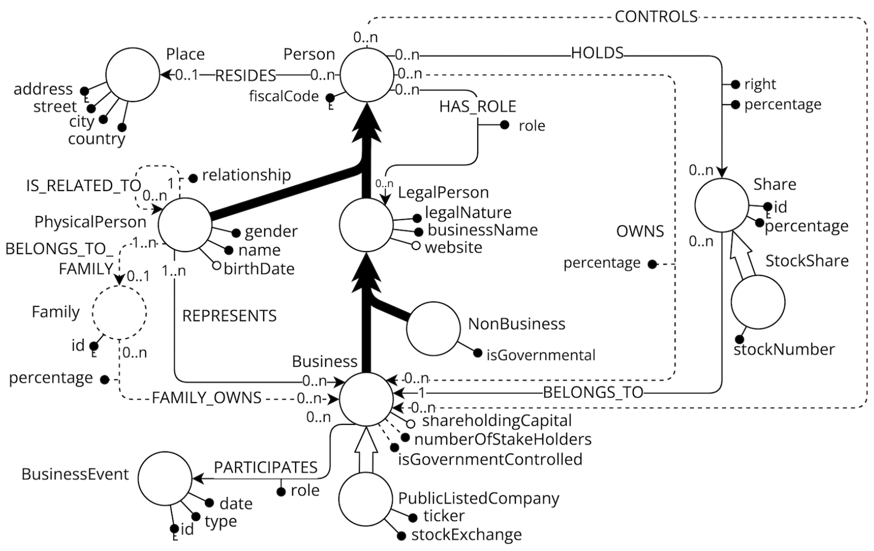

Figure 2: A portion of the Bank of Italy KG designed with KGModel methodology.

4. Conclusion

We could appreciate how a technology-independent super-model guides the designer through the modeling activity, offering a toolkit of lenses to capture real-world objects, understand their characteristics and relationships, and communicate the design choices with stakeholders. Our methodology fulfills the presented desiderata and extends existing meta-level approaches [15, 16] to the KG realm.

5. Acknowledgments

This work has been funded by the Vienna Science and Technology Fund (WWTF) [10.47379/VRG18013, 10.47379/NXT22018, 10.47379/ICT2201]; and the Christian Doppler Research Association (CDG) JRC LIVE.

References

[1] R. Angles, The property graph database model, in: AMW, volume 2100 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018.

[2] L. Bellomarini, D. Fakhoury, G. Gottlob, E. Sallinger, Knowledge graphs and enterprise AI: the promise of an enabling technology, in: ICDE, IEEE, 2019, pp. 26–37.

[3] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutiérrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. F. Sequeda, S. Staab, A. Zimmermann, Knowledge graphs, ACM Comput. Surv. 54 (2021) 71:1–71:37.

[4] P. Atzeni, L. Bellomarini, M. Iezzi, E. Sallinger, A. Vlad, Weaving enterprise knowledge graphs: The case of company ownership graphs, in: EDBT, OpenProceedings.org, 2020, pp. 555–566.

[5] L. Bellomarini, M. Benedetti, S. Ceri, A. Gentili, R. Laurendi, D. Magnanimi, M. Nissl, E. Sallinger, Reasoning on company takeovers during the COVID-19 crisis with knowledge graphs, in: RuleML+RR, 2020.

[6] L. Bellomarini, E. Laurenza, E. Sallinger, Rule-based anti-money laundering in financial intelligence units: Experience and vision, in: RuleML+RR (Supplement), volume 2644 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 133–144.

[7] L. Bellomarini, M. Nissl, E. Sallinger, Rule-based blockchain knowledge graphs: Declarative ai for solving industrial blockchain challenges, in: RuleML+RR (To Appear), CEUR Workshop Proceedings, CEUR-WS.org, 2021.

[8] M. Y. Vardi, A theory of regular queries, in: PODS, ACM, 2016, pp. 1–9.

[9] B. Glimm, C. Ogbuji, S. Hawke, I. Herman, B. Parsia, A. Polleres, A. Seaborne, SPARQL 1.1 entailment regimes, 2013. W3C Recommendation 21 March 2013, 2013.

[10] F. A. Fontana, H. Brunelière, H. A. Müller, C. Raibulet, Guest editors' introduction to the special issue on model driven engineering and reverse engineering: Research and practice, J. Syst. Softw. 159 (2020).

[11] L. Bellomarini, A. Gentili, E. Laurenza, E. Sallinger, Model-independent design of knowledge graphs - lessons learnt from complex financial graphs, in: EDBT, OpenProceedings.org, 2022, pp. 2:524–2:526.

[12] L. Bellomarini, E. Sallinger, G. Gottlob, The vadalog system: Datalog-based reasoning for knowledge graphs, PVLDB 11 (2018).

[13] A. Bonifati, P. Furniss, A. Green, R. Harmer, E. Oshurko, H. Voigt, Schema validation and evolution for graph databases, in: ER, volume 11788 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 448–456.

[14] G. Gottlob, A. Pieris, Beyond SPARQL under OWL 2 QL entailment regime: Rules to the rescue, in: IJCAI, 2015, pp. 2999–3007.

[15] P. Atzeni, L. Bellomarini, F. Bugiotti, G. Gianforme, MISM: A platform for model-independent solutions to model management problems., in: J. Data Semantics, 2009, pp. 133–161.

[16] P. Atzeni, P. Cappellari, P. A. Bernstein, Modelgen: Model independent schema translation, in: ICDE, IEEE Computer Society, 2005, pp. 1111–1112.