



Transactions on  
**Graph Data and Knowledge**

**Volume 2 | Issue 2 | December, 2024**

**Special Issue: Resources for Graph Data and Knowledge**

Edited by

Aidan Hogan  
Ian Horrocks  
Andreas Hotho  
Lalana Kagal  
Uli Sattler



## ISSN 2942-7517

*TGDK Special Issue Editors*

### **Aidan Hogan**

DCC, Universidad de Chile, IMFD, Chile  
ahogan@dcc.uchile.cl

### **Ian Horrocks**

University of Oxford, U.K.  
ian.horrocks@cs.ox.ac.uk

### **Andreas Hotho**

University of Würzburg, Germany  
hotho@informatik.uni-wuerzburg.de

### **Lalana Kagal**

Massachusetts Institute of Technology, Cambridge,  
MA, USA  
lkagal@csail.mit.edu

### **Uli Sattler**

University of Manchester, U.K.  
Uli.Sattler@manchester.ac.uk

#### *ACM Classification 2012*

Computing methodologies → Knowledge representation and reasoning; Information systems → Semantic web description languages; Information systems → Graph-based database models; Computing methodologies → Machine learning; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory

#### *Published online and open access by*

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany.

Online available at

<https://www.dagstuhl.de/dagpub/2942-7517>.

#### *Publication date*

December, 2024

#### *Digital Object Identifier*

10.4230/TGDK.2.2.0

*Bibliographic information published by the Deutsche Nationalbibliothek*

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <https://dnb.d-nb.de>.

#### *License*

This work is licensed under a Creative Commons Attribution 4.0 International license (CC BY 4.0): <https://creativecommons.org/licenses/by/4.0>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

#### *Aims and Scope*

Transactions on Graph Data and Knowledge (TGDK) is an Open Access journal that publishes original research articles and survey articles on graph-based abstractions for data and knowledge, and the techniques that such abstractions enable with respect to integration, querying, reasoning and learning. The scope of the journal thus intersects with areas such as Graph Algorithms, Graph Databases, Graph Representation Learning, Knowledge Graphs, Knowledge Representation, Linked Data and the Semantic Web. Also in-scope for the journal is research investigating graph-based abstractions of data and knowledge in the context of Data Integration, Data Science, Information Extraction, Information Retrieval, Machine Learning, Natural Language Processing, and the Web.

The journal is Open Access without fees for readers or for authors (also known as Diamond Open Access).

#### *Editors in Chief*

- Aidan Hogan
- Andreas Hotho
- Lalana Kagal
- Uli Sattler

#### *Editorial Office*

Schloss Dagstuhl – Leibniz-Zentrum für Informatik  
TGDK, Editorial Office

Oktavie-Allee, 66687 Wadern, Germany

[tgdk@dagstuhl.de](mailto:tgdk@dagstuhl.de)

<https://www.dagstuhl.de/tgdk>



# ■ Contents

List of Authors	0:vii–0:viii
-----------------	--------------

## Preface


Resources for Graph Data and Knowledge <i>Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler</i>	1:1–1:2
--	---------


## Resources for Graph Data and Knowledge


NEOntometrics – A Public Endpoint for Calculating Ontology Metrics <i>Achim Reiz and Kurt Sandkuhl</i>	2:1–2:22
The dblp Knowledge Graph and SPARQL Endpoint <i>Marcel R. Ackermann, Hannah Bast, Benedikt Maria Beckermann, Johannes Kalmbach, Patrick Neises, and Stefan Ollinger</i>	3:1–3:23
FAIR Jupyter: A Knowledge Graph Approach to Semantic Sharing and Granular Exploration of a Computational Notebook Reproducibility Dataset <i>Sheeba Samuel and Daniel Mietchen</i>	4:1–4:24
The Reasonable Ontology Templates Framework <i>Martin Georg Skjæveland and Leif Harald Karlsen</i>	5:1–5:54
TØIRoads: A Road Data Model Generation Tool <i>Grunde Haraldsson Wesenberg and Ana Ozaki</i>	6:1–6:12
Whelk: An OWL EL+RL Reasoner Enabling New Use Cases <i>James P. Balhoff and Christopher J. Mungall</i>	7:1–7:17
MELArt: A Multimodal Entity Linking Dataset for Art <i>Alejandro Sierra-Múnera, Linh Le, Gianluca Demartini, and Ralf Krestel</i>	8:1–8:22
Horned-OWL: Flying Further and Faster with Ontologies <i>Phillip Lord, Björn Gehrke, Martin Larralde, Janna Hastings, Filippo De Bortoli, James A. Overton, James P. Balhoff, and Jennifer Warrender</i>	9:1–9:14





## ■ List of Authors

Marcel R. Ackermann  (3)  
Schloss Dagstuhl – Leibniz Center for Informatics,  
dblp computer science bibliography, Trier,  
Germany


James P. Balhoff  (7, 9)  
Renaissance Computing Institute, University of  
North Carolina, Chapel Hill, NC, USA


Hannah Bast  (3)  
University of Freiburg, Department of Computer  
Science, Freiburg, Germany


Benedikt Maria Beckermann  (3)  
Schloss Dagstuhl – Leibniz Center for Informatics,  
dblp computer science bibliography, Trier,  
Germany

Filippo De Bortoli  (9)  
TU Dresden, Germany;  
Center for Scalable Data Analytics and Artificial  
Intelligence (ScaDS.AI), Dresden/Leipzig, Germany


Gianluca Demartini  (8)  
The University of Queensland, Brisbane, Australia


Björn Gehrke  (9)  
Institute for Implementation Science in Health  
Care, Faculty of Medicine, University of Zurich,  
Switzerland


Janna Hastings  (9)  
Institute for Implementation Science in Health  
Care, Faculty of Medicine, University of Zurich,  
Switzerland;  
School of Medicine, University of St. Gallen,  
Switzerland; Swiss Institute of Bioinformatics,  
Switzerland


Aidan Hogan  (1)  
DCC, Universidad de Chile, IMFD, Chile

Ian Horrocks  (1)  
University of Oxford, U.K


Andreas Hotho  (1)  
Department of Informatics,  
University of Würzburg, Germany

Lalana Kagal  (1)  
Massachusetts Institute of Technology,  
Cambridge, MA, USA


Johannes Kalmbach  (3)  
University of Freiburg, Department of Computer  
Science, Freiburg, Germany


Leif Harald Karlsen  (5)  
Department of Informatics,  
University of Oslo, Norway

Ralf Krestel  (8)  
ZBW – Leibniz Information Centre for Economics,  
Kiel, Germany; Kiel University, Kiel, Germany


Martin Larralde  (9)  
Leiden University Medical Center,  
The Netherlands;  
Structural and Computational Biology Unit,  
EMBL, Heidelberg, Germany


Linh Le  (8)  
The University of Queensland, Brisbane, Australia


Phillip Lord  (9)  
School of Computing, Newcastle University,  
United Kingdom


Daniel Mietchen  (4)  
FIZ Karlsruhe – Leibniz Institute for Information  
Infrastructure, Germany;  
Institute for Globally Distributed Open Research  
and Education (IGDORE)

Christopher J. Mungall  (7)  
Lawrence Berkeley National Laboratory,  
Berkeley, CA, USA


Patrick Neises  (3)  
Schloss Dagstuhl – Leibniz Center for Informatics,  
dblp computer science bibliography, Trier,  
Germany

Stefan Ollinger  (3)  
Schloss Dagstuhl – Leibniz Center for Informatics,  
dblp computer science bibliography, Trier,  
Germany

James A. Overton  (9)  
Knocean Inc., Toronto, Canada

Ana Ozaki  (6)  
Department of Informatics,  
University of Oslo, Norway;  
Department of Informatics,  
University of Bergen, Norway

Achim Reiz  (2)  
Rostock University, Germany

Sheeba Samuel  (4)  
Distributed and Self-organizing Systems, Chemnitz  
University of Technology, Chemnitz, Germany

*Transactions on Graph Data and Knowledge*, Vol. 2, Issue 2, Article No. 0, pp. 0:i–0:viii  
Special Issue: Resources for Graph Data and Knowledge.

Editors: Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler




TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


## 0:viii Authors


Kurt Sandkuhl  (2)  
Rostock University, Germany

Uli Sattler  (1)  
University of Manchester, U.K

Alejandro Sierra-Múnera  (8)  
Hasso Plattner Institute,  
University of Potsdam, Potsdam, Germany

Martin Georg Skjæveland  (5)  
Department of Informatics,  
University of Oslo, Norway

Jennifer Warrender  (9)  
School of Computing, Newcastle University,  
United Kingdom

Grunde Haraldsson Wesenberg  (6)  
Department of Informatics,  
University of Bergen, Norway;  
Institute of Transport Economics,  
Oslo, Norway



# Resources for Graph Data and Knowledge

**Aidan Hogan** ✉ 🏠 

DCC, Universidad de Chile, IMFD, Chile

**Ian Horrocks** ✉ 🏠 

University of Oxford, U.K.

**Andreas Hotho** ✉ 🏠 

Department of Informatics, University of Würzburg, Germany

**Lalana Kagal** ✉ 🏠 

Massachusetts Institute of Technology, Cambridge, MA, USA

**Uli Sattler** ✉ 🏠 

University of Manchester, U.K.

---

## — Abstract —

In this Special Issue of Transactions on Graph Data and Knowledge – entitled “Resources for Graph Data and Knowledge” – we present eight articles that describe key resources in the area. These resources cover a wide range of topics within the scope

of the journal, including graph querying, graph learning, information extraction, and ontologies, addressing applications of knowledge graphs involving art, bibliographical metadata, research reproducibility, and transport networks.

**2012 ACM Subject Classification** Computing methodologies → Knowledge representation and reasoning; Information systems → Semantic web description languages; Information systems → Graph-based database models; Computing methodologies → Machine learning; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory

**Keywords and phrases** Graphs, Data, Knowledge

**Digital Object Identifier** 10.4230/TGDK.2.2.1

**Category** Preface

**Acknowledgements** We warmly thank Dagstuhl Publishing for their continued collaboration, the Semantic Web Science Association (SWSA) for their support, our colleagues on the SWSA Task Force who helped to plan this new journal, as well as our Advisory and Editorial Boards for their contributions towards getting the journal up and running and ensuring its continued operation and development.

**Published** 2024-12-18

**Editors** Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler

**Special Issue** Resources for Graph Data and Knowledge

## 1 Resources Articles

Resources play an essential role in many areas of computer science research, including in the area of Graph Data & Knowledge. Such resources may involve benchmarks, datasets, engines, frameworks, interfaces, knowledge graphs, languages, ontologies, pre-trained models, software libraries, standards, tools, user logs, web applications and services, etc. High-quality resources along these lines offer significant value to the community, facilitating rapid prototyping of novel applications and tools; experimentation over real-world datasets, ontologies, queries, etc.; transitioning novel research findings into practice; etc. Despite the advances that such resources can enable within a particular research community, and the amount of effort required in designing, building, and maintaining them, they can often be undervalued in an academic setting.



© Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler; licensed under Creative Commons License CC-BY 4.0

*Transactions on Graph Data and Knowledge*, Vol. 2, Issue 2, Article No. 1, pp. 1:1–1:2



Transactions on Graph Data and Knowledge

TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1:2 Resources for Graph Data and Knowledge

In this context, Transactions on Graph Data & Knowledge (TGDK) has introduced a new submission type for Resource Articles that describe two types of resources relevant to research on Graph Data & Knowledge:

- *Mature resources* that have already enjoyed significant adoption by third parties, or that complement resources with significant adoption. Such adoption may be in, for example, the context of research, industry or a specific user community.
- *Emerging resources* that may have only recently been made available, but that provide novel scientific results. An example of such a resource could be, for example, a bespoke benchmark that provides novel insights into the performance of state-of-the-art tools on a specific task.

Resource Articles include discussion of the motivation for the resource and its novelty, a technical description of the resource, relevance to the journal, key statistics or other metadata underlying the resource and its adoption, how the resource is made available, how the sustainability of the resource is assured, an assessment of the limitations of the resource, and future directions for the resource. Mature resources must further discuss the impact of the resource, while emerging resources must present a scientific contribution enabled by the resource. Such submissions are then peer-reviewed with respect to six criteria: novelty, relevance, clarity, technical soundness, impact, and resource quality. In the case of novelty, the emphasis is put on the novelty of the resource itself in the context of related resources rather than expecting novelty in a research sense.

This new submission type was inaugurated via the current Special Issue, entitled “Resources for Graph Data & Knowledge” (described below). Based on the success of this Special Issue, the Editors-in-Chief now solicit Resource Articles as part of the regular call for submissions.

## 2 Resources for Graph Data & Knowledge

As the Editors-in-Chief, we are pleased to present the TGDK Special Issue titled “Resources for Graph Data & Knowledge”, which is the second issue of the second volume of the journal, and the third issue overall since the inauguration of the journal. This Special Issue presents a collection of eight Resource Articles. The resources covered by the Special Issue reflect the broad applicability of graphs for representing data and knowledge. The resources themselves cover various sub-topics relevant for TGDK that include graph querying, graph learning, information extraction and knowledge representation, covering also applications relating to art, bibliographical metadata, research reproducibility, and transport networks.

Though the call for this Special Issue is now over, based on its success, we have opted to include Resource Articles in the regular call for the journal. Hence we invite the reader who may be an author, creator and/or maintainer of high-quality resources in the area to consider submitting a description of their artefact(s) to the journal!

# NEOntometrics – A Public Endpoint for Calculating Ontology Metrics

Achim Reiz  

Rostock University, Germany

Kurt Sandkuhl  

Rostock University, Germany

## Abstract

Ontologies are the cornerstone of the semantic web and knowledge graphs. They are available from various sources, come in many shapes and sizes, and differ widely in attributes like expressivity, degree of interconnection, or the number of individuals. As sharing knowledge and meaning across human and computational actors emphasizes the reuse of existing ontologies, how can we select the ontology that best fits the individual use case? How do we compare two ontologies or assess their different versions? Automatically calculated ontology metrics offer a starting point for an objective assessment. In the past years, a multitude of metrics have been proposed. However, metric implementations and validations for real-world data are scarce. For most

of these proposed metrics, no software for their calculation is available (anymore). This work aims at solving this implementation gap. We present the emerging resource NEOntometrics, an open-source, flexible metric endpoint that offers (1.) an explorative help page that assists in understanding and selecting ontology metrics, (2.) a public metric calculation service that allows assessing ontologies from online resources, including GIT-based repositories for calculating evolutionary data, with (3.) a scalable and adaptable architecture. In this paper, we first evaluate the state of the art, then show the software and its underlying architecture, followed by an evaluation. NEOntometrics is today the most extensive software for calculating ontology metrics.

**2012 ACM Subject Classification** Computing methodologies → Ontology engineering; Information systems → Web Ontology Language (OWL); General and reference → Metrics; General and reference → Evaluation

**Keywords and phrases** Ontology Metrics, Ontology Quality, Knowledge Graph Semantic Web, OWL, RDF

**Digital Object Identifier** 10.4230/TGDK.2.2.2

**Category** Resource Paper

**Related Version** *Previous Version*: <https://ceur-ws.org/Vol-3235/paper16.pdf> [25]

**Supplementary Material** The source code for NEOntometrics is published on Github under the MIT license, where version 1.1.0 was used for the performance results presented in Section 5.3. The evaluations are available on Zenodo under a CC-BY license.

*InteractiveResource (Project Website)*: <http://neontometrics.com> [23]

*Software (Source Code)*: <https://github.com/achiminator/neontometrics> [21]

archived at [swh:1:dir:a0a2d612a4de911f171dadcefb66dcc1c5b42bd9](https://swh.1:dir:a0a2d612a4de911f171dadcefb66dcc1c5b42bd9)

*Dataset (Evaluation and Supporting Materials)*: <https://zenodo.org/records/14047141> [30]

**Received** 2023-12-01 **Accepted** 2024-10-31 **Published** 2024-12-18

**Editors** Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler

**Special Issue** Resources for Graph Data and Knowledge



© Achim Reiz and Kurt Sandkuhl;  
licensed under Creative Commons License CC-BY 4.0

*Transactions on Graph Data and Knowledge*, Vol. 2, Issue 2, Article No. 2, pp. 2:1–2:22



Transactions on Graph Data and Knowledge

TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

Ontologies facilitate the shared understanding of a domain between people and systems [15]. They allow the structuring and contextualizing of data for detecting implicit knowledge, accessing this knowledge using complex queries, and integrating and interlinking data from various sources while facilitating a common understanding.

Over time, the semantic web community developed numerous ontologies. To give a perspective, the vocabulary repository “Linked Open Vocabulary (LOV)” [33] contains 860 ontologies. The portal “ontologydesignpatterns.org” collects small, reusable ontology patterns and provides 240 artifacts. Bioportal [34], a large repository for biomedical ontologies, contains 1,140 ontologies. Moreover, many more ontologies are available on different sources like GitHub or private company repositories<sup>1</sup>.

While the number of developed ontologies is extensive, just a few means are available to assess these artifacts quantitatively. For the development team that likes to integrate an ontology into their system, it is cumbersome to numerically compare the main attributes of two or more available ontologies that serve the same purpose. For the knowledge engineer, the missing assessment capabilities hinder tracking how ontology structure evolves throughout the ontology lifetime.

As shown in the next section, the lack of means for numerical assessments does not originate from a lack of proposed metrics - over time, various metric frameworks have been developed. What is missing are practical implementations of these metrics. Without a means to put these metrics into use, further empirical research cannot proceed, and the potential of ontology metrics remains theoretical.

This work aims to close this gap by presenting a flexible, extensible metric calculation endpoint for RDFS and OWL-based ontologies. The software enables metrics to be calculated and retrieved using a graphical user interface (GUI) or an application programming interface (API). It further aids users in learning about different metrics, their calculations, and possible interpretations through an interactive explorer for ontology metrics. If several versions of an ontology are available in a GIT-based repository, the development of the metric values over time can be tracked.

NEOntometrics is the successor of the Ontometrics software [11] and its API endpoint [26]. The new software allows for evolutionary analysis, comes with visualization capabilities, calculates more metrics with improved calculation performance, and is better extensible. Parts of the NEOntometrics application have been previously published. We presented a poster of an earlier version of NEOntometrics at the Semantics Conference 2022 [25]. The metric ontology was presented at the KEOD conference 2022 [26], and the visualization capabilities at the Voilá Workshop 2023 [29]. This contribution combines the various works and extends them with (A.) a more thorough review of the state of the art, (B.) a more detailed description of the software’s capabilities and structure, and (C.) an evaluation of the practical relevance and the performance gains of the calculation engine.

The paper is structured as follows: Section two summarizes the current state-of-the-art regarding ontology metrics and calculation software. Section three presents NEOntometrics with its architecture and usage of the API and GUI. In section four, we illustrate the use of NEOntometrics by presenting a case study and means to adapt the software, followed by the evaluation of the framework and a conclusion.

---

<sup>1</sup> Accessed July 2024

## 2 Related Work

Significant for our research are metric calculation proposals and ontology metric frameworks, covered in the first part of this section, and possible calculation implementations, covered in the second part.

There are many different evaluation methods available. Please note that we only consider criteria-based frameworks that allow for an automatic evaluation based on the structural attributes of the ontology. Metrics that need human intervention or additional input parameters are not considered relevant. That excludes evaluation methods based on a gold standard (additional input parameter is a “perfect” ontology), task-based (additional input parameter is the task fulfillment level that an ontology can provide in a given context), or corpus-based (additional input parameters are domain-related documents like a text corpus). Raad and Cruz further describe the underlying categorization [20].

### 2.1 Related Quality Frameworks

Lozano-Tello and Gómez-Pérez published the Ontometric framework in 2004. It proposes evaluation attributes in five criteria: (Development) tool, (ontology) language, context, methodology, and cost. Arguably, some metrics have become obsolete due to standardization in the past years. In 2004, the web ontology language (OWL) was just released, and other representations like OIL, DAML+OIL, and SHOE were still actively used. Here, Ontometric is targeted to make the influences of the languages explicit and comparable [13]. Today though, regarding the category languages, RDFS-based ontologies can be considered state-of-the-art and are mostly compatible with each other. Further, the standardization decoupled the tools from the ontology. Thus, the tool capabilities do not influence the semantic artifact. Other proposed elements, however, can be supported by an automatic calculation, like the metric maximum depth in the category content. While the relevance today might be somewhat limited, Ontometric considerably influenced the newer frameworks.

Gangemi et al. proposed an ontology evaluation framework based on a semiotic meta-ontology  $O^2$  that provides a formal definition for ontologies and their usage. Further, the authors define an ontology evaluation design pattern (oQual). Based on their  $O^2$  definition, measurements assessing structural, task, corpus, and usability-based attributes are proposed [8]. A technical report by the same authors further suggested 32 metrics in seven categories assessing mostly graph-related structures like depth, width, modularity, the distribution of siblings, or tangledness [9].

In 2005, Burton-Jones et al. presented the semiotic metric suite. It comprises four main categories (syntactic, semantic, pragmatic, and social quality) and ten quality metrics. While some of these metrics are based solely on the structure of the ontology itself, others need further additional external information. Nine of these measurements, in theory, can be calculated automatically [2]. Practically, some of the required data for some measures will probably not be available. Examples are the access count of the ontology or the links from other ontologies to the currently assessed one.

OQuaRE was first proposed in 2011 by Duque-Ramos et al. It has since been used in several publications, always involving the core team of the proposing authors. OQuaRE offers 19 calculatable metrics and associates these metrics with quality dimensions like *readability* or *accountability*. Further, the framework ties metric results to quality ratings, thus providing an interpretation of the measurements. This holistic approach to quality is a unique characteristic among the frameworks [6]. However, during implementation, we experienced several heterogeneities in the metric definitions of the framework, with metrics having the same name, created by the same authors, being defined differently in their publications. To facilitate further research on

■ **Table 1** Metric frameworks with their first publishing date and citations (Semantic Scholar, July 2024).

Framework	[13]	[8]	[2]	[6]	[32]	[35]	[14]
Published	2004	2006	2005	2011	2007	2005	2010
# Citations	424	356	370	115	356	180	52

the framework and to integrate the framework into NEOntometrics, we reworked the OQuaRE measurements. However, empirical research with the newly implemented metrics showed that their proposed linkages from measures to ontology quality scores do not sufficiently work [27]. Our study was the first made by authors not part of the team that proposed the framework; however, the NEOntometrics applications shall allow more thorough analysis in the future.

Tartir et al. published 19 metrics in the OntoQA framework in 2005 and 2007. While the framework does not provide a grading system for metrics like OQuaRE, it aids the interpretation by describing how modeling decisions influence the metric results. Further, the authors propose measurements applicable not to the ontology as a whole but to the elements in an ontology. OntoQA also defines class- and relation-specific measurements. The relationship importance, for example, is calculated once for each relationship [32, 31].

There are further metric frameworks that consider the cohesion of an ontology. Yao et al. propose a set of measures based on an inheritance tree [35]. In a consecutive paper, the authors further provide an empirical analysis and interpretation context [17]. Ma et al. examined the ontology partitions with special consideration of the open world assumption [14].

Over the years, a lot of frameworks have been proposed. As Table 1 shows, these papers have gathered many citations over the years. Some of these frameworks are merely theoretical in their proposals; others came with prototypical implementations. Further, tools that do not correspond to one of the proposed frameworks have been developed.

## 2.2 Related Metric Calculation Software

The following section shows historical and current software for ontology metric calculation.

OntoKBEval by Lu and Haarslev [19] analyzes the structure of ontologies by providing graph-related measures like the number of levels or the number of concepts per level. The tool offers means to grasp clusters in the ontology and developed its own visualization “Xmas”-tree.

Tartir et al. developed a standalone java application for the OntoQA framework [32], implementing measures of the OntoQA framework, including metrics for the individual classes.

OntoCat, proposed by Cross and Pal [3], is a plugin for the Protégé editor and provides size- and structure-related metrics. They allow the assessment of the ontology as a whole but also provide metrics concerned with specific subsets of the given ontology.

S-OntoEval by Dividino et al. [4] serves as a calculation tool for, among others, the framework of Gangemi et al. Its primary focus is on structural evaluation. However, the tool also calculates usability based on annotations and task performance based on ontology querying.

The Protégé editor [16] offers basic metrics on its landing page that counts the usage of OWL-specific language constructs like the number of object property domain declarations or the number of classes.

The developers of the OQuaRE framework introduced a web tool to calculate their proposed metrics. It integrates a statistical correlation analysis of the metrics and a web service. The tool suffers from the same issues as the framework [27], and the implemented metrics are heterogeneous and do not adhere to a clear definition.

Amith et al. developed the Semiotic-based Evaluation Management System (SEMS), later renamed OntoKeeper [1], which implements the semiotic suite by Burton-Jones et al.

■ **Table 2** The availability of the developed metric software (type: S: Standalone, P+: Protégé plugin, API: REST-API, WT: Web Tool).

Tool	Date	Type	Available	Open Source	Ref
Onto-KBEval	2006	S	No	No	[19]
OntoQA	2005	S	Yes	(No) <sup>2</sup>	[32]
OntoCat	2006	P+	No	No	[3]
S-Onto-Eval	2008	S	No	No	[4]
Protégé	2015	S	Yes	Yes	[16]
OQuaRE	2018	WT, API	Yes <sup>3</sup>	No	[27]
Onto-Keeper	2017	WT	No	No	[1]
OOPS	2012	WT, API	Yes <sup>4</sup>	No	[18]
OntoMetrics	2015	WT, API	Yes <sup>5</sup>	No	[22, 11]

The “Ontology Pitfall Scanner” (OOPS) by Poveda-Villalón et al. [18] approaches automatic ontology evaluation differently: They detect common modeling pitfalls like the use of *is* relationship instead of *rdfs:subClassOf* or wrongful equivalent relations.

OntoMetrics, first developed by Lantow [11], is a web service for calculating several ontology metrics. It covers most of the OntoQA and oQual ontology metrics and integrates the OWL-based axiom counts that are also part of Protégé. It was later extended with a web service by Reiz et al. [22].

### 2.3 The Need for Another Calculation Tool

As the previous section has shown, many frameworks and tools have been developed over the past years. That raises the question of whether a new calculation tool is necessary. We argue that our application fills essential gaps:

**Missing Practicality.** As Table 2 shows, most of the developed tools are no longer available. Even if they are available, their usability is often low. Many of the tools were used for the authors’ evaluation efforts and do not come with a state-of-the-art user interface. Further, most of the software is not maintained. This problem is amplified by the fact that most of the software is:

**Closed Source.** None of the evaluated tools is fully open source (cf. Table 2). Not only hinders this reproducibility. It also prevents the community from maintaining the software and building on this previous research. If there is a need for another kind of evaluation, one has to start from scratch. We, thus, argue that the closed source leads to:

**Isolation.** The implementation efforts have stayed mainly isolated from one another. Hardly any tool has reached a broad acceptance within the community, and the ontology evaluation efforts of researchers using different tools are often not comparable. While there is a consensus that ontology evaluation is meaningful, there is no common understanding of how to do it.

<sup>2</sup> <https://github.com/Samir-Tartir/OntoQA.Thebinary.jarfilesareavailableunderCClicense>.  
Thesourcecodeitselfisnotpublic.

<sup>3</sup> <http://sele.inf.um.es/ontology-metrics>

<sup>4</sup> <http://oops.linkeddata.es>

<sup>5</sup> <https://ontometrics.informatik.uni-rostock.de,opi.informatik.uni-rostock.de>

■ **Table 3** A comparison of the existing OntoMetrics, its API and the new NEOntometrics software.

	NEOntometrics	OntoMetrics/OPI
GUI	Flutter / Material Design	OntoMetrics: Static Web Page
API	GraphQL	OPI: REST
Technology	Microservices	Java Webpage
Evolutional Analysis	Public Git Repositories	No
Async	Yes	No
Metrics	159	72 <sup>7</sup>
Extensibility	Good	Poor
Open Source	Yes	No
Performance	Fast (cf. Section 5.3)	Moderate

### 3 NEOntometrics

NEOntometrics is the successor of the Ontometrics tool [22, 11] (thus NEOntometrics). The old Ontometrics is one of the few ontology evaluation tools still available, but it does not scale well, provides fewer functionalities, sometimes redundant calculations, and is complicated to adapt [26].

The new tool, NEOntometrics, is a complete overhaul of the old software. It consumes public GIT-based repositories, iterates through all of the commits (a commit is a published change in a repository), and calculates the metrics of the available ontology files. The software seeks to solve many of the previously named challenges. It comes with a state-of-the-art user experience, a GraphQL endpoint, calculates various metrics, is quickly extensible through an ontology for creating and describing metrics, and is open source under the MIT License<sup>6</sup>. Table 3 further shows the application differences.

The following section details the software itself: It presents the different components of the service, how they interact, and the underlying development decisions. We also present how our ontology-based metric calculations are extensible for future usage. Afterward, we give an overview of how to put the software to use.

#### 3.1 The Architecture of the Metric Calculation

One design goal was to create a flexible application for integrating new metrics. A researcher shall be able to adapt the application to their needs and quickly implement new required metrics.

To achieve this adaptability, we did not encode all of the metrics of the various frameworks directly in the software but decomposed them into their building blocks. For example, the metric *Axiom/Class Ratio* is not calculated during the ontology analysis. Instead, their building blocks *Axioms* and *Classes* are analyzed and saved in the database. The compositional values are then calculated at the time of querying.

The information on the ontology metrics is stored in an OWL-based metric ontology<sup>13</sup>. On startup of the application, multiple SPARQL-queries extract the codified knowledge and set up the backend and frontend. Thus, changing and adapting the ontology is sufficient to adapt the measures. The work [26] further details the underlying metric ontology.

Figure 1 presents an example of the metric elements in the ontology. *Elemental Metrics* contain the atomic measures that are used to build the compositions. For *Axiom Class Ratio*, the *Elemental Metrics* are *Axioms* (the number of axioms) and *Classes* (the number of classes). The

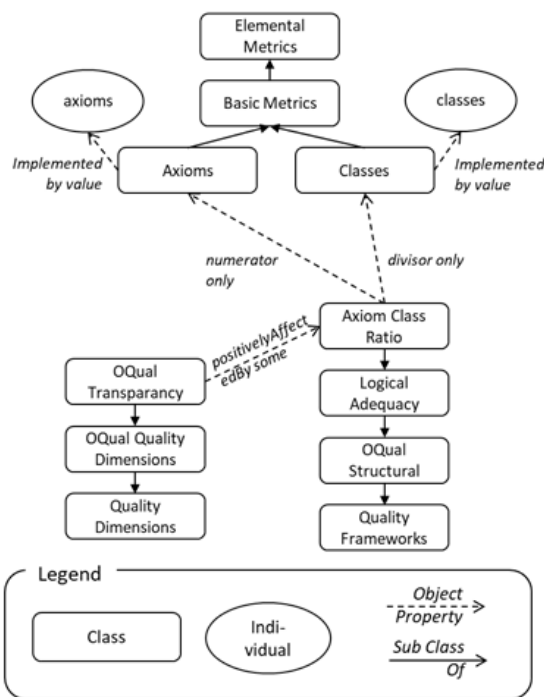
<sup>6</sup> <https://github.com/achinator/NEOntometrics>

<sup>7</sup> As some frameworks propose similar measurements, not all of the metrics are unique.



■ **Table 4** The Metric Frameworks that are implemented in NEOntometrics.

Name in NEOntometrics	Proposed By	Ref
Cohesion Metrics	Yao et. al	[35]
Complexity Metrics	Zhang, Ye, Yang	[37]
Good Ontology	Fernandez et. al	[7]
OQuaRE	Duque-Ramot et. al	[6, 27]
OQual	Gangemi et al.	[9, 8]
OntoQA	Tartir, Apinar	[32, 31]
Complexity Cohesion	Orme, Yao, Etkorn	[17]



■ **Figure 1** The metric ontology (image adapted from [26]).

ontology further specifies mathematical relationships between the metrics. In the given example, *Axiom Class Ratio* is the *subClassOf* (*divisor only Classes*) and (*numerator only Axioms*). The *Elemental Metrics* are connected to metric instances named identically to the implementation names in the calculation service and the elements in the database. In the example of the *Axioms*, this element has a relationship *implementedBy value axioms*.

All elements have rich annotations, providing human-centered meaning to the metrics. Some elements have links to further online resources or scientific publications. The annotations are the foundation for the *Metric Explorer*, where users find guidance on the available metrics.

New metrics that build upon the available *Elemental Metrics* can be created by modeling them in the ontology. Upon start, the application will make these custom metrics automatically available in the front- and backend. Table 4 shows the frameworks that are already implemented in NEOntometrics and part of the *Metric Explorer* and the *Calculation Unit* at the time of publication. The case study in Section 4.2 details how to create new frameworks, e.g., for individual metric frameworks in an organization.

### 3.2 The Architecture Of The Application

The application is based on a dockerized microservice architecture and consists of five components: the calculation-unit OPI (Ontology Programming Interface), the API, the worker application, a database for storing the calculated metrics, and a Redis interface for queueing jobs. The API and worker share a common codebase. Figure 3 depicts the interaction of the involved services.

The **frontend** contains the GUI. It is written using the multi-platform UI language *flutter* with its underlying client language *dart*<sup>8</sup>. Upon loading, the frontend first queries the API for available ontology metrics based on the metric ontology. This data fills the help section *Metric Explorer*, which allows users to inform themselves about the various available metrics and the options for the calculation page. Afterward, the user can retrieve the requested ontology metrics or put them into the queue if they do not yet exist.

The **API** is the django-based<sup>9</sup> endpoint for accessing already calculated metric data or requesting the analysis of new repositories. During the startup of the software, the application queries the metric ontology. It builds the frontend data and dynamically creates calculation code to provide the measurements of the frameworks that build upon the Elemental Metrics. After startup, a client can exploit GraphQL to check whether the data he requests exists already in the database. If so, he is able to retrieve all the selected metrics for a given repository. If not, it is possible to put the calculation of a given repository in the queue and track its progress.

The **worker** is responsible for the calculation of the metrics itself. It checks whether jobs are available in the **scheduler** Redis database. If that is the case, it starts the analysis by first cloning or pulling the GIT-repository, then iterating through every new file and commit, analyzing the ontologies using the **OPI** metrics endpoint. Afterward, the calculation results are stored in the **database**. The scheduling mechanism is based on *django-rq*<sup>10</sup>. Even though the worker shares a code base with the API, it runs as a separate application. The number of parallel calculations can be scaled by increasing the number of workers.

The calculation service **OPI** is responsible for calculating metrics out of ontology documents. While it is based on the calculation service published in [22], most underlying code has been replaced. The old application struggled with ontology files larger than 10 MB due to inefficient memory allocation and had no separation of the calculation of the elemental metrics and the composed metrics of the metric frameworks. The old application was designed as standalone software, while the new calculation engine is hidden from the user and only accessed by the **API**.

The backend utilizes two languages: The **API** is written in python, and the calculation service **OPI** builds on Java. While the two languages add complexity to the application design, they allow the use of established frameworks for their given tasks.

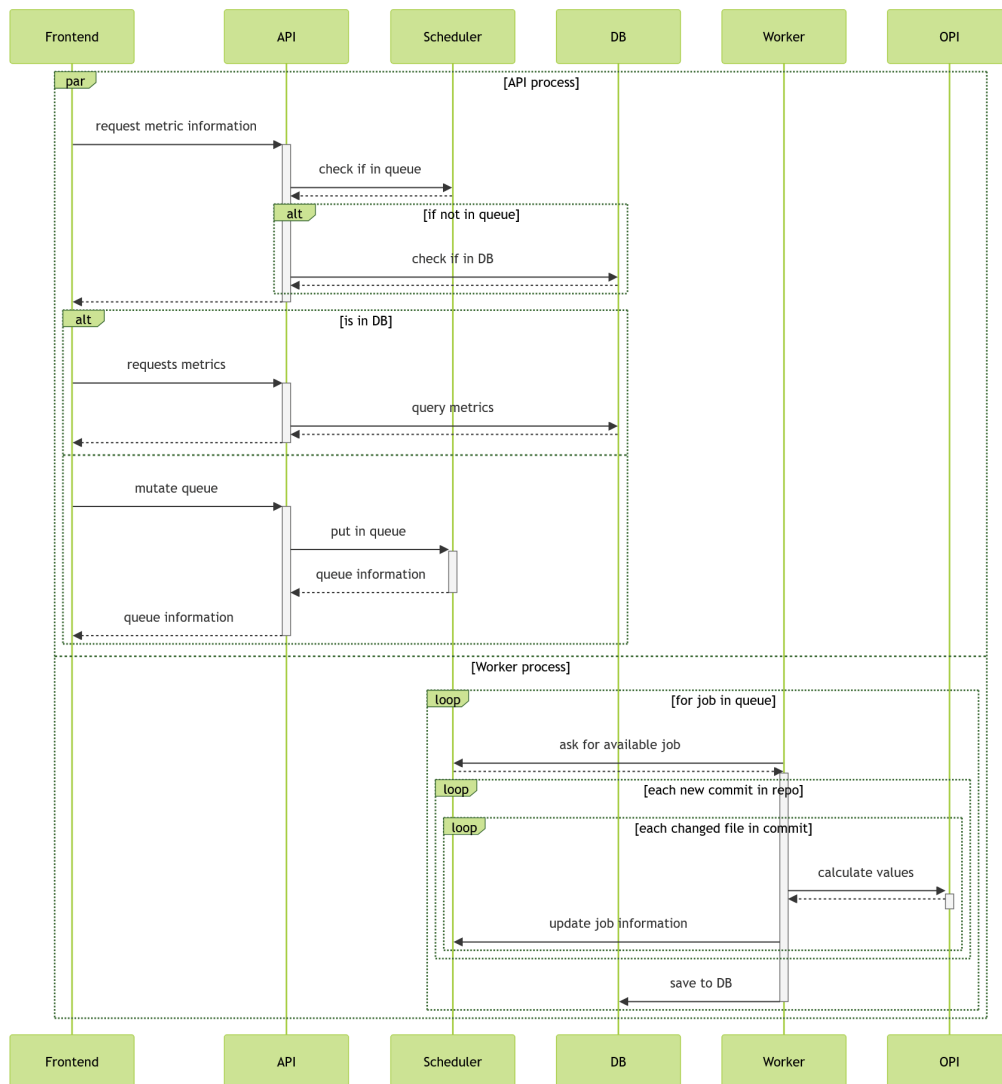
The calculation and retrieval process as a whole is depicted in Figure 2. At first, the frontend requests whether an ontology is already known in the system. If not, it either returns the queue information to the end-user or starts another request for the ontology metrics. At the same time, the worker applications and OPI handle the queued tasks.

The microservice architecture allows utilizing the strengths of the various languages. The Java calculation unit builds on the OWL-API for an efficient graph traversal. The Python/django application is easily extensible through the availability of multiple plugins, e.g., for the GraphQL endpoint and the asynchronous metric calculation. The flutter-based frontend comes with built-in material design support. Further, the microservices allow for potential horizontal scaling of the calculation.

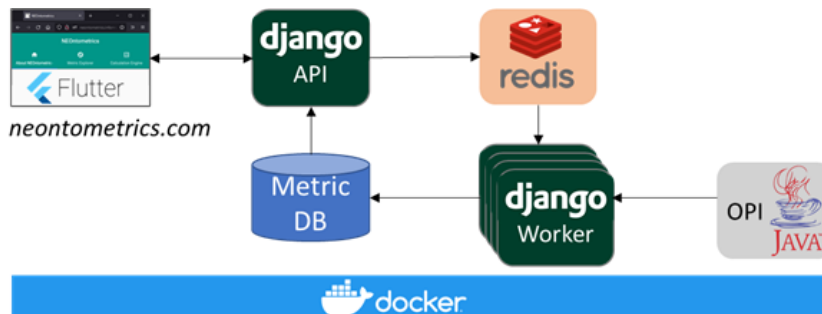
<sup>8</sup> <https://flutter.dev/>, <https://dart.dev/>

<sup>9</sup> <https://www.djangoproject.com/>, <https://www.django-rest-framework.org/>

<sup>10</sup> <https://python-rq.org/patterns/django/>



■ **Figure 2** The process of analyzing and retrieving ontologies with NEOntometrics (without application startup).



■ **Figure 3** The NEOntometrics microservice architecture.

### 3.3 The Metric Explorer

The page *Metric Explorer* is a dynamic help page of available metrics in NEOntometrics. The two main categories are *Elemental Metrics* and *Quality Frameworks*. The former contains the underlying atomic measurements of the ontologies. The authors of NEOntometrics create all the information shown in this category. *Quality Frameworks*, on the other hand, present the ontology metrics developed by other researchers, like the OntoQA Metrics [32] by Tartir et al., shown in Section 2.1. Here, all information originates from the authors of the given frameworks.

The *Metric Explorer* provides information on five categories (though not all are filled for all the metrics). *Metric Definition* contains the formal definition of the metrics and how they are calculated, while *Metric Description* supplements a more human-readable explanation and, at times, an example. The *Metric Interpretation* guides practical usage. *Calculation* explains their decomposition into the *Elemental Metrics* using the metric names that are returned by the API, and *seeAlso* links to further resources like the corresponding papers or additional reads.

The interactive help page is closely bound to the metric ontology. Every change will be reflected in the *Metric Explorer* after a restart, and the ontology provides annotation properties for the *metric definitions*, *descriptions*, and *interpretations*. The nesting of the measurements is defined by the subclass relations in the ontology, and the *calculation* field is defined by the object properties representing the mathematical relationships (cf. Figure 1).

### 3.4 A Frontend for Humans and an Interface for Machines

For direct consumption by a user, the tab *Calculation Engine* (as shown in Figure 5) is the main entry point for the metric calculation. The end-user first selects the required metrics. The “Already Calculated” button shows the calculated repositories already stored in the database. While these repositories can be a starting point for further exploration, the user can also place a URL in the textbox that points to a new public GIT repository or the location of an ontology file.

A click on the arrow starts the metric request. Once the data is analyzed, clicking the arrow leads to the metric results presented as a paginated table, representing the metric values for the different ontology versions. A drop-down menu in the header allows selecting the various ontology files, and the download button exports the metrics into a .csv. A click on the button “*show the analytic*” opens internal visualizations for displaying the ontology evolution, comparing the various ontologies in a repository, and assessing the recent changes.

One goal of NEOntometrics is to allow the integration of ontology metrics into semantic web applications, which requires exposing the service using a standardized interface. Relevant open standards are REST and GraphQL for the web and SPARQL for querying the semantic web. NEOntometrics builds on GraphQL.

GraphQL (together with REST) has become a new de facto standard for sharing information on the web, and there is broad support in various programming languages and frameworks. This support includes the django web framework used in this project, where the graphene plugin<sup>11</sup> allows utilizing the internal Object Relational Mapping. It allowed us to build the interface with comparatively little implementation effort, as the requests are translated to database queries automatically. While these integrations are also available for REST, GraphQL allows for the traversing of relationships and the precise selection of attributes for querying. Avoiding over-fetching is highly relevant for this use case, as one ontology version has over 100 ontology metrics, and the user likely selects just a few.

---

<sup>11</sup> <https://graphene-python.org/>

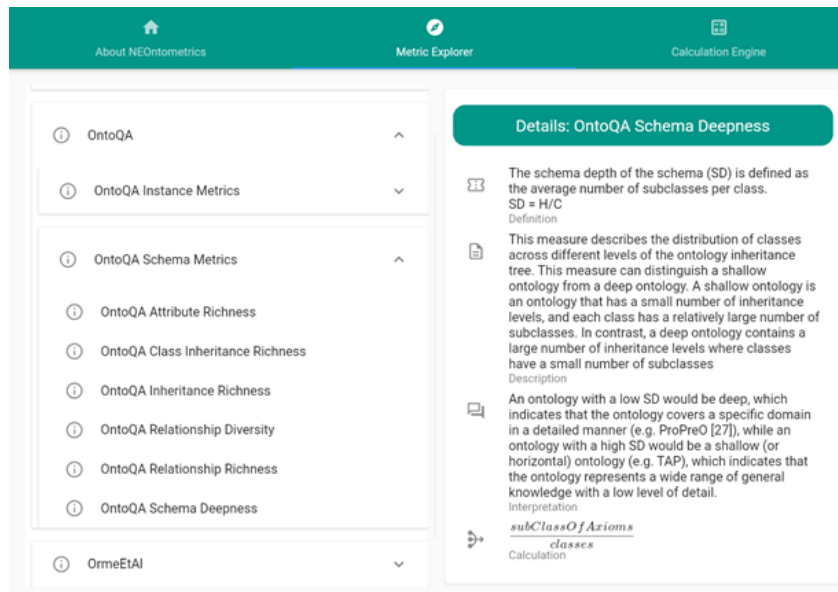


Figure 4 The Metric Explorer page in NEOntometrics.

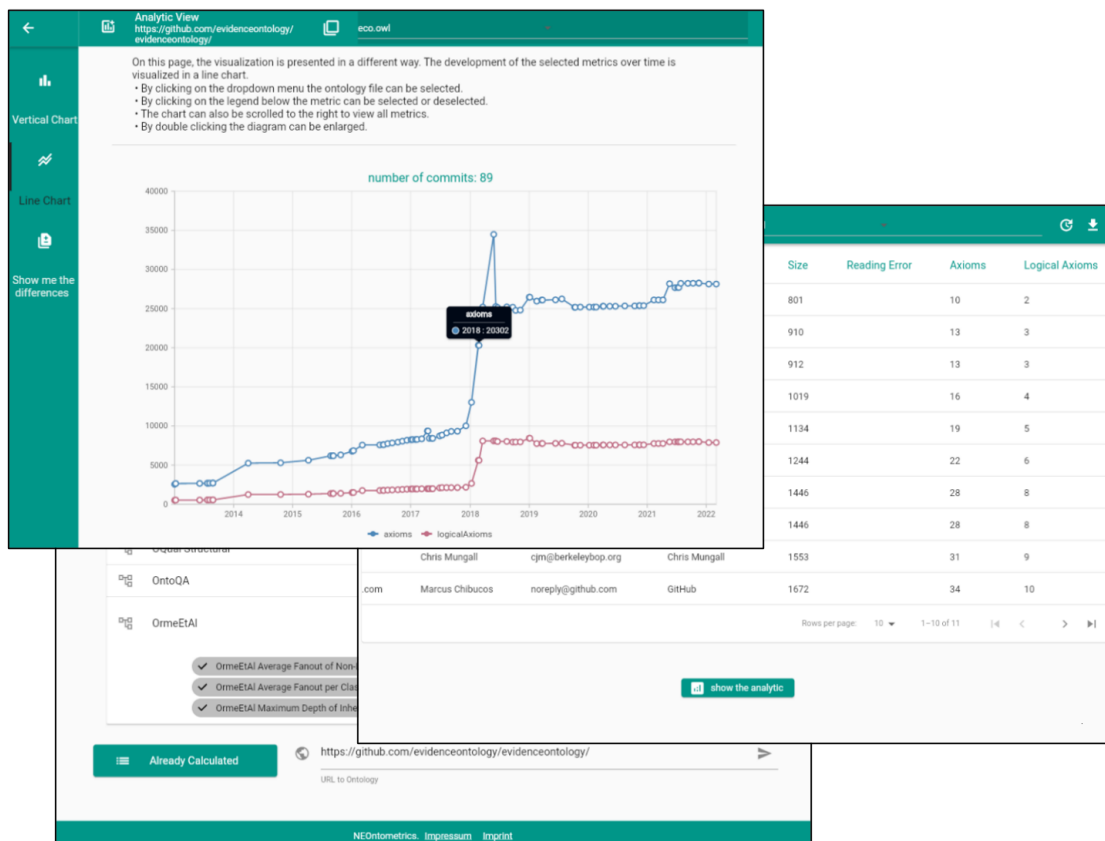


Figure 5 The NEOntometrics Frontend.

The GraphQL endpoint further provides documentation on the various available requests and possible return values, thus enabling the guided development of new queries. The interface is accessible through a browser on a GraphiQL interface or any other GraphQL client.

SPARQL, as a graph-based query language, has similar attributes to GraphQL regarding relationship traversal and attribute selection. Additionally, proving a SPARQL endpoint would further allow the integration of the metric calculations into existing knowledge bases. Unfortunately, there is (currently) no support in the form of plugins for integrating such an endpoint into the used django framework. This lack makes the creation of such an endpoint dissimilar costlier.

### 4 Bringing NEOntometrics Into Use

The following presents application scenarios for the NEOntometrics application. The first case study shows the potential of analyzing ontology evolution. The second part presents possible integration and adaptation scenarios for NEOntometrics.

#### 4.1 Analyzing Ontology Evolution with NEOntometrics

Analyzing ontology metrics over time can tell a lot about underlying design decisions. The size of the changes indicates if an ontology evolves gradually or has disruptive changes, thus measuring stability and identifying the disruptive changes. They also allow us to assess how attributes like the logical complexity (e.g., measures through the number of axioms that incorporate meaning), the coverage (e.g., measured through the number of classes or individuals), or the shape of the graph (e.g., measured through depth or breadth) change over time.

This case study analyzes the *Evidence and Conclusion Ontology (ECO)*. *ECO* captures the biological coherences like “gene product X has function Y as supported by evidence Z” [10]. The NEOntometrics authors have no affiliation with the authors of the ontology nor with the biomedical field of research. Further, the goal of the section is not to evaluate quality but to observe the development of the ontology over time to give an impression of possible assessments. Previous work discussed the connection between metrics and development decisions from an ontology engineering perspective [24, 36].

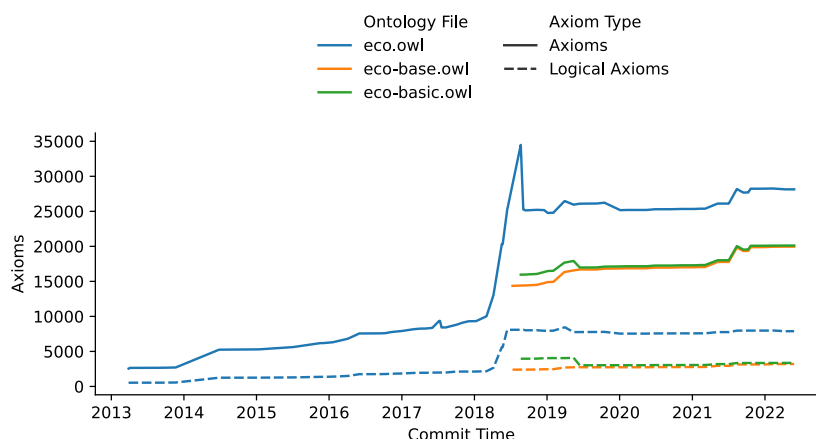
The *ECO* repository has 856 commits in 17 ontology files. For this analysis, we were interested in the main ontologies in this repository. Thus, we only assessed the ontologies in the root structure, resulting in three ontology files. *eco.owl* with 89 versions, *eco-basic.owl* with 44, and *eco-base.owl* with 45 versions. We first examined the axiom count of the ontologies and then used Tartir et al.’s OntoQA framework [32, 31] for further analysis. The corresponding source files in Jupyter Notebooks are available online<sup>12</sup>.

The first analysis is concerned with the development of the ontology size. Figure 6 presents the three ontology files in their different versions and plots the development of axioms with time. While the solid line represents all axioms overall, the dashed line only accounts for such that incorporate a logical meaning in RDFS or OWL syntax. The difference between the dashed and the solid lines are, thus, annotations or custom-defined properties.

An insight of the chart in Figure 6 is the variances of the logical axioms and the axioms in general. While the size of the ontology overall fluctuates intensely, the number of parts of the ontology that incorporate logical meaning stays relatively stable. One significant spike occurred between 2018 and 2019, which we will scrutinize further. Analysis reveals that a more extensive restructuring of the ontology drives this increase in logical axioms. The classes in *eco* doubled

---

<sup>12</sup> [doi.org/10.5281/zenodo.14047141](https://doi.org/10.5281/zenodo.14047141)



■ **Figure 6** The change of axioms over time in the ECO-ontologies.

from around 900 to a little over 2,000, then increased to over 3,000. The number of defined object properties jumps from three to above 50, and the relation on classes through object properties increases from 350 to almost 2,500, then drops to around 1,600. This change event also marked the introduction of *eco-base* and *eco-basic*.

Figure 7 shows the relationship richness and schema deepness defined in the OntoQA framework. The former is defined as the number of non-inheritance relationships divided by the sum of non-inheritance relationships and inheritance relationships, and the latter is the number of subclasses per class [32].

Figure 7 (left) shows that, after an initial increase due to the rise in object properties, the relationship richness of *eco* drops with the increase in classes and `subClassOf` statements. Also, object properties were introduced. Later, a decline in object properties, combined with the further increase in classes and `subClassOf` statements, partially reverses the growth.

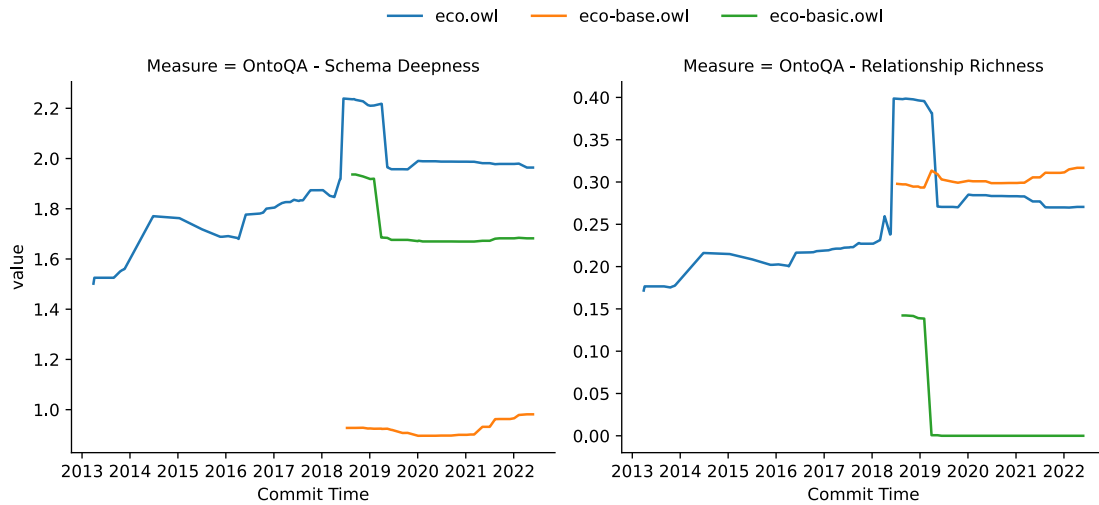
The right diagram in Figure 7 provides more insights into the role of sub-class relationships. At first, a lot more `subClassOf` relationships than classes were introduced. However, the number of `subClassOf` relations later stagnated, even getting smaller. In contrast, the number of classes increased steadily. This suggests that the rebound in the relationship richness is driven more by the decline in object properties than the increase in `subClassOf` relationships. While the number of logical axioms is more or less stable, the underlying logical attributes of the ontology that constitute how the ontology is structured are still subjected to changes.

There are many more aspects that one could analyze for the given repository, and this section is merely a short demonstration of the value of environmental metrics. As the last diagram indicates, many more fluctuations are worth looking at. The variations affect the relationships between non-hierarchical and hierarchical relationships, classes, and graph-related structures like the width or depth, individuals, or data properties.

## 4.2 Adapting NEOntometrics by Adapting the Metric Ontology

A recent empirical analysis of 69 ontology evolution processes (based on NEOntometrics) has shown that the developments are highly heterogeneous and that assumptions on stereotypical development processes do not apply: There is no common rule or joint history that ontologies share [28]. If the ontologies are highly diverse, so is the required evaluation. This diversity of ontologies and their metrics emphasizes the careful selection of the latter. One person might

## 2:14 NEOntometrics – Calculating Ontology Metrics



■ **Figure 7** The development of OntoQA’s Relationship Richness and Schema Deepness.

build a taxonomy with rich human-readable annotations and other targets to infer knowledge by modeling complex class relationships. A successfully applied metric by the first person might not work for the second. While the Metric Explorer supports the selection process, a person might develop their measure to intertwine two metrics in a way that has not been done before. Organizations may want to select and reorder the metrics or limit the display to only relevant ones. The following subsection explains how to adapt the application by altering the metric ontology. While every ontology editor can be utilized for editing, this section builds on the open-source software Protégé [16]. The metric ontology is stored in the GitHub repository<sup>13</sup>

### 4.2.1 Restructuring the Ontology Metrics

The two classes *Elemental\_Metrics* and *Quality\_Frameworks* are at the core of the metric calculation. The former represents the measurable ontology attributes and their implementation in NEOntometrics, and should only be changed if there is a need to define additional measurable ontology attributes. The *Elemental\_Metrics* subclasses are essential for the startup of the application, and their alteration or deletion can lead to undesired behaviors. Thus, the individualization effort should occur in the subclasses of the *Quality\_Frameworks*.

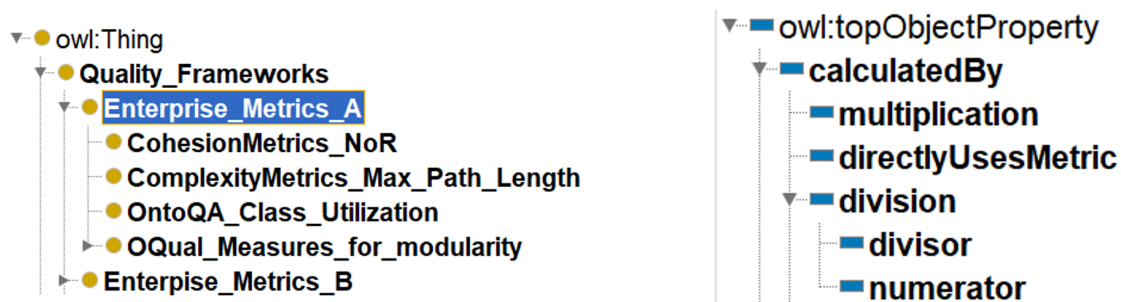
Reusing the existing metrics is possible by creating new, individual subclass structures for dedicated purposes. After restart, the software reads the new structure and injects it into the code. As an effect, the frontend displays the new categories, and the new subgroups can be quickly selected, reducing the complexity for the metric consumer. The example of Figure 8 illustrates custom-ordered metric categories in Protégé.

### 4.2.2 Creating New Ontology Metrics

The currently implemented quality frameworks based on the literature (cf. Table 4) already provide various metrics covering many use cases. However, reusing the existing metrics might not be sufficient depending on the individual challenges. In these cases, creating custom metric classes can provide a possible solution.

<sup>13</sup> <https://raw.githubusercontent.com/achinator/NEOntometrics/master/Git-Extension/rest/metricOntology/OntologyMetrics.owl>





■ **Figure 8** Screenshots Protégé: Left: Example of custom quality frameworks, Right: The formalized mathematical relationships for connecting the subclasses of *Quality\_Frameworks* to *Elemental\_Metrics*.

The ontology provides formalized properties for the extension of the ontology. The annotation properties *MetricDefinition*, *MetricDescription*, and *MetricInterpretation* fill the respective fields in the Metric Explorer (cf. Figure 4) to help the metric consumer to select suitable measures.

The object properties facilitate the connection of the reused or self-created *Quality\_Frameworks* to the *Elemental\_Metrics* and are the backbone for setting up the calculation unit. The subclasses of *calculatedBy* contain relations to describe the mathematical calculation operations of the application (cf. Figure 8).

The relation *directlyUsesMetric* states that a metric from a quality framework directly accesses an *Elemental\_Metric*, e.g., the *OQual\_Absolute\_Depth* metric is a subclass of *directlyUsesMetric only Total\_Depth*. Commutative operations like sum or multiplication are combined using the AND operator, e.g., *sum only (Subclasses\_Of\_Thing and Super\_Classes)*. Division and subtraction have further subclass for linking the elements. The mathematical relationships can be nested to create more complex queries.

As an example, the class with the name *Average\_Paths\_Per\_Concept*, having a relationship *SubClassOf (divisor only Classes) and (numerator only (sum only (Subclasses\_Of\_Thing and Super\_Classes)))* is first connected to the names of the given implemented database fields, represented by the connected individuals. Afterward, it is injected into NEOntometrics as:

$$\frac{\text{rootClasses+superClasses}}{\text{classes}}$$

## 5 Evaluation

While Section 4 focused on demonstrating the applicability of NEOnto, this section describes how the systematic evaluation of NEOnto against the objectives motivating our research was performed. As indicated in the introductory part of this paper, the development of NEOnto aimed at (a) supporting understanding and selecting ontology metrics, (b) a public metric calculation service that allows assessing ontologies from online resources, including calculation of evolutionary data, and (c) with a scalable and adaptable architecture. Section 5.1 introduces the evaluation strategy applied, Section 5.2 summarizes the different evaluation episodes and their results, and Section 5.3 presents an additional evaluation episode focusing on performance improvement.

### 5.1 Evaluation Strategy

The aims motivating the development of NEOntometrics express the importance of creating an approach that is mature enough to be applied in the ontology engineering community without substantial development efforts. Our assumption is that the more an approach has been evaluated

■ **Table 5** Validation steps according to Lincoln and Guba.

	<b>Theory</b>	<b>Practice</b>
<i>Internal, Development Team</i>	Validation against state of research, internal consistency checks	Prototype implementation for checking feasibility, test in lab environment
<i>External, in validation context</i>	Peer-review of publications describing approach and concepts, comparison to known best practices of the domain.	Case studies with application partners using the artifacts for evaluation purposes, Application of the developed artifacts in cooperation / under instruction from developers
<i>External, in application context</i>	Development of extensions or enhancements of the concepts and approaches by external actors Application of the artifacts for creation of new theoretical knowledge, Comparison with related approaches	Use of the artifacts developed (e.g. algorithms, methods, software components) for solutions

in theory and practice, the more mature and useful it is. Among the many scientific approaches for evaluating research results, we base our evaluation strategy on the work of Lincoln and Guba [12, p. 289 ff.] on “naturalistic inquiry”.

Lincoln and Guba distinguish between theoretical and practical validation. Theoretical validation means assessing an approach within the theories of the domain to which the approach is part or supposed to contribute. In the context of ontology metrics, this means assessing the soundness, feasibility, and consistency within the body of knowledge, such as ontology engineering and knowledge engineering. Practical validation encompasses all kinds of application of the approach for evaluation purposes, which requires defined procedures and documenting results. This could be simple lab examples illustrating the approach, controlled experiments in a lab setting, applications in industrial cases, etc.

Furthermore, Lincoln and Guba also consider the context of validation and distinguish between validation by the approach’s developers in their internal environment, validation by the developers outside the internal environment, and validation by actors other than the developers. Combining these two perspectives leads to a two-by-three matrix, as depicted in Table 5. The cells of this table show typical ways of validation for the different combinations of the two perspectives.

Usually, validation starts on the “internal, development team” level with validation in theory followed by validation in practice, and proceeds “downward” in the matrix with alternating theory and practice validation to “external, in application context”. Thus, the highest validation status would be reached if all cells in the matrix were covered.

As described above, Lincoln and Guba focus on validating research results, i.e., to check whether or not a certain result from research is appropriate for its purpose. In our case, the purpose of NEOntometrics is defined by the objectives (see introduction to this section 5). Validation and evaluation, even though different from each other, are very much linked. Evaluation is the process of assessing (and often computing) key characteristics of the research results, which can be used for validation purposes. As many of the NEOntometrics objectives require measurements instead of only checking characteristics, we use the term evaluation episodes in the next section.

## 5.2 Evaluation Episodes

Table 6 shows the performed validation episodes following Lincoln and Guba’s naturalistic inquiry framework. This section aims to summarize the intention and results of the different episodes. To emphasize the importance of the usefulness and applicability of the tool for the ontology

engineering community, we put a focus on external validation. Many external validation episodes were published in peer-reviewed publications, and peer-reviewing was also used as an instrument for external validation. In total, six publications contribute to the evaluation steps.

**Internal, Development Team.** Internally, we first validated that the results of the newly designed measures are consistent with the ones from the old application. This shall ensure continuity for analyses regardless of the used calculation backend. Further, the shortcomings of the predecessor OntoMetrics motivated the creation of the NEOntometrics tool. The technical shortcomings of the predecessor are described in section 3 and originated from a practical application with an industry partner, resulting in the feature list of Table 3. Regarding the practical evaluation, we tested the performance of the reworked measures. The results are shown in 5.3.

**External, in Validation Context.** Two papers are part of the external validation context: The homogenization of ontology metrics in the metric ontology [26] and an early validation of the usefulness of analyzing evolutionary ontology metrics [24].

The metric ontology (cf. Figure 1) was presented at the Conference on Knowledge Engineering and Ontology Design [26]. While many frameworks have been published over the years (cf. Section 2.1), no common language exists for naming measured elements. That led to similar metrics included in different frameworks, which is only apparent after close examination. The metric ontology homogenizes the various notions into one machine-readable notation that serves as the backbone of the NEOntometrics application and is served through the Metric Explorer.

A practical validation of the usefulness of analyzing evolutionary ontology metrics was performed before the development of the NEOntometrics tooling and presented at the Business Informatics Research Conference [24]. This paper showed the potential of analyzing changes in ontology metrics over time by giving an abstract yet objective account of how development decisions influenced the ontology structure. Also, several obsolete axioms and areas that had not been developed further were identified.

**External, in Application Context.** For the theoretical validation context, NEOntometrics was extended with a visualization capability [29]. In a practical setting, the tool examined various ontology versions to identify stereotypical development behavior [28], investigated and invalidated the broad quality claims made by the OQuaRE framework [27], and analyzed semantic media wikis [5].

One theoretical validation in the application context was performed as part of the Voilá Workshop for ontology visualization [29]. Here, we described how newly added visualization features can be used to compare various ontologies in a repository, analyze the evolution of one ontology throughout its lifetime, and a feature to compare the changes between the two most recent versions.

For the practical validation, we looked out for active use of the tool in research papers. First, the tool scrutinized stereotypical development behavior in dormant ontologies, published as part of the invited extended conference papers on Knowledge Engineering and Ontology Development (KEOD) 2022 [28]. Using public git repositories, 7,053 ontology versions of 69 ontologies showed the heterogeneous development of ontologies throughout their lifetime. We could invalidate many commonly held assumptions with the numerical assessments, like “*ontologies tend to get larger over their lifetime*” or “*ontologies get more complex with increasing maturity*”.

The next analysis concerned the quality statements of the often-cited OQuaRE framework; this analysis was published as an extended version of the contribution of the International Conference on Information Systems (ICEIS) 2022 [27]. OQuaRE not only recommends ontology measures but

■ **Table 6** Evaluation Steps according to Lincoln and Guba’s Naturalistic Inquiry Framework.

	<b>Theory</b>	<b>Practice</b>
<i>Internal, Development Team</i>	Internal Checks for consistent Measurement of the old and new metric calculation engine. Feature requirements out of the predecessors’ shortcomings. (cf. Section 3.1)	Test of calculation performance (cf. Section 5.3).
<i>External, in validation context</i>	Metric Ontology containing the implemented ontology metrics [26].	Case-Study on the value of evolutionary metric analysis [24].
<i>External, in application context</i>	Extending the software with metric Visualization Capabilities [29].	Analysis on stereotypical ontology evolutionary processes [28]. An empiric examination of the OQuaRE quality claims [27]. Evaluating Semantic Media Wikis [5].

also desirable value ranges for the given measurements. However, while integrating the ontology metrics in the NEOntometrics metric ontology, we identified several inconsistencies in their metric proposal, leading to a homogenization effort of the various measures. Further examination based on the data collected with NEOntometrics on the validity of the made quality statements through the analysis of 4,094 ontologies found that the framework’s quality statements likely do not reflect the actual performance of the modeled artifacts.

Finally, the calculation software was used by Dobriy et al. [5] in an analysis of semantic media wikis, presented at the Extended Semantic Web Conference (ESWC) 2024. The authors analyzed a corpus of 1,029 datasets containing wikis for various use cases. The authors found significant deviations between the structures of the semantic media wikis and the linked open data cloud, especially regarding the use of RDFS and OWL semantics.

### 5.3 Evaluation of Computational Performance

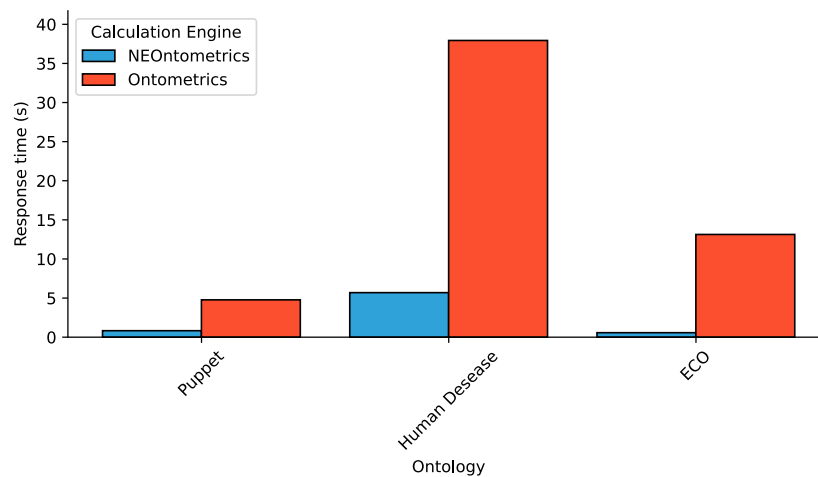
Besides the additional functionality powered by the microservice architecture of NEOntometrics, the calculation engine has also been reworked to improve performance and resource consumption. The following evaluation compares the old OntoMetrics [25] with the performance of the reworked metric calculation (cf. Figure 3). The old Ontometrics engine calculates 72 measurements, 16 of these measurements are ratio-based metrics. The new calculation engine only measures the underlying atomic measures, 83 in total.

For the test, we run both calculation services in dockerized environments on the same machine (Lenovo z13 G2, 64GB Ram, AMD Ryzon Pro 7840U). We run a calculation of three different ontologies: The puppet-Disco inferred ontology<sup>14</sup>, the larger human disease (DOID) ontology with 24MB<sup>15</sup>, and the ECO ontology of Section 4.1 with 7.2MB<sup>16</sup>. Ten times, we send the puppet ontology, then human disease, and finally, the ECO ontology to the new calculation service and then the old one. We measured the response time and queried the docker stats to analyze memory footage. The jupyter notebook used for analysis is available online<sup>12</sup>. The first analysis presented in Figure 9 displays the average calculation times for each of the ontologies. The small Puppet Ontology is sped up 35 times in its analysis, while the human disease ontology calculates 6.5 times faster, and ECO 29 times faster.

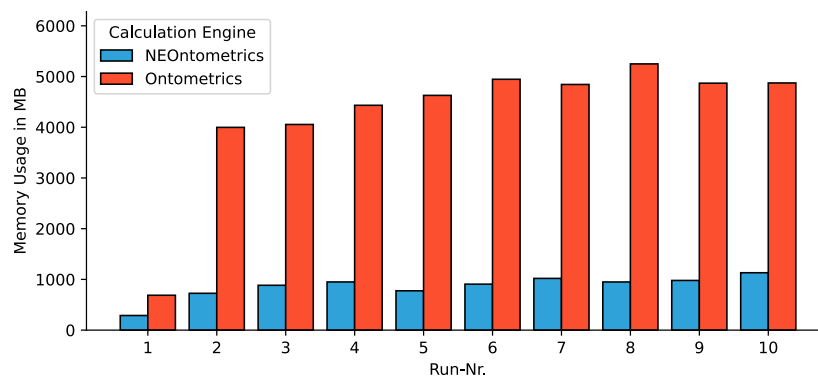
<sup>14</sup><https://raw.githubusercontent.com/kbarber/puppet-ontologies/master/puppet-disco/inferred.owl>

<sup>15</sup><https://raw.githubusercontent.com/DiseaseOntology/HumanDiseaseOntology/main/src/ontology/doid.owl>

<sup>16</sup><https://raw.githubusercontent.com/evidenceontology/evidenceontology/master/eco.owl>



■ **Figure 9** The average calculation time of the old and new calculation service.



■ **Figure 10** The memory consumption of the old and new microservices throughout the runs.

The second analysis in Figure 10 measures the engine’s memory consumption by extracting the docker statistics after each run. Here, the rework of the calculation engine cut the required memory of the analysis to a fifth.

## 6 Conclusion

Ontologies are in use in various applications, facilitating meaning between human and computational actors and enabling these actors to harness the full potential of structured knowledge. The rising number of developed ontologies emphasizes the need for practical evaluations.

The research community has identified this need for quite some time. Many ontology metric frameworks have been proposed that assess a variety of ontology attributes. However, implementations of these frameworks have been scarce. The missing software hinders the research progress: While the definition of measurements is important, it is then crucial to put these metrics into use to perform further evaluations.

The application proposed in this paper aims at closing this gap. We presented NEOntometrics, an open-source software to calculate ontology metrics. The application integrates several metric frameworks and is easily extensible. It is possible to analyze the development of metrics over time

by analyzing GIT-based ontology repositories. Further, the user can inform themselves of available calculations and possible implications using an interactive Metric Explorer. The ontology metrics can be calculated and retrieved either using a graphical user interface or a GraphQL-API. While the former is targeted at knowledge engineers, the latter shall allow developers of semantic-based applications to integrate metrics into their software.

In a case study, we briefly presented possible applications. The evaluation further shows that the software works with large ontologies on an average machine and demonstrates how it has already enabled research on ontology evolution and existing metric frameworks.

The software still has limitations, which motivates further work. One active task is adding more potential metric sources, like private repositories or enabling the manual upload of new graph versions. Further, we aim to add metrics on the specific elements within an ontology, like class-specific and relation-specific measurements. The OntoQA framework by Tartir et al. [32, 31] has some element-specific measures that are a potential starting point. Finally, since most of the frameworks were proposed over 10 years ago, the semantic web community moved forward quite considerably in terms of new vocabularies: The Shape Constraint Language (SHACL) was proposed and is increasingly adopted, and there is a growing need to create evaluations to the constraint specifics of the language.

Our perception is that quantitative ontology research offers much potential for future research, which benefits from continuous interaction in the community. In this context, we are interested in the aspects the community would like to see implemented in our tool and ask for participation<sup>17</sup>.

Further research will be concerned with analyzing the metric data itself. There are many more aspects worth looking at regarding empirical ontology development studies, like comparing typical development processes in different fields (e.g., industrial vs. biomedical ontologies), the usefulness of the proposed frameworks, and the modeling preferences of different persons, to name a few. In the long term, we hope that NEOntometrics impacts the use and research of ontology metrics and that it can help us empirically understand ontology modeling better.

---

## References

- 1 Muhammad Amith, Frank Manion, Chen Liang, Marcelline Harris, Dennis Wang, Yongqun He, and Cui Tao. OntoKeeper: Semiotic-driven Ontology Evaluation Tool For Biomedical Ontologists. *Journal of biomedical semantics*, 8(1), 2017. doi:10.1109/BIBM.2018.8621458.
- 2 Andrew Burton-Jones, Veda C. Storey, Vijayan Sugumaran, and Punit Ahluwalia. A semiotic metrics suite for assessing the quality of ontologies. *Data & Knowledge Engineering*, 55(1):84–102, 2005. doi:10.1016/j.datak.2004.11.010.
- 3 Valerie Cross and Anindita Pal. Ontocat: An ontology consumer analysis tool and its use on product services categorization standards. In *Proceedings of the First International Workshop on Applications and Business Aspects of the Semantic Web*, 2006.
- 4 Renata Dividino, Massimo Romanelli, and Daniel Sonntag. Semiotic-based ontology evaluation tool S-OntoEval. In *Proceedings of the International Conference on Language Resources and Evaluation*, 2008.
- 5 Daniil Dobriy, Martin Beno, and Axel Polleres. Smw Cloud: A Corpus of Domain-Specific Knowledge Graphs from Semantic MediaWikis. In *The Semantic Web - 21st International Conference, ESWC 2024, Hersonissos, Crete, Greece, May 26–30, 2024, Proceedings, Part II*, pages 145–161, 2024. doi:10.1007/978-3-031-60635-9\_9.
- 6 Astrid Duque-Ramos, J. T. Fernández-Breis, R. Stevens, and Nathalie Aussenac-Gilles. OQuARE: A SQuARE-based Approach for Evaluating the Quality of Ontologies. *Journal of Research and Practice in Information Technology*, 43(2):159–176, 2011. URL: <http://ws.acs.org.au/jrpit/JRPITvolumes/JRPIT43/JRPIT43.2.159.pdf>.
- 7 Miriam Fernández, Chwhynny Overbeeke, Marta Sabou, and Enrico Motta. What Makes a Good Ontology? a Case-Study in Fine-Grained Knowledge Reuse. In *The semantic web - Fourth Asian Conference, ASWC 2009, Shanghai, China, December 6-9, 2008. Proceedings*, volume 5926 of *Lecture notes in computer science*, pages

---

<sup>17</sup>Contributors or users are asked to create an issue in the GitHub repository to start a discussion on a new feature or potential bug: <https://github.com/achinator/NEOntometrics/issues>

- 61–75, Berlin, 2009. Springer. doi:10.1007/978-3-642-10871-6\_5.
- 8 Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, and Jos Lehmann. Modelling Ontology Evaluation and Validation. In *The semantic web: research and applications*, volume 4011 of *Lecture notes in computer science*, pages 140–154, Berlin, 2006. Springer. doi:10.1007/11762256\_13.
  - 9 Aldo Gangemi, Carola Catenacci, Massimiliano Ciaramita, Jos Lehmann, Rosa Gil, Francesco Bolici, and Strignano Onofrio. Ontology evaluation and validation: An integrated formal model for the quality diagnostic task, 2005.
  - 10 Michelle Giglio, Rebecca Tauber, Suvarna Nadendla, James Munro, Dustin Olley, Shoshannah Ball, Elvira Mitraka, Lynn M. Schriml, Pascale Gaudet, Elizabeth T. Hobbs, Ivan Erill, Deborah A. Siegele, James C. Hu, Chris Mungall, and Marcus C. Chibucos. ECO, the Evidence & Conclusion Ontology: community standard for evidence information. *Nucleic Acids Research*, 47(D1):D1186–D1194, 2019. doi:10.1093/nar/gky1036.
  - 11 Birger Lantow. OntoMetrics: Putting Metrics into Use for Ontology Evaluation. In *Proceedings of the 8th IC3K 2016 International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KEOD)*, pages 186–191, 2016. doi:10.5220/0006084601860191.
  - 12 Yvonna Lincoln and Egon Guba. *Naturalistic Inquiry*. SAGE Publications, Inc, 1985.
  - 13 Adolfo Lozano-Tello and Asunción Gómez-Pérez. ONTOMETRIC: A Method to Choose the Appropriate Ontology. *Journal of Database Management*, 15(2):1–18, 2004. doi:10.4018/jdm.2004040101.
  - 14 Yinglong Ma, Beihong Jin, and Yulin Feng. Semantic oriented ontology cohesion metrics for ontology-based systems. *Journal of Systems and Software*, 83(1):143–152, 2010. doi:10.1016/j.jss.2009.07.047.
  - 15 C.J.H. Mann. Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce. *Kybernetes*, 33(7), 2004. doi:10.1108/k.2004.06733gae.001.
  - 16 Mark A. Musen. The Protégé Project: A Look Back and a Look Forward. *AI matters*, 1(4):4–12, 2015. doi:10.1145/2757001.2757003.
  - 17 Anthony Mark Orme, Haining Yao, and Letha H. Etzkorn. Indicating Ontology Data Quality, Stability, and Completeness Throughout Ontology Evolution. *Journal of Software Maintenance and Evolution*, 19(1):49–75, 2007. doi:10.1002/smr.341.
  - 18 María Poveda-Villalón, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa. OOPS! (Ontology Pitfall Scanner!). *Semantic Web and Information Systems*, 10(2):7–34, 2014. doi:10.4018/ijswis.2014040102.
  - 19 Qing Lu and Volker Haarslev. OntoKBEval: A Support Tool for DL-based Evaluation of OWL Ontologies. In *OWLED - OWL: Experiences and Directions*, 2006.
  - 20 Joe Raad and Christophe Cruz. A Survey on Ontology Evaluation Methods. In *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 179–186, Setúbal, 2015. SciTePress. doi:10.5220/0005591001790186.
  - 21 Achim Reiz. neontometrics. Software, swId: swh:1:dir:a0a2d612a4de911f171dadcefb66dcc1c5b42bd9 (visited on 2024-12-09). URL: <https://github.com/achiminator/neontometrics>, doi:10.4230/artifacts.22597.
  - 22 Achim Reiz, Henrik Dibowski, Kurt Sandkuhl, and Birger Lantow. Ontology Metrics as a Service (OMaaS). In *Proceedings of the 12th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 250–257, 02.11.2020 - 04.11.2020. doi:10.5220/0010144002500257.
  - 23 Achim Reiz and Kurt Sandkuhl. neontometrics online calculation. InteractiveResource (visited on 2024-12-09). URL: <http://neontometrics.com>, doi:10.4230/artifacts.22599.
  - 24 Achim Reiz and Kurt Sandkuhl. Design Decisions and Their Implications: An Ontology Quality Perspective. In *Perspectives in Business Informatics Research*, volume 398 of *Lecture Notes in Business Information Processing (LNBIP)*, pages 111–127, Vienna, 2020. doi:10.1007/978-3-030-61140-8\_8.
  - 25 Achim Reiz and Kurt Sandkuhl. NEOntometrics: A Flexible and Scalable Software for Calculating Ontology Metrics. In *Proceedings of Poster and Demo Track and Workshop Track of the 18th International Conference on Semantic Systems co-located with 18th International Conference on Semantic Systems (SEMANTiCS 2022)*, Vienna, 2022. CEUR-WS.
  - 26 Achim Reiz and Kurt Sandkuhl. An Ontology for Ontology Metrics: Creating a Shared Understanding of Measurable Attributes for Humans and Machines. In *Proceedings of the 14th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 193–199. SCITEPRESS - Science and Technology Publications, 2022. doi:10.5220/0011551500003335.
  - 27 Achim Reiz and Kurt Sandkuhl. A Critical View on the OQuRE Ontology Quality Framework. In *Enterprise Information Systems*, volume 487 of *Lecture Notes in Business Information Processing*, pages 273–291. Springer Nature Switzerland, Cham, 2023. doi:10.1007/978-3-031-39386-0\_13.
  - 28 Achim Reiz and Kurt Sandkuhl. Evolution of Computational Ontologies: Assessing Development Processes Using Metrics. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management - 14th International Joint Conference, IC3K 2022, Valletta, Malta, October 24–26, 2022, Revised Selected Papers*, volume 1842 of *Communications in Computer and Information Science*, pages 217–238. Springer Nature Switzerland, Cham, 2023. doi:10.1007/978-3-031-43471-6\_10.
  - 29 Achim Reiz and Kurt Sandkuhl. Visualizing Ontology Metrics In The NEOntometrics Application. In *Proceedings of the 8th International Workshop on the Visualization and Interaction for Ontologies, Linked Data and Knowledge Graphs co-located with the 22nd International Semantic Web Conference (ISWC 2023)*, 2023.

- 30 Achim Reiz and Kurt Sandkuhl. neontometrics TGDK dataset, November 2024. doi:10.5281/zenodo.14047141.
- 31 Samir Tartir and I. Budak Arpinar. Ontology Evaluation and Ranking using OntoQA. In *International Conference on Semantic Computing, 2007*, pages 185–192, Los Alamitos, Calif., 2007. IEEE Computer Society. doi:10.1109/ICSC.2007.19.
- 32 Samir Tartir, I. Budak Arpinar, Michael Moore, Amit P. Sheth, and Boanerges Aleman-Meza. Ontoqa: Metric-Based Ontology Quality Analysis. In *IEEE Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources*, 2005.
- 33 Pierre-Yves Vandenbussche, Ghislain A. Atemez-ing, María Poveda-Villalón, and Bernard Vatant. Linked Open Vocabularies (LOV): A gateway to reusable semantic vocabularies on the web. *Semantic web*, 8(3):437–452, 2016. doi:10.3233/SW-160213.
- 34 Patricia L. Whetzel, Natasha Noy, Nigam Haresh Shah, Paul R. Alexander, Csongor Nyulas, Tania Tudorache, and Mark A. Musen. BioPortal: enhanced functionality via new web services from the National Center for Biomedical Ontology to access and use ontologies in software applications. *Nucleic Acids Research*, 39, 2011. doi:10.1093/nar/gkr469.
- 35 Haining Yao, Anthony Mark Orme, and Letha Etzkorn. Cohesion Metrics for Ontology Design and Application. *Journal of Computer Science*, 1(1), 2005. doi:10.3844/jcssp.2005.107.113.
- 36 Jonathan Yu, James A. Thom, and Audrey Tam. Requirements-oriented methodology for evaluating ontologies. *Information Systems*, 34(8):766–791, 2009. doi:10.1016/j.is.2009.04.002.
- 37 Dalu Zhang, Chuan Ye, and Zhe Yang. An Evaluation Method for Ontology Complexity Analysis in Ontology Evolution. In *Managing knowledge in a world of networks - 15th International Conference, EKAW 2006, Podebrady, Czech Republic, October 6-10, 2006, Proceedings*, volume 4248, 2006. doi:10.1007/11891451\_20.



# The dblp Knowledge Graph and SPARQL Endpoint

**Marcel R. Ackermann** ✉ 

Schloss Dagstuhl – Leibniz Center for Informatics, dblp computer science bibliography, Trier, Germany

**Hannah Bast** ✉ 

University of Freiburg, Department of Computer Science, Freiburg, Germany

**Benedikt Maria Beckermann** ✉ 

Schloss Dagstuhl – Leibniz Center for Informatics, dblp computer science bibliography, Trier, Germany

**Johannes Kalmbach** ✉ 

University of Freiburg, Department of Computer Science, Freiburg, Germany

**Patrick Neises** ✉ 

Schloss Dagstuhl – Leibniz Center for Informatics, dblp computer science bibliography, Trier, Germany

**Stefan Ollinger** ✉ 

Schloss Dagstuhl – Leibniz Center for Informatics, dblp computer science bibliography, Trier, Germany

---

## Abstract

For more than 30 years, the dblp computer science bibliography has provided quality-checked and curated bibliographic metadata on major computer science journals, proceedings, and monographs. Its semantic content has been published as RDF or similar graph data by third parties in the past, but most of these resources have now disappeared from the web or are no longer actively synchronized with the latest dblp data. In this article, we introduce the *dblp Knowledge Graph (dblp KG)*, the first semantic representation of the dblp data that is designed and maintained by the dblp team.

The dataset is augmented by citation data from the OpenCitations corpus. Open and FAIR access to the data is provided via daily updated RDF dumps, persistently archived monthly releases, a new public SPARQL endpoint with a powerful user interface, and a linked open data API. We also make it easy to self-host a replica of our SPARQL endpoint. We provide an introduction on how to work with the dblp KG and the added citation data using our SPARQL endpoint, with several example queries. Finally, we present the results of a small performance evaluation.

**2012 ACM Subject Classification** Information systems → Digital libraries and archives; Information systems → Graph-based database models; Computing methodologies → Knowledge representation and reasoning

**Keywords and phrases** dblp, Scholarly Knowledge Graph, Resource, RDF, SPARQL

**Digital Object Identifier** 10.4230/TGDK.2.2.3

**Category** Resource Paper

**Supplementary Material** All described data is available under the CC0 1.0 Universal license.<sup>a</sup>

*Dataset (dblp KG, monthly snapshot):* <https://doi.org/10.4230/dblp.rdf.ntriples> [15]

*Dataset (dblp SPARQL query service, daily updated source triples):* <https://sparql.dblp.org/download/>

*Service (dblp SPARQL query service, user interface):* <https://sparql.dblp.org/>

*Service (dblp SPARQL query service, endpoint):* <https://sparql.dblp.org/sparql>

*Software (QLever SPARQL engine, source code):* <https://github.com/ad-freiburg/qllever>

archived at [swh:1:dir:594019ed1ffe83fef266ac215ba69dff30185d72](https://swh.1:dir:594019ed1ffe83fef266ac215ba69dff30185d72)

*Text (dblp KG, ontology reference documentation):* <https://dblp.org/rdf/docu/>

**Funding** *Benedikt Maria Beckermann:* DFG-LIS project Unknown Data (GEPRIS 460676019)

*Stefan Ollinger:* consortium NFDI4DataScience (GEPRIS 460234259); consortium NFDIxCS (GEPRIS 501930651)



© Marcel R. Ackermann, Hannah Bast, Benedikt Maria Beckermann, Johannes Kalmbach, Patrick Neises, and Stefan Ollinger;

licensed under Creative Commons License CC-BY 4.0

*Transactions on Graph Data and Knowledge*, Vol. 2, Issue 2, Article No. 3, pp. 3:1–3:23



Transactions on Graph Data and Knowledge

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**Acknowledgements** The dblp team would like to thank Silvio Peroni, Ralf Schenkel, and Tobias Zeimetz for the many fruitful discussions and practical help with the specification of the dblp RDF schema. We would also like to thank the many members of the dblp community who sent us their comments, thoughts, criticisms, and suggestions from working with the early versions of the dblp RDF data. Many thanks to Michael Wagner and Michael Didas from the Dagstuhl Publishing team for their support in creating a sustainable workflow for publishing and preserving persistent dblp RDF snapshot releases.

**Received** 2024-06-28 **Accepted** 2024-11-02 **Published** 2024-12-18

**Editors** Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler

**Special Issue** Resources for Graph Data and Knowledge

<sup>a</sup> <https://creativecommons.org/publicdomain/zero/1.0/>

## 1 Introduction

The ever-increasing volume of academic research requires advanced methods for managing and accessing the wealth of information about scholarly publications. To harness the full potential of modern research information systems, it is essential to represent knowledge in a structured, interlinked, and semantically rich manner. Knowledge graphs [23] allow such a representation by providing structured and interlinked data and improving the ability to understand the interconnected nature of scholarly knowledge.

The *dblp computer science bibliography* is a comprehensive online reference for bibliographic information on important computer science publications. It was launched in 1993 by Michael Ley at the University of Trier and has developed from a small experimental website about databases and logic programming (hence, “dblp”) into a popular open data service for the entire computer science community [26]. As of June 2024, dblp indexes over 7.2 million publications written by more than 3.5 million authors. The database indexes more than 57,000 journal volumes, more than 58,000 conference and workshop proceedings, and more than 150,000 monographs.

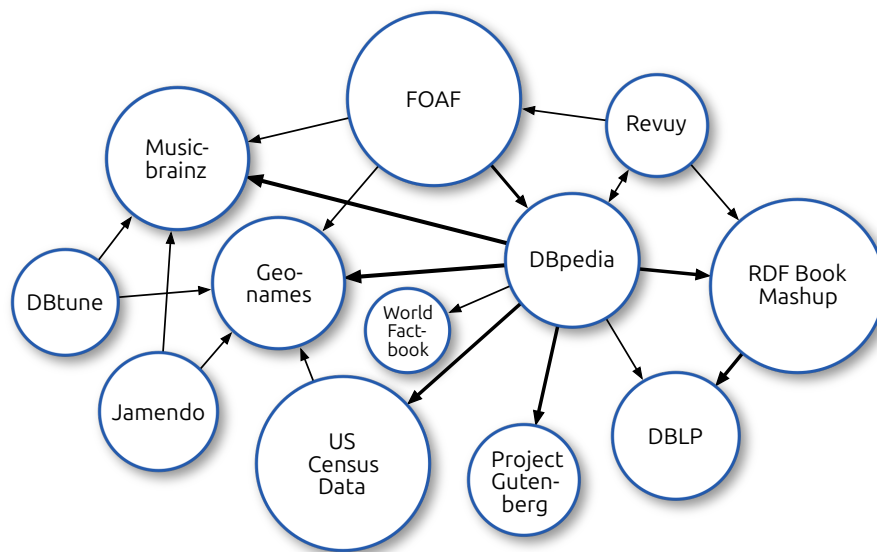
Although the term “open data” had not yet been coined in 1993, dblp was open from the very beginning. Individual data entries have always been freely accessible, and complete dump downloads of the entire dblp data in its own custom XML format have been available since at least 2002 [27, 28]. Since 2015, dblp XML snapshots are archived as persistent monthly releases [16].

### 1.1 Related work

The idea of providing access to dblp data as linked open data is not new. In the very first iteration of the linked open data cloud from 2007 (see Figure 1), dblp was already linked as one of the few early data sources [3, 11]. However, these were always independent contributions from the international computer science community and not an original contribution of the dblp team. These earlier contributions were based on snapshots (current at the time) of the public XML dump export and were generally not updated after creation. Given the continuous additions and maintenance by the dblp team, which make dblp a “living” dataset, these conversions were quickly out of sync with the live data. Many of the external live services, in particular SPARQL endpoints, have vanished from the web in the meantime.

The probably earliest RDF conversion of the dblp dataset has been exercised as a benchmark example for the declarative mapping language D2RQ in 2004 [10]. The data was later released together with an accompanying SPARQL endpoint using the D2R Server tool [9].<sup>1</sup> This conversion provided entities for up to 800,000 articles and 400,000 authors and hasn’t been updated since 2007.

<sup>1</sup> [https://web.archive.org/web/\\*/http://www4.wiwiss.fu-berlin.de/dblp/](https://web.archive.org/web/*/http://www4.wiwiss.fu-berlin.de/dblp/) (archived)



■ **Figure 1** The first snapshot of the LOD cloud, from May 2007, according to <https://lod-cloud.net/>.

At about the same time, further conversions and SPARQL endpoints were the D2R Server in the context of the *Faceted DBLP*<sup>2</sup> search engine [17] and the *RKBExplorer*<sup>3</sup> [21, 20]. These RDF datasets have been actively used for a long time and are a subject or component of numerous computer science publications. E.g., a simple Internet search using Google Scholar with query "fu-berlin.de/dblp" OR "dblp.13s.de/d2r" OR "dblp.RKBExplorer.com" finds at least 380 results,<sup>4</sup> with citing papers still being published today in 2024.

Other graph datasets used dblp data as a starting point to build improved and extended semantic information. For example, the *SwetoDblp* ontology and dataset<sup>5</sup> augmented dblp XML data with relationships to other entities such as publishers and affiliations [2, 1]. The *GraphDBLP* tool<sup>6</sup> models the dblp data from 2016 as a graph database and, in doing so, allows for performing graph-based queries and social network analyses [31, 30]. The *COLINDA* dataset<sup>7</sup> provided a linked data collection of 15,000 conference events, augmenting dblp proceedings data with location, start and end times, geodata and further links [39, 38]. More recently, the *EVENTSKG* knowledge graph<sup>8</sup> provided a semantic description of publications, submissions, start date, end date, location, and homepage for events of top-prestigious conference series in different computer science communities [18]. Semantically structured metadata on scientific events was later also made accessible via the *ConfIDent* platform<sup>9</sup> [22, 19].

Independently of dblp and beyond the discipline of computer science, several international efforts have been launched in recent years to provide open scientific knowledge graphs. *Wikidata*<sup>10</sup> is the collaborative, omnithematic, and multilingual knowledge graph hosted by the Wikimedia

<sup>2</sup> [https://web.archive.org/web/\\*/http://dblp.13s.de/d2r/](https://web.archive.org/web/*/http://dblp.13s.de/d2r/) (archived)

<sup>3</sup> [https://web.archive.org/web/\\*/dblp.RKBExplorer.com](https://web.archive.org/web/*/dblp.RKBExplorer.com) (archived)

<sup>4</sup> Accessed on 2024-06-12.

<sup>5</sup> [https://web.archive.org/web/\\*/http://knoesis.wright.edu/library/ontologies/swetodblp](https://web.archive.org/web/*/http://knoesis.wright.edu/library/ontologies/swetodblp) (archived)

<sup>6</sup> <https://github.com/fabiomercorio/GraphDBLP>

<sup>7</sup> [https://web.archive.org/web/\\*/http://www.colinda.org/](https://web.archive.org/web/*/http://www.colinda.org/) (archived)

<sup>8</sup> <http://w3id.org/EVENTSKG-Dataset/ekg>

<sup>9</sup> <https://www.confident-conference.org/>

<sup>10</sup> <https://www.wikidata.org>

### 3:4 The dblp Knowledge Graph and SPARQL Endpoint

Foundation [40]. Within Wikidata, the *WikiCite* project<sup>11</sup> aims to create an open, collaborative repository of bibliographic data. *OpenCitations*<sup>12</sup> maintains and publishes open citation data as linked open data, thereby providing the first truly open alternative to proprietary citation indexes [35]. Initially an outcome of the EU Horizon 2020, the *OpenAIRE Graph*<sup>13</sup> was one of the first comprehensive research knowledge graphs [29]. Since then, OpenAIRE has consolidated its organizational structure and the OpenAIRE Graph is now the authoritative source for the European Open Science Cloud (EOSC).<sup>14</sup> *OpenAlex*<sup>15</sup> is a recent open infrastructure service, built on the data of the now abandoned Microsoft Academic Graph<sup>16</sup>. OpenAlex is a massive, cross-disciplinary research knowledge graph of publications, authors, venues, institutions, and concepts [36]. Furthermore, the *Open Research Knowledge Graph (ORKG)* aims to make scientific knowledge fully human- and machine-actionable by describing research contributions in a structured manner, e.g., by connecting research papers, datasets, and used methods [25]. The ORKG aims to build a community of contributors in order to collect, curate, and organize descriptions of scientific contributions in a crowd-sourcing manner.

## 1.2 Our contribution

In this article, we introduce the *dblp Knowledge Graph (dblp KG)*. The dblp KG aims to make all semantic relationships modeled in the dblp computer science bibliography explicit. In contrast to previous approaches, the dblp KG is not merely based on a one-time snapshot of dblp data, but is actively synchronized with the current data. In particular, the dblp KG thus benefits from the continuing curation work of the dblp team.

The dblp KG aims to complement other open knowledge graphs by bringing in dblp’s unique strengths in author disambiguation, semantic enrichment of bibliographies, and its role as a directory of computer science journals and conferences. We demonstrate this by augmenting our dataset with identifiers and citation data from the OpenCitations corpus.

Open and FAIR access to the data is provided via daily updated RDF dumps, persistently archived monthly releases, a new public SPARQL endpoint with a powerful user interface, and a linked open data API. We also make it easy to self-host a replica of our SPARQL endpoint.

The rest of this article is organized as follows. In Section 2, we introduce the ontology of the dblp KG and present statistics about the graph. Section 3 describes the different ways to access the dblp KG and the citation data. Section 4 provides an introduction on how to work with the dblp KG and the added citation data using our SPARQL endpoint, with several example queries, and complemented by a small performance evaluation. Section 5 concludes the article with a short discussion and an outlook.

## 2 dblp as a Knowledge Graph

Since its earliest stages, the semantic organization of bibliographic metadata has been a primary concern of the dblp team’s editorial work. This includes linking publications with their true authors, research papers with their proceedings, and conference events with the history of their conference series. The dblp team puts a lot of (often manual) effort into providing such information

---

<sup>11</sup> <https://www.wikidata.org/wiki/Wikidata:WikiCite>

<sup>12</sup> <https://opencitations.net>

<sup>13</sup> <https://graph.openaire.eu>

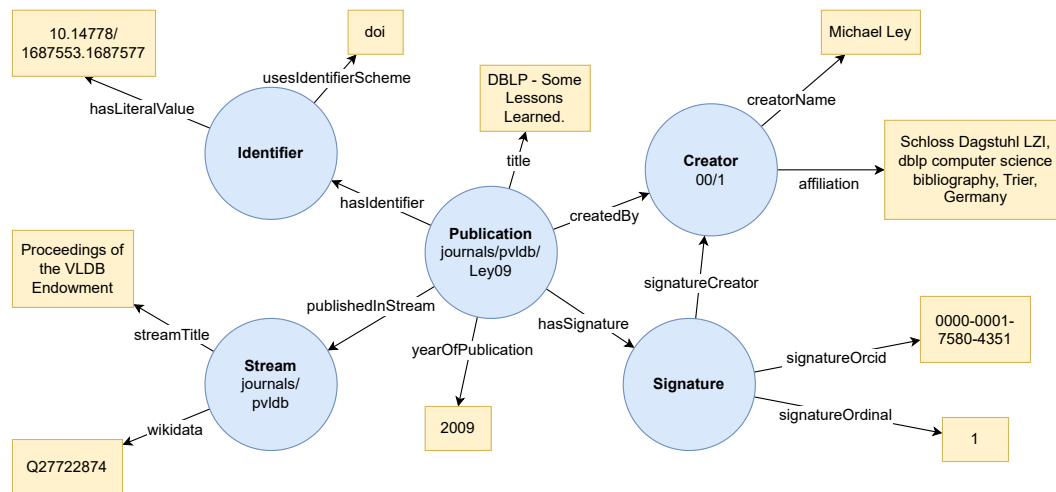
<sup>14</sup> <https://eosc-portal.eu/>

<sup>15</sup> <https://openalex.org/>

<sup>16</sup> <https://www.microsoft.com/en-us/research/project/microsoft-academic-graph/>

as accurately, completely, and up-to-date as possible. Editors manually annotate individual entries with further metadata, like alternative names, external identifiers, or links to relevant web resources. However, most semantic relations have only been provided implicitly on the (once manually crafted) dblp HTML webpages, and so far have not been made explicit in a machine-friendly way.

The dblp Knowledge Graph (dblp KG) aims to make these semantic relations explicit and machine-actionable. This includes structured information already available via APIs, like the authorship of publications [28], as well as information that has not been published explicitly before. In the current, second major iteration of the knowledge graph, this additional information includes the concrete linkage of published works with the publication venue they appeared in (conferences, journals, etc.), metadata about these venues, and information about known relations between venues. Future iterations of the graph will expand the model even further. This will include such information as metadata about conference events within a conference series, and author affiliations. A simplified excerpt of the current graph is shown in Figure 2.



■ **Figure 2** Simplified excerpt from the dblp knowledge graph. The excerpt is centered on the paper “DBLP – Some Lessons Learned” from Michael Ley [28].

## 2.1 The dblp ontology

As there already is a whole range of ontologies that model bibliographic information about scientific works (e.g., see [14, 32, 24]), the dblp ontology is explicitly not intended to replace them. Instead, it is designed to model the way dblp handles and provides bibliographic metadata, including all possible quirks and oddities that may arise from dblp’s unique approach. For example, dblp’s author disambiguation uses certain “pseudo-author entities” (described in more detail in Section 2.1.1.3 below) to model cases where the true authorship of a work is currently unknown or ambiguous. Also, dblp’s records are incompatible with the more fine-grained FRBR model [34] that is standard in the library community.<sup>17</sup> Therefore, it was not viable to reuse existing ontologies, as is usually recommended. However, links to related types and predicates from existing ontologies are provided in the dblp RDF schema whenever possible.

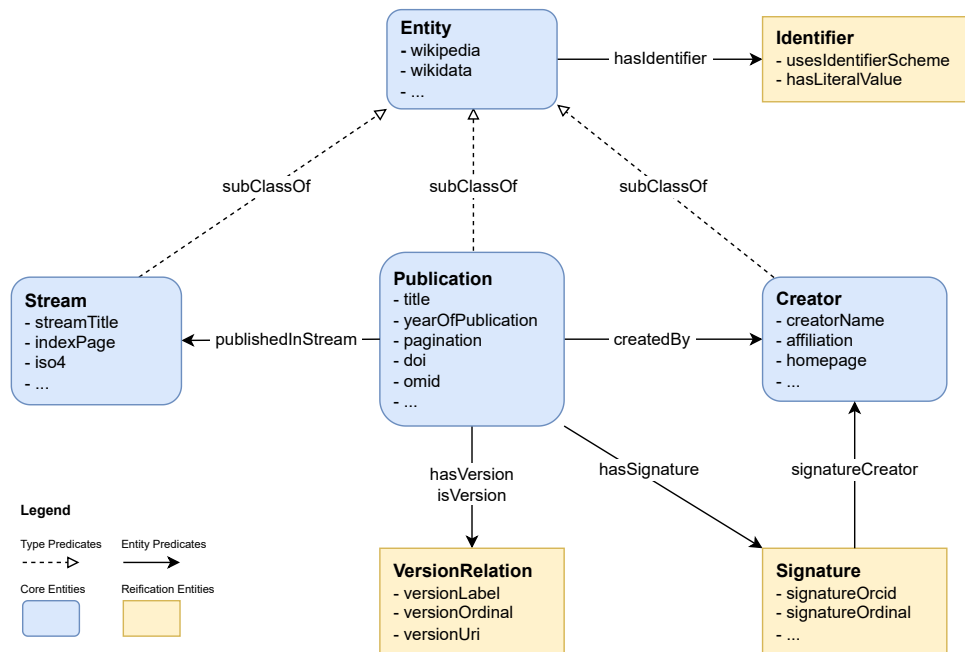
<sup>17</sup>In particular, the dblp data model generally does not distinguish between the FRBR layers *Work* and *Expression* and does not address the layers *Manifestation* or *Item* at all.

### 3:6 The dblp Knowledge Graph and SPARQL Endpoint

In the remainder of this section, *entity* refers to any resource accessible via an IRI, a literal, or an anonymous node. It is synonymous with the term *resource* as defined in [12]. Entities in the dblp ontology are assigned certain core and reification *types* that stem from the dblp internal data model, and relations between entities are modeled using *predicates*. The types of the dblp ontology and their connections are described below, and a simplified view of the ontology is presented in Figure 3. The full dblp ontology reference documentation can be found at <https://dblp.org/rdf/docu/>.

■ **Table 1** IRI prefixes for the core entities of the dblp KG.

Type	IRI Prefix
Publication	<a href="https://dblp.org/rec/">https://dblp.org/rec/</a>
Creator	<a href="https://dblp.org/pid/">https://dblp.org/pid/</a>
Stream	<a href="https://dblp.org/streams/">https://dblp.org/streams/</a>



■ **Figure 3** Excerpt from the dblp ontology showing the relationships between core and reification types. Each of the entity boxes shows the type at the top, followed by a list of predicates of entities of that type. The figure shows only a small selection of all predicates and omits most of the finer-grained subtypes.

#### 2.1.1 Core entities

The current iteration of the dblp ontology contains named entities for publications, their creators, and the publication venues (which we call streams) they appeared in. These core entities have persistent IRIs and are accessible via open data APIs and as HTML web pages within the dblp website. For an overview of the IRI prefixes of the different types, see Table 1.

### 2.1.1.1 Entities

The dblp ontology defines an abstract supertype `dblp:Entity` as a parent to all core entity types. In the dblp ontology, this type represents any core entity that can be associated with an identifier. The main purpose of this abstract supertype is to provide a common `rdfs:range` subject for predicates in the dblp RDF schema.

### 2.1.1.2 Publications

Entities of type `dblp:Publication` represent any academic work indexed in dblp. This includes traditionally published articles, authored or edited volumes, and (more recently) also published data artifacts.

Like on the dblp websites, publications are linked to their authors or editors (modeled as `dblp:Creator` entities, see Section 2.1.1.3). This is done redundantly in two ways. First, there is a direct link towards authors and editors using the predicate `dblp:createdBy`. Additionally, special reification entities called *signatures* (of type `dblp:Signature`, see Section 2.1.2) are provided using the predicate `dblp:hasSignature`. This redundancy enables convenient and elegant queries via the first option when nothing other than the link from a publication to its creators is needed, and can provide more in-depth metadata about the authorship via the signature entities if required.

Publications are also linked to their publication venue (of type `dblp:Stream`, see Section 2.1.1.4), such as the conference or journal in which they are published. The link is modeled via the predicate `dblp:publishedInStream`. There are no dedicated reification entities for these links in the current iteration of the dblp ontology. Related metadata, such as issue or volume numbers, is given as literal values via predicates on the publications. In the future, reification entities might be introduced here.

To provide external identifiers, publications are linked to identifier entities (`datacite:Identifier`, see Section 2.1.2) using the `datacite:hasIdentifier` predicate. Redundantly and for convenience, links to the IRIs of the most important external identifiers are provided via direct predicates, namely DOIs (`dblp:doi`), ISBN (`dblp:isbn`), Wikidata entity (`dblp:wikidata`), and OpenCitations Meta IDs (`dblp:omid`).

Publication entities also carry further metadata fields, such as their titles, the year of publication, or pagination information.

All publications in dblp are classified by a rudimentary system of publication types. Similar to many modeling decisions made at dblp in the early days, these types were originally derived from classic BibTeX, but have evolved. Publication types are modeled as subtypes of `dblp:Publication`, like `dblp:Inproceedings` for conference publications, `dblp:Book` for monographs, or `dblp:Data` for research data and artifacts. A list of all types can be found in Table 2. We are aware that due to the evolving publication landscape, a BibTeX-inherited classification might no longer be a best fit for modern publication practices, and many of the decisions behind the dblp type classification system are disputable.

### 2.1.1.3 Creators

The type `dblp:Creator` represents any individual or group listed as the author or editor of a publication. Analogous to the case of `dblp:Publication` entities, creators are linked to their publications redundantly in two ways: First, they are linked directly via predicates (such as `dblp:creatorOf`) and indirectly via `dblp:Signature` reification entities.

Creator entities also carry metadata such as their names, alternative names, current and former affiliations, and homepages. Manually curated identifiers are provided via identifier entities (`datacite:Identifier`) linked using the `datacite:hasIdentifier` predicate. For convenience, ORCID IRIs (`dblp:orcid`) and Wikidata IRIs (`dblp:wikidata`) are provided via direct predicates.

■ **Table 2** List of all publication types within the dblp ontology.

Publication Type	Description
Inproceedings	Conference and workshop publications
Article	Journal articles
Book	Monographs and PhD theses
Editorship	Edited volumes, prefaces, and editorials
Incollection	Chapters within a monograph
Reference	Reference material and encyclopedia entries
Data	Research data and artifacts
Informal	Preprints, non-peer-reviewed and other publications
Withdrawn	Withdrawn publications

The standard creator subtype (c.f. Table 3) used for individual authors or editors is `dblp:Person`. In some cases, where a listed author of a publication is not a single person but represents a known group or consortium, the type `dblp:Group` is used.

One major contribution of the dblp team is the continuous work to identify and disambiguate the “true authors” behind the plain character strings given in bibliographic metadata. This work often leaves a fair number of disambiguation cases unresolved as the information at hand does not allow for a reliable decision. These situations are handled by introducing certain pseudo-persons that represent more than one individual and are fully known to be ambiguous. Publications assigned to such a pseudo-person are known to have their true author not yet determined, and the collected bibliography of such a pseudo-person is known to not represent the coherent scholarly work of an actual person.

For example, assume we have several publications written by people called “Jane Doe”. Further, assume that we know for some of those publications that they are written by two different individuals, called “Jane Doe 0001” and “Jane Doe 0002” (following dblp’s scheme to distinguish different individuals with the same name). These two individuals will be modeled using the subtype `dblp:Person`. However, for the remaining “Jane Doe” publications, the true authorship is currently unknown. In that case, the remaining publications will be linked to neither “Jane Doe 0001” nor “Jane Doe 0002”, but to a different pseudo-person “Jane Doe” of type `dblp:AmbiguousCreator`.

For all purposes, `dblp:AmbiguousCreator` entities are used and referenced just like normal, unambiguous creator entities in dblp, and they are linked to publications, signatures, etc. in the usual way. However, when retrieved in complex queries, their ambiguous nature should be understood and results should be handled accordingly. E.g., if an `dblp:AmbiguousCreator` is retrieved as a common coauthor, there is no guarantee that this is really the same person linking both authors. `dblp:AmbiguousCreator` entities do provide several unique predicates, like `dblp:possibleActualCreator` and `dblp:proxyAmbiguousCreator` that link between ambiguous and actual creators that may be related.

Please be aware that due to the continuous curation work done by the dblp editorial team, long-standing disambiguation cases can be (and are regularly) resolved at any time. Hence, `dblp:proxyAmbiguousCreator` entities and their links to publications and signatures are among the most volatile content of the dblp Knowledge Graph.

#### 2.1.1.4 Streams

In dblp, we use the term *stream* to refer to any journal, conference series, book series, or repository that acts as a regular source for publications. Such streams are modeled using the type `dblp:Stream`. Publications are linked to the streams they appeared in using the predi-



■ **Table 3** List of all creator types within the dblp ontology.

Creator Type	Description
Person	An individual person
Group	A group or organisation
AmbiguousCreator	An unknown number of unidentified individuals of the same name

cate `dblp:publishedInStream`. A single publication might be linked to multiple streams in that way. For example, an HCI conference paper might appear both in the stream of its conference event series `<https://dblp.org/streams/conf/hci>` as well as, say, the LNCS book series `<https://dblp.org/streams/series/lncs>` that publishes the conference proceedings.

`dblp:Stream` entities may be linked to other `dblp:Stream` entities using `dblp:relatedStream` or one of its subpredicates. These include hierarchical relations (`dblp:subStream` and `dblp:superStream`) in cases of streams that take place or are published as part of another stream, and temporal relations (`dblp:predecessorStream` and `dblp:successorStream`) in cases where streams merge with or are replaced by another stream.

Stream entities have further metadata like their (past, current, or alternative) titles, homepage URLs, a URL of their dblp index page, or their ISO4 journal title abbreviation. Identifiers are again provided via identifier entities (`datacite:Identifier`) linked using the `datacite:hasIdentifier` predicate. For convenience, ISSN IRIs (`dblp:issn`) and Wikidata IRIs (`dblp:wikidata`) are provided via direct predicates.

The dblp ontology uses the following subtypes of `dblp:Stream` (see Table 4): `dblp:Journal` for periodically published journals, `dblp:Conference` for conference or workshop series, and `dblp:Series` for series of published volumes like monographs and proceedings. Only very recently, we expanded the dblp data model also to include a fourth, new subtype `dblp:Repository` for sources of research data and artifacts, such as Zenodo.<sup>18</sup>

■ **Table 4** List of all stream types within the dblp ontology.

Stream Type	Description
Journal	Periodically published journals
Conference	Conference or workshop series
Series	Series of monographs or proceedings volumes
Repository	Sources of research data and artifacts

### 2.1.2 Reification entities

Reification entities, also known as linking entities, link core entities to other core or external entities and provide further metadata about that link. Reification entities are represented by blank nodes in the dblp Knowledge Graph.

**Identifiers.** Identifier entities (of type `datacite:Identifier`) are used to annotate identifiable entities in the dblp KG with external identifiers, such as DOI or ORCID. These external identifiers allow users to connect the dblp KG entities with information from other knowledge graphs. The type `datacite:Identifier` is reused from and defined in the DataCite Ontology [37].

<sup>18</sup><https://zenodo.org/>

## 3:10 The dblp Knowledge Graph and SPARQL Endpoint

A dblp KG core entity is linked to identifier entities using the predicate `datacite:hasIdentifier`. Identifier entities are linked to their identifier schema (such as `datacite:doi`) via the predicate `datacite:usesIdentifierScheme`. They are linked to their literal value stating the actual ID string via the predicate (`litre:hasLiteralValue`). It is important to note that identifier entities do not link directly to the IRIs of the external identifier to support a wider range of identifier schemas.

**Signatures.** In dblp, we use the term *signature* to refer to the reification entity (of type `dblp:Signature`) that links a publication (of type `dblp:Publication`) to one of its authors (of type `dblp:Creator`). The purpose of these entities is to provide more context to this otherwise simple link. Signature entities may link to an ORCID IRI that has been stated in the publication’s metadata using the `dblp:signatureOrcid` predicate and provide the relative position of a publication’s creator in the complete creator list using the `dblp:signatureOrdinal` predicate. We aim to provide additional context via the signature entities in future iterations of the dblp KG, such as affiliation information provided in the publication.

To distinguish between the roles of an editor and an author for a published work, the two subtypes `dblp:AuthorSignature` and `dblp:EditorSignature` are used.

**Version Relations.** With the recent inclusion of the publication type `dblp:Data`, an optional hierarchy between publications has been introduced to dblp to model cases where one publication is an instance of another publication. In dblp, we call the instanced publication a *version*, while the instantiated publication is called a *concept*. These relations are represented by the type `dblp:VersionRelation`. Publications are linked to version-relation entities via the predicates `dblp:versionConcept` and `dblp:versionInstance`. For convenience, the redundant predicate `dblp:isVersionOf` is provided to directly link between the `dblp:Publication` entities in cases when nothing other than this link is needed.

The relation contains further metadata like a label for the instance (such as “Version 1.3”), their relative order compared to other instances of the same concept, and an identifying IRI of the concept.

## 2.2 Key statistics

This section presents several key statistics of the dblp Knowledge Graph to give an overview of its content and dimension. Table 5 shows the number of entities per type, and Table 6 shows the number of identifiers by schema. The SPARQL queries used to create the statistics can be found online<sup>19</sup>, each with a direct link to our SPARQL endpoint that will then execute the corresponding query. The statistics in this paper are based on the dump from September 11th, 2024.

The majority of publication entities are conference proceedings papers (47.43%) and journal articles (39.22%). Informal publications (9.25%), such as preprint publications on arXiv, also form a significant share of publications. The other publication types each form less than 3% of the corpus.

Less than 0.01% of all creator entities are of type `dblp:Group` because we aim to present individual authorship where possible. Individual authors of type `dblp:Person` form the vast majority (99.58%) of creator entities. The manually identified `dblp:AmbiguousCreator` entities only form a small fraction (0.41%).

In contrast to many other research fields, where journals are the predominant medium, conferences play a crucial role in disseminating research in computer science. Many conferences only take place once or twice, while journals tend to be much more long-lived. In addition,

---

<sup>19</sup><https://github.com/dblp/kg/wiki/Paper-TGDK-2024#statistics-queries>

■ **Table 5** Number of entities in the dblp KG by type, as of September 11th, 2024.

(a) Count of the main dblp type entities.

Type	Count	%
Signature	24,559,211	41.09
Identifier	24,157,894	40.42
Publication	7,446,698	12.46
Creator	3,595,686	6.02
Stream	8,864	0.01
VersionRelation	899	< 0.01

(b) Count of the publication subtype entities.

Publication Type	Count	%
Inproceedings	3,532,088	47.43
Article	2,920,903	39.22
Informal	689,075	9.25
Book	154,185	2.07
Editorship	61,847	0.83
Incollection	43,220	0.58
Reference	27,366	0.37
Withdrawn	10,425	0.14
Data	7589	0.10

(c) Count of the creator subtype entities.

Creator Type	Count	%
Person	3,580,548	99.58
AmbiguousCreator	14,774	0.41
Group	364	0.01

(d) Count of the stream subtype entities.

Stream Type	Count	%
Conference	6,753	76.18
Journal	1,885	21.27
Series	220	2.48
Repository	6	0.07

conferences are often divided into workshop series which may also form their own entities. All this explains why there are three times as many conference entities (76.18%) as journal entities (21.27%) in the dblp KG. Currently, there are only 6 repository streams in the dblp KG because the support for data publications, which are modeled to be published in repositories, has only recently been added to dblp.

The almost 6 million DOIs are the most often occurring identifier in the dblp KG. In addition to about 170,000 distinct ORCID IRIs manually linked to creators using `dblp:orcid` in the dblp KG, we also link to more than one million distinct ORCID IRIs on signatures via `dblp:signatureOrcid`. Other than ORCID IRIs linked to creator entities, ORCID IRIs linked to signatures are automatically harvested from metadata and have not been manually verified by the dblp team.

### 3 How to access the dblp Knowledge Graph and the citation data

We provide a variety of ways to access the dblp Knowledge Graph and the associated citation data: via RDF dumps, via a public SPARQL endpoint with an associated user interface, via SPARQL queries embedded into the dblp website, via a linked open data API, and by providing an easy way to set up one's own SPARQL endpoint. Each of these is briefly described in one of the following subsections. All data is released openly under the CC0 1.0 Universal license.<sup>20</sup>

<sup>20</sup><https://creativecommons.org/publicdomain/zero/1.0/>

## 3:12 The dblp Knowledge Graph and SPARQL Endpoint

■ **Table 6** Count of external identifier entities in the dblp KG by type, as of September 11th, 2024.

Identifier Type	Count	%	Identifier Type	Count	%
doi	6,205,594	47.35	math genealogy	9938	0.08
omid	5,312,165	40.53	linkedin	5764	0.04
wikidata	685,613	5.23	twitter	4162	0.03
arxiv	409,965	3.14	issn	3567	0.03
orcid	167,117	1.28	research gate	2356	0.02
isbn	90,777	0.69	github	2157	0.02
handle	44,665	0.34	isni	1412	0.01
dnb	41,713	0.32	viaf	1071	< 0.01
google scholar	30712	0.23	oclc	448	< 0.01
urn	24386	0.19	lattes	442	< 0.01
ieee	14876	0.11	repec	244	< 0.01
gnd	13387	0.10	gepris	128	< 0.01
zbmath	12215	0.09	openalex	10	< 0.01
acm	11486	0.09	gitlab	10	< 0.01
loc	10266	0.08			

### 3.1 RDF dumps of the dblp Knowledge Graph

We provide daily updated RDF exports of the dblp KG in RDF/XML, N-Triples, and Turtle formats.<sup>21</sup> These are useful for tools and services that need the latest version of the data. Further, we publish persistent monthly releases of the dblp KG in N-Triples format [15] and recommend using these persistent releases for reproducible experiments and similar purposes. We also provide serializations of the RDF schema of the dblp KG ontology described in Section 2.1. This schema is rarely changed, and all previous versions of the schema are persistently available.

### 3.2 RDF dumps of the dblp KG with citation data

We also provide daily updated dblp RDF exports augmented with citation data<sup>22</sup>, obtained from the OpenCitations project, which provides open-access citation data for publications across all areas of science [13]. OpenCitations assigns each publication an identifier (called OMID), which is also provided in the dblp KG; see Section 2.1.1. We filter the whole OpenCitations corpus to the subset of citations that are concerned with publications listed in dblp and provide the combined graph. The connection between dblp and OpenCitation entities is done via the predicate `dblp:omid` (see Section 2.1.1.2). Please be aware that while the OpenCitations data is updated only infrequently, the dblp KG is updated daily. Thus, the combined dataset is also updated daily.

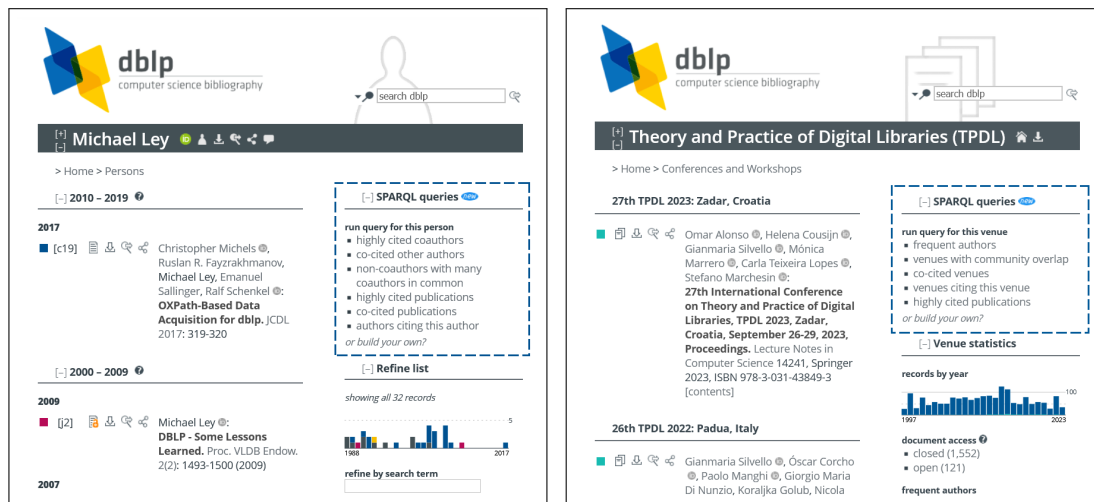
### 3.3 Public SPARQL endpoint with associated user interface

We provide a public SPARQL endpoint for the combined data described in Section 3.2 under <https://sparql.dblp.org/sparql>, powered by the QLever SPARQL engine [6] [8].<sup>23</sup> The SPARQL endpoint is updated daily, in sync with the daily releases of these datasets. The endpoint

<sup>21</sup> <https://dblp.org/rdf/>

<sup>22</sup> <https://sparql.dblp.org/download/>

<sup>23</sup> <https://github.com/ad-freiburg/qlever>



(a) Example of person queries.

(b) Example of venue queries.

■ **Figure 4** Example SPARQL queries embedded into the dblp website.

conforms with the SPARQL 1.1 Protocol<sup>24</sup>, currently still with minor deviations. These are documented in the dblp KG Wiki<sup>25</sup> and will be fixed in the near future. The SPARQL endpoint makes it very easy to build services or tools on top of the dblp KG. Section 4.6 briefly analyzes its performance.

The endpoint also comes with a user interface, available under <https://sparql.dblp.org>, for the interactive formulation and execution of SPARQL queries. The user interface provides various features to help people that are unfamiliar with the intricacies of the dataset or of the SPARQL query language, most notably context-sensitive suggestions (autocompletion) after each keystroke. This is explained in more detail in Section 4.2, for the example query from Section 4.1. The user interface also features a button, which opens a searchable list of example queries.

### 3.4 Setting up your own SPARQL endpoint

The SPARQL endpoint described in the previous section is publicly and freely available. To enable a continuous service, there is a fixed timeout for each query, and at some point, we might also introduce rate limits or quotas. For users with high query volumes or other special requirements, we provide instructions for setting up their own SPARQL endpoint and user interface,<sup>26</sup> with the exact same functionality as described in Section 3.3, using the exact same dataset. This setup requires only a few commands and works with standard hardware; see Section 4.6.

### 3.5 SPARQL queries embedded into the dblp website

Beyond the examples of the query interface, we also provide links with preformulated SPARQL queries embedded into various pages across the dblp website, as shown in Figure 4. In particular, each author page now features a box with links to related SPARQL queries, such as a query to calculate the most highly cited co-authors of the author described on the page. Similarly, each

<sup>24</sup> <https://www.w3.org/TR/sparql11-protocol/>

<sup>25</sup> <https://github.com/dblp/kg/wiki/Known-Issues>

<sup>26</sup> <https://github.com/dblp/kg/wiki/SPARQL-server-setup>

## 3:14 The dblp Knowledge Graph and SPARQL Endpoint

■ **Table 7** Recognized file extensions and MIME types.

Format	API file extension	MIME type for content negotiation
RDF/XML	.rdf	application/rdf+xml
N-Triples	.nt	application/n-triples
Turtle	.ttl	text/turtle
HTML	.html	text/html
dblp XML	.xml	application/xml

conference or journal page features a box with links to related SPARQL queries, such as a query to calculate frequent authors for this conference, or conferences with a large overlap regarding authors. These embedded queries are useful in three respects: (1) they provide interesting information that complements the information provided on the respective page, (2) they draw attention to the SPARQL endpoint for users that might otherwise miss this opportunity, and (3) it is an easy and motivating way to learn by example what is in the dblp KG and how to query it.

### 3.6 Linked Open Data API

Finally, we provide an API for individual pieces of RDF data for creators, publications, and streams. This data is guaranteed to always be up-to-date with the current state of the dblp database. The URLs of this API all follow the same structure: a dblp resource IRI, followed by a file extension corresponding to the requested file format as given in Table 7. For example, for the creator resource IRI <https://dblp.org/pid/71/4882> there is

<https://dblp.org/pid/71/4882.rdf> for retrieving RDF/XML,  
<https://dblp.org/pid/71/4882.nt> for retrieving N-Triples,  
<https://dblp.org/pid/71/4882.ttl> for retrieving Turtle,  
<https://dblp.org/pid/71/4882.html> for the dblp HTML website.

Similarly, for publications, there is

<https://dblp.org/rec/conf/semweb/AuerBKLCI07.rdf> for retrieving RDF/XML,  
<https://dblp.org/rec/conf/semweb/AuerBKLCI07.nt> for retrieving N-Triples,  
<https://dblp.org/rec/conf/semweb/AuerBKLCI07.ttl> for retrieving Turtle  
<https://dblp.org/rec/conf/semweb/AuerBKLCI07.html> for the dblp HTML website,

and for stream entities, there is

<https://dblp.org/streams/conf/semweb.rdf> for retrieving RDF/XML,  
<https://dblp.org/streams/conf/semweb.nt> for retrieving N-Triples,  
<https://dblp.org/streams/conf/semweb.ttl> for retrieving Turtle,  
<https://dblp.org/streams/conf/semweb.html> for the dblp HTML website.

## 4 SPARQL queries and performance

This section provides an introduction on how to work with the dblp KG and the added citation data using the SPARQL endpoint we provide. We provide four example queries: a basic one (Section 4.1), a more advanced one (Section 4.3), a federated query that queries two endpoints (Section 4.4), and a query that uses both the dblp KG and the added citation data (Section 4.5). We use the basic example query to explain how the autocompletion works (Section 4.2). The

section closes with a brief performance evaluation of a selection of SPARQL queries (Section 4.6). All queries discussed in this section can also be found on the web<sup>27</sup>, each with a direct link to our SPARQL endpoint that will then execute the corresponding query.

## 4.1 A basic SPARQL query

SPARQL is the standard query language for querying RDF data. The language can be seen as a variant of SQL (the standard query language for relational databases), adapted to the RDF data model. Namely, just like RDF data is a set of triples, the core of a typical SPARQL query is a set of triples, with variables in some places. Following is an example query that asks for the titles of all papers in dblp published until 1940, their authors, and the year of publication:

```
PREFIX dblp: <https://dblp.org/rdf/schema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?title ?author ?year WHERE {
  ?paper dblp:title ?title .
  ?paper dblp:authoredBy ?author_id .
  ?author_id rdfs:label ?author .
  ?paper dblp:yearOfPublication ?year .
  FILTER (?year <= "1940"^^xsd:gYear)
}
ORDER BY DESC(?year) ASC(?title)
```

Run this query [↗](#)

Conceptually, the result of a SPARQL query is a table. For the query above, that table has three columns (labeled `?title`, `?author`, and `?year`), and one row for each possible assignment to these three variables such that all the corresponding triples in the query body exist and all the additional constraints (in this case, the `FILTER` condition) are fulfilled. For example, the first five result rows for the query above are as follows:

<code>?title</code>	<code>?author</code>	<code>?year</code>
A Correction to Lewis and Langford's Symbolic Logic.	J. C. C. McKinsey	1940
A Formulation of the Simple Theory of Types	Alonzo Church	1940
Einkleidung der Mathematik in Schröderschen Relativkalkül	Leopold Lowenheim	1940
Elimination of Extra-Logical Postulates.	Willard Van Orman Quine	1940
Elimination of Extra-Logical Postulates.	Nelson Goodman	1940

Note that if a paper has  $k$  authors, there are  $k$  rows for that paper in the result (as in rows four and five above). If a paper had  $k_1$  authors and  $k_2$  titles, there would be  $k_1 \cdot k_2$  result rows for that paper, one for each combination. Such Cartesian products are unexpected for many SPARQL beginners and can lead to very large results, in particular, when making mistakes in the query formulation.

<sup>27</sup> <https://github.com/dblp/kg/wiki/Paper-TGDK-2024#example-queries>

## 4.2 SPARQL autocompletion

Writing a correct SPARQL query requires knowledge about the SPARQL query language in general as well as about the structure of the concrete knowledge graph. For the example query above, the following skills are required:

1. Getting the general syntax right: how to write a `SELECT` statement, how to write a `FILTER` expression, how to write an `ORDER BY` clause.
2. Knowing the names of the RDF types and entities needed for the query, here: `dblp:title`, `dblp:authoredBy`, and `dblp:yearOfPublication`.
3. Knowing the right `PREFIX` definitions and where to put them (the first few lines in the query above).

The user interface helps with all of these by offering incremental context-sensitive autocompletion after each keystroke. We recommend to go to <https://sparql.dblp.org> and try the following instructions live.

By simply typing “S”, the UI suggests the whole template for a `SELECT` clause (needed for most queries). After having typed the first variable `?paper` in the body of the SPARQL query, the UI will suggest predicate names, which are searchable by typing a prefix such as “ti” (for `title`). When selecting a predicate, the corresponding `PREFIX` statement will be automatically added at the top of the query. After entering the variable `?paper` a second time, the UI will suggest only those predicates that would lead to a non-empty result together with the already typed `?paper dblp:title ?title`. This narrows down the selection considerably. Continuing this way, the user can type a query from left to right, top to bottom relatively easily with minimal input and minimal knowledge of the syntax and the details of the knowledge graph. The details of this mechanism, along with example queries for other RDF datasets, are described in [7].

## 4.3 A more advanced SPARQL query

The following query returns all papers published at STOC 2018, and for each paper the number of all its authors, the number of its ORCID-certified authors, and the ratio between the two. The results are ordered by that ratio, largest first. The query makes use of the signatures in the dblp KG (see Section 2) as well as of several more advanced SPARQL features.

```
PREFIX dblp: <https://dblp.org/rdf/schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?paper (COUNT(?signature) AS ?num_authors) (COUNT(?orcid) AS ?num_orcid)
              (ROUND(100 * ?num_orcid / ?num_authors) AS ?perc) WHERE {
  ?paper dblp:publishedInStream <https://dblp.org/streams/conf/stoc> .
  ?paper dblp:yearOfEvent "2018"^^xsd:gYear .
  ?paper dblp:hasSignature ?signature .
  OPTIONAL { ?signature dblp:signatureOrcid ?orcid }
}
GROUP BY ?paper
ORDER BY DESC(?perc)
```

Run this query ↗



Here are the top-5 results:

?paper	?num_authors	?num_orcid	?perc
<a href="https://dblp.org/rec/conf/stoc/Cheraghchi18">https://dblp.org/rec/conf/stoc/Cheraghchi18</a>	1	1	100
<a href="https://dblp.org/rec/conf/stoc/Filos-RatsikasG18">https://dblp.org/rec/conf/stoc/Filos-RatsikasG18</a>	2	2	100
<a href="https://dblp.org/rec/conf/stoc/BergBKMZ18">https://dblp.org/rec/conf/stoc/BergBKMZ18</a>	5	4	80
<a href="https://dblp.org/rec/conf/stoc/ChattopadhyayKL18">https://dblp.org/rec/conf/stoc/ChattopadhyayKL18</a>	4	3	75
<a href="https://dblp.org/rec/conf/stoc/ByrkaSS18">https://dblp.org/rec/conf/stoc/ByrkaSS18</a>	3	2	67

Let us break down the main components of this query:

- Each paper has one signature per author, that is, the pattern `?paper dblp:hasSignature ?signature` will have one match for each author of each paper.
- A signature might or might not have an ORCID associated with it. The `OPTIONAL` ensures that no signature will be left out, but if there is no ORCID, the value for `?orcid` is undefined.
- The `GROUP BY` groups the information by paper, that is, there will be one row per paper in the result. As a consequence, any other variable used in the `SELECT` clause has to be aggregated so that we get one value for each paper: `COUNT(?signature)` counts the number of authors, `COUNT(?orcid)` counts the number of ORCIDs that are not undefined, and `?perc` is computed as the ratio between the two, expressed as a percentage and rounded to the nearest integer.
- The `ORDER BY` ensures that the results are ordered by the percentage, highest percentage first. Note that by default, the result of a SPARQL query is unordered (and an endpoint can produce them in an arbitrary order).

#### 4.4 Federated queries

Federated queries request data from more than one SPARQL endpoint. By design, RDF and SPARQL are particularly well suited for such queries because there is no dataset-specific schema (conceptually, every RDF dataset is just a set of triples) and because all identifiers are globally unique (just like IRIs). Connections between datasets are established by reusing identifiers from the other dataset or by having extra triples that relate the identifiers to each other.

For example, the following query returns all SIGIR authors that exist in Wikidata with a link to dblp (via Wikidata's `wdt:P2456` predicate), and the location of their birthplace (which can then be shown on a map).

```
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX dblp: <https://dblp.org/rdf/schema#>
SELECT ?author_dblp ?author_name ?num_papers ?location WHERE {
  { SELECT ?author_dblp (COUNT(?paper) AS ?num_papers) WHERE {
    ?paper dblp:authoredBy ?author_dblp .
    ?paper dblp:publishedInStream <https://dblp.org/streams/conf/sigir> .
  } GROUP BY ?author_dblp }
  ?author_dblp dblp:primaryCreatorName ?author_name .
  ?author_dblp dblp:wikidata ?person_wd .
  SERVICE <https://query.wikidata.org/sparql> {
    # ?person_wd wdt:P2456 [] .
    ?person_wd wdt:P19 ?place_of_birth .
    ?place_of_birth wdt:P625 ?location .
  }
}
```

Run this query [↗](#)

### 3:18 The dblp Knowledge Graph and SPARQL Endpoint

Let us break down the three main components of this query:

- The first four lines of the query body are a so-called subquery, which is a full SPARQL query enclosed in `{ ... }`. The result of that subquery is a table with one row per dblp author and two columns: the author ID from dblp and the number of papers from that author in dblp.
- The next two lines augment that table by two more columns: the name of the author and the Wikidata IRI of that author. Note that both of these predicates are *functional*, that is, for each distinct subject there is at most one object.
- The remaining four lines of the query body are a SPARQL query to another SPARQL endpoint, with the URL `https://query.wikidata.org/sparql`. The result is a table with one row per entity in Wikidata that has a birthplace (these are mostly people) and three columns: the IRI of that person, their birthplace, and the coordinates of that birthplace.<sup>28</sup> If the IRIs for `?person_wd` from Wikidata are compatible with the IRIs for `?person_wd` from dblp (which they are), the join of the two tables then gives the desired result.
- The reason for the commented out first line of the `SERVICE` query is as follows. Without that line (or when it's commented out), the `SERVICE` query produces a large result, namely a table of all people in Wikidata together with their birthplace and the respective coordinates (3.5 million rows at the time of this writing). Transferring this result to the dblp SPARQL endpoint would be very expensive, and there are two ways to avoid that. One way makes use of the fact that the part of the query before the `SERVICE` gives only relatively few results, namely one row per author who has published at SIGIR (around one thousand at the time of this writing). The corresponding matches for `person_wd` can be sent to the Wikidata SPARQL endpoint using a `VALUES` clause to restrict the result of the `SERVICE` query.<sup>29</sup> QLever indeed automatically performs this optimization for small sub-results, so also for the given query (The exact threshold is configurable). The other way is to comment in the commented out line, which would restrict the result of the `SERVICE` query to only those persons with a dblp ID (around 71K at the time of this writing). For this particular query, the second way has the disadvantage that the query excludes dblp authors who do have an entry in Wikidata, but where the `wdt:P2456` predicate, which links authors to their dblp identifier, is missing (18 authors at the time of this writing). The second way would however be necessary if we wanted to perform the query for all 3.6 million dblp authors (not limited to SIGIR).

#### 4.5 Querying both the dblp KG and the citation data

It is very natural to query the combined data from the dblp KG (Section 3.1) and the added citation data (Section 3.2). We could have set up a separate endpoint for each of these datasets and then use federated queries as shown in the previous section. However, there is always an overhead associated with federated queries because potentially large amounts of data have to be transferred between endpoints in one of the standard serialization formats. We therefore decided to provide both datasets in a single endpoint, as explained in Section 3.3.

Here is a typical example query to our endpoint that makes use of both datasets. It results in a list of all publications of Donald E. Knuth with at least one citation, ordered by the number of citations (most cited paper first).

---

<sup>28</sup>We here assume that both of these predicates are functional, which may not always be true. This could be addressed by a slightly more complex query, or by accepting that there will be multiple rows for the same person.

<sup>29</sup>This optimization is even discussed and suggested in the official SPARQL standard, see <https://www.w3.org/TR/sparql11-federated-query/#values>.

```

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX cito: <http://purl.org/spar/cito/>
PREFIX dblp: <https://dblp.org/rdf/schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?publ (COUNT(?cite) as ?count_cite) (SAMPLE(?label) as ?sample_label)
WHERE {
  ?publ rdf:type dblp:Publication .
  ?publ dblp:authoredBy ?author .
  ?publ rdfs:label ?label .
  ?author dblp:creatorName "Donald E. Knuth" .
  ?publ dblp:omid ?omid .
  ?cite cito:hasCitedEntity ?omid .
}
GROUP BY ?publ
ORDER BY DESC(?count_cite)

```

Run this query [↗](#)

This query is easy to understand given the concepts already explained in the previous sections. The second to last pattern of the query connects each publication to its OMID (the identifier used by OpenCitations, see Section 2.1.1). The last pattern produces one match for `?cite` for every citation. Grouping by `?publ` and using `COUNT(?cite)` for aggregation gives us the number of citations per publication.

Note that the result will only include publications that have an OMID and at least one citation. For the query above, this is true for only 100 of Donald Knuth’s 184 publications in dblp. To include all publications, the last two patterns could be included in an `OPTIONAL { ... }`, see Section 4.3.

## 4.6 Performance

Our SPARQL endpoint and its user interface described in Section 3.3, as well as the embedded queries described in Section 3.5, are all powered by the QLever SPARQL engine. QLever is free and open-source software (FOSS), provided under a permissive license. QLever’s primary design goal is to be efficient even on very large knowledge graphs (with up to hundreds of billions of triples), on a single machine using (relatively cheap) standard hardware. Compared to other knowledge graphs, the dblp KG is medium-sized (around 400 million triples), even if combined with the OpenCitations data (giving a total of around 1.2 billion triples). But even on a graph of this size, queries can be expensive and an efficient engine is key for a good user experience.

**Indexing time.** Like all SPARQL engines except the most basic ones, QLever precomputes special index data structures based on the input data, in order to enable fast queries. This pre-computation is called *indexing*. On a PC with an AMD Ryzen 9 7950X processor with 16 cores, 128 GB of RAM, and a 2 TB NVMe SSD, indexing the dblp KG takes around 4 minutes, while indexing the combined data takes around 12 minutes. Indexing times for QLever are roughly proportional to the number of triples in the input data.

**Query times.** Table 8 shows the query times (including the time to download the complete result) for a selection of six queries, against a SPARQL endpoint running on the same machine as above. The queries were chosen manually to cover a spectrum of queries that users typically ask and

## 3:20 The dblp Knowledge Graph and SPARQL Endpoint

■ **Table 8** Query times in seconds and result sizes (number of rows × number of columns) for a selection of six queries on the dblp KG. Clicking on the query name takes you to the full query. The time for downloading the full result is included, hence the larger time for the fifth query.

Query	Result size	QLever	Comment
All papers published in SIGIR <a href="#">↗</a>	6,264 x 3	0.02 s	Two simple joins, nothing special
Number of papers by venue <a href="#">↗</a>	19,954 x 2	0.02 s	Scan of a single predicate with GROUP BY and ORDER BY
Author names matching REGEX <a href="#">↗</a>	513 x 3	0.05 s	Joins, GROUP BY, ORDER BY, FILTER REGEX
All papers in DBLP until 1940 <a href="#">↗</a>	70 x 4	0.11 s	Three joins, a FILTER, and an ORDER BY
All papers with their title <a href="#">↗</a>	7,167,122 x 2	4.2 s	Simple, but must materialize large result (problematic for many SPARQL engines)
All predicates ordered by size <a href="#">↗</a>	68 x 3	0.01 s	Conceptually requires a scan over all triples, but huge optimization potential

different complexities regarding the query processing. This is not a complete evaluation and just meant to give an impression. For a more extensive performance evaluation and for a comparison against other SPARQL engines, see the QLever Wiki and the publications listed there.<sup>30</sup>

We remark that all queries except the first are non-trivial and pose significant performance challenges to other SPARQL engines. The second query requires a scan over all 7.1M publication-venue pairs in the data. The third query requires the materialization of over 10K strings and a REGEX evaluation on each. The fourth query filters out 63 publications from over 7.2M. The fifth query requires the materialization and downloading of 7.2M paper IDs and titles. The sixth query conceptually requires a scan of the complete dataset.

## 5 Discussion and outlook

In this article, we introduced the dblp Knowledge Graph (dblp KG), an up-to-date semantic representation of the knowledge contained in the dblp computer science bibliography. We also introduced our new public SPARQL endpoint as a powerful new tool to explore dblp’s bibliographic data and to create new insights. One particular advantage of the dblp KG is that it can be easily combined with other scholarly graphs using common identifiers. We have demonstrated this by combining the dblp and OpenCitations data in our query service. Just as easily, dblp could be joined with with, e.g., biographical data from Wikidata or the subject area classification from the ORKG.

The dblp KG is already in active use by the community. The linked open data live API (Section 3.6) alone receives more than one million requests from more than 25,000 IPs each month. The dblp KG RDF dump is downloaded about one thousand times each month. In recent research,

<sup>30</sup> Go to <https://github.com/ad-freiburg/qllever/wiki/> and search for “performance”.

[5] uses the dblp KG to create a dataset for training and testing of question answering over Knowledge Graph (KGQA) systems. In [41], a natural language interface is built for the dblp KG, while [33] evaluates their universal question-answering platform using the dblp KG. In [4], an entity linking method has been proposed which links entities mentioned in text to their corresponding unique identifiers in the dblp KG.

Building upon the current iteration of the dblp KG and expanding its capabilities is an ongoing endeavor of the dblp team. A particular priority is the further utilization of only weakly structured semantic information listed on the dblp website, such as event dates or publisher information, as well as making it machine-actionable. The immediate next steps ahead are already clear: In the current DFG-LIS project *SmartER affiliations*,<sup>31</sup> the dblp team is intensifying its coverage of author affiliation information. Future iterations of the dblp KG will expand its model to add institution entities as first-class citizens to the graph and link affiliation information for authors and signatures. Also, the event history of conference and workshop series, together with metadata about the time and date of events, is contained aplenty in the dblp webpages and will be added to the dblp KG.

Having said that, there are several limitations that are probably out of the scope of what the dblp team can deliver. For example, the breakdown of person names into first, last, or middle name parts, gender information, or the annotation of the language a published work is written in, is out of reach since dblp has no comprehensive, reliable open data source for this kind of information. For the same reason, we cannot provide finer-grained type classifications, e.g., there will be no distinguishing between conference and workshop series, no distinguishing of full-paper from poster contributions, and no distinguishing of editorial articles from book review articles. Furthermore, email addresses or contact information (even if stated on published articles) will not be added to the dblp KG because of their privacy-sensitive nature. Finally, we have deliberately removed all links to authors and editors from `dblp:Withdrawn` publications to allow authors to exercise their right to be forgotten.

---

## References

- 1 Boanerges Aleman-Meza. Swetodblp. <https://lod-cloud.net/dataset/sweto-dblp>, 2007. Accessed on 2024-04-22.
- 2 Boanerges Aleman-Meza, Farshad Hakimpour, Ismailcem Budak Arpinar, and Amit P. Sheth. Swetodblp ontology of computer science publications. *J. Web Semant.*, 5(3):151–155, 2007. doi:10.1016/J.WEBSEM.2007.03.001.
- 3 Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In *The Semantic Web, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2007. doi:10.1007/978-3-540-76298-0\_52.
- 4 Debayan Banerjee, Arefa, Ricardo Usbeck, and Chris Biemann. Dblplink: An entity linker for the DBLP scholarly knowledge graph. In *ISWC 2023 Posters and Demos: 22nd International Semantic Web Conference, Athens, Greece, November 6-10, 2023*, volume 3632 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL: [https://ceur-ws.org/Vol-3632/ISWC2023\\_paper\\_428.pdf](https://ceur-ws.org/Vol-3632/ISWC2023_paper_428.pdf).
- 5 Debayan Banerjee, Sushil Awale, Ricardo Usbeck, and Chris Biemann. DBLP-QuAD: A question answering dataset over the DBLP scholarly knowledge graph. In *BIR 2023: 13th International Workshop on Bibliometric-enhanced Information Retrieval ECIR 2023, Dublin, Ireland, April 2, 2023*, volume 3617 of *CEUR Workshop Proceedings*, pages 37–51. CEUR-WS.org, 2023. URL: <https://ceur-ws.org/Vol-3617/paper-05.pdf>.
- 6 Hannah Bast and Björn Buchhold. QLever: A query engine for efficient SPARQL+Text search. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, pages 647–656. ACM, 2017. doi:10.1145/3132847.3132921.
- 7 Hannah Bast, Johannes Kalmbach, Theresa Klumpp, Florian Kramer, and Niklas Schnelle. Efficient and effective SPARQL autocompletion on very large knowledge graphs. In *Proceedings of CIKM 2022, Atlanta, GA, USA, Octo-*

---

<sup>31</sup><https://gepris.dfg.de/gepris/projekt/515537520>

- ber 17-21, 2022, pages 2893–2902. ACM, 2022. doi:10.1145/3511808.3557093.
- 8 Hannah Bast, Johannes Kalmbach, Claudius Kozren, and Theresa Klumpp. Knowledge graphs. In Omar Alonso and Ricardo Baeza-Yates, editors, *Information Retrieval: Advanced Topics and Techniques*, volume 60 of *ACM Books*. Association for Computing Machinery, New York, NY, USA, 1 edition, 2025. doi:10.1145/3674127.
  - 9 Christian Bizer. DBLP bibliography database in RDF (fu berlin). <https://lod-cloud.net/dataset/fu-berlin-dblp>, 2007. Accessed on 2024-04-22.
  - 10 Christian Bizer and Andy Seaborne. D2RQ - treating non-RDF databases as virtual RDF graphs. In *Proceedings of ISWC 2004) Posters, Hiroshima, Japan, November 7-11, 2004*. Springer, 2004. URL: <http://iswc2004.semanticweb.org/posters/PID-SMCVRKBT-1089637165.pdf>.
  - 11 LOD W3C SWEO Community. Sweoig/taskforces/communityprojects/linkingopendata. <https://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData/>, April 2007. Accessed on 2024-04-22.
  - 12 Richard Cyganiak, David Wood, and Markus Lanthaler. Rdf 1.1 concepts and abstract syntax. <https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>, february 2014. Accessed on 2024-05-23.
  - 13 Marilena Daquino, Silvio Peroni, David M. Shotton, Giovanni Colavizza, Behnam Ghavimi, Anne Lauscher, Philipp Mayr, Matteo Romanello, and Philipp Zumstein. The opencitations data model. In *The Semantic Web - ISWC 2020, Athens, Greece, November 2-6, 2020, Proceedings, Part II*, volume 12507 of *Lecture Notes in Computer Science*, pages 447–463. Springer, 2020. doi:10.1007/978-3-030-62466-8\_28.
  - 14 Bruce D’Arcus and Frederick Giasson. The bibliographic ontology. <https://www.dublincore.org/specifications/bibo/>, May 2016. Accessed on 2024-05-13.
  - 15 dblp Team. dblp computer science bibliography – Monthly Snapshot RDF/N-Triple Release. doi:10.4230/dblp.rdf.ntriples.
  - 16 dblp Team. dblp computer science bibliography – Monthly Snapshot XML Release. doi:10.4230/dblp.xml.
  - 17 Jörg Diederich. DBLP in RDF (l3s). <https://lod-cloud.net/dataset/l3s-dblp>, 2007. Accessed on 2024-04-22.
  - 18 Said Fathalla, Christoph Lange, and Sören Auer. EVENTSKG: A 5-star dataset of top-ranked events in eight computer science communities. In *The Semantic Web - 16th International Conference, ESWC 2019, Portorož, Slovenia, June 2-6, 2019, Proceedings*, volume 11503 of *Lecture Notes in Computer Science*, pages 427–442. Springer, 2019. doi:10.1007/978-3-030-21348-0\_28.
  - 19 Julian Franken, Aliaksandr Birukou, Kai Eckert, Wolfgang Fahl, Christian Hauschke, and Christoph Lange. Persistent identification for conferences. *Data Sci. J.*, 21:11, 2022. doi:10.5334/DSJ-2022-011.
  - 20 Hugh Glaser. DBLP computer science bibliography (rkbexplorer). <https://lod-cloud.net/dataset/l3s-dblp>, 2007. Accessed on 2024-04-22.
  - 21 Hugh Glaser and Ian Millard. RKB explorer: Application and infrastructure. In *Proceedings of the Semantic Web Challenge 2007 co-located with ISWC 2007 + ASWC 2007, Busan, Korea, November 13th, 2007*, volume 295 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007. URL: <https://ceur-ws.org/Vol-295/paper13.pdf>.
  - 22 Stephanie Hagemann-Wilholt, Margret Plank, and Christian Hauschke. ConfIDent – an open platform for FAIR conference metadata. In *21st International Conference on Grey Literature “Open Science Encompasses New Forms of Grey Literature”, Hannover, Germany, October 22-23, 2019*, volume 21 of *GL Conference Series*, pages 47–51, 2019. doi:10.15488/9424.
  - 23 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d’Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. *Knowledge Graphs*. Synthesis Lectures on Data, Semantics, and Knowledge. Morgan & Claypool Publishers, 2021. doi:10.2200/S01125ED1V01Y202109DSK022.
  - 24 Google Inc., Yahoo Inc., Microsoft Corporation, and Yandex. Schema.org v26.0. <https://schema.org/version/26.0>, February 2024. Accessed on 2024-05-13.
  - 25 Mohamad Yaser Jaradeh, Allard Oelen, Kheir Eddine Farfar, Manuel Prinz, Jennifer D’Souza, Gábor Kismihók, Markus Stocker, and Sören Auer. Open research knowledge graph: Next generation infrastructure for semantic scholarly knowledge. In *Proceedings of the 10th International Conference on Knowledge Capture, K-CAP 2019, Marina Del Rey, CA, USA, November 19-21, 2019*, pages 243–246. ACM, 2019. doi:10.1145/3360901.3364435.
  - 26 Michael Ley. Die trierer informatik-bibliographie DBLP. In *Informatik ’97, Informatik als Innovationsmotor, 27. Jahrestagung der Gesellschaft für Informatik, Aachen, 24.-26. September 1997*, Informatik Aktuell, pages 257–266. Springer, 1997. doi:10.1007/978-3-642-60831-5\_34.
  - 27 Michael Ley. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *String Processing and Information Retrieval, 9th International Symposium, SPIRE 2002, Lisbon, Portugal, September 11-13, 2002, Proceedings*, volume 2476 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2002. doi:10.1007/3-540-45735-6\_1.
  - 28 Michael Ley. DBLP – Some lessons learned. *Proc. VLDB Endow.*, 2(2):1493–1500, 2009. doi:10.14778/1687553.1687577.
  - 29 Paolo Manghi, Alessia Bardi, Claudio Atzori, Miriam Baglioni, Natalia Manola, Jochen Schirrwagen, and Pedro Principe. The openaire research graph data model, April 2019. doi:10.5281/zenodo.2643198.
  - 30 Fabio Mercorio, Mario Mezzanzanica, Vincenzo Moscato, Antonio Picariello, and Giancarlo Sperli.

- A tool for researchers: Querying big scholarly data through graph databases. In *Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019*, volume 11908 of *Lecture Notes in Computer Science*, pages 760–763. Springer, 2019. doi:10.1007/978-3-030-46133-1\_46.
- 31 Mario Mezzanatica, Fabio Mercorio, Mirko Cesarini, Vincenzo Moscato, and Antonio Picariello. Graphdblp: a system for analysing networks of computer scientists through graph databases - graphdblp. *Multim. Tools Appl.*, 77(14):18657–18688, 2018. doi:10.1007/S11042-017-5503-2.
- 32 United States Library of Congress. Bibframe 2 ontology. <http://id.loc.gov/ontologies/bibframe-2-3-0/>, 2016. Accessed on 2024-05-13.
- 33 Reham Omar, Ishika Dhall, Panos Kalnis, and Essam Mansour. A universal question-answering platform for knowledge graphs. *Proc. ACM Manag. Data*, 1(1):57:1–57:25, 2023. doi:10.1145/3588911.
- 34 Silvio Peroni and David Shotton. Frbr-aligned bibliographic ontology (fabio). <http://www.sparontologies.net/ontologies/fabio>, 2012. Accessed on 2024-05-13.
- 35 Silvio Peroni and David M. Shotton. Opencitations, an infrastructure organization for open scholarship. *Quant. Sci. Stud.*, 1(1):428–444, 2020. doi:10.1162/QSS\_A\_00023.
- 36 Jason Priem, Heather A. Piwowar, and Richard Orr. Openalex: A fully-open index of scholarly works, authors, venues, institutions, and concepts. In *Proceedings of the 26th International Conference on Science and Technology Indicators (STI 2022)*, Granada, Spain, Sptember 7-9, 2022, 2022. doi:10.5281/zenodo.6936227.
- 37 David Shotton and Silvio Peroni. Datacite ontology. <http://www.sparontologies.net/ontologies/datacite>, September 2022. Accessed on 2024-05-13.
- 38 Selver Softic. Colinda - conference linked data. <https://lod-cloud.net/dataset/colinda>, 2015. Accessed on 2024-04-22.
- 39 Selver Softic, Laurens De Vocht, Erik Mannens, Martin Ebner, and Rik Van de Walle. COLINDA: modeling, representing and using scientific events in the web of data. In *Proceedings of DeRiVE 2015, Protoroz, Slovenia, May 31, 2015*, volume 1363 of *CEUR Workshop Proceedings*, pages 12–23. CEUR-WS.org, 2015. URL: [https://ceur-ws.org/Vol-1363/paper\\_2.pdf](https://ceur-ws.org/Vol-1363/paper_2.pdf).
- 40 Denny Vrandečić. The rise of wikidata. *IEEE Intell. Syst.*, 28(4):90–95, 2013. doi:10.1109/MIS.2013.119.
- 41 Ruijie Wang, Zhiruo Zhang, Luca Rossetto, Florian Ruosch, and Abraham Bernstein. Nlqxformui: A natural language interface for querying DBLP interactively. *CoRR*, abs/2403.08475, 2024. doi:10.48550/arXiv.2403.08475.





# FAIR Jupyter: A Knowledge Graph Approach to Semantic Sharing and Granular Exploration of a Computational Notebook Reproducibility Dataset

Sheeba Samuel<sup>1</sup> ✉ 

Distributed and Self-organizing Systems, Chemnitz University of Technology, Chemnitz, Germany

Daniel Mietchen<sup>1</sup> ✉ 

FIZ Karlsruhe – Leibniz Institute for Information Infrastructure, Germany

Institute for Globally Distributed Open Research and Education (IGDORE)

## — Abstract —

The way in which data are shared can affect their utility and reusability. Here, we demonstrate how data that we had previously shared in bulk can be mobilized further through a knowledge graph that allows for much more granular exploration and interrogation. The original dataset is about the computational reproducibility of GitHub-hosted Jupyter notebooks associated with biomedical publications. It contains rich metadata about the publications, associated GitHub repositories and Jupyter notebooks, and the notebooks' reproducibility. We took this dataset, converted it into semantic triples and loaded these into a triple store to create a knowledge graph – FAIR Jupyter – that we made accessible via a web service. This enables granular data exploration and analysis through queries that can be

tailored to specific use cases. Such queries may provide details about any of the variables from the original dataset, highlight relationships between them or combine some of the graph's content with materials from corresponding external resources. We provide a collection of example queries addressing a range of use cases in research and education. We also outline how sets of such queries can be used to profile specific content types, either individually or by class. We conclude by discussing how such a semantically enhanced sharing of complex datasets can both enhance their FAIRness – i.e., their findability, accessibility, interoperability, and reusability – and help identify and communicate best practices, particularly with regards to data quality, standardization, automation and reproducibility.

**2012 ACM Subject Classification** Information systems → Entity relationship models; Information systems → Information extraction

**Keywords and phrases** Knowledge Graph, Computational reproducibility, Jupyter notebooks, FAIR data, PubMed Central, GitHub, Python, SPARQL

**Digital Object Identifier** 10.4230/TGDK.2.2.4

**Category** Resource Paper

**Related Version** *Full Version*: <https://doi.org/10.48550/arXiv.2404.12935>

**Supplementary Material** see also Supplementary Material Statement in Section 7

*Software (Source Code)*: <https://doi.org/10.5281/zenodo.14197755> [69]

*Service (Website)*: <https://w3id.org/fairjupyter> [68]

**Funding** *Sheeba Samuel*: German Research Foundation (DFG), TRR-386, project number 514664767

*Daniel Mietchen*: MaRDI: DFG 460135501, BASE4NFDI/KGI4NFDI: DFG 521453681

**Acknowledgements** We thank the providers of infrastructure, data, and code that we used in this study. These include the PubMed Central repository at the U.S. National Center for Biotechnology Information and the Ara Cluster at the University of Jena as well as the Jupyter, Python and Conda communities and their respective dependencies. The authors gratefully acknowledge the computing

<sup>1</sup> Corresponding author. Both authors contributed equally to this work.



time made available to them on the high-performance computer at the NHR Center of TU Dresden. This center is jointly supported by the Federal Ministry of Education and Research and the state governments participating in the NHR ([www.nhr-verein.de/unsere-partner](http://www.nhr-verein.de/unsere-partner)). Special thanks go to JupyterCon, which provided the nucleus for our collaboration. We also thank Ramy-Badr Ahmed and Moritz Schubotz for help with registering the GitHub repositories from our corpus in the Software Heritage archive.

**Received** 2024-07-01 **Accepted** 2024-11-04 **Published** 2024-12-18

**Editors** Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler

**Special Issue** Resources for Graph Data and Knowledge

## 1 Introduction

In an age where research findings shape policy decisions and impact our understanding of the world, ensuring the reproducibility of scientific work is paramount. Jupyter notebooks [43] have revolutionized the way researchers share code, results, and documentation, all within an interactive environment, promising to make science more transparent and reproducible [30]. In research contexts, Jupyter notebooks often coexist with other software and various resources such as data, instruments, and mathematical models, all of which may affect scientific reproducibility. The evolving Jupyter ecosystem and the growing popularity of code sharing platforms like GitLab, Gitee, or Codeberg in parallel with GitHub require systematic approaches in future assessments of Jupyter reproducibility.

The FAIR principles – Findable, Accessible, Interoperable, and Reusable – play a crucial role in promoting effective data sharing practices across scientific disciplines, thereby enhancing reproducibility [80]. Data sharing has many facets, including the nature of the data, the purpose of the sharing, reuse considerations as well as various practicalities like the choice of file formats, metadata standards, licensing and location for the data to be shared. Here, we look into some of the practical and reuse aspects by mobilizing a previously shared reproducibility dataset in a more user-friendly fashion. That previous dataset arose from a study [71] of the computational reproducibility of Jupyter notebooks associated with biomedical publications. It is already publicly available [70] as a 1.5GB SQLite database contained within a ZIP archive of 415.6 MB (compressed) but in order to be explored, it needs to be unzipped, loaded into a SQLite server and queried via SQL. While these steps are routine for many, they nonetheless present a technical hurdle that stands in the way of broader use of the data, both in research and educational contexts.

For instance, imagine an instructor of a programming course for wet lab researchers who wants to present to her students some real-world Jupyter notebooks from their respective research field that use a specific Python module and are either fully reproducible or exhibit a given type of error. Wouldn't it be nice – and quicker than via the route outlined above – if she could get her students to run such queries directly in their browser, with no need to install anything on their system? Enabling students and instructors to do this is what we are aiming at. Likewise, reproducibility researchers can explore what aspects of notebooks, repositories, journals or papers correlate with reproducibility-related variables, editors can become aware of common issues in notebooks associated with their publications venue, while maintainers of software packages, ontologies or Jupyter-based services can explore the use of their resources in a reproducibility context.

One way to build a webservice addressing such use cases would be to use a web framework like Flask or Django in combination with a library like sqlite3 [10] to interact with the database. However, this approach complicates semantic integration with other resources, and so we chose instead to leverage semantic web standards [53] and build a demonstrator for converting a dataset

into a knowledge graph (KG). For people interested in granular exploration of datasets by way of knowledge graphs, it provides both a blueprint on how to get started and a working prototype to check against.

In this paper, we thus introduce **FAIR Jupyter**, a KG created from Jupyter notebooks hosted on GitHub and linked to biomedical publications sourced from PubMedCentral [59]. This resource is aimed at the intersection between KGs, reproducibility and Jupyter/Python: readers can engage with these individual areas according to their expertise, optionally in a way that is assisted by knowledge in the other areas. Here, we outline our two primary contributions: (i) The **FAIR Jupyter Ontology** developed using the NeOn methodology [76] reuses and extends the state-of-the-art ontologies to describe metadata related to the reproducibility of Jupyter notebooks, GitHub repositories, publications, and journals. (ii) The **FAIR Jupyter Knowledge Graph**, a KG containing 190 million triples about GitHub-hosted Jupyter notebooks extracted from PubMedCentral, and their reproducibility. In addition to metadata on repositories, journals, publications, and authors, the KG also includes fine-grained information on atomic elements of notebooks including cells, input, output, data and code dependencies, modules, libraries, styling, executions, execution order, cell features, and errors.

This emerging resource available as a KG identifies various issues, including imprecise statements of requirements such as data and code dependencies, the use of custom software libraries, inadequate documentation, the use of hard-coded paths, non-descriptive filenames, non-open data as well as the nature of default settings or policies used by deployment frameworks and infrastructures. We provide real-world examples highlighting specific problems, serving as a foundation for addressing such problems in order to improve computational reproducibility. Besides using the KG to showcase Jupyter-related information, we use Jupyter notebooks to showcase information about the KG. Additionally, we present SPARQL queries designed to achieve several objectives: (1) making it straightforward to reproduce results from [71] (this was already possible with the original data, but required more technical expertise and user-side infrastructure); (2) making it easier to filter the dataset for subgraphs of interest, e.g. things not covered in detail in [71], such as notebooks associated with a particular journal or written in languages other than Python; (3) combining information from this KG with information from external sources through federated queries.

The platform does not only target authors of research software like Jupyter notebooks but also addresses what other stakeholders in the research ecosystems can do to enhance computational aspects of reproducibility, such as providers of research infrastructures on which such computations are run, as well as reviewers, editors, and publishers of manuscripts reporting on computational aspects of research. It can be used by educators to provide practical tips to avoid common pitfalls and improve the reproducibility of computational analyses, particularly when conducted and shared through notebook environments like Jupyter.

Our work represents a significant milestone as the first systematic and large-scale effort to crawl, integrate, and semantically publish repositories of research-related Jupyter notebooks as a KG and to enrich that KG with reproducibility information pertaining to these notebooks. The current setup paves the way from the original one-off snapshots of Jupyter reproducibility to a future service that can provide a routinely updated dashboard-like overview of the Jupyter reproducibility landscape in biomedical research, while at the same time stimulating exploration and education around the role of knowledge graphs in this space.

## 2 Motivation and Use Cases

The need for reproducibility in computational research has become increasingly critical, particularly in fields that rely heavily on data science and computational methods. Jupyter notebooks are widely used to document and share these computational workflows, but ensuring their reproducibility poses

significant challenges [55]. By creating a structured resource that integrates Jupyter notebooks with their associated publications and repositories, we aim to provide a comprehensive platform for assessing and enhancing computational reproducibility and contributes to scientific reproducibility.

One of the key motivations for converting the existing dataset into a KG is to enable more granular and flexible querying of reproducibility information. KGs offer a dynamic and interconnected way to represent data, enabling users to explore and analyze relationships between different entities, in this context, journals, articles, authors, repositories, and specific notebook cells, as highlighted in Table 5. For example, users can easily query for specific error types across notebooks from different journals or research fields. With federated queries, a KG allows users to combine data from multiple sources, such as Wikidata, to enrich the analysis with additional contextual information about papers, authors, affiliations, and software packages. With KGs, researchers find relevant data and reproducibility information more efficiently, enhancing the discoverability of datasets by exposing relationships and metadata that might be buried in traditional databases. The motivation for developing this resource is also based on discussions from our community engagements [6] for reuse in a more user-friendly way.

Through a KG, researchers can query aggregated data across journals, articles, and repositories to analyze trends and patterns in computational reproducibility. They can investigate GitHub repositories to understand the structure and dependencies of computational workflows. Detailed queries at the notebook level allow for the examination of individual Jupyter notebooks, assessing their reproducibility and computational methods. Queries at the author level help study patterns of computational practices and reproducibility across different contributors. Granular queries at the level of individual cells or markdowns within notebooks provide insights into the execution and documentation of computational steps. Researchers can target specific requirements and dependencies stated within notebooks, highlighting areas critical for reproducibility. Subject-level queries enable the exploration of computational reproducibility trends within specific research fields or disciplines. Information from different entity types can also be combined in various fashions, e.g. to highlight exceptions that are common for notebooks associated with publications in a given field or journal. Additionally, queries can focus on the stylistic aspects of notebooks, examining conventions and best practices for documentation and presentation. Table 5 provides motivating examples of queries that users can explore. These use cases demonstrate the versatility of the resource in supporting varied research needs and enhancing computational reproducibility across different dimensions of scholarly communication and practice.

### **3 Related Works**

This section explores the current state-of-the-art approaches in computational reproducibility, along with the ontologies and KGs that have been developed to support these efforts.

#### **Computational Reproducibility**

Multiple studies have explored the reproducibility of computational research. For example, Gruning et al. [33] examined the specifics of computational reproducibility in the life sciences. Nust et al. [52] investigated the use of Docker, a containerization tool, in reproducibility contexts. Trisovic et al. [77] focused on the reproducibility of R scripts archived in an institutional repository.

Several studies have been conducted in recent years to explore the reproducibility of Jupyter notebooks [60, 55, 74, 79, 81]. Rule et al. [61] examined one million notebooks available on GitHub, exploring repositories, languages, packages, notebook length, and execution order, with a focus on the structure and formatting of computational notebooks. This study resulted in the proposal of ten best practices for writing and sharing computational analyses in Jupyter notebooks [60].

Another study [56] focused on the reproducibility of 1.4 million notebooks collected from GitHub, providing an extensive analysis of the factors impacting reproducibility in Jupyter notebooks. With respect to the use of Jupyter notebooks in research contexts, Chattopadhyay et al. [19] reported on a survey conducted among 156 data scientists, highlighting the challenges they face when working with notebooks, while Schröder et al. [74] manually examined the reproducibility of Jupyter notebooks linked to five publications from PubMed Central .

The datasets generated from these studies, although not directly linked to any specific publications, face significant challenges. These datasets are often not easily queryable or reusable in a user-friendly manner. Additionally, they cannot be seamlessly linked or integrated using federated queries with other valuable sources such as Wikidata or DBpedia. This lack of interoperability and accessibility hinders the potential for broader analysis and insights, limiting their usability and impact within the research community.

## Ontologies

Semantic web technologies and ontologies play a crucial role in enhancing computational reproducibility. Ontologies, in particular, offer a standardized vocabulary and relationships that facilitate the consistent annotation of computational research, making it easier to understand and reproduce experiments [28, 29, 45, 67]. For instance, the use of ontologies such as PROV-O [45] and P-Plan [28] allows researchers to describe the provenance of data and the specifics of computational processes, ensuring that every step of analysis can be traced. Here, we focus on the ontologies that represent computational workflows. The existence of numerous well-established ontologies points to the maturity of the scientific workflow domain [28, 45]. A significant number of ontologies have also been developed in the computer science and artificial intelligence domain to describe computational processes [25, 63]. The REPRODUCE-ME ontology [64, 65] is the pioneering effort to describe the provenance of Jupyter notebooks by extending the PROV-O [45] and P-Plan [28] ontologies. As this work touches upon different areas like publication [54], journals, repositories [3], and notebooks [67], it is important that well-established ontologies are reused to describe the domain of scholarly publications [21] and computational reproducibility.

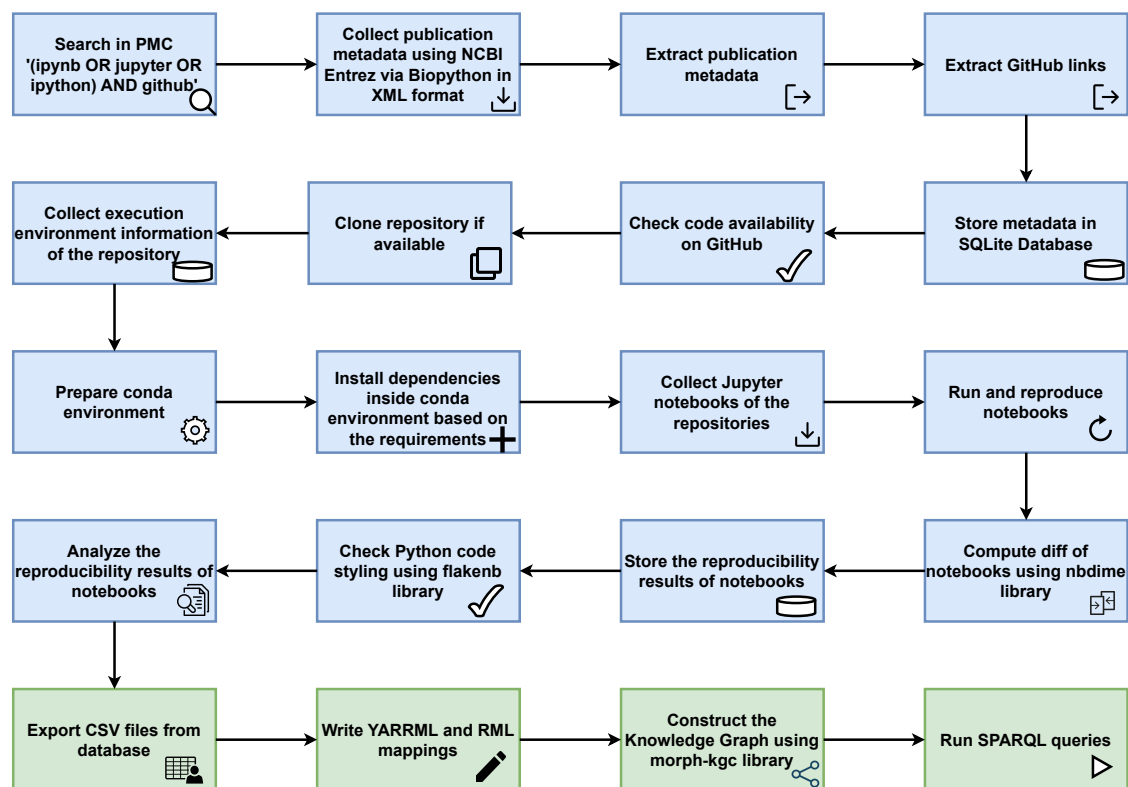
## Knowledge Graphs

While we are not aware of KGs about Jupyter notebooks or GitHub repositories, there have been a number of related efforts. These include the creation of KGs about FAIR computational workflows [29], PubMed [83], scientific software [40] and artificial intelligence tasks and benchmarks [15], along with tools for creating schemas and KGs from data [37]. Of particular relevance here is the Open Research Knowledge Graph (ORKG) [14] which provides a framework to represent, curate, and discover scholarly knowledge in a structured manner. It has also shown potential in assisting with reproducibility [39, 38]. To address scientific reproducibility in biomedicine, Liu et al. [48] developed ProvCaRe, a semantic provenance resource that was aggregating information extracted using NLP from the full text of 1.6 million biomedical articles. The repository (now defunct) required users to log in and contained 166 million provenance triples focusing on Study Method, Study Data, and Study Tool (including software, though computational reproducibility was not assessed). Another study [62] presents a semantic provenance graph from 75 sleep medicine articles, mapping provenance information to PROV-O [45] for querying reproducibility-related information, albeit not in a Jupyter context. The Microsoft Academic Knowledge Graph [26] was a large-scale RDF dataset with information on scholarly publications, authors, institutions, journals, and subject areas. It has been used to analyze repositories and their associated papers based on metrics like stars, forks, and the number of contributors [27]. Information about GitHub

repositories and GitHub contributors has been combined with information from the Wikidata KG to highlight gender-specific contribution patterns in open-source software projects [46]. An initial attempt has also been made to create a KG of Git repositories using a prototype tool called GitGraph. This tool extracts metadata from repositories, including commits and files, and constructs a KG. Graph4Code [12] is another relevant effort that is designed to generate KGs from code, supporting applications like program search, code understanding, bug detection, and automation. It complements our efforts, as its focus is on representing the structure of Python scripts and the flow of data through a script's components, rather than computational reproducibility.

These efforts collectively contribute to a better understanding and improvement of reproducibility in computational research, highlighting the importance of structured metadata, provenance, and advanced tools for maintaining the integrity and reliability of scientific workflows. To our knowledge, no prior work has undertaken a systematic effort to describe and semantically publish large-scale reproducibility analyses of Jupyter notebooks from research publications as a KG.

## 4 Methods



■ **Figure 1** Workflow overview. The blue workflow was used to construct the original dataset [70] and is described in [71], while the subsequent KG construction workflow in green represents the current study.

Our workflow has two main components, as shown in Figure 1. The first is the generation of the Jupyter notebook reproducibility dataset. The second is the conversion of this dataset into the FAIR Jupyter KG, which forms the focus of the present study.

## 4.1 Computational Reproducibility Dataset Generation

This section represents a summary of the methodology used in [71]. Briefly, we queried PubMed Central (PMC, cf. [59]) for publications mentioning GitHub alongside keywords such as “Jupyter”, “ipynb” (the file extension for Jupyter notebooks), or “IPython” (a predecessor to Jupyter).<sup>2</sup> Utilizing the primary PMC IDs obtained this way, we then retrieved publication records in XML format using the *efetch* function and collected publication metadata from PMC using NCBI Entrez utilities via Biopython [22].

Next, we processed the XML data retrieved from PMC by storing it in an SQLite database. Our database encompassed details regarding journals and articles, populating it with metadata including ISSN (International Standard Serial Number), journal and article titles, PubMed IDs, PMC IDs, DOIs, subjects, submission, acceptance, and publication dates, licensing information, copyright statements, keywords, and GitHub repository references mentioned in the publication. Additionally, we extracted associated Medical Subject Headings (MeSH terms) [8] for each article. These terms, assigned during indexing in the PubMed database, are hierarchical. We obtained the top-level MeSH term by querying the MeSH RDF API through SPARQL queries to the SPARQL endpoint [9]. This aggregation resulted in 108 top-level MeSH terms in our dataset, serving as proxies for the subject areas of the articles.

We extracted the GitHub repositories mentioned in each article, including the abstract, body, data availability statement, and supplementary sections.<sup>3</sup> After this preprocessing, we associated each article with the GitHub repositories extracted from it as well as with the journal in which the article had been published, and we gathered author information in a separate database table, including first and last names, ORCID, and email addresses.

We verified the availability of GitHub repositories mentioned in the articles and, if existing, cloned them, based on the main branch, and gathered repository details including creation, update, and push dates, releases, issues, license details, etc. using the GitHub REST API [5]. Additionally, we extracted details for each notebook provided in the repository, such as name, nbformat, kernel, language, cell types, and maximum execution count, and extracted source code and output from each cell using Python Abstract Syntax Tree (AST) for further analysis.

After the notebook collection, we gathered execution environment details by examining dependency declarations in repository files like requirements.txt, setup.py, and pipfile. After collecting the necessary Python notebook execution information, we prepared a conda environment based on the declared Python version, installing dependencies listed in files such as requirements.txt, setup.py, and pipfile. For repositories lacking specified dependencies, the pipeline executed notebooks by installing all Anaconda dependencies, leveraging Anaconda’s comprehensive data science package suite. We also conducted Python code styling checks using the flake8 [4] library, which enforces code style guidelines outlined in PEP 8, to collect all detected errors, obtaining information on the error code and description.

We executed our pipeline on 27<sup>th</sup> March 2023, and it ran until 9 May 2023, for a total of 43 days. The code has been adapted from [55, 66]. We utilize this method to reproduce Jupyter notebooks from GitHub repositories, as outlined in [55]. Additionally, we leverage ReproduceMeGit [66], which uses the nbdime library [57] to compare execution results with the original results. This forms the foundation of our code for the reproducibility study.

---

<sup>2</sup> We used the search query “(ipynb OR jupyter OR ipython) AND github”.

<sup>3</sup> Since the GitHub repositories had been stated in a number of different formats, we harmonized them to “https://github.com/username/repositoryname”.

## 4.2 FAIR Jupyter Ontology and KG Construction

### Competency Questions (CQs)

The requirements for ontology construction are driven by the requirements used for generating the initial dataset and the CQs are based on the research questions that arose from the initial pipeline. We used the NeOn methodology [76] for constructing the FAIR Jupyter ontology. We present the Ontology Requirement Specification Document (ORSD) which outlines the purpose, scope, implementation language, intended end-users and uses of the ontology, and the set of requirements the ontology should fulfill, presented in the form of competency questions. These competency questions are organized into eight categories, addressing the domain knowledge that needs to be represented, as detailed in Table 5 and 6.

### Data modeling

In this section, we provide a brief description of the ontology model used in the construction of the FAIR Jupyter KG. Overall, it contains 22 classes, as outlined in Figure 2. They are centred around notebooks, notebook cells, repositories and articles, each of which are linked to several other classes. We reused the following ontologies for describing these entities: PROV-O [45], REPRODUCE-ME [64], P-Plan [28], PAV [21], DOAP [3] and FaBiO [54].

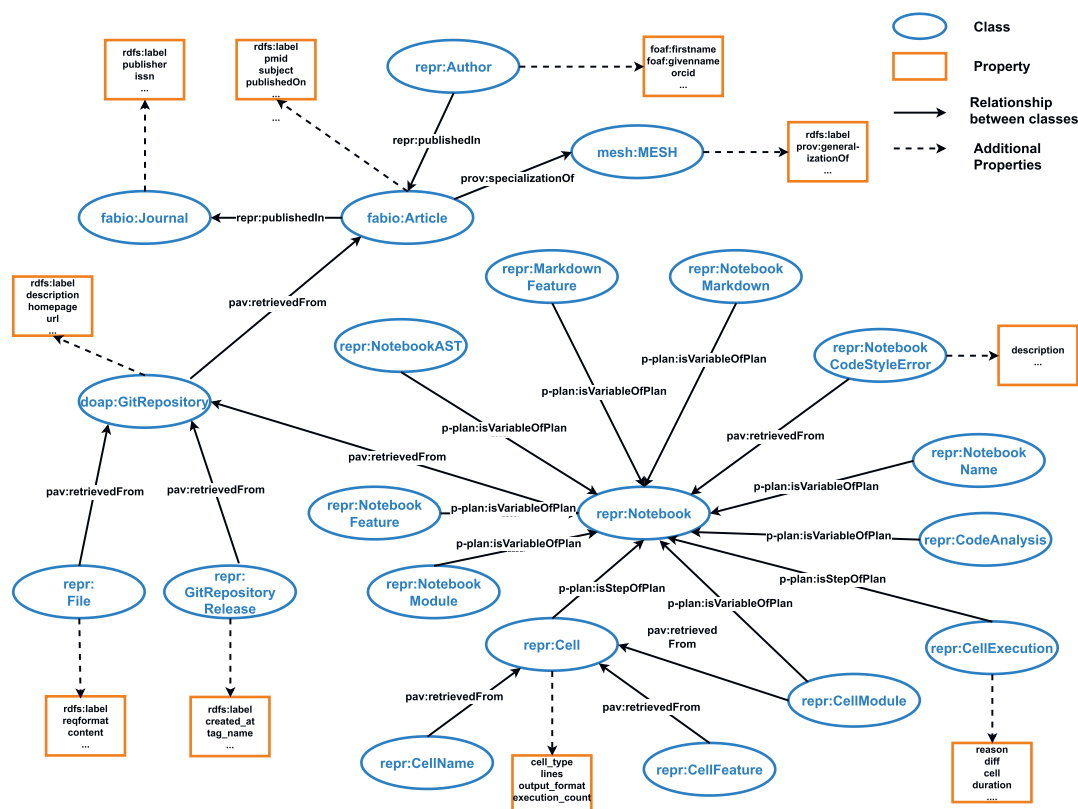
Building on PROV and P-Plan, the REPRODUCE-ME ontology captures provenance information for individual Jupyter Notebook cells [64], in addition to the end-to-end scientific experiment with real-life entities like instruments and specimens, as well as human activities such as lab protocols and screening [67]. Hence, we used and extended this model to construct the FAIR Jupyter KG.

We reused `fabio:Article` to link the publications in our dataset and `fabio:Journal` to represent the journal where the article was published. FaBiO is an ontology that helps represent information about publishable works (articles, books, etc.) and the bibliographic references that connect them [54]. A GitHub repository is represented using the class `doap:GitRepository` from the DOAP ontology [3], which is used to describe open source software projects. We reused the class `repr:Notebook` to represent the Jupyter Notebooks, which is a subclass of `p-plan:Plan` extending the class `prov:Plan`. The PROV-O [45] is a W3C recommendation providing foundation to implement provenance applications. We use two object properties of the PROV-O ontology: `prov:specializationOf` and `prov:generalizationOf` to show that the article is a specialization of a MESH term and the corresponding MESH term a generalisation of the top level MESH term in its hierarchy. To denote where a resource is retrieved from (e.g., `repr:Notebook` is `pav:retrievedFrom doap:GitRepository`), we have used the object property `pav:retrievedFrom`. PAV which builds on the PROV-O ontology, is a lightweight ontology for tracking Provenance, Authoring and Versioning and describes information about authorship, curation, and versioning of online resources [21]. Each individual cell of a computational notebook `repr:Cell` is described as a step of a `repr:Notebook` using the object property `p-plan:isStepOfPlan`. To describe the different features of notebooks and their cells, we reused the object property `p-plan:isVariableOfPlan`.

### Mappings and KG construction

We used the CSV files exported from the database [70] as input for the YARRRML mapping [36]. YARRRML is a human-readable text-based format for expressing declarative rules to generate Linked Data from different data sources. Listing 1 shows an example of a YARRRML mapping used for expressing the rules for “repositories” and “notebooks”. To facilitate adaptation to a variety of





■ **Figure 2** Partial outline of the data model used in FAIR Jupyter. Classes of entities (represented by ellipses) and the class properties (represented by orange rectangles) were inferred from the original dataset, and – along with relationships between them (arrows) – expressed in terms of relevant ontologies. Note that requirement files and repository files are both represented as `repr:File`. While some properties have not been depicted here for clarity and some classes have not been included in our endpoint for performance reasons, all the underlying files – CSV exports of the original data, the RML and YARRML mapping files used for KG construction, and the resulting RDF triples – are available via [72].

use cases, we created mappings for each entity types in separate files. The mappings are created for the classes of the ontology (see Figure 2). We chose the same name for the user-chosen keys as the name provided in the database tables. The key source has the value of the corresponding CSV file of the entity type. We used the namespace `https://w3id.org/reproduceme/` for generating the IRI. We reused the properties from the existing ontologies wherever possible to generate the combination of predicates and objects. For others, we added them to the REPRODUCE-ME ontology. The YARRRML mappings were written using the YARRRML editor, Matey [7]. These mappings were then converted to RML mappings [24]. The generated RML mappings were further used as input to the Morph-KGC library [13]. Morph-KGC is a tool designed to build RDF knowledge graphs from different data sources using the R2RML and RML mapping languages. For large datasets, Morph-KGC’s use of mapping partitions significantly speeds up processing and reduces memory usage. For some classes (CellName, CodeAnalysis, RepositoryFile, and MarkdownFeature), their size triggered import errors, and so we excluded them from this prototype but we plan to include them in the future as we continue to develop the platform. All the triples generated were stored in N-triples format and are downloadable in bulk from [72].

■ **Listing 1** Part of a YARRRML mapping for repositories and notebooks.

```

mappings:
repositories:
  sources:
    - [data/repositories.csv~csv]
  s: https://w3id.org/reproduceme/repository_$(id)
  po:
    - [a, doap:GitRepository]
    - [rdfs:label, $(repository)]
    - p: pav:retrievedFrom
      o:
        - mapping: article
          condition:
            function: equal
            parameters:
              - [str1, $(article_id), s]
              - [str2, $(id), o]
notebooks:
  sources:
    - [data/notebooks.csv~csv]
  s: https://w3id.org/reproduceme/notebook_$(id)
  po:
    - [a, repr:Notebook]
    - [rdfs:label, $(name)]
    - [repr:kernel, $(kernel)]
    - [repr:language, $(language)]
    - p: pav:retrievedFrom
      o:
        - mapping: repositories
          condition:
            function: equal
            parameters:
              - [str1, $(repository_id), s]
              - [str2, $(id), o]

```

### Triple store

We use Apache Jena Fuseki [18] as a triple store to query our KG. The FAIR Jupyter KG is available at <https://reproduceme.uni-jena.de/#/dataset/fairjupyter/query>.

### Web interface for FAIR Jupyter: visual exploration and querying

We provide a web service at <https://reproduceme.uni-jena.de/fj> that facilitates the visual exploration and interactivity of the FAIR Jupyter ontology and knowledge graph. Through this platform, users can directly explore the KG via a browser interface written using vis.js [11] that displays all KG entities. By selecting an entity, users can view detailed attributes, access links to relevant ontologies, and examine all associated properties. The service also enables users to run their own SPARQL queries, providing a flexible environment for custom exploration. In addition, the website features a dedicated section showcasing all queries used in this paper, including those that reproduce the original results from [71], as well as additional queries – including federated ones – highlighting aspects of the data that were not discussed there. To assist users from various backgrounds, a comprehensive documentation page is provided, consolidating all resources and guidance in one place for ease of use.

## 5 Results

Table 1 shows some general statistics about the FAIR Jupyter KG, which consists of about 190 million triples taking up a total of about 20.6 GB in space. Taking inspiration from the Scholia frontend to Wikidata [51], we anticipate using the KG for creating profiles based on specific entity types. To this end, we created the FAIR Jupyter KG from multiple smaller graphs based on separate mappings for each entity type. We present the time it took to construct the KG, the number of mapping rules retrieved, the total triples generated for each entity and the total file size of the RDF file generated.

■ **Table 1** General statistics of the FAIR Jupyter KG. The graphs for four entity types (CellName, CodeAnalysis, RepositoryFile, and MarkdownFeature) were omitted from the prototype implementation for performance reasons but included here, as we plan to include them in future versions of the KG.

Entity	Time (in sec)	No. of mappings	Triples generated	File Size
Article	5.2	17	60589	7.8 MB
Author	5.1	6	121247	14.1 MB
Cell	39.8	10	5940797	645.9 MB
CellFeature	15.4	11	1340610	168.4 MB
CellModule	8.4	2	917367	120.2 MB
CellName	176.2	12	38609232	4.2 GB
CellExecution	8.4	15	125383	48.9 MB
CodeAnalysis	462.4	162	77295103	7.9 GB
Journal	4.7	7	4497	0.49 MB
MarkdownFeature	203.5	125	36693762	3.8 GB
MESH	43.2	3	9078	1.2 MB
Notebook	10.5	24	627127	65.2 MB
NotebookAST	32.0	159	3281939	327.7 MB
NotebookCodeStyle	14.5	4	216105	26.4 MB
NotebookFeature	8.5	29	374071	42.3 MB
NotebookMarkdown	37.5	125	3389551	353.2 MB
NotebookModule	9.4	31	498504	63.3 MB
NotebookName	47.4	115	619562	137.2 MB
Repository	7.1	38	183592	20.1 MB
RepositoryFile	100.6	5	19223736	2.6 GB
RepositoryRelease	6.4	9	252500	33.3 MB
RequirementFile	5.5	6	27865	14.3 MB
Total	1251.7	915	189812217	20.6 GB

With the architecture laid out in Figure 2 and Table 1, it becomes possible to interrogate the dataset about any of the entities, classes or their relationships in a granular fashion. An example query is provided in Figure 2, which asks for notebooks where the reproducibility run produced results identical to those reported in the original publication. It then sorts this set of notebooks by the number of cells and the execution duration of the notebook (both of these parameters can serve as a proxy for the complexity of the notebook itself or the computations triggered by it), and then it limits the results to 10.

In the following, we illustrate the diversity of possible queries by presenting three sets of further example queries. First, Table 2 shows queries corresponding to some figures and tables from the publication describing the original dataset [71]. Second, Table 3 shows a brief selection of other queries that can be queried over the FAIR Jupyter graph. Third, federated queries can be run that combine information from our KG with other KGs, and some examples of such federated queries are given in Table 4. A more comprehensive list of queries is available at

■ **Listing 2** Example query (for live version, see Table 3): Ten successfully reproduced Jupyter notebooks associated with articles indexed in PubMed Central.

```
# Ten successfully reproduced Jupyter notebooks associated with articles indexed in PubMed Central
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX repr: <https://w3id.org/reproduce/>
PREFIX p-plan: <http://purl.org/net/p-plan>
PREFIX pav: <http://purl.org/pav/>

SELECT DISTINCT
?notebook_url ?total_cells ?duration
WHERE {
    ?execution a repr:CellExecution ;
    p-plan:isStepOfPlan ?notebook ;    # notebook that was executed
    repr:duration ?duration .        # execution duration in seconds
    ?execution repr:processed ?processed_same_result .
    FILTER ((xsd:integer(?processed_same_result) = 51)) # identical results
    ?notebook pav:retrievedFrom ?repository ;    # repo with this notebook
        rdfs:label ?notebook_label ;    # notebook filename
        repr:total_cells ?total_cells .    # number of cells in notebook
    ?repository repr:url ?repo_url_base . # find repo on GitHub
    # create clickable link to notebook on GitHub
    BIND(
        URI(CONCAT( ?repo_url_base, "/blob/main/", ENCODE_FOR_URI(?notebook_label)))
        AS ?notebook_url)
}
# sort by number of cells, then duration, both in descending order
ORDER BY DESC(xsd:float(?total_cells)) DESC(xsd:float(?duration))
LIMIT 10 # limit the output to 10 results
```

■ **Table 2** SPARQL queries to the KG that reproduce materials from the original manuscript describing the dataset [71].

Figure no. in [71]	SPARQL query
Fig. 3	Research articles by research field
Fig. 4	Research field (MeSH terms) by the number of GitHub repositories that contain at least one Jupyter notebook.
Fig. 5	Journals with the highest number of articles that had a valid GitHub repository and at least one Jupyter notebook.
Fig. 6	Journals by the number of GitHub repositories and by the number of GitHub repositories with at least one Jupyter notebook.
Fig. 7	Journals by number of GitHub repositories with Jupyter notebooks.
Fig. 9	Programming languages of the notebooks.
Fig. 10	Relative proportion of the most frequent programming languages used in the notebooks per year.
Fig. 11	Python notebooks by minor Python version by year of last commit to the GitHub repository containing the notebook.
Fig. 12	Python notebooks by major Python version by year of first commit to the notebook's GitHub repository.
Fig. 19	Exceptions occurring in Jupyter notebooks in our corpus.
Fig. 22	Jupyter notebook exceptions by research field, taking as a proxy the highest-level MeSH terms of the article associated with the notebook.
Table 2	Notebooks with successful executions with same and different results
Table 4	Common Python code warnings/ style errors in our notebook corpus.

<https://w3id.org/fairjupyter>. Multiple queries with results involving the same entity types can be combined into a profile for that entity type, as described in [51], and we plan on working in this direction. In addition, we combined example queries into a Jupyter notebook, through which users can explore the queries and results, all available at [73], along with a notebook that demonstrates some simple benchmarking measurements for our KG, which are shown in Figure 3.

## 6 Discussion

In this work, we have demonstrated how the accessibility of an already openly and FAIRly shared dataset can be further improved by leveraging semantic web approaches for representing the data in a KG that can be readily explored from a web browser. That dataset had been created using a workflow for reproducing Jupyter notebooks from biomedical publications, and in the present work, we have enhanced it by representing the dataset's components – publications, GitHub repositories, Jupyter notebooks and reproducibility aspects thereof – using web ontologies.

Besides the central point of using KGs to increase the FAIRness of a dataset, the present manuscript has a number of other features that are rare or novel in the context of KGs – especially in combination – yet likely of interest to the KG community: (a) using a KG to reproduce a paper's figures and tables, (b) using a Jupyter notebook to document KG queries and results, (c) systematic archiving of in-scope software on Software Heritage, (d) paying attention to a plurality of both programming and natural languages (Julia next to Python, Malayalam in addition to English), (e) environmental footprint assessment, (f) ethics statement. While the latter two also apply to our initial manuscript, a-d do not.

## 4:14 FAIR Jupyter

■ **Table 3** General queries over the FAIR Jupyter graph.

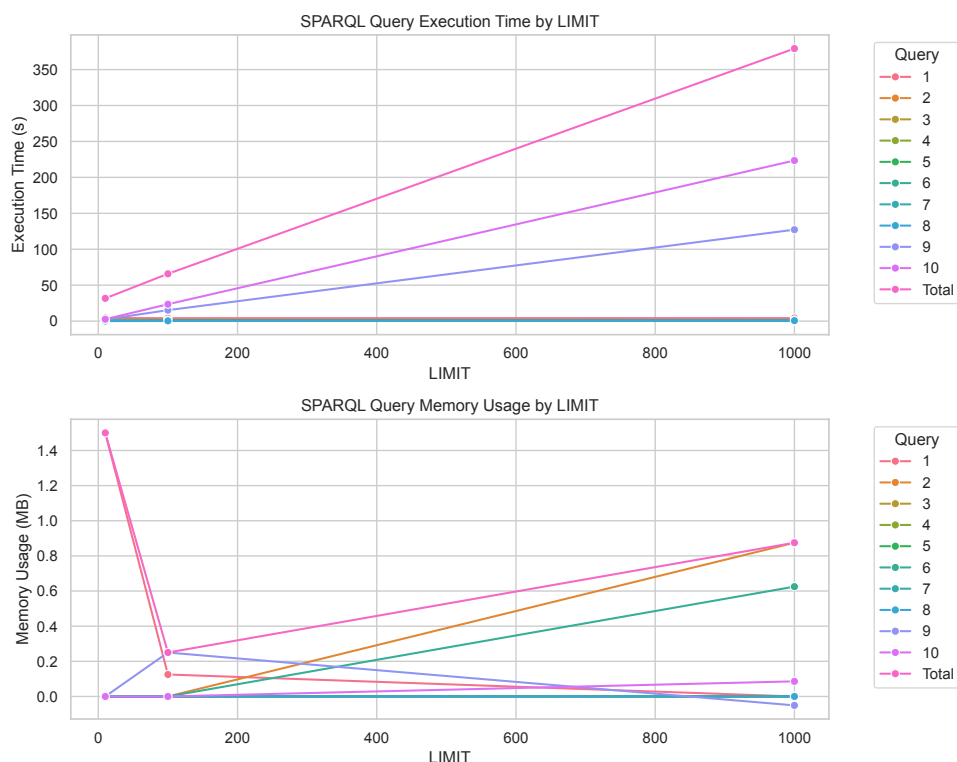
Description	SPARQL
Ten successfully reproduced notebooks, as per Figure 2	Query URL
Notebooks by search term: “immun” AND (“stem” OR “differentiation”)	Query URL
Article by keywords, e.g., “open source”	Query URL
Most common errors in immunology	Query URL
Most common errors in Nature journal	Query URL
MeSH terms ranked by “ModuleNotFoundError” frequency	Query URL
GitHub repositories by their stargazers count	Query URL
GitHub repositories and their Software Heritage snapshot	Query URL
Articles and repositories with notebooks in Julia	Query URL

■ **Table 4** Federated queries between FAIR Jupyter and external KGs. Wikidata [78] operates a SPARQL endpoint at <https://query.wikidata.org/>, while MaRDI – the Mathematical Research Data Initiative [23] – operates one at <https://query.portal.mardi4nfdi.de/>.

Description	SPARQL
Match articles between FAIR Jupyter and Wikidata via DOI	Query URL
Match articles between FAIR Jupyter and Wikidata via PMC ID	Query URL
Match articles between FAIR Jupyter and Wikidata via MeSH in a different language (here: Malayalam)	Query URL
Match articles between FAIR Jupyter and MaRDI via DOI and get co-used software	Query URL

The dataset is of particular interest for trainings and education as well as for showcasing real-world examples of actual research practices, from which both best practices and things to avoid can be synthesized. In order to enhance the dataset’s usefulness in such contexts, we have paid special attention to its Findability, Accessibility, Interoperability and Reusability, as per the FAIR Principles [80] for sharing research data. Since our data is about software, we also took into account adaptations of the FAIR Principles to software [44].

The original dataset as a whole was already well aligned with the FAIR Principles, and we added an additional layer of alignment by sharing individual elements of the original dataset in a FAIR way, so as to make it easier for humans and machines to engage with them. At a very basic level, the original dataset as a whole becomes more findable by virtue of the current manuscript and the FAIR Jupyter website pointing to it, more accessible by being available in additional formats (e.g. RDF), more interoperable by compatibility with additional standards (particularly ontologies, as per Figure 2) and more reusable through combinations of the above as well as the enhanced granularity. At a more profound level, the dataset’s individual facets and features – be it the classes shown in Figure 2 or the entities listed in Table 1 or any of the relationships between them – have become more FAIR, as they can now be searched, explored, aggregated, filtered and reused in additional user friendly ways both individually or in various combinations, both manually and programmatically. This KG approach facilitates additional routes for engaging with the original data and makes it easy to ask and address questions that were not included in the paper describing the original dataset, hence encouraging readers to build on our work. It also provides an additional level of reproducibility in that it allows for reproducing specific aspects of the original paper, as demonstrated with Table 2. Finally, the coupling of the KG with a



■ **Figure 3** Demonstration of performance measurements of the FAIR Jupyter SPARQL endpoint. Execution time (top) and memory usage (bottom) for a set of 10 SPARQL queries that have been taken from Tables 2, 3 and 4 and were run with different LIMIT values (10, 100, 1000). These measurements were run with a Jupyter notebook that is available at [73], along with the respective query results. With such a simple setup – and perhaps some additional measures like randomized order to reduce position-related artifacts – queries could be identified that would be suitable for assessing performance of the system.

public-facing web frontend, a visual graph browser and user-friendly example queries essentially turns the dataset from FAIR data into a FAIR service of potential utility in both research and education settings.

Enhancing a dataset this way and setting up such a KG service represents an additional effort in terms of data sharing. We cannot say at this point whether those efforts – which we have tried to outline in this manuscript – merit the actual benefits in our case, yet our example queries outline some of the potential benefits. We are factoring usage assessment and user feedback into our plans guiding future development of the platform. We are welcoming others to experiment with our workflows or to collaborate with us to adapt them to their use cases.

Assuming some level of usefulness of a service like FAIR Jupyter, keeping it useful requires additional steps like continued maintenance as well as updates in a timely manner. When we ran our Jupyter reproducibility pipeline that resulted in the original dataset, our search query in PubMed Central (cf. 4.1) yielded 3467 articles in March 2023, as opposed to 1419 in February 2021, 5126 in April 2024 and 5941 in October 2024. This translates into an average of several such articles being indexed in PubMed Central per day. We are working on establishing a pipeline that would regularly feed new articles and their notebooks into our KG and on highlighting the successful reproductions, e.g. by badges [42] displayed next to them or through nanopublications [17] sharing reproducibility information about them.

We are working towards extending the KG in other ways, too. For instance, we are exploring to include – or to federate – information about institutions and citations pertaining to articles with Jupyter notebooks, perhaps together with information about resources used alongside Jupyter (e.g. as per Figure 29 in [71] or the MaRDI query in Table 4). We are also conscious that our way of using MeSH terms – which are assigned by PubMed at an article level – does not necessarily represent an optimal way to associate topics with notebooks, for which approaches like a BERTopic [32] analysis of the notebooks themselves might be more suitable.

Furthermore, we could combine FAIR Jupyter with a ReproduceMeGit [66] service that would take Jupyter notebooks as input, assess their reproducibility (potentially even before the corresponding article is submitted to a publication venue) and – depending on the outcome – invoke the KG to suggest similar notebooks (e.g. based on dependencies or topics) with better reproducibility, better documentation or fewer styling issues as a mechanism to help notebook authors familiarize themselves with best practices. Likewise, the FAIR Jupyter KG would provide fertile ground for automated approaches at studying or enhancing computational reproducibility [16, 75].

The reproducibility of any particular notebook is not set in stone, and research questions could be formed around that (e.g. which dependencies contribute most to reproducibility decay, and how that changes over time), which an updated KG could help address, perhaps seeded by incorporating data from our initial run of the pipeline in 2021. This opens up questions around how best to represent time series data in KGs, e.g. as per [20]. Other lines of research could look at deep dependencies [50] of the Jupyter notebooks or their repositories and relate these to aspects of reproducibility – as discussed here – or of security, e.g. as per [47]. As our pipeline is automated, it lends itself to further integration with other automated workflows, for example in the context of workflow systems (e.g. [29]) or machine-actionable data management plans (e.g. [49]).

In its current state, FAIR Jupyter can already support various uses in educational settings, be that the identification of articles to choose for reprohack events [35], materials for lessons by The Carpentries [58], for general reproducibility activities in libraries [31] or for self-guided study by anyone wishing to learn about Jupyter, Python, software dependencies or computational reproducibility. These use cases could be expanded to include, for instance, correlates of reproducibility with individual Python modules or their versions.

We welcome contributions from others – including from other disciplines (e.g. as per [82]) – to the reproducibility pipeline, to the KG or to any of their suggested improvements or applications.

## 7 Supplementary Material Statement

The FAIR Jupyter service with links to the SPARQL endpoint, code and all the corresponding resources is available at <https://w3id.org/fairjupyter> under the GPL-3.0 license. We are committed to keeping it up for five years, i.e. until April 2029. The code used for generating the original dataset is available at <https://github.com/fusion-jena/computational-reproducibility-pmc>. The CSV files, the YARRRML and RML mappings used for constructing the KG are available at <https://github.com/fusion-jena/fairjupyter> and in Zenodo [69].

### Environmental footprint

To estimate the environmental impact of our computation, we leveraged a tool from the Green Algorithms project (<http://www.green-algorithms.org/>). This tool calculates the environmental footprint based on hardware configuration, total runtime, and location. To generate the original dataset, the pipeline consumed 373.78 kWh, resulting in a carbon footprint of approximately



126.58 kg CO<sub>2</sub>e, equivalent to 11.51 tree years when using default values for Germany. The pipeline for constructing the KG took 20.8 minutes, resulting in a carbon footprint of 7.33 g CO<sub>2</sub>e. The carbon footprint of the query execution from Table 2 is around 151.48mg CO<sub>2</sub>e.

### Ethical considerations

The original dataset contained the email addresses of the corresponding authors, which are available from PubMed Central as part of the respective article's full text. We have not included these author email addresses within the publicly available KG, since the increased accessibility of the data in the graph also increases the potential for misuse. However, we retain these emails internally to facilitate communication with authors regarding their repositories and the reproducibility of their work.

## 8 Impact of the Resource

This emerging resource which enhances reproducibility and facilitates information retrieval through KGs, has the potential for impact not just in the Semantic Web community but also other domain-specific communities that extensively work with Jupyter notebooks. Its methodologies are transferable to other domains, making it pertinent to the broader KG community. A bottleneck of our workflow is the availability of full-text access to research articles and the permission to mine them. These conditions are met for biomedicine through the PubMed Central platform that we used, and it provided us with JATS XML, which is straightforward to mine. In principle, our workflow could be adapted to use any other such full-text component instead or in addition. This seems technically doable for Biodiversity PMC [2], an initiative at the Swiss Institute of Bioinformatics that builds on the efforts of PubMed Central and enriches it with additional articles and other information from the biodiversity domain. Another full-text repository amenable to mining is of course arXiv [1], where much of the KG literature can be found already. In both cases, the number of additional Jupyter notebooks that the modified workflow would yield is small relative to the route we chose, but we will keep an eye on the respective developments. FAIR Jupyter can be leveraged for executing federated queries with other resources like Wikidata, as demonstrated in Table 4. This could be expanded further: for instance, one could retrieve additional information such as demographics for papers, authors, affiliations, and details about software packages – see [46] for some gender-based examples. Given the widespread use of Jupyter notebooks across various disciplines in data science, including KG applications, this resource targets a central artifact critical to computational research. It could serve as a guideline for developing recommendations and best practices for creating, maintaining, and executing computational notebooks [56, 41] or setting up services around that, e.g. as per [34]. The resource also offers opportunities to uncover new insights from the dataset that we have not yet explored. These could include, for instance, institutional rankings, or research background of the people involved in successful replications vs. replication problems.

We aim to distill insights from this study into recommendations and infrastructure that enhance the reproducibility of Jupyter notebooks and facilitate the validation of fundamental reproducibility standards. Addressing this challenge requires continued engagement from users and providers of computational resources. We are investigating ways to integrate our materials, workflows, and findings with educational initiatives such as The Carpentries. We are currently incorporating these materials into our own teaching, e.g. we are using and testing it while teaching a “Management of Scientific Data” course at the University of Jena. We are in touch with organizers of ReproHack events, who are interested in using our graph and pipeline to help event attendees find suitable examples to work on. The participants can collaborate to fix notebooks and

submit pull requests to repositories, fostering a community effort to improve reproducibility not only of Jupyter notebooks but also of associated repositories, datasets, and results. In the context of the National Research Data Initiative (NFDI) in Germany, two new projects (Jupyter4NFDI and KGI4NFDI) have recently been approved that will provide Jupyter and KG services to dozens of projects involving hundreds of institutions across Germany. We are involved with both and working on including FAIRJupyter into the education and training materials being prepared and disseminated in this context. Both through [71] and through our own editorial activities, we are in touch with a number of journals about their respective reproducibility efforts, where our pipeline and/or the graph are considered to become either part of those efforts or a reference or to be named as a collection of best practice examples or things to avoid. One of us is involved with reproducibility-focused research projects like TIER2 and also serves on the Steering Committee of the German Reproducibility Network. Last but not least, we have been invited to contribute a piece to the Jupyter community blog. Through all these channels, we are receiving feedback and suggestions for developing the resource further and adapting it to specific use cases. As for running our PMC pipeline continuously, we are currently writing this up as a preregistration study aimed at testing the reproducibility of [71] with data from the full year of 2024. We would pipe those 2024 data into a graph in early 2025 and then make the pipeline available to FAIRJupyter users.

## 9 Conclusion

In this paper, we present the FAIR Jupyter ontology and knowledge graph designed for the semantic sharing and granular exploration of a computational notebook reproducibility dataset. The proposed FAIR Jupyter dataset and knowledge graph are derived from GitHub-hosted Jupyter notebooks linked to biomedical publications. Using state-of-the-art processes and tools, this resource is made available on the web, adhering to best practices. The resulting KG, accessible via a SPARQL endpoint and a web service, exemplifies how to create such resources using advanced methodologies and tools. Although it primarily focuses on the biomedical domain, the methodologies are broadly applicable, making the resource valuable to the wider KG research community and for KG-related education. The central focus on Jupyter notebooks, a common tool in data science, underscores the resource's relevance across various disciplines. Our paper emphasizes technical documentation to enhance usability for both educational purposes and research. We hope that focus on the intersection between KGs, reproducibility and research software engineering will provide fertile ground for engagement from the respective communities and stimulate interactions between them. An example of further research benefiting from our graph would be reproducibility studies focused on deep dependencies of Jupyter notebooks (zooming in on landscape analyses of deep dependencies like [50]). We also plan to establish a pipeline to regularly add new articles and their notebooks into our KG, while highlighting successful reproductions.

---

## References

- 1 arXiv. URL: <https://arxiv.org/>.
- 2 Biodiversity PMC. URL: <https://biodiversitypmc.sibils.org/>.
- 3 DOAP: Description of a project. URL: <http://usefulinc.com/ns/doap#>.
- 4 flake8nb. URL: <https://github.com/s-weigand/flake8-nb>.
- 5 GitHub REST API. URL: <https://docs.github.com/en/rest/guides/getting-started-with-the-rest-api>.
- 6 JupyterCon'23. URL: <https://cfp.jupytercon.com/2023/talk/FSMWLQ/>.
- 7 Matey. URL: <https://rml.io/yarrml/matey/>.
- 8 MeSH (Medical Subject Headings). URL: <https://www.ncbi.nlm.nih.gov/mesh>.
- 9 MeSH SPARQL Endpoint. URL: <https://id.nlm.nih.gov/mesh/sparql>.
- 10 SQLite. URL: <https://www.sqlite.org>.
- 11 vis.js. URL: <https://visjs.org/>.
- 12 Ibrahim Abdelaziz, Julian Dolby, Jamie P. McCusker, and Kavitha Srinivas. A toolkit for gener-

- ating code knowledge graphs. In Anna Lisa Gentile and Rafael Gonçalves, editors, *K-CAP '21: Knowledge Capture Conference, Virtual Event, USA, December 2-3, 2021*, pages 137–144. ACM, 2021. doi:10.1145/3460210.3493578.
- 13 Julián Arenas-Guerrero, David Chaves-Fraga, Jhon Toledo, María S. Pérez, and Oscar Corcho. Morph-KGC: Scalable knowledge graph materialization with mapping partitions. *Semantic Web*, 15(1):1–20, 2024. doi:10.3233/SW-223135.
  - 14 Sören Auer, Viktor Kovtun, Manuel Prinz, Anna Kasprzik, Markus Stocker, and Maria Esther Vidal. Towards a knowledge graph for science. In *Proceedings of the 8th international conference on web intelligence, mining and semantics*, pages 1–6, 2018. doi:10.1145/3227609.3227689.
  - 15 Kathrin Blagec, Adriano Barbosa-Silva, Simon Ott, and Matthias Samwald. A curated, ontology-based, large-scale knowledge graph of artificial intelligence tasks and benchmarks. *Scientific Data*, 9(1):322, 2022.
  - 16 Ben Bogin, Kejuan Yang, Shashank Gupta, Kyle Richardson, Erin Bransom, Peter Clark, Ashish Sabharwal, and Tushar Khot. Super: Evaluating agents on setting up and executing tasks from research repositories, 2024. doi:10.48550/arXiv.2409.07440.
  - 17 Cristina-Iulia Bucur, Tobias Kuhn, Davide Ceolin, and Jacco van Ossenbruggen. Nanopublication-based semantic publishing and reviewing: a field study with formalization papers. *PeerJ Computer Science*, 9:e1159, 2023. doi:10.7717/peerj-cs.1159.
  - 18 Jeremy J Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83, 2004. doi:10.1145/1013367.1013381.
  - 19 Souti Chattopadhyay, Ishita Prasad, Austin Z Henley, Anita Sarma, and Titus Barik. What's wrong with computational notebooks? pain points, needs, and design opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2020. doi:10.1145/3313831.3376729.
  - 20 Zheyuan Chen, Yuwei Wan, Ying Liu, and Agustin Valera-Medina. A knowledge graph-supported information fusion approach for multi-faceted conceptual modelling. *Inf. Fusion*, 101:101985, 2024. doi:10.1016/j.inffus.2023.101985.
  - 21 Paolo Ciccarese, Stian Soiland-Reyes, Khalid Belhajjame, Alasdair J. G. Gray, Carole A. Goble, and Tim Clark. PAV ontology: provenance, authoring and versioning. *J. Biomed. Semant.*, 4:1–22, 2013. doi:10.1186/2041-1480-4-37.
  - 22 Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J. L. de Hoon. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 2009. doi:10.1093/bioinformatics/btp163.
  - 23 The MaRDI consortium. MaRDI: Mathematical Research Data Initiative Proposal, May 2022. doi:10.5281/zenodo.6552436.
  - 24 Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. Rml: A generic language for integrated rdf mappings of heterogeneous data. *Ldow*, 1184, 2014. URL: [https://ceur-ws.org/Vol-1184/ldow2014\\_paper\\_01.pdf](https://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf).
  - 25 Diego Esteves, Diego Moussallem, Ciro Baron Neto, Tommaso Soru, Ricardo Usbeck, Markus Ackermann, and Jens Lehmann. Mex vocabulary: a lightweight interchange format for machine learning experiments. In *Proceedings of the 11th International Conference on Semantic Systems*, pages 169–176, 2015. doi:10.1145/2814864.2814883.
  - 26 Michael Färber. The microsoft academic knowledge graph: A linked data source with 8 billion triples of scholarly data. In *The Semantic Web – ISWC 2019*, pages 113–129, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-30796-7\_8.
  - 27 Michael Färber. Analyzing the github repositories of research papers. In Ruhua Huang, Dan Wu, Gary Marchionini, Daqing He, Sally Jo Cunningham, and Preben Hansen, editors, *JCDL '20: Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020, Virtual Event, China, August 1-5, 2020*, JCDL '20, pages 491–492, New York, NY, USA, 2020. ACM. doi:10.1145/3383583.3398578.
  - 28 Daniel Garijo and Yolanda Gil. Augmenting PROV with plans in P-PLAN: scientific processes as linked data. In Tomi Kauppinen, Line C. Pouchard, and Carsten Kefler, editors, *Proceedings of the Second International Workshop on Linked Science 2012 - Tackling Big Data, Boston, MA, USA, November 12, 2012*, volume 951 of *CEUR Workshop Proceedings*. CEUR Workshop Proceedings, CEUR-WS.org, 2012. URL: <https://ceur-ws.org/Vol-951/paper6.pdf>.
  - 29 Carole Goble, Sarah Cohen-Boulakia, Stian Soiland-Reyes, Daniel Garijo, Yolanda Gil, Michael R. Crusoe, Kristian Peters, and Daniel Schober. FAIR Computational Workflows. *Data Intelligence*, 2(1-2):108–121, 2020. doi:10.1162/dint\_a\_00033.
  - 30 Brian E. Granger and Fernando Perez. Jupyter: Thinking and Storytelling With Code and Data. *Computing in Science and Engineering*, 23(2):7–14, 2021. doi:10.1109/MCSE.2021.3059263.
  - 31 Sabrina Granger. How research reproducibility challenges librarians' skill sets. A French librarian's perspective. *Journal for Reproducibility in Neuroscience*, 2, 2020. <https://jrn.trialanderror.org/pub/french-librarians-perspective>.
  - 32 Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based TF-IDF procedure. *CoRR*, abs/2203.05794, 2022. doi:10.48550/arXiv.2203.05794.
  - 33 Björn Grüning, John Chilton, Johannes Köster, Ryan Dale, Nicola Soranzo, Marius van den Beek, Jeremy Goecks, Rolf Backofen, Anton Nekrutenko, and James Taylor. Practical Computational Re-

- producibility in the Life Sciences. *Cell systems*, 6(6):631–635, 2018.
- 34 Björn Hagemeier, Arnim Bleier, Bernd Flemisch, Matthias Lieber, Klaus Reuter, and George Dogaru. Jupyter4nfdi, July 2024. doi:10.5281/zenodo.12699382.
  - 35 Kristina Hettne, Ricarda Proppert, Linda Nab, L. Paloma Rojas-Saunero, and Daniela Gawehns. Reprohacknl 2019: how libraries can promote research reproducibility through community engagement. *IASSIST quarterly*, 44(1-2):1–10, 2020.
  - 36 Pieter Heyvaert, Ben De Meester, Anastasia Dimou, and Ruben Verborgh. Declarative rules for linked data generation at your fingertips! In *The Semantic Web: ESWC 2018 Satellite Events: ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers 15*, pages 213–217. Springer, 2018. doi:10.1007/978-3-319-98192-5\_40.
  - 37 Nicolas Hubert, Pierre Monnin, Mathieu d’Aquin, Armelle Brun, and Davy Monticolo. Pygraft: Configurable generation of schemas and knowledge graphs at your fingertips. *CoRR*, abs/2309.03685, 2023. doi:10.48550/arXiv.2309.03685.
  - 38 Hassan Hussein, Kheir Eddine Farfar, Allard Oelen, Oliver Karras, and Sören Auer. Increasing reproducibility in science by interlinking semantic artifact descriptions in a knowledge graph. In *International Conference on Asian Digital Libraries*, pages 220–229. Springer, 2023. doi:10.1007/978-981-99-8088-8\_19.
  - 39 Mohamad Yaser Jaradeh, Allard Oelen, Kheir Eddine Farfar, Manuel Prinz, Jennifer D’Souza, Gábor Kismihók, Markus Stocker, and Sören Auer. Open research knowledge graph: next generation infrastructure for semantic scholarly knowledge. In *Proceedings of the 10th International Conference on Knowledge Capture*, pages 243–246, 2019. doi:10.1145/3360901.3364435.
  - 40 Aidan Kelley and Daniel Garijo. A framework for creating knowledge graphs of scientific software metadata. *Quantitative Science Studies*, 2(4):1423–1446, 2021. doi:10.1162/qss\_a\_00167.
  - 41 Dominik Kerzel, Birgitta König-Ries, and Sheeba Samuel. MLProvLab: Provenance management for data science notebooks. In *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS), 06.-10. März 2023, Dresden, Germany, Proceedings*, volume P-331 of *LNI*, pages 965–980. Gesellschaft für Informatik e.V., 2023. doi:10.18420/BTW2023-66.
  - 42 Mallory C Kidwell, Ljiljana Lazarevic, Erica Baranski, Tom E. Hardwicke, Sarah Piechowski, Lina-Sophia Falkenberg, Curtis Kennett, Agnieszka Slowik, Carina Sonnleitner, Chelsey Hess-Holden, Timothy M Errington, Susann Fiedler, and Brian A Nosek. Badges to Acknowledge Open Practices: A Simple, Low-Cost, Effective Method for Increasing Transparency. *PLOS Biology*, 14(5):e1002456, 2016. arXiv:27171007.
  - 43 Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E. Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B. Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter Development Team. Jupyter notebooks - a publishing format for reproducible computational workflows. In Fernando Loizides and Birgit Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas, 20th International Conference on Electronic Publishing, Göttingen, Germany, June 7-9, 2016*, pages 87–90. IOS Press, 2016. doi:10.3233/978-1-61499-649-1-87.
  - 44 Anna-Lena Lamprecht, Leyla J. García, Mateusz Kuzak, Carlos Martínez-Ortiz, Ricardo Arcila, Eva Martin Del Pico, Victoria Dominguez Del Angel, Stephanie van de Sandt, Jon C. Ison, Paula Andrea Martínez, Peter McQuilton, Alfonso Valencia, Jennifer L. Harrow, Fotis E. Psomopoulos, Josep Lluis Gelpí, Neil P. Chue Hong, Carole A. Goble, and Salvador Capella-Gutiérrez. Towards FAIR principles for research software. *Data Sci.*, 3(1):37–59, 2020. doi:10.3233/ds-190026.
  - 45 Timothy Lebo, Satya Sahoo, Deborah McGuinness, Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. Prov-o: The prov ontology. *W3C recommendation*, 30, 2013.
  - 46 Ekaterina Levitskaya, Gizem Korkmaz, Daniel Mietchen, and Lane Rasberry. Analysis of linked github and wikidata, December 2022. doi:10.5281/zenodo.7443339.
  - 47 Mario Lins, René Mayrhofer, Michael Roland, Daniel Hofer, and Martin Schwaighofer. On the critical path to implant backdoors and the effectiveness of potential mitigation techniques: Early learnings from xz, 2024. doi:10.48550/arXiv.2404.08987.
  - 48 Chang Liu, Matthew Kim, Michael Rueschman, and Satya S. Sahoo. *ProvCaRe: A Large-Scale Semantic Provenance Resource for Scientific Reproducibility*, pages 59–73. Springer International Publishing, Cham, 2021. doi:10.1007/978-3-030-67681-0\_5.
  - 49 Tomasz Miksa, Stephanie Renee Simms, Daniel Mietchen, and Sarah Jones. Ten principles for machine-actionable data management plans. *PLoS Comput. Biol.*, 15(3):e1006750, 2019. doi:10.1371/journal.pcbi.1006750.
  - 50 Andrew Nesbitt, Boris Veytsman, Daniel Mietchen, Eva Maxfield Brown, James Howison, João Felipe Pimentel, Laurent Hébert-Dufresne, and Stephan Druskat. Biomedical open source software: Crucial packages and hidden heroes. *CoRR*, abs/2404.06672, 2024. doi:10.48550/arXiv.2404.06672.
  - 51 Finn Årup Nielsen, Daniel Mietchen, and Egon L. Willighagen. Scholia, scientometrics and wikidata. In Eva Blomqvist, Katja Hose, Heiko Paulheim, Agnieszka Lawrynowicz, Fabio Ciravegna, and Olaf Hartig, editors, *The Semantic Web: ESWC 2017 Satellite Events - ESWC 2017 Satellite Events, Portorož, Slovenia, May 28 - June 1, 2017, Revised Selected Papers*, volume 10577 of *Lecture Notes in Computer Science*, pages 237–259. Springer, 2017. doi:10.1007/978-3-319-70407-4\_36.
  - 52 Daniel Nüst, Vanessa V. Sochat, Ben Marwick, Stephen J. Eglén, Tim Head, Tony Hirst, and Benjamin D Evans. Ten simple rules for writing

- Dockerfiles for reproducible data science. *PLoS Computational Biology*, 16(11):e1008316, 2020. doi:10.1371/journal.pcbi.1008316.
- 53 Jeff Z Pan. Resource description framework. In *Handbook on ontologies*, pages 71–90. Springer, 2009. doi:10.1007/978-3-540-92673-3\_3.
  - 54 Silvio Peroni and David Shotton. Fabio and cito: Ontologies for describing bibliographic resources and citations. *Journal of Web Semantics*, 17:33–43, 2012. doi:10.1016/j.websem.2012.08.001.
  - 55 João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. A large-scale study about quality and reproducibility of jupyter notebooks. In *Proceedings of the 16th International Conference on Mining Software Repositories, MSR '19*, pages 507–517, Piscataway, NJ, USA, 2019. IEEE Press. doi:10.1109/MSR.2019.00077.
  - 56 João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. Understanding and improving the quality and reproducibility of jupyter notebooks. *Empir. Softw. Eng.*, 26(4):65, 2021. doi:10.1007/s10664-021-09961-9.
  - 57 Project Jupyter. nbtime: Jupyter notebook diff and merge tools. <https://github.com/jupyter/nbtime>, 2021. Accessed 22 November 2024.
  - 58 Sarah Pugachev. What are “the carpentries” and what are they doing in the library? *portal: Libraries and the Academy*, 19(2):209–214, 2019.
  - 59 Richard J. Roberts. Pubmed central: The genbank of the published literature. *Proceedings of the National Academy of Sciences*, 98(2):381–382, 2001.
  - 60 A Rule, A Birmingham, C Zuniga, I Altintas, SC Huang, R Knight, N Moshiri, MH Nguyen, SB Rosenthal, F Pérez, et al. Ten simple rules for writing and sharing computational analyses in jupyter notebooks. *Plos Computational Biology*, 15(7):e1007007–e1007007, 2019. doi:10.1371/journal.pcbi.1007007.
  - 61 Adam Rule, Aurélien Tabard, and James D. Hollan. Exploration and explanation in computational notebooks. In Regan L. Mandryk, Mark Hancock, Mark Perry, and Anna L. Cox, editors, *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*, CHI '18, page 32, New York, NY, USA, 2018. ACM. doi:10.1145/3173574.3173606.
  - 62 Satya S Sahoo, Joshua Valdez, Michael Rueschman, and Matthew Kim. Semantic provenance graph for reproducibility of biomedical research studies: Generating and analyzing graph structures from published literature. In *MEDINFO 2019: Health and Wellbeing e-Networks for All*, pages 328–332. IOS Press, 2019. doi:10.3233/SHTI190237.
  - 63 Angelo A Salatino, Thiviyan Thanapalasingam, Andrea Mannocci, Francesco Osborne, and Enrico Motta. The computer science ontology: a large-scale taxonomy of research areas. In *The Semantic Web—ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part II 17*, pages 187–205. Springer, 2018. doi:10.1007/978-3-030-00668-6\_12.
  - 64 Sheeba Samuel and Birgitta König-Ries. Combining P-Plan and the REPRODUCE-ME ontology to achieve semantic enrichment of scientific experiments using interactive notebooks. In *The Semantic Web: ESWC 2018 Satellite Events: Heraklion, Crete, Greece, June 3-7, 2018*, pages 126–130. Springer, 2018. doi:10.1007/978-3-319-98192-5\_24.
  - 65 Sheeba Samuel and Birgitta König-Ries. Provenance-based semantic enrichment of interactive notebooks for reproducibility. In *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th - to - 12th, 2018*, volume 2180 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018. URL: <https://ceur-ws.org/Vol-2180/paper-57.pdf>.
  - 66 Sheeba Samuel and Birgitta König-Ries. ReproduceMeGit: A visualization tool for analyzing reproducibility of jupyter notebooks. In *Provenance and Annotation of Data and Processes*, pages 201–206, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-80960-7\_12.
  - 67 Sheeba Samuel and Birgitta König-Ries. End-to-end provenance representation for the understandability and reproducibility of scientific experiments using a semantic approach. *Journal of biomedical semantics*, 13(1):1, 2022. doi:10.1186/s13326-021-00253-1.
  - 68 Sheeba Samuel and Daniel Mietchen. FAIR Jupyter. Service, DFG 514664767, DFG 460135501, DFG 521453681 (visited on 2024-11-29). URL: <https://w3id.org/fairjupyter>, doi:10.4230/artifacts.22527.
  - 69 Sheeba Samuel and Daniel Mietchen. FAIR Jupyter Knowledge Graph: v1.0. Software, version 1.0., DFG 514664767, DFG 460135501, DFG 521453681 (visited on 2024-11-29). doi:10.5281/zenodo.14197755.
  - 70 Sheeba Samuel and Daniel Mietchen. Dataset of a Study of Computational reproducibility of Jupyter notebooks from biomedical publications. <https://doi.org/10.5281/zenodo.8226725>, 2023.
  - 71 Sheeba Samuel and Daniel Mietchen. Computational reproducibility of Jupyter notebooks from biomedical publications. *GigaScience*, 13:giad113, 2024.
  - 72 Sheeba Samuel and Daniel Mietchen. FAIR Jupyter Knowledge Graph, September 2024. doi:10.5281/zenodo.13845701.
  - 73 Sheeba Samuel and Daniel Mietchen. FAIR Jupyter Knowledge Graph: SPARQL Queries and Performance Evaluation and Benchmark, September 2024. doi:10.5281/zenodo.13847627.
  - 74 Max Schröder, Frank Krüger, and Sascha Spors. Reproducible research is more than publishing research artefacts: A systematic analysis of jupyter notebooks from research articles. *CoRR*, abs/1905.00092, 2019. arXiv:1905.00092, doi:10.48550/arXiv.1905.00092.
  - 75 Zachary S. Siegel, Sayash Kapoor, Nitya Nagdir, Benedikt Stroebel, and Arvind Narayanan. Core-bench: Fostering the credibility of published

- research through a computational reproducibility agent benchmark, 2024. doi:10.48550/arXiv.2409.11363.
- 76 Mari Carmen Suárez-Figueroa, Asunción Gómez-Pérez, and Mariano Fernández-López. The neon methodology for ontology engineering. In *Ontology engineering in a networked world*, pages 9–34. Springer, 2011. doi:10.1007/978-3-642-24794-1\_2.
- 77 Ana Trisovic, Matthew K Lau, Thomas Pasquier, and Mercè Crosas. A large-scale study on research code quality and execution. *Scientific Data*, 9(1):60, 2022.
- 78 Denny Vrandečić, Lydia Pintscher, and Markus Krötzsch. Wikidata: The making of. In Ying Ding, Jie Tang, Juan F. Sequeda, Lora Aroyo, Carlos Castillo, and Geert-Jan Houben, editors, *Companion Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, pages 615–624. ACM, 2023. doi:10.1145/3543873.3585579.
- 79 Jiawei Wang, Tzu-yang Kuo, Li Li, and Andreas Zeller. Restoring reproducibility of jupyter notebooks. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 288–289, 2020. doi:10.1145/3377812.3390803.
- 80 Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9, 2016.
- 81 Alistair Willis, Patricia Charlton, and Tony Hirst. Developing students’ written communication skills with jupyter notebooks. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 1089–1095, 2020. doi:10.1145/3328778.3366927.
- 82 Morgan F. Wofford, Bernadette M. Boscoe, Christine L. Borgman, Irene V. Pasquetto, and Milena S. Golshan. Jupyter notebooks as discovery mechanisms for open science: Citation practices in the astronomy community. *Computing in Science & Engineering*, 22(1):5–15, 2020. doi:10.1109/MCSE.2019.2932067.
- 83 Jian Xu, Sunkyu Kim, Min Song, Minbyul Jeong, Donghyeon Kim, Jaewoo Kang, Justin F Rousseau, Xin Li, Weijia Xu, Vetle I Torvik, et al. Building a pubmed knowledge graph. *Scientific data*, 7(1):205, 2020.

■ **Table 5** The FAIR Jupyter Ontology Requirements Specification Document.

The FAIR Jupyter Ontology Requirements Specification Document	
<b>1. Purpose</b>	
The purpose of this ontology is to provide a knowledge model for analyzing, categorizing, and understanding the reproducibility, dependencies, errors, and metadata of Jupyter notebooks cited in academic research.	
<b>2. Scope</b>	
The ontology has to focus on the computational and non-computational processes of an experiment and the data used and generated in an experiment. The level of granularity is directly related to the competency questions and terms identified	
<b>3. Implementation Language</b>	
The ontology will be implemented in the Web Ontology Language (OWL).	
<b>4. Intended End-Users</b>	
User 1. Researchers and academics conducting reproducibility studies. User 2. Data scientists and developers analyzing Jupyter notebooks. User 3. Journals and publishers interested in ensuring the reproducibility of published research. User 4. Educational institutions integrating Jupyter notebooks into curricula. User 5. Software tools and platforms providing support for notebook analysis and management.	
<b>5. Intended Uses</b>	
Use 1. Facilitating reproducibility studies by providing a standardized framework for analyzing notebook execution and outcomes. Use 2. Identifying common errors, dependencies, and issues in Jupyter notebooks to improve their reliability. Use 3. Enabling the integration of best practices and coding standards compliance checks in Jupyter notebooks. Use 4. Providing a basis for educational and training materials on best practices for using Jupyter notebooks in research.	
<b>6. Ontology Requirements</b>	
<b>a. Non-Functional Requirements</b>	
NFR 1. The ontology must be published on the Web with an open license. NFR 2. The ontology must reuse other ontologies if required.	
<b>b. Functional Requirements: Groups of Competency Questions</b>	
CQG1. Journals	CQG2. Research Fields
CQ1. What is the timeline of journals (by year) that publish their first and tenth article mentioning a GitHub repository with a Jupyter/iPython notebook? CQ2. Which journals have significantly higher or lower reproducibility rates?	CQ3. What are the top-level MeSH terms as a proxy for their research field ranked by the number of GitHub repositories, and which of these repositories contain at least one Jupyter notebook? CQ4. How does the error rate in notebooks differ by research field?
CQG3. Articles	CQG4. Repositories
CQ5. Are notebooks associated with more recent articles more or less likely to have certain errors? CQ6. How does the error rate in notebooks differ between the top and bottom 10th percentiles of articles? CQ7. How many repositories are mentioned in the articles? CQ8. What is the timeline of articles (by year) that mention a GitHub repository with a Jupyter/iPython notebook? CQ9. What is the timeline of subjects (by year) that publish their first and tenth article mentioning a GitHub repository with a Jupyter/iPython notebook?	CQ10. What percentage of repositories mentioned in articles are still available at the indicated GitHub address? CQ11. What percentage of repositories have official releases? CQ12. What is the number of collaborators, forks, and stars for each repository? CQ13. How many commits have been made to the repository after the publication of the article? CQ14. Which repositories contain declared dependencies (e.g., requirements.txt, setup.py, or Pipfile)? CQ15. What is the percentage of Jupyter code compared to the overall code in the repository? CQ16. How many repositories do not contain any notebooks?
CQG5. Notebooks	CQG6. Programming Languages
CQ17. How many notebooks are PEP8 compliant? CQ18. What are the most popular modules/libraries used in the notebooks? CQ19. What are the most popular APIs used in the notebooks? CQ20. How many notebooks are in repositories that contain a requirements.txt, setup.py, or Pipfile? CQ21. How many notebooks are shared in the actual order of execution of their cells? CQ22. How many cells are code cells? CQ23. How many cells are Markdown cells? CQ24. Which notebooks have the highest ratio of Markdown cells to code cells?	CQ25. Are there groupwise differences between R and Python notebooks/repositories in terms of errors and reproducibility? CQ26. What “nbformat” versions are used in the notebooks? CQ27. How many notebooks use “nbformat” version 3 or lower? CQ28. What are the top 15 programming languages used in the notebooks? CQ29. How many Python notebooks are valid or invalid? CQ30. How many notebooks have an undeclared programming language? CQ31. What versions of Python are used in the notebooks?

■ **Table 6** The FAIR Jupyter Ontology Requirements Specification Document.

CQG7. Notebook Reproducibility		CQG8. General	
CQ32. What are the common data dependencies in the notebooks? CQ33. What types of file errors occur in the notebooks? CQ34. How often do notebooks reference local files? CQ35. What are the best practices for file naming in the notebooks? CQ36. How often are URLs used in the notebooks, and what issues arise from their usage? CQ37. How frequently are persistent identifiers used in the notebooks? CQ38. What types of HTTP errors are encountered in the notebooks? CQ39. What are the common causes of failed dependencies in the notebooks? CQ40. What are the common reasons for installation failures in the notebooks? CQ41. How often do notebooks fail due to missing files? CQ42. How many notebooks finish execution successfully? CQ43. How many notebooks produce different results on re-execution? CQ44. How many notebooks produce the same results on re-execution?		CQ45. Is the usage of other best practices (e.g., some level of CI in the repo, PEP8 compliance in the notebooks, usage of DOIs for article, data, and code, ORCID for authors, or having a data/code availability statement) predictive of reproducibility success? CQ46. How does the presence of best practices correlate with reproducibility success? CQ47. What percentage of authors on a article have an ORCID? CQ48. What is the environmental footprint of reproducing a Jupyter notebook? CQ49. What is the distribution by country of author affiliations for the notebooks?	
7. Pre-Glossary of Terms			
a. Terms from Competency Questions + Frequency			
Notebook 37 Article 10 Code 7 Fail 3 Journal 2	Repository 15 GitHub 5 Python 6 File 6 Module 1	Dependencies 3 Cell 6 Execution 5 Reproducibility 4 Author 3	Error 6 Programming language 2 Research field 2 Markdown 2 Release 1
b. Terms from Answers + Frequency			
Notebook 37 Article 10 Code 7 Fail 3 Journal 2	Repository 15 GitHub 5 Python 6 File 6 Module 1	Dependencies 3 Cell 7 Execution 5 Reproducibility 4 Author 3	Error 6 Programming language 2 Research field 2 Markdown 2 Release 1
c. Objects			
No objects were identified.			



# The Reasonable Ontology Templates Framework

Martin Georg Skjæveland  

Department of Informatics, University of Oslo, Norway

Leif Harald Karlsen  

Department of Informatics, University of Oslo, Norway

---

## Abstract

Reasonable Ontology Templates (OTTR) is a templating language for representing and instantiating patterns. It is based on simple and generic, but powerful, mechanisms such as recursive macro expansion, term substitution and type systems, and is designed particularly for building and maintaining RDF knowledge graphs and OWL ontologies.

In this resource paper, we present the formal specifications that define the OTTR framework. This includes the fundamentals of the OTTR language and the adaptations to make it fit with standard semantic web languages, and two serialization formats developed for semantic web practitioners.

We also present the OTTR framework's support for documenting, publishing and managing template libraries, and for tools for practical bulk instantiation of templates from tabular data and queryable data sources. The functionality of the OTTR framework is available for use through Lutra, an open-source reference implementation, and other independent implementations. We report on the use and impact of OTTR by presenting selected industrial use cases. Finally, we reflect on some design considerations of the language and framework and present ideas for future work.

**2012 ACM Subject Classification** Computing methodologies → Ontology engineering; Information systems → Data management systems; Computing methodologies → Modeling methodologies

**Keywords and phrases** Ontology engineering, Ontology design patterns, Template mechanism, Macros

**Digital Object Identifier** 10.4230/TGDK.2.2.5

**Category** Resource Paper

**Supplementary Material** The homepage for the project is `ottr.xyz`. Versioned releases of Lutra and OTTR specifications are published at Zenodo. The source code for Lutra is published on GitLab under the GNU Lesser General Public License v2.1 license. Maven artefacts for Lutra are published at Sonatype.

*InteractiveResource (Website):* <https://ottr.xyz/>

*Software:* <https://gitlab.com/ottr>

*Software:* <https://zenodo.org/communities/ottr/>

*Software (Maven artefacts):* <https://central.sonatype.com/namespace/xyz.ottr.lutra>

**Funding** We acknowledge financial support from The Research Council of Norway through SIRIUS, Centre for Scalable Data Access (237898).

*Martin Georg Skjæveland:* The author acknowledges funding from the EU projects RE4DY (101058384) and Tec4MaaSEs (101138517).

**Acknowledgements** We wish to thank all past members of the OTTR project: Chris Kindermann, Daniel Lupp, Evgenij Thorstensen, Henrik Forssell, Laura Slaughter, Oliver Stahl; its associated master students: Erik Snilsberg, Lars Ivar Bull Larssen, Magnus Wiik Eckhoff, Marlen Jarholt, Preben Zahl, Shanshan Qu; part-time programmers: Fariha Hossain, Humza Ahmad, Vinicius Gracioli, Yiyao Chen; and SHS for their contributions to the OTTR framework. We are also grateful to the users of the OTTR framework for all their feedback and support.

**Received** 2024-07-01 **Accepted** 2024-11-07 **Published** 2024-12-18

**Editors** Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler

**Special Issue** Resources for Graph Data and Knowledge



© Martin Georg Skjæveland and Leif Harald Karlsen; licensed under Creative Commons License CC-BY 4.0

*Transactions on Graph Data and Knowledge*, Vol. 2, Issue 2, Article No. 5, pp. 5:1–5:54



Transactions on Graph Data and Knowledge

TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

Abstraction is a fundamental concept in computer science, particularly in software engineering and information modelling. In these disciplines, abstraction entails identifying and describing the relevant entities and structures for the problem at hand at a suitable level of detail. Done correctly, abstraction helps to hide unnecessary detail and presents the essence of the content that is described to the effect that the content is clearly conveyed and easily understood, and can hence be more efficiently processed by operations acting on the representation.

Figure 1 displays a simplified comparison of different abstraction levels with two code snippets that both write “Hello world!” to screen. The first snippet is written in x86 Linux assembly language<sup>1</sup> and the second is written in the high-level programming language Python. There is a striking difference between the two snippets: the Python code is succinct, easy to read and write and understand, while the assembly code is far more verbose as it must orchestrate a series of low-level resources and steps, such as memory locations and sizes, file descriptors and interrupt handlers, in order to solve the task at hand.<sup>2</sup> For most users, the high level of abstraction provided by the Python snippet is appropriate when all one wants is to write messages to screen – all other details are hidden. This code is safe and robust for this use case; its succinctness makes it difficult to use the code in the wrong way. The assembly code is arguably incomprehensible for most users and appears inefficient to use and manage if the task is just to write messages to screen. However, for some expert users or use cases the level of detail and control offered by assembly languages to interact more directly with hardware is exactly what is needed.

Most modern programming languages, like Python, offer mechanisms for different kinds of user-defined abstractions, such as functions, classes, interfaces, and modules, and it is common to package and distribute a set of such abstractions in an application programming interface (API). A well-designed API offers a suitable abstraction level using terminology that is familiar and natural for its intended users and hides the details of underlying lower-level APIs or systems. Understanding, managing and designing APIs is a central part of modern software engineering.

The Resource Description Framework (RDF) [27] and the Web Ontology Language (OWL) [40] are the standard languages for representing knowledge graphs and ontologies. A challenge for the wider adoption of semantic web languages, and ontology languages in particular, is its inherent complexity, a steep learning curve and the lack of developer-, and end user-friendly ways to interact with their artefacts. Thus, interfaces and simplifications for eliciting the content of ontologies are identified as opportunities for future research [56]. In this regard, it is worth noting that these languages offer very limited options for user-defined abstractions and provide no means to represent modelling patterns or templates that can be instantiated in a precise and deterministic manner, and that can hide details that appear unnecessary and complex to users.

As a case in point, consider the Protégé Pizza Ontology Tutorial<sup>3</sup> which models the domain of pizzas for the purpose of demonstrating and teaching features of OWL and Protégé [37]. This OWL ontology contains 22 types of pizza that are modelled following the same pattern; the description logic axioms that represent the Margherita pizza are listed in Figure 2 (the numbers that follow in parentheses refer to axioms in the figure): every pizza is represented as a subclass of `NamedPizza` (1), some pizzas have a country of origin (2), and toppings are expressed by stating that they are both required (3, 4) and permissible (5) for the pizza. Figure 2 also contains two different standard serialization formats for OWL: Manchester syntax [17] and RDF Turtle [41, 2].

<sup>1</sup> The code snippet is taken from <https://gist.github.com/pablocorbalann/f9d39a80e30b8d8230a9760048d0e575>.

<sup>2</sup> The interested reader can find more information about the assembly code in the following article: <https://pablocorbalann.medium.com/>.

<sup>3</sup> <http://protege.stanford.edu/ontologies/pizza/pizza.owl>

x86 Linux assembly language:

```

section      .text                ; declare the .text section
global      _start                ; has to be declared for the linker (ld)
_start:
    mov     edx, len                ; "invoke" the len of the message
    mov     ecx, msg                ; "invoke" the message itself

    mov     ebx, 1                  ; set the file descriptor (fd) to stdout

    mov     eax, 4                  ; system call for "write"
    int     0x80                   ; call the kernel

    mov     eax, 1                  ; system call for "exit"
    int     0x80                   ; call the kernel

section      .data                ; here you declare the data
msg          db "Hello world!"     ; the actual message to use
len          equ $ -msg            ; get the size of the message

```

Python:

```
print("Hello world!")
```

■ **Figure 1** “Hello world!” printed to screen in x86 Linux assembly language, and in Python.

The Manchester syntax lies close to the description logics representation, while the RDF Turtle serialization is more verbose as all statements are on the form of triples, and must use blank nodes and resources such as `owl:Restriction` in order to represent the same information.

The case demonstrates two points: The first point is that RDF and OWL, the standard knowledge representation languages for the web, appear as expert languages that operate on a too low level of abstraction for the task of representing ordinary compound modelling patterns, such as pizzas, in a succinct and readable manner. The representations arguably expose too many details in the form of logical constructs and language peculiarities in order to be easy to read and understand for non-experts. The second point is that the lack of abstraction mechanisms for RDF and OWL forces all statements to be on the form of RDF triples and OWL axioms and limited to the constructs defined by these standards, such as `some/owl:someValuesFrom`. This makes the representations repetitive and verbose. In the code samples this is shown with the repetition of the existentially quantified axiom and the fact that, e.g., a “macro” symbol [58] that allows to express the required and permissible pizza toppings in a single statement is not possible to declare. For the full Pizza Ontology, repetition is also visible with the 22 pizzas using the same pizza modelling patterns by replicating all the axiom schemata. As a result, the representation of only the pizzas, according to the pattern in Figure 2, comprises in total 198 OWL axioms and 1106 RDF triples.

The overall effect is that the current standard representation formats for knowledge graphs and ontology will often appear too far removed from most users’ understanding and conceptualization of the domain, and is therefore difficult to understand and use. Also, the fact that there is no explicit representation of a pattern and its instances makes it difficult to identify that any pattern is followed, which again makes it difficult to ensure consistent modelling. Furthermore, it complicates consistent and efficient updates of the pattern instances as they are spread across multiple sets of OWL axioms or RDF triples. The lack of established representation for consistently reusable

## 5:4 The Reasonable Ontology Templates Framework

Description Logic:

- Margherita  $\sqsubseteq$  NamedPizza (1)
- Margherita  $\sqsubseteq \exists$  hasCountryOfOrigin.{Italy} (2)
- Margherita  $\sqsubseteq \exists$  hasTopping.Mozzarella (3)
- Margherita  $\sqsubseteq \exists$  hasTopping.Tomato (4)
- Margherita  $\sqsubseteq \forall$  hasTopping.(Mozzarella  $\sqcup$  Tomato) (5)

Manchester OWL:

```
Class: Margherita
SubClassOf:
  NamedPizza,
  hasCountryOfOrigin some { Italy },
  hasTopping some Mozzarella,
  hasTopping some Tomato,
  hasTopping only (Mozzarella or Tomato)
```

RDF Turtle:

```
ex:Margherita
  rdfs:subClassOf p:NamedPizza ,
  [ a owl:Restriction ;
    owl:onProperty p:hasCountryOfOrigin ;
    owl:hasValue ex:Italy ] ,
  [ a owl:Restriction ;
    owl:onProperty p:hasTopping ;
    owl:allValuesFrom [ a owl:Class ;
                        owl:unionOf ( ex:Mozzarella ex:Tomato ) ] ] ,
  [ a owl:Restriction ;
    owl:onProperty p:hasTopping ;
    owl:someValuesFrom ex:Tomato ] ,
  [ a owl:Restriction ;
    owl:onProperty p:hasTopping ;
    owl:someValuesFrom ex:Mozzarella ] .
```

■ **Figure 2** Margherita pizza represented as description logic axioms, in OWL Manchester syntax, and in RDF Turtle.

modelling patterns is also evident in today's documentation of vocabularies and ontologies and ontology design patterns [11, 4]. Here, current practice is usually limited to at most textual descriptions, illustrative and informal diagrams, and samples of OWL files that describe and illustrate how to use the resource. These offer little tangible practical help in building knowledge graphs and ontologies at scale. Following best practice descriptions requires considerable manual effort and the result is prone to errors due to the tolerant nature of RDF and RDFS vocabularies unless some constraint language like SHACL [28] is used.

The *Reasonable Ontology Templates (OTTR)* framework [50, 51, 52] is created to fill these gaps. OTTR is a macro-like [58] templating mechanism with which modelling patterns can be represented and instantiated by nested and parameterized templates. Using the OTTR framework, the pizza pattern used in Figure 2 can be represented by an OTTR Template `o-p:NamedPizza` (presented in detail in Section 2), and instances of the template can be used to express replicas of

```
o-p:NamedPizza(ex:Margherita, ex:Italy, (ex:Mozzarella, ex:Tomato)) .
```

■ **Figure 3** Margherita pizza represented as an OTTR template instance.

the pattern; the Margherita pizza in Figure 2 can be represented succinctly and precisely with the OTTR template instance found in Figure 3 that specifies the arguments to parameterized template. Templates can be documented and shared as template libraries targeted for different users at different abstraction levels, and be efficiently instantiated using the OTTR framework's bulk instantiation tools.

Introducing the use of succinctly represented patterns and pattern instances to knowledge graph engineering allows interaction with RDF and OWL knowledge bases at a higher level of abstraction than that of RDF triples and OWL axioms. This brings with it many favourable properties such as adherence to the do-not-repeat-yourself (DRY) principle, encapsulation of complexity, separation of concerns, and better support for different user groups. Templates are also useful for documenting typical modelling use cases, such as vocabulary uses and ontology design patterns. Representing modelling patterns as identifiable templates, allows them to be shared online in a precise and actionable manner, and leads arguably to more modelling uniformity and increased efficiency and quality of knowledge base modelling tasks.

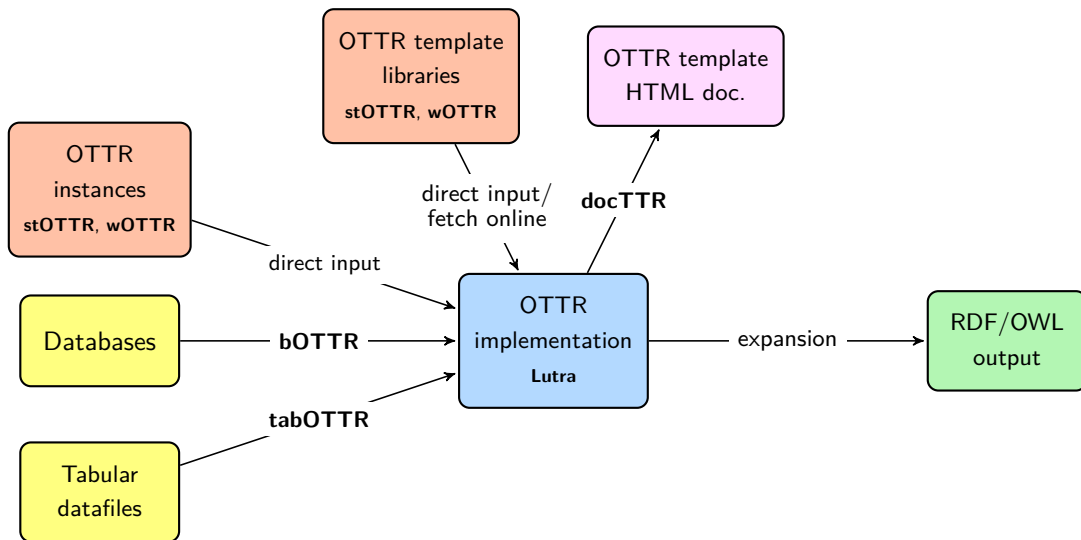
While OTTR at its core is a generic templating language, it is one of few practical pattern-based frameworks that is specifically designed for the construction of knowledge graphs and ontologies to be serialized in RDF, and with demonstrated use in the construction of large-scale ontologies and knowledge graphs [50, 49, 6, 55]. As such, the OTTR framework is an advance of the state of the art of ontology engineering [23, 56] and ontology design pattern [11, 4] tools and methodologies.

The OTTR framework has been presented in a series of papers [50, 51, 52, 34]. These papers have presented and characterized the OTTR language at a conceptual level and demonstrated different uses of the framework. The OTTR framework has since then gradually matured to a stable state with multiple different independent implementations and applications by prominent ontology development projects. The purpose of this resource paper is to give a complete and self-contained presentation of the resources that now comprise the OTTR framework: specifications, core template library, reference implementation, and project infrastructure. The paper gives emphasis to the specifications of the formal syntax and semantics of the OTTR language and its implementation for semantic web, which is given in Section 3 and Section 4, and the formal specification of the mapping languages for instantiating templates, which is presented in Section 6. These introduce an abstract and formal model and vocabulary for characterizing the OTTR language that form the basis of the reference implementation. These specifications have not been published before in this rigorous form and are necessary to fully understand the OTTR framework. Section 5 gives an overview of the motivation and support for developing and maintaining template libraries. We also give an updated overview of the OTTR framework's impact, including publicly available template libraries, implementations of the OTTR framework in Section 7, and a selection of industrial and academic uses in Section 8. Section 9 presents related work and Section 10 presents lessons learned and ideas for future development of the OTTR framework collected throughout the project from experience and interaction with its users. Section 11 concludes the paper. First, Section 2 presents an overview of the OTTR framework to tie all the resources together and gives examples to establish intuitions for the following more technical sections.

## 2 Overview

The OTTR framework is formally described by a series of specifications that define:

## 5:6 The Reasonable Ontology Templates Framework



■ **Figure 4** High-level OTTR framework architecture.

- an abstract language for characterizing templates and template instances and the process of expanding template instances,
- serialization formats for representing templates and instances (stOTTR and wOTTR),
- a mapping language for consuming data from queryable databases as template instances (bOTTR), and
- a mapping language for annotating and consuming data from tabular datafiles as template instances (tabOTTR).

Additionally, the framework consists of:

- a template library of basic templates called the core template library that mostly contains templates that represent basic modelling patterns over the vocabularies RDF, RDFS and OWL,
- a tool-supported best practice description of how to document and publish template libraries (docTTR), and
- a reference implementation that supports all the specifications of the framework (Lutra).

Figure 4 shows a high-level architecture diagram of the OTTR framework.

The primary uses of the OTTR framework are to represent and document useful modelling patterns in a precise and actionable manner as (shared) OTTR template libraries, and to use such libraries to expand OTTR template instances to RDF data. The consumed instances can be described either directly using one of the OTTR serialization formats or by way of mappings that extract or identify instances in tabular data sources such as database query results or tabular datafiles. The OTTR template language has different features to guarantee the correctness of the output, and verifying the input according to these correctness measures is a core feature of the framework. These features also help to reveal the intended and correct instantiations of templates and play an important role in the documentation of templates. The following sections present an overview of the OTTR language, the concept behind template libraries and bulk instantiation of templates.

ottr:Triple base template:

```

1 ottr:Triple [
2   ottr:IRI ?subject, ! ottr:IRI ?predicate, rdfs:Resource ?object ] ::
3   BASE .

```

o-owl-ax:SubClassOf template:

```

1 o-owl-ax:SubClassOf [
2   owl:Class ?subclass, owl:Class ?superclass ] ::
3   {
4     ottr:Triple(?subclass, rdfs:subClassOf, ?superclass)
5   } .

```

o-p:NamedPizza template:

```

1 o-p:NamedPizza [
2   owl:Class ?pizza, ? owl:NamedIndividual ?country, NElList<owl:Class> ?toppings ] ::
3   {
4     o-owl-ax:SubClassOf(?pizza, pz:NamedPizza),
5     o-owl-ax:SubObjectHasValue(?pizza, pz:hasCountryOfOrigin, ?country),
6     cross | o-owl-ax:SubObjectSomeValuesFrom(?pizza, pz:hasTopping, ++?toppings),
7     o-owl-ax:SubObjectAllValuesFrom(?pizza, pz:hasTopping, _:toppingUnion),
8     o-owl-re:ObjectUnionOf(_:toppingUnion, ?toppings)
9   } .

```

o-p:NamedPizza instances:

```

1 o-p:NamedPizza(ex:Margherita, ex:Italy, (ex:Mozzarella, ex:Tomato)) .
2
3 o-p:NamedPizza(ex:PlainHam, none, (ex:Mozzarella, ex:Tomato, ex:Ham)) .
4
5 o-p:NamedPizza(ex:Hawaiian, ex:Canada,
6   (ex:Mozzarella, ex:Tomato, ex:Pineapple, ex:Ham)) .

```

■ **Figure 5** OTTR templates and instances representing different pizzas.

## 2.1 Language

The OTTR language and its features will be introduced in an incremental and example-driven approach that builds on the example established in the introduction. The complete specification of the OTTR language is found in Section 3.

### 2.1.1 Templates, base templates and instances

A *template* has a *signature* that assigns an IRI to the template and lists its *parameters* that specify its permissive *instances*. An *instance* refers to a template's IRI and lists *arguments* that must match the parameters of the referenced template. The template *body* contains instances of other templates and specifies hence how its instances can be *expanded* into instances of templates at a lower level of abstraction; this hierarchy of templates is required to be non-cyclic. At the lowest level of abstraction in the hierarchy of templates are *base templates* that specify how instances should be interpreted into a different representation language, such as RDF. Base templates do not have a body; the translation of base template instances to the underlying representation language is handled by an OTTR implementation that must follow a textual specification of how base templates must be interpreted.

## 5:8 The Reasonable Ontology Templates Framework

`o-owl-ax:SubClassOf` and `ottr:Triple` instances:

```
1 o-owl-ax:SubClassOf(ex:A, ex:B) .  
2  
3 ottr:Triple(ex:A, rdfs:subClassOf, ex:B) .
```

Expansion result:

```
< ex:A, rdfs:subClassOf, ex:B >
```

■ **Figure 6** `o-owl-ax:SubClassOf` and `ottr:Triple` instances, and their expansion result.

► **Example 1.** Figure 5 contains three templates, the base template `ottr:Triple`, and the (regular) templates `o-owl-ax:SubClassOf` and `o-p:NamedPizza`; and instances of the `o-p:NamedPizza` template. All examples in this section are serialized using the stOTTR format. The example templates are formatted so that their signatures are contained in the two first lines of each of the code listings. The remaining lines contain the template body. Instead of a body, the `ottr:Triple` base template is marked with the token `BASE`.

Template instances are *expanded* by recursively replacing an instance with its referenced template's body's instances where the parameters are appropriately substituted by the instance's arguments, akin to unfolding macros. This process terminates with a set of base template instances that can be translated to the underlying representation language as per the specification. A template can hence be understood to represent a mapping from its signature instance format to a set of statements over an underlying language represented by base templates, via a nested non-cyclic template structure.

► **Example 2.** The signature of the `ottr:Triple` template in Figure 5 specifies three parameters: `?subject`, `?predicate` and `?object`. (The example also includes parameter types and modifiers which will be explained shortly.) The body of the `o-owl-ax:SubClassOf` template contains one instance of the `ottr:Triple` template where the parameters of the `o-owl-ax:SubClassOf` template are used as parameters. Figure 6 demonstrates the expansion of instances; the example instance of the `o-owl-ax:SubClassOf` instance in line 1 is expanded in one step to the `ottr:Triple` instance in line 3, which represents the RDF triple as shown in the figure.

► **Example 3.** The `o-p:NamedPizza` template in Figure 5 is a faithful representation of the pizza modelling pattern used in the Pizza Ontology. The body of the `o-p:NamedPizza` template contains instances of the `o-owl-ax:SubClassOf` template and other templates that represent common OWL axioms and constructs. The first template instance in Figure 5 expressing a Margherita pizza expands in multiple steps to an RDF graph that is equivalent to the RDF graph found in Figure 2 on page 4.

### 2.1.2 Parameter types and non-blank flags

*Parameter types* are used to check that templates are correctly instantiated and specified; the arguments' types must be *compatible* with the types of the parameters where the arguments are used, and this must also hold when parameters are used as arguments in template bodies. The OTTR language also contains *parameter modifiers*, where *non-blank* is one such parameter modifier that forbids RDF blank nodes as arguments. OTTR implementations must emit errors when instances and template violate these parameter type specifications.

► **Example 4.** The signature of the `ottr:Triple` template assigns types to its parameters; the `?subject` and `?predicate` parameters have the type `ottr:IRI`, and the `?object` has the type `rdfs:Resource`. These parameter types guarantee that no `ottr:Triple` instance can, for example,



have a literal in subject position, which would be a violation of the RDF specification [27], since the type assigned to literals is specified by the type system as incompatible with the parameter type `ottr:IRI`. The following `ottr:Triple` instance contains two type errors: the literal values "A" and "B" are arguments to parameters with the type `ottr:IRI`.

```
ottr:Triple("A", "B", "C") .
```

► **Example 5.** In the body of the `o-owl-ax:SubClassOf` template, the types of the parameters `?subclass` (`owl:Class`) and `?superclass` (also `owl:Class`) must be compatible with the types of the first (`ottr:IRI`) and third parameter (`rdfs:Resource`) of the `ottr:Triple` template, respectively – which they are. Furthermore, the parameter types of `o-owl-ax:SubClassOf` template force for example the parameter type of `o-p:NamedPizza`'s `?pizza` parameter to have a type that is compatible with `owl:Class`, since `?pizza` is passed on as an argument to a parameter with this type.

► **Example 6.** The `ottr:Triple` signature specifies, using an exclamation mark `!`, the `?predicate` parameter to be non-blank. This ensures that no RDF triple constructed using this template will end up with a blank node in predicate position, which would be a violation of the RDF specification [27]. The following `ottr:Triple` instance violates the non-blank modifier.

```
ottr:Triple(ex:A, _:blank, "C") .
```

### 2.1.3 Optional parameters and none values

Parameters may be specified as being *optional*, whereas parameters that are not optional are called mandatory. Whether a parameter is optional or not has consequences for the treatment of *none values*, which in OTTR is represented by the reserved token `none` and is used to indicate a missing value. In the expansion of instances, a none value given as an argument to a mandatory parameter is simply ignored and will not contribute to the end result of the expansion – the instance is simply removed. A none value given to an optional argument, on the other hand, will be passed on to body template instances just like other arguments.

► **Example 7.** The second argument of the `o-p:NamedPizza` template is marked as optional, using a question mark `?`. This means that instances of the template do not need to specify a country of origin. The `ex:PlainHam` example instance demonstrates this. Here, the none value will be passed on as an argument to the third parameter of the `o-owl-ax:SubObjectHasValue` template. This parameter is mandatory, hence there will be no OWL axiom in the expansion result that expresses the country of origin of the `ex:PlainHam` pizza, however, the other axioms will remain. If the `?country` parameter of the `o-p:NamedPizza` had not been marked as optional, then the `ex:PlainHam` instance would have been simply removed in the first expansion step.

### 2.1.4 Default values

Parameters may be given a *default value*. This default value is used whenever a none value is given as an argument to the parameter.

► **Example 8.** Figure 7 demonstrates the use of a default valued parameter using an alternative signature to the `o-p:NamedPizza` template that assigns `ex:Italy` as the default value to the second argument. The `ex:PlainHam` example instance in Figure 5 would under this signature get `ex:Italy` as its country of origin.

## 5:10 The Reasonable Ontology Templates Framework

```
1 o-p:NamedPizza[
2   owl:Class ?pizza,
3   owl:NamedIndividual ?country = ex:Italy,
4   NEList<owl:Class> ?toppings
5 ] .
```

■ **Figure 7** `o-p:NamedPizza` with default valued parameter.

### 2.1.5 Expansion modes and list values

Template instances can be marked with an *expansion mode* which is only applicable to instances that have arguments that are lists. An expansion mode applied to an instance with one list argument specifies that the selected instance will be instantiated multiple times, one per element in the marked argument list. There are different expansion modes that behave differently when multiple lists are marked in an instance.

► **Example 9.** The `o-p:NamedPizza` template makes use of expansion modes, indicated with the token `cross` and by marking the list-typed parameter `?toppings` with `++`:

```
6 cross | o-owl-ax:SubObjectSomeValuesFrom(?pizza, pz:hasTopping, ++?toppings),
```

The effects of the expansion mode are that one instance of the `o-owl-ax:SubObjectSomeValuesFrom` template will be created for each element in the `?toppings` list, e.g.,

```
cross | o-owl-ax:SubObjectSomeValuesFrom(ex:Margherita, pz:hasTopping,
++(ex:Mozzarella, ex:Tomato)) .
```

will expand in one step to:

```
o-owl-ax:SubObjectSomeValuesFrom(ex:Margherita, pz:hasTopping, ex:Mozzarella) .
o-owl-ax:SubObjectSomeValuesFrom(ex:Margherita, pz:hasTopping, ex:Tomato) .
```

## 2.2 Template Libraries

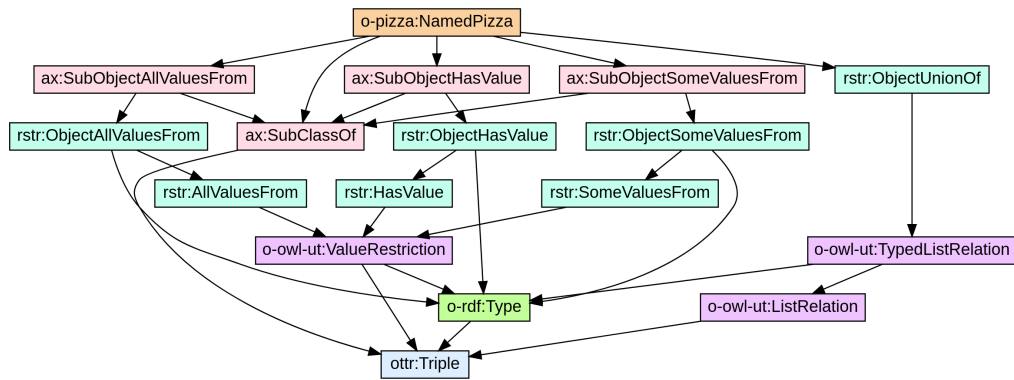
A *template library* is a collection of templates developed and curated for a particular purpose, such as representing patterns for a given vocabulary, domain, or project. The ability to share and reuse templates for common modelling patterns is central to the OTTR framework and will be further elaborated in Section 5. By following best practices and principles similar to linked open data [19] and ontology publication, templates and template libraries are expected to be published and interconnected in a distributed and decentralized manner, promoting their reuse and community-driven curation. Our intention is that template libraries should be developed alongside the development of vocabularies and ontologies which are intended for reuse, in order to promote and simplify correct and consistent typical use of the vocabulary or ontology. Given that a template's signature is clearly documented and understood, there is no need to understand how the template is implemented in order to correctly instantiate the template. Templates at different abstraction levels, and templates and their instances, target different users and use cases, and can hence be created and managed separately and by different users.

► **Example 10.** The `o-p:NamedPizza` template is published at its IRI, <https://tpl.ottr.xyz/p/pizza/0.2/NamedPizza>, using content negotiation [44] to serve different presentations of the template,<sup>4</sup> including an HTML documentation page generated by the OTTR's docTTR tool which

<sup>4</sup> <https://tpl.ottr.xyz/p/pizza/0.2/NamedPizza.html>,



## 5:12 The Reasonable Ontology Templates Framework



■ **Figure 9** Layered dependencies between templates used by the `o-p:NamedPizza` template.

is shown in Figure 8. The `o-p:NamedPizza` template is an example template in the Core OTTR template library [52], which is available at <https://tpl.ottr.xyz>. The Core OTTR template library contains all the templates used in the examples.

Figure 9 shows the dependency graph with the `o-p:NamedPizza` on top and the base template `ottr:Triple` at the bottom. Observe that the graph is divided into different layers: the “user-facing” `o-p:NamedPizza`, “logical” OWL templates, including `o-owl-ax:SubClassOf`, “utility” templates that represent OWL restrictions and different RDF list patterns, and the low-level base template `ottr:Triple`. Each layer represents different a level of abstractions that hide the complexity of lower levels.

### 2.3 Template Instantiation

Efficient instantiation of templates is also central to the OTTR framework. For this task, it is natural to consider a template as a mapping from its signature input format to the pattern of its expansion. The OTTR framework provides two specifications, `bOTTR` and `tabOTTR`, for selecting and translating data from structured sources into template instances, which in turn can be expanded into a knowledge graph or ontology according to the corresponding template definitions. These are presented in Section 6. The `bOTTR` specification defines an RDF vocabulary with which mappings from database query results to templates may be specified. The `tabOTTR` specification describes a simple “markup” language for defining mappings to templates directly in tabular datafiles, such as CSV, TSV or Excel files. These specifications permit the OTTR framework to become a part of a complete data transformation pipeline, where external tools may be used to cleanse and prepare data for template instantiation, and the OTTR framework’s mapping specification may be used to collect and integrate data from multiple sources to build knowledge graphs and ontologies at scale.

► **Example 11.** Figure 10 demonstrates the use of `tabOTTR`. It shows a spreadsheet that contains arguments to 22 instances of the `o-p:NamedPizza` template in Figure 5, and that uses `tabOTTR` processing instructions to describe how the data is to be understood as instances.

---

<https://tpl.ottr.xyz/p/pizza/0.2/NamedPizza.stottr>,  
<https://tpl.ottr.xyz/p/pizza/0.2/NamedPizza.ttl>.

	a	b	c
1	#OTTR	prefix	
2	p	http://example.com#	
3	#OTTR	end	
4			
5	#OTTR	template	<a href="http://pl.ottr.xyz/p/pizza/0.2/NamedPizza">http://pl.ottr.xyz/p/pizza/0.2/NamedPizza</a>
6	1	2	3
7	iri	iri	iri+
8			
9	p:Veneziana	p:Italy	p:SultanaTopping   p:OnionTopping   p:TomatoTopping   p:PineKernels   p:OliveTopping   p:MozzarellaTopping   p:CaperTopping
10	p:AmericanHot	p:America	p:HotGreenPepperTopping   p:MozzarellaTopping   p:JalapenoPepperTopping   p:TomatoTopping   p:PeperoniSausageTopping
11	p:Margherita		p:TomatoTopping   p:MozzarellaTopping
12	p:FourSeasons		p:TomatoTopping   p:AnchoviesTopping   p:MozzarellaTopping   p:PeperoniSausageTopping   p:CaperTopping   p:MushroomTopping   p:OliveTopping
13	p:Florentina		p:GarlicTopping   p:SpinachTopping   p:MozzarellaTopping   p:OliveTopping   p:TomatoTopping   p:ParmesanTopping
14	p:PrinceCarlo		p:MozzarellaTopping   p:ParmesanTopping   p:LeekTopping   p:TomatoTopping   p:RosemaryTopping
15	p:LaReine		p:MushroomTopping   p:MozzarellaTopping   p:HamTopping   p:OliveTopping   p:TomatoTopping
16	p:American	p:America	p:PeperoniSausageTopping   p:TomatoTopping   p:MozzarellaTopping
17	p:Caprina		p:SundriedTomatoTopping   p:GoatsCheeseTopping   p:MozzarellaTopping   p:TomatoTopping
18	p:PolloAdAstra		p:SweetPepperTopping   p:CajunSpiceTopping   p:GarlicTopping   p:RedOnionTopping   p:ChickenTopping   p:TomatoTopping   p:MozzarellaTopping
19	p:Capricciosa		p:TomatoTopping   p:PeperonataTopping   p:HamTopping   p:CaperTopping   p:MozzarellaTopping   p:OliveTopping   p:AnchoviesTopping
20	p:Mushroom		p:TomatoTopping   p:MushroomTopping   p:MozzarellaTopping
21	p:FruttiDiMare		p:TomatoTopping   p:GarlicTopping   p:MixedSeafoodTopping
22	p:SloppyGiuseppe		p:MozzarellaTopping   p:GreenPepperTopping   p:TomatoTopping   p:OnionTopping   p:HotSpicedBeefTopping
23	p:Cajun		p:TobascoPepperSauce   p:PrawnsTopping   p:MozzarellaTopping   p:PeperonataTopping   p:TomatoTopping   p:OnionTopping
24	p:Napoletana	p:Italy	p:CaperTopping   p:TomatoTopping   p:OliveTopping   p:MozzarellaTopping   p:AnchoviesTopping
25	p:Soho		p:ParmesanTopping   p:GarlicTopping   p:RocketTopping   p:TomatoTopping   p:MozzarellaTopping   p:OliveTopping
26	p:QuattroFormaggi		p:TomatoTopping   p:FourCheesesTopping
27	p:Giardiniera		p:PeperonataTopping   p:SlicedTomatoTopping   p:TomatoTopping   p:MushroomTopping   p:LeekTopping   p:MozzarellaTopping   p:OliveTopping   p:PetitPoisTopping
28	p:Siciliana		p:AnchoviesTopping   p:TomatoTopping   p:HamTopping   p:OliveTopping   p:MozzarellaTopping   p:GarlicTopping   p:ArtichokeTopping
29	p:Parmense		p:AsparagusTopping   p:ParmesanTopping   p:TomatoTopping   p:MozzarellaTopping   p:HamTopping
30	p:Rosa		p:TomatoTopping   p:MozzarellaTopping   p:GorgonzolaTopping
31	#OTTR	end	
32			

■ **Figure 10** Spreadsheet using tabOTTR to specify 22 instances of the `o-p:NamedPizza` template.

## 2.4 History

An early predecessor and inspiration to OTTR templates dates back to 2008 [26]. Here, a template mechanism was developed for “lifting” compact data representations, typically tabular data, to rich semantic format according to a complex upper level ontology. A prototype of the template mechanism was implemented using OWL and SWRL rules [18].

The practical and theoretical aspects of OTTR templates were first introduced in 2017, where OTTR templates were defined parameterized knowledge bases using a dedicated OWL ontology [48], and as description logic macros [10].<sup>5</sup>

The formulation of the OTTR language later matured into a dedicated representation of templates and template instances [50, 51]. The OTTR language has since then evolved into a framework and reached the state of stable resource, far beyond a research prototype – with multiple users from different communities and industries, multiple independent implementations initiated, and several publicly available template libraries.

## 2.5 Resources

All publicly available resources managed by the OTTR team are available from the project web page: <https://ottr.xyz>. The formal specifications and software are hosted in the Git repository at: <https://gitlab.com/ottr/>. Stable releases are also published at Zenodo: <https://zenodo.org/communities/ottr/>.

## 3 Fundamentals

This section defines the formal templating mechanism that underlies the OTTR framework. The presentation follows three tracks that are given in tandem: (1) definition blocks define the conceptual and formal aspects of the OTTR framework, such as template, template instances, validity of

<sup>5</sup> The name “Reasonable Ontology Templates” comes partly from the fact that in the first version of OTTR, templates were parameterized OWL ontologies that could be directly reasoned over. Also, “reasonable” has a suitable double meaning of “being reasonable” and “being subject to reasoning”. The acronym OTTR is inspired by OWL.

templates, and instance expansion; (2) implementation blocks describe how the conceptual model is adapted to semantic web technologies, with the specific purpose of using the OTTR framework to produce RDF graphs; and (3) syntax blocks specify the stOTTR serialization format for OTTR; more details on OTTR serialization format are given in Section 4.

We start by defining terms and types, before we introduce template instances and template objects. We then introduce template expansion, and end by defining libraries and datasets and their properties, such as correctness.

### 3.1 Terms and Types

OTTR is a language for expressing statements, in particular for ontologies and knowledge graphs represented using OWL and RDF. The basic building blocks for such statements are terms. As different terms may play different roles within these statements, and denote, e.g., relationships, entities, or data values, OTTR introduces a type system over the terms to ensure that terms are used correctly. The type system assigns a type to each term and uses subtype and compatibility relationships between types to check correct and consistent use of terms across ontologies and knowledge graphs. Below we introduce these terms and type systems.

► **Definition 12.** *We assume we have a (possibly countably infinite) set  $\mathcal{C}$  of constants at least containing the elements  $nil$  and  $none$ . Furthermore, we assume we have a countably infinite set  $\mathcal{V}$  of variables. Let the set of terms  $\mathcal{E}$  be defined inductively as follows: All elements of  $\mathcal{C}$  and  $\mathcal{V}$  are terms, and for any finite list of terms  $e_1, e_2, \dots, e_n \in \mathcal{E}$  then  $\langle e_1, e_2, \dots, e_n \rangle \in \mathcal{E}$ . Elements of  $\mathcal{E}$  of the form  $nil$  and  $\langle e_1, e_2, \dots, e_n \rangle$  are called list terms or simply lists. We let  $length(l)$  denote the length of the list  $l$  and  $l(i)$  denote the  $i$ th term (1-indexed) of the list  $l$  if  $i \leq length(l)$  and  $none$  otherwise.*

Note that list terms can be nested arbitrarily, so if  $e_1, e_2, e_3$  are terms, then, e.g.,  $\langle e_1, \langle e_2, e_3 \rangle, nil \rangle$  is also a term. The special constant  $none$  denotes a missing value, similar to how `NULL` or `null` is used in SQL and many programming languages. This constant is not intended to be used in the final statements that become part of constructed the knowledge graph, but only within the OTTR framework's definitions.

► **Definition 13.** *The set of basic types  $\mathcal{B}$  is a set that contains at least the elements  $\top$  and  $\perp$ , and that is partially ordered by the subtype relation  $\leq$  such that for any  $t \in \mathcal{B}$  we have  $\perp \leq t$  and  $t \leq \top$ . The inverse relation of subtype is called supertype.*

► **Definition 14.** *The set of types  $\mathcal{T}$  is the smallest set such that*

- $\mathcal{B} \subseteq \mathcal{T}$
- if  $t \in \mathcal{B}$ , then  $LUB\langle t \rangle \in \mathcal{T}$
- if  $t \in \mathcal{T}$ , then  $List\langle t \rangle \in \mathcal{T}$
- if  $t \in \mathcal{T}$ , then  $NEList\langle t \rangle \in \mathcal{T}$

*Furthermore,  $\leq$  is extended to  $\mathcal{T}$  as follows:*

- if  $t_1 \leq t_2$  for  $t_1, t_2 \in \mathcal{T}$ , then  $NEList\langle t_1 \rangle \leq NEList\langle t_2 \rangle$  and  $List\langle t_1 \rangle \leq List\langle t_2 \rangle$
- if  $t \in \mathcal{B}$ , then  $NEList\langle t \rangle \leq List\langle t \rangle$
- if  $t \in \mathcal{B}$ , then  $LUB\langle t \rangle \leq t$

*All non-list terms have a unique type given by the typing relation written  $e : t$  for a term  $e$  with type  $t$ , where  $none : \perp$ . The typing relation is extended to list-terms as follows:*

- $nil : List\langle \perp \rangle$
- $\langle e_1, e_2, \dots, e_n \rangle : NEList\langle LUB\langle \top \rangle \rangle$

We call types of the form  $List\langle t \rangle$  and  $NEList\langle t \rangle$  (non-empty list) for *list types*. These types have terms that are lists, i.e., ordered collections of terms. Note that we distinguish, at the type level, between empty and non-empty lists. Types of the form  $LUB\langle t \rangle$  are called *LUB-types* where LUB is short for *least upper bound*. The motivation for the latter type constructor builds on the following definition.

► **Definition 15.** Let  $\triangleright$  be the least relation between types such that whenever  $t_1 \leq t_2$  then:

- $t_1 \triangleright t_2$
- $LUB\langle t_2 \rangle \triangleright t_1$

If  $t_1 \triangleright t_2$  we say that  $t_1$  is compatible with  $t_2$ .

The intuition behind the compatibility relation between types is to permit the use of terms in places that are compatible with their type. We use the notion of compatible types to check *correct use of terms* and *consistent use of terms*. Correct use means that a term may only be used in places where its type is compatible with the expected type. Consistent use, which is only relevant for LUB-typed terms, means that a term is not used in multiple places that are incompatible.

LUB-types are required when the lexical form of terms is alone not sufficient to determine its type, which is typically when there are more types than different lexical forms. In these cases, one needs to examine the expected types of where the terms are used to establish if the term is used consistently. For the semantic web languages RDF and OWL, this is relevant as the IRIs and blank nodes of RDF may be used to designate different types of entities in OWL that are necessary to keep apart to ensure their correct and consistent use. For instance in OWL, object properties and datatype properties are disjoint types, yet, it is not possible to determine based on the lexical representation alone if an IRI represents an object property or a datatype property. For these terms, we only give an upper bound (LUB) of what types they can have. A term to which we assign the type  $LUB\langle t \rangle$  may have a type that is a subtype of  $t$ , and may therefore be used any place where any subtype  $t'$  of  $t$  is expected. However, terms must also be used consistently, so the same term cannot then be used in a place where a type not compatible with  $t'$  is expected.

Note how we exploit LUB-types when we assign non-empty list terms the type  $List\langle LUB\langle \top \rangle \rangle$ . Since all list elements are type-checked, it is unnecessary to give a more specific type to the list itself. Any type violation of a non-empty list is either due to the list itself is given to an argument expecting a non-list, or that a term “inside” the list is of an incompatible type. The type assigned to the list term itself need only account for the first of these two cases. However, to type-check the terms inside lists we need to know how deeply nested a term is inside a list, and how deeply nested a given type is inside a list type. This is captured in the following definitions.

► **Definition 16.** We define  $\delta_{\mathcal{E}}$  to be a binary function from pairs of terms to natural numbers as follows:

- $\delta_{\mathcal{E}}(a, a) = 0$  for any  $a \in \mathcal{E}$
  - if  $\delta_{\mathcal{E}}(a, b) = n$ , then  $\delta_{\mathcal{E}}(a, \langle e_1, \dots, b, \dots, e_n \rangle) = n + 1$ , for any  $a, b, e_1, \dots, e_n \in \mathcal{E}$
- If  $\delta_{\mathcal{E}}(e_1, e_2) = n$ , we say that  $e_1$  occurs at depth  $n$  in  $e_2$ .

► **Definition 17.** Let  $\delta_{\mathcal{T}}$  be a binary function from pairs of types to natural numbers as follows:

- $\delta_{\mathcal{T}}(t, t) = 0$ , for any  $t \in \mathcal{T}$
- $\delta_{\mathcal{T}}(t_1, t_2) = n$  then  $\delta_{\mathcal{T}}(t_1, List\langle t_2 \rangle) = n + 1$  and  $\delta_{\mathcal{T}}(t_1, NEList\langle t_2 \rangle) = n + 1$  for any pair  $t_1, t_2 \in \mathcal{T}$

If  $\delta_{\mathcal{T}}(t_1, t_2) = n$ , we say that  $t_1$  occurs at depth  $n$  in  $t_2$ .

As an example, in the type  $NEList\langle List\langle t \rangle \rangle$ ,  $t$  occurs at depth 2.

The definition is used to relate depths of terms in lists to the type at corresponding depths in nested list types. However, note that for  $LUB\langle t \rangle$ ,  $t$  must be a basic type and, e.g., not a list type. It is therefore no  $n$  such that  $t$  occurs at depth  $n$  in  $LUB\langle t \rangle$ , but, e.g.,  $LUB\langle t \rangle$  occurs at depth 1 in  $NEList\langle LUB\langle P \rangle \rangle$ .

## 5:16 The Reasonable Ontology Templates Framework

We now introduce the implementation of terms and types to be used for creating knowledge graphs and ontologies in RDF and OWL.

► **Implementation 18.** All vocabulary terms defined in the OTTR framework use the following namespace, unless otherwise noted:

```
@prefix ottr:    <http://ns.ottr.xyz/0.4/> .
```

► **Implementation 19.** Let  $\mathcal{E}$  be the set of all valid RDF terms, i.e., IRIs, literals and blank nodes [27]. Variables are designated by blank nodes, so let  $\mathcal{V}$  be an infinite set of blank nodes. All IRI terms have type  $LUB\langle\text{ottr:IRI}\rangle$ , all non-list blank nodes have type  $LUB\langle\text{rdfs:Resource}\rangle$ , and all literals have a type equal to their specified datatype or `xsd:string` if no datatype is given. The term `rdf:nil` denotes *nil* and has the type  $List\langle\text{rdfs:Resource}\rangle$ . All other RDF lists denote the corresponding list term and have the type  $NEList\langle LUB\langle\text{rdfs:Resource}\rangle\rangle$ .

► **Syntax 20.** Terms in stOTTR share the same syntax as terms in Turtle [2], both for IRIs, blank nodes, literals and lists, except that lists are written surrounded with parenthesis with elements *separated by commas*. stOTTR also adopts Turtle’s syntax for defining prefixes. Variable terms are written using Turtle’s syntax for blank node labels, prefixed by a question mark. We may write `none` for the term *none*, and `()` for the empty list *nil*. stOTTR is space-insensitive.

► **Implementation 21.** All basic types are listed in Table 1. All of these, except those prefixed by `ottr:` are IRIs taken from the RDF, RDFS, OWL and XSD standards. `ottr:Bot` denotes  $\perp$ , whereas `rdfs:Resource` denotes  $\top$ . The types are presented with a description taken from the respective standards, and possibly given a supertype that follows this description and which forms the basis of determining compatibility between types. The type hierarchy is published at Zenodo: <https://zenodo.org/records/12607216>.

► **Syntax 22.** Basic types are denoted by their IRI as defined above, using the syntax for IRIs from Turtle. For complex types, we write  $LUB\langle t \rangle$ ,  $List\langle t \rangle$  and  $NEList\langle t \rangle$ , where  $t$  is a type.

► **Example 23.** The term `"3"^^xsd:int` has type `xsd:int`, and since `xsd:int` is a subtype of `xsd:long`, and `xsd:long` is a subtype of `xsd:integer`, we have that `xsd:int` is compatible with `xsd:integer` and can use `"3"^^xsd:int` where a value of type `xsd:integer` is expected.

The term `ex:mary` is an IRI, and therefore has type  $LUB\langle\text{ottr:IRI}\rangle$ . Since `owl:NamedIndividual` is a subtype of `ottr:IRI`, we have that  $LUB\langle\text{ottr:IRI}\rangle$  is compatible with `owl:NamedIndividual`, and can therefore use the term `ex:mary` where a term of type `owl:NamedIndividual` is expected.

The following illustrates an interesting feature of OTTR’s type system. Some types of the OWL ontology language are defined to be disjoint, such as OWL object properties and datatype properties, and should raise an error in the case that an IRI is assigned multiple such types. Other cases of assigning multiple types to the same IRI can result in what is called *punning*, e.g., stating that `Eagle` is both a `owl:NamedIndividual` and a `owl:Class`, which is permissible in OWL, but may not always be desirable. The type hierarchy presented in Table 1 above does not permit punning, as there is no subtype of, e.g., `owl:NamedIndividual` and `owl:Class` that is different from  $\perp$ . However, it is easy to extend the type hierarchy with types to allow for punning. Table 2 lists the necessary extensions of types to allow for punning according to the OWL standard. This example is further developed in Example 53 on page 26.



■ **Table 1** The basic types of the OTTR type system.

Type	Supertype	Description
rdfs:Resource		All things described by RDF
ottr:Bot		Empty type
ottr:IRI	rdfs:Resource	An IRI (Internationalized Resource Identifier)
owl:Class	ottr:IRI	OWL Classes (understood as sets of individuals)
owl:NamedIndividual	ottr:IRI	Individuals in OWL 2
owl:ObjectProperty	ottr:IRI	Properties connecting pairs of individuals
owl:DatatypeProperty	ottr:IRI	Properties connecting individuals with literals
owl:AnnotationProperty	ottr:IRI	Properties used to provide an annotation for an ontology, axiom, or an IRI
rdfs:Datatype	ottr:IRI	Data values
rdfs:Literal	rdfs:Resource	Literal values such as strings and integers
ottr:string	rdfs:Literal	Character strings with or without language tag
xsd:string	ottr:string	Character strings
xsd:normalizedString	xsd:string	Whitespace-normalized strings
xsd:token	xsd:normalizedString	Tokenized strings
xsd:language	xsd:token	Language tags per [BCP47]
rdf:langString	ottr:string	Character strings with language tag
xsd:Name	xsd:token	XML Names
xsd:NCName	xsd:Name	XML NCNames
xsd:NMTOKEN	xsd:Name	XML NMTOKENs
owl:real	rdfs:Literal	All real numbers
owl:rational	owl:real	All rational numbers
xsd:decimal	owl:rational	Arbitrary-precision decimal numbers
xsd:integer	xsd:decimal	Arbitrary-size integer numbers
xsd:long	xsd:integer	64 bit signed integers
xsd:int	xsd:long	32 bit signed integers
xsd:short	xsd:int	16 bit signed integers
xsd:byte	xsd:short	8 bit signed integers
xsd:nonNegativeInteger	xsd:integer	Integer numbers $\geq 0$
xsd:positiveInteger	xsd:nonNegativeInteger	Integer numbers $> 0$
xsd:unsignedLong	xsd:positiveInteger	64 bit unsigned integer
xsd:unsignedInt	xsd:unsignedLong	32 bit unsigned integer
xsd:unsignedShort	xsd:unsignedInt	16 bit unsigned integer
xsd:unsignedByte	xsd:unsignedShort	8 bit unsigned integer
xsd:nonPositiveInteger	xsd:integer	Integer numbers $\leq 0$
xsd:negativeInteger	xsd:nonPositiveInteger	Integer numbers $< 0$
xsd:double	rdfs:Literal	64-bit floating point numbers incl. $+\infty$ , $-\infty$ , $+0$ , $\text{NaN}$
xsd:float	rdfs:Literal	32-bit floating point numbers incl. $+\infty$ , $-\infty$ , $+0$ , $\text{NaN}$
xsd:date	rdfs:Literal	Dates (yyyy-mm-dd) with or without timezone
xsd:dateTime	rdfs:Literal	Date and time with or without timezone
xsd:dateTimeStamp	xsd:dateTime	Date and time with timezone
xsd:time	rdfs:Literal	Times (hh:mm:ss.sss...) with or without timezone
xsd:gYear	rdfs:Literal	Gregorian calendar year
xsd:gMonth	rdfs:Literal	Gregorian calendar month
xsd:gDay	rdfs:Literal	Gregorian calendar day of the month
xsd:gYearMonth	rdfs:Literal	Gregorian calendar year and month
xsd:gMonthDay	rdfs:Literal	Gregorian calendar month and day
xsd:duration	rdfs:Literal	Duration of time
xsd:yearMonthDuration	xsd:duration	Duration of time (months and years only)
xsd:dayTimeDuration	xsd:duration	Duration of time (days, hours, minutes, seconds only)
xsd:hexBinary	rdfs:Literal	Hex-encoded binary data
xsd:base64Binary	rdfs:Literal	Base64-encoded binary data
xsd:boolean	rdfs:Literal	<b>true</b> , <b>false</b>
xsd:anyURI	rdfs:Literal	Absolute or relative URIs and IRIs
rdf:HTML	rdfs:Literal	HTML content
rdf:XMLLiteral	rdfs:Literal	XML content

■ **Table 2** Examples of types that would allow punning.

Type	Supertypes
:Punned-Class-NamedIndividual	owl:Class, owl:NamedIndividual
:Punned-Class-ObjectProperty	owl:Class, owl:ObjectProperty
:Punned-Class-DatatypeProperty	owl:Class, owl:DatatypeProperty
:Punned-Class-AnnotationProperty	owl:Class, owl:AnnotationProperty
:Punned-Datatype-NamedIndividual	rdfs:Datatype, owl:NamedIndividual
:Punned-Datatype-ObjectProperty	rdfs:Datatype, owl:ObjectProperty
:Punned-Datatype-DatatypeProperty	rdfs:Datatype, owl:DatatypeProperty
:Punned-Datatype-AnnotationProperty	rdfs:Datatype, owl:AnnotationProperty
:Punned-NamedIndividual-ObjectProperty	owl:NamedIndividual, owl:ObjectProperty
:Punned-NamedIndividual-DatatypeProperty	owl:NamedIndividual, owl:DatatypeProperty
:Punned-NamedIndividual-AnnotationProperty	owl:NamedIndividual, owl:AnnotationProperty
:Punned-Class-NamedIndividual-ObjectProperty	owl:Class, owl:NamedIndividual, owl:ObjectProperty
:Punned-Class-NamedIndividual-DatatypeProperty	owl:Class, owl:NamedIndividual, owl:DatatypeProperty
:Punned-Class-NamedIndividual-AnnotationProperty	owl:Class, owl:NamedIndividual, owl:AnnotationProperty
:Punned-Datatype-NamedIndividual-ObjectProperty	rdfs:Datatype, owl:NamedIndividual, owl:ObjectProperty
:Punned-Datatype-NamedIndividual-DatatypeProperty	rdfs:Datatype, owl:NamedIndividual, owl:DatatypeProperty
:Punned-Datatype-NamedIndividual-AnnotationProperty	rdfs:Datatype, owl:NamedIndividual, owl:AnnotationProperty

### 3.2 Template Instances

Statements in OTTR are expressed by *template instances*. Before we can introduce these, we need a couple of utility definitions specific to the use of lists in statements. OTTR has special support for lists in the form of list expanders, which are functions that allow a single statement to expand to multiple statements by replacing a list of terms with the elements of the list, in different ways.

► **Definition 24.** A list expander is a function from a list of list terms to a set of lists of terms.

► **Implementation 25.** We define the following list expanders:

$$\begin{aligned}
 id(\langle l_1, \dots, l_n \rangle) &= \{\langle l_1, \dots, l_n \rangle\} \\
 cross(\langle l_1, \dots, l_n \rangle) &= \{\langle e_1, \dots, e_n \rangle \mid e_i \in l_i\} \\
 zipMin(\langle l_1, \dots, l_n \rangle) &= \{\langle l_1(i), \dots, l_n(i) \rangle \mid 1 \leq i \leq \min_{k \leq n} length(l_k)\} \\
 zipMax(\langle l_1, \dots, l_n \rangle) &= \{\langle l_1(i), \dots, l_n(i) \rangle \mid 1 \leq i \leq \max_{k \leq n} length(l_k)\}
 \end{aligned}$$

That is,  $id$  is the identity function,  $cross$  is the cross product of its argument lists,  $zipMin$  is the convolution restricted to the shortest list, and  $zipMax$  is the convolution where all lists are made of equal length by padding *none*-terms at the end (remember that  $L(i) = none$  if  $i > length(L)$ ).

► **Example 26.** This shows the behaviour of the list expanders on the same input.

$$\begin{aligned} cross(\langle 1, 2, 3 \rangle, \langle 4, 5 \rangle) &= \{\langle 1, 4 \rangle, \langle 2, 4 \rangle, \langle 3, 4 \rangle, \langle 1, 5 \rangle, \langle 2, 5 \rangle, \langle 3, 5 \rangle\} \\ zipMin(\langle 1, 2, 3 \rangle, \langle 4, 5 \rangle) &= \{\langle 1, 4 \rangle, \langle 2, 5 \rangle\} \\ zipMax(\langle 1, 2, 3 \rangle, \langle 4, 5 \rangle) &= \{\langle 1, 4 \rangle, \langle 2, 5 \rangle, \langle 3, none \rangle\} \end{aligned}$$

► **Definition 27.** Let  $\mathcal{L}$  be a set of list expanders that contains at least  $cross$ ,  $zipMin$  and  $zipMax$ .

The list expanders defined above will be used in the definition of instance expansion, Section 3.4. With this, we are ready to define the notion of template instance.

► **Definition 28.** A template instance (or just instance) is a 4-tuple  $(t, A, E, e)$  of

- a constant term  $t$ , called the instance's template name,
- a list of terms  $A$  called the instance's arguments,
- a set of indices  $E$ , denoting which arguments to apply a list expander to,
- and a list expander  $e$ .

The arity of an instance is the size of its argument list. A ground template instance is a template instance where the value of every argument is a constant.

A template instance can be viewed as a *call* to a template. A template is a definition of a pattern of statements, and a template instance denotes one instance of the template's pattern.

► **Syntax 29.** Instances have the form  $t(a_1, \dots, a_n)$ . where  $t$  is a template name in the form of an IRI, and each  $a_i$  is an argument term. List expanders are written before the template name followed by a  $|$  (where the  $id$ -expander is always omitted), with the argument terms to expand marked with  $++$ . Examples:

Structure	Syntax
$(t, \langle a_1, \dots, a_n \rangle, \emptyset, id)$	$t(a_1, \dots, a_n)$ .
$(t, \langle a_1, a_2, a_3 \rangle, \{1\}, cross)$	$cross \mid t(++a_1, a_2, a_3)$ .
$(t, \langle a_1, a_2, a_3 \rangle, \{1, 3\}, zipMin)$	$zipMin \mid t(++a_1, a_2, ++a_3)$ .

► **Example 30.** The following are examples of written instances:

```
ex:Person(ex:mary, "Mary Smith", "1980-02-03"^^xsd:date) .

ex:Person(ex:peter, "Peter Smith", "1984-10-01"^^xsd:date) .

ex:Person(ex:bob, "Bob Green", none) .

cross | ex:HasFamilyRelation(
  ++(ex:peter, ex:mary),
  ++(ex:carl, ex:nora),
  ex:parentOf) .

zipMax | ex:HasFamilyRelation(
  ++(ex:eric, ex:hannah, ex:bob),
  ex:peter,
  ++(ex:father, ex:mother)) .
```

The corresponding templates to these instances are defined in Example 44 and their corresponding RDF statements can be seen in Example 48. The first three instances each describe a person, where the first argument is the person's IRI, the second argument is the person's name, and the final argument is the person's birthdate. Note that for the final instance, there is no value (i.e., *none*) given for the birthdate, e.g., the birthdate is unknown. The fourth instance uses the *cross* list-expander to create `ex:HasFamilyRelation` instances with the `ex:parentOf` property, for all combinations of elements from the two lists, thus stating that `ex:peter` and `ex:mary` are the parents of `ex:carl` and `ex:nora`. The final instance also instantiates `ex:HasFamilyRelation`, but uses a *zipMax* list expander to pair people with their relation to `ex:peter`, thus making `ex:eric` the `ex:father` of `ex:peter`, whereas `ex:hannah` is his `ex:mother`. For `ex:bob`, there is no given property, i.e., the value is *none*, and it's up to the definition of the template, whether this parameter is optional or not, how this is handled.

### 3.3 Templates

A *template* is a parameterized set of statements – which themselves are template instances. Thus, templates are a recursive structure where a template is defined in terms of other templates. Base templates are the exception, and are used to represent basic statements in a different data representation language.

Before we can define our notion of templates, we need to establish some preliminary definitions.

► **Definition 31.** Let  $\mathcal{M}$  be a set of tokens, called *modifiers*, that contains at least the token *optional*, denoting an optional value.

The *optional* modifier is used to control how the *none* term behaves during expansion. This is defined in Section 3.4. However, the intuition is that we specify a parameter as *optional* if *none* is a meaningful argument, and omit *optional* when it is not. All statements with a *none* as arguments to a non-*optional* are discarded. This allows templates to contain subpatterns that are only used when specific values are present (i.e., not *none*).

► **Implementation 32.** We extend  $\mathcal{M}$  with an additional modifier called *nonBlank*, that specifies that the value is not a blank node. Its behaviour is defined in Implementation 55.

Recall that our implemented terms contain blank nodes, and that blank nodes are, according to the RDF specification [27], not permitted as predicates in RDF triples. Blank nodes can also be undesirable in certain other settings, for example, if concrete values are required to be meaningful for their intended use. The *nonBlank* modifier is introduced to control where blank nodes are permitted and not. That is, a blank node is not allowed as an argument to a parameter marked with *nonBlank*.

► **Definition 33.** A parameter is a 4-tuple  $(v, t, d, M)$  consisting of:

- a variable term  $v$  different from *none*, called the parameter variable
- a type  $t$
- a (possibly *none*) constant term  $d$ , called the parameter's default value
- a (possibly empty) set of modifiers  $M$ .

In the above definition, we use *none* to denote that a parameter does not have a default value, and say that a parameter *does not have a default value* if the parameter's default value is *none*.

► **Syntax 34.** A parameter is written with modifiers first, where a question mark denotes *optional* and an exclamation mark denotes *nonBlank*, and nothing is written for the empty set of modifiers. Following this comes the parameter's type. We can omit writing the type `rdfs:Resource`. Then

follows the parameter's variable. Finally, if the default value is not *none*, the value is written at the end separated from the variable with an equals sign. Examples:

Structure	Syntax
$(v, t, \text{none}, \emptyset)$	$t \ ?v$
$(v, t, \text{none}, \{\text{optional}\})$	$? \ t \ ?v$
$(v, t, d, \{\text{optional}, \text{nonBlank}\})$	$! ? \ t \ ?v=d$

- **Definition 35.** A template signature (or just signature) is a triple  $(t, P, N)$  of
- a constant term  $t$ , called the signature's template name,
  - a list of parameters  $P$  such that all parameter variables in a signature's list of parameters are different,
  - and a set of annotations  $N$ , which is a set of ground template instances; we call these annotation instances.

The arity of a signature is the size of its parameter list. The type of a variable is the type of its parameter within the signature's parameter list.

As we shall see, a signature is part of the definition of a template. However, a signature is also meaningful on its own, as documentation of how to use a template, similar to function and method signatures in programming languages.

A signature may contain a set of ground instances called annotations. These are meant to be used for documenting the signature (similar to Javadoc in Java or Docstrings in Python), such as who created the template, the version of the template, and a description of the template pattern.

- **Syntax 36.** A signature is written starting with the template name, followed by the list of parameters enclosed in square braces. Annotations, if any, are listed after this, separated by commas and prefixed with @@. Examples:

Structure	Syntax
$(t, P, \emptyset)$	$t [P] \ .$
$(t, P, \{(i_1, a_1), (i_2, a_2)\})$	$t [P] \ @@i_1(a_1), @@i_2(a_2) \ .$

A signature ends with a dot.

- **Definition 37.** A base template is a pair  $(S, \text{base})$  of a template signature and the token base.

A base template denotes a parameterized basic statement that cannot be broken down into a set of smaller (parameterized) statements. Base template instances can either be used directly in an OTTR serialization, or, more commonly, be transformed into statements in a different language and serialization format.

- **Syntax 38.** A base template is written similarly to a template, except that the pattern is replaced with the BASE keyword, that is:

Structure	Syntax
$(S, \text{base})$	$S \ :: \ \text{BASE} \ .$

- **Implementation 39.** Our implementation contains one base template that denotes an RDF triple:

```
((ottr:Triple,
  ((subject, ottr:IRI,  $\emptyset$ ),
   (predicate, ottr:IRI, {nonBlank}),
   (object, rdfs:Resource,  $\emptyset$ )),  $\emptyset$ ),
 base)
```

or, equivalently in stOTTR format:

```

ottr:Triple [
  ottr:IRI ?subject,
  ! ottr:IRI ?predicate,
  rdfs:Resource ?object
] :: BASE .

```

One can imagine implementations supporting other base templates: Base templates for RDF quadruples, OWL expressions, rows in tabular files, or SQL INSERT statements.

We are now ready to define the central concept of a template.

► **Definition 40.** A template is pair  $(S, B)$  of a template signature  $S$  and a set of template instances  $B$  called the template's pattern; we call these pattern instances.

A template is the core construct in OTTR, and is the primary means of abstraction. Using templates, we can create complex parameterized statements that are easy to reuse. A template can either be defined in terms of base templates directly, or by instantiating other templates (or a combination). Taking a bottom-up approach, this supports layers of abstractions, each layer creating more complex statements that are closer to the terminology of that of a concrete domain to be modelled. To use a template, all one needs to know is its signature. The signature states the arguments a user must provide, and may also contain annotations that further describe the intended use of the template.

► **Syntax 41.** A template is written with the signature first (as described above) except the final dot, followed by `::`, and then the pattern instances separated by comma and enclosed in curly braces, and finally ends with a dot. Examples:

Structure	Syntax
$(S, \emptyset)$	<code>S :: .</code>
$(S, \{i_1, i_2\})$	<code>S :: i_1, i_2 .</code>

► **Definition 42.** Let  $\mathbb{S}$  be the set of all signatures,  $\mathbb{B}$  be the set of all base templates, and  $\mathbb{T}$  be the set of all templates. Let  $\mathbb{O} = \mathbb{S} \cup \mathbb{B} \cup \mathbb{T}$ , and let the elements of  $\mathbb{O}$  be called template objects.

Furthermore, let  $\sigma$  be a function from sets of template objects to sets of signatures, such that  $\sigma(O)$  is the set of all template signatures contained either directly in  $O$ , or within a template or base template in  $O$ .

► **Definition 43.** We say that a template instance  $I$  is the instance of a template signature  $T$  if  $T$  has the same template name as  $I$ . For an argument  $a$  in instance  $I$  of signature  $T$ , we say that its corresponding parameter of  $T$  is the parameter with the same index in the parameter list as the index of  $a$  in the argument list of  $I$ .

► **Example 44.** The templates used in Example 30 are defined as follows:

```

ex:Person[ owl:NamedIndividual ?p, xsd:string ?name, ? xsd:date ?born ]
  @@ottr:Triple(ex:person, ex:madeBy, ex:leifhka),
  @@ottr:Triple(ex:person, rdfs:label, "Person Template")
:: {
  o-rdf:Type(?p, ex:Person),
  ottr:Triple(?p, ex:hasName, ?name),
  ottr:Triple(?p, ex:born, ?born)
} .

```

```

ex:HasFamilyRelation [
  owl:NamedIndividual ?p1,
  owl:NamedIndividual ?p2,
  ! owl:ObjectProperty ?r=ex:isFamilyRelatedTo
] :: {
  ottr:Triple(?p1, ?r, ?p2)
} .

```

Note that we have given the property parameter a default value, so if *none* is given as argument, the default value `ex:isFamilyRelatedTo` is used. The `?born` parameter is specified as optional, hence a missing birthdate will still create a person with an IRI and name, but no birthdate.

One can now use these templates to register complete families, where input is given as lists of IRIs and names for parents and children per family. We can capture both the creation of the persons and their relations with a single template as follows:

```

ex:NuclearFamily[
  List<owl:NamedIndividual> ?parents,
  List<xsd:string> ?parentNames,
  List<owl:NamedIndividual> ?children,
  List<xsd:string> ?childrenNames
] :: {
  zipMax | ex:Person(++?parents, ++?parentNames, none),
  zipMax | ex:Person(++?children, ++?childrenNames, none),
  cross | ex:HasFamilyRelation(++?parents, ++?children, ex:parentOf)
}

```

As we assume that input does not contain any dates of birth, the `ex:NuclearFamily` template uses a *none* value as argument for the corresponding parameter in the `ex:Person` template.

### 3.4 Instance Expansion

We have now defined the core constructs in the OTTR framework, and will proceed to define the process of *instance expansion*, which is to iteratively transform instances into ultimately instances of base templates only.

We treat list expansion separately first, as this is technically the most complex part of the expansion process. List expansion is specified using two functions, where the first selects the lists to expand from the instance and expands them using the given list expander function, while the second creates one instance per element in the result of this function application. The full list expansion is the composition of these two functions.

► **Definition 45.** Let  $I = (t, A, E, e)$  be an instance of arity  $m$  with list expander indices  $E = \{i_1, \dots, i_n\}$  where  $i_k < i_{k+1}$ . Define the function  $\epsilon_1$  from instances to set of argument lists as follows:

$$\epsilon_1((t, A, \{i_1, \dots, i_n\}, e)) = e(\langle A(i_1), \dots, A(i_n) \rangle)$$

Here we use  $A(i_k)$  to denote the  $i_k$ 'th element of  $A$ . Furthermore, let

$$\epsilon_2(A, L, E, i) = \begin{cases} L(E'(i)), & \text{if } i \in E \\ A(i), & \text{otherwise} \end{cases}$$

where  $E'(i)$  is the position of  $i$  in  $E$  in ascending order. Finally, let

$$\epsilon((t, A, E, e)) = \{t(\epsilon_2(A, L, E, 1), \dots, \epsilon_2(A, L, E, m)) \mid L \in \epsilon_1(t, A, E, e)\}$$

## 5:24 The Reasonable Ontology Templates Framework

The function  $\epsilon$  takes an instance and produces a set of instances by first selecting the argument lists that are to be expanded and applies the list expander function (with  $\epsilon_1$ ), and then creates new instances based on the expanded lists by combining elements of the expansion with the original non-expanded values of the argument instance (with  $\epsilon_2$ ).

► **Definition 46.** The direct expansion  $\delta(I)$  of a ground instance  $I = (t, A, E, e)$ , where  $t$  corresponds to the template object  $T$ , is defined as follows:

1. if  $E \neq \emptyset$ , then the direct expansion of  $I$  is  $\epsilon(I)$ .
2. if there is an  $i$  such that  $a_i = \text{none}$  and its corresponding parameter is not optional and has no default value, then  $\delta(I) = \emptyset$ .
3. if  $T$  is a base template or a signature (and not a template), then  $\delta(I) = \{I\}$ .
4. otherwise, let  $T = (S, B)$  and build the induced substitution  $\sigma$  of  $T$  and  $I$  by considering each argument  $a_i$  of  $I$  and its corresponding parameter  $P_i$  with variable  $x_i$  in  $T$ :
  - if  $a_i$  has value  $\text{none}$  and  $p_i$  has a default value  $d$ , then  $\sigma := \sigma \cup \{x_i/d\}$
  - otherwise,  $\sigma := \sigma \cup \{x_i/a_i\}$

Then let  $\delta(I) = B\sigma$ , that is,  $\sigma$  applied to the pattern  $B$  of  $T$ .

In the above definition, the first case is performing the list expansion defined in the previous definition. The second case handles *none* values, where *none* values given to non-optional parameters (without default value) result in an empty expansion, i.e., the instance is discarded, and *none* values given to parameters with a default value are replaced with that default value. The third case states that the expansion of a base template (or a signature, i.e., the case where we do not have the full definition of a template object) is just the base template itself (however, note that step comes after the former two, so these steps apply first). The final case is the replacement of an instance to a template with the pattern the template denotes, where parameter values are substituted with argument values.

This denotes a single step in the expansion, the full expansion of an instance is simply the fix-point of this process.

► **Definition 47.** The expansion of a set of ground instances  $\mathcal{I}$  is the fix-point of the following function:

$$\eta(\mathcal{I}) = \bigcup \{\delta(I) \mid I \in \mathcal{I}\}$$

Example 48 gives an example of the expansion process of the previously exemplified instances and templates.

► **Example 48.** The example demonstrates the expansion of selected instances from Example 30. For the two first examples, we show the step-wise expansion process.

The following instance:

```
ex:Person(ex:mary, "Mary Smith", "1980-02-03"^^xsd:date) .
```

... expands in one step to:

```
o-rdf:Type(ex:mary, ex:Person),
ottr:Triple(ex:mary, ex:hasName, "Mary Smith"),
ottr:Triple(ex:mary, ex:born, "1980-02-03"^^xsd:date)
```

... which expands in one step to:

```
ottr:Triple(ex:mary, rdf:type, ex:Person),
ottr:Triple(ex:mary, ex:hasName, "Mary Smith"),
ottr:Triple(ex:mary, ex:born, "1980-02-03"^^xsd:date)
```



... which is equivalent to the following RDF graph:

```
ex:mary rdf:type ex:Person ;
        ex:hasName "Mary Smith" ;
        ex:born "1980-02-03"^^xsd:date .
```

The following instance:

```
ex:Person(ex:bob, "Bob Green", none) .
```

... expands in one step to:

```
o-rdf:Type(ex:bob, ex:Person),
ottr:Triple(ex:bob, ex:hasName, "Bob Green"),
ottr:Triple(ex:bob, ex:born, none)
```

... which expands in one step to:

```
ottr:Triple(ex:bob, rdf:type, ex:Person),
ottr:Triple(ex:bob, ex:hasName, "Bob Green")
```

... which is equivalent to the following RDF graph:

```
ex:bob rdf:type ex:Person ;
        ex:hasName "Bob Green" .
```

The following instance:

```
cross | ex:HasFamilyRelation(
  ++(ex:peter, ex:mary),
  ++(ex:carl, ex:nora),
  ex:parentOf) .
```

... expands to the following RDF graph:

```
ex:peter ex:parentOf ex:carl, ex:nora .
ex:mary ex:parentOf ex:carl, ex:nora .
```

The following instance:

```
zipMax | ex:HasFamilyRelation(
  ++(ex:eric, ex:hannah, ex:bob),
  ex:peter,
  ++(ex:fatherOf, ex:motherOf)) .
```

... expands to the following RDF graph:

```
ex:eric ex:fatherOf ex:peter .
ex:hannah ex:motherOf ex:peter .
ex:bob ex:isFamilyRelatedTo ex:peter .
```

Finally, we define the process of annotation expansion.

► **Definition 49.** *The annotation expansion of a template signature is the result of replacing the annotation instances of the template signature with their expansion.*

### 3.5 Template Library and Dataset

In this section, we will define what it means for a set of template objects and instances to be correct, e.g., with respect to the type system and template signature specifications. We start by defining the notions of template library and dataset.

► **Definition 50.** A template library is a set of template objects. A template dataset is a pair  $(\mathcal{L}, \mathcal{I})$  of a template library  $\mathcal{L}$  and a set of ground template instances  $\mathcal{I}$ .

► **Definition 51.** For a term  $v$  occurring in an instance  $(t, \langle a_1, \dots, a_n \rangle, E, e)$ , we say that  $v$  has inferred type  $p$  if  $v$  is a term in an argument  $a_i$  at depth  $n$  and either:

- $i \notin E$ , with a corresponding parameter with a type having the type  $p$  at depth  $n$
- $i \in E$ , with a corresponding parameter with a type having  $p$  at depth  $n - 1$

The inferred type of a term is the type the term is used as. A term may therefore have many inferred types, one for each time the term occurs in any instance.

► **Definition 52.** A term  $v$  is consistently typed in a set of instances if there exists a type  $p$  unequal to  $\perp$  such that

- $p$  is a subtype of all inferred types of  $v$ , and
- the type of  $v$  is compatible with  $p$ .

In other words, a term is consistently typed if there is a type one can assign it that is a subtype of all of its inferred types and that is compatible with the actual type of the term. Note that this definition covers both the consistent use of terms and correct typing as discussed above. For example, if a term  $v$  is used both as an `xsd:int` and as a `xsd:string`, this is a case of inconsistent use of the term  $v$ , as there is no subtype for these inferred types (unequal to  $\perp$ ), which violates the first point. If the term  $v$  has type `xsd:int` but is used as a `xsd:string`, then it is a case of incorrect typing and a violation of the second point of the definition.

► **Example 53.** Assume the IRI `ex:Eagle` is used both as a `owl:NamedIndividual` and a `owl:Class`. Under the type hierarchy given in Table 1 there exists no subtype of these types, hence `ex:Eagle` is not consistently typed. Under the type hierarchy given in Table 2 there exists a subtype of these types, `:Punned-Class-NamedIndividual`, hence `ex:Eagle` is consistently typed.

► **Definition 54.** An instance is modifier correct if all of its arguments satisfy the corresponding parameter modifiers.

► **Implementation 55.** Any non-blank constant and any variable of a parameter marked with `nonBlank` satisfies the `nonBlank` modifier.

Note that the definition above ensures that the `nonBlank` modifier is propagated upwards in the dependency graph of templates so that any variable used as an argument to a `nonBlank`-parameter must be marked as `nonBlank`.

► **Definition 56.** A set of instances is consistently typed (modifier correct) if every term occurring in it is consistently typed (modifier correct). A template library is consistently typed (modifier correct) if the set of all instances occurring in it is consistently typed (modifier correct).

This covers the correct use of terms.

We now define correctness of the interplay between template objects and between templates and instances by way of several properties that combined form the notion of correctness.

► **Definition 57.** A template  $T$  directly depends on a template object  $S$  if  $T$  has a pattern that contains an instance of  $S$ . A template library is acyclic if the directly depends relation is acyclic.

Acyclicity ensures that instance expansion terminates and is finite. Note that this also disallows recursively defined templates. However, we have no means of manipulating or producing new terms apart from through list expansion. Under the current type system, we are unable to define a template that can apply a list expander to an instance of itself, as this would not be consistently typed. Thus, a recursive call within a template's pattern can only reuse the same arguments it was originally given or have constants as arguments, thus creating an infinite loop in the expansion.

► **Definition 58.** *A set of instances  $\mathcal{I}$  has referential integrity with respect to a template library  $\mathcal{L}$  if every instance has a name corresponding to a template signature in  $\sigma(\mathcal{L})$ , and that the arity of the instance equals the arity of the corresponding template signature.*

*A template library has referential integrity if no two non-signature template objects have the same name and the set of all instances occurring in it has referential integrity with respect to it.*

Referential integrity ensures that all instances refer to a unique template object, and that the number of arguments equals the number of parameters in the corresponding signature.

► **Definition 59.** *A template object is well-founded if it is a base template or if it is a template that depends only on well-founded templates. A template library is well-founded if it contains only well-founded templates.*

Well-foundedness is a property that ensures that all templates are properly defined, that is, there are no templates that depend on a template object that is a signature only. It characterizes the fact that instances can be expanded all the way to instances of base templates only. Note that well-foundedness is not the same as acyclicity. A template that depends on a template object which is a signature is non-well-founded but acyclic, while a template that directly depends on a base template and itself (recursively) is well-founded and cyclic.

► **Definition 60.** *A semi-valid template library is a template library that is consistently typed, modifier correct, acyclic, and has referential integrity.*

► **Definition 61.** *A valid template library is a semi-valid template library that is well-founded. A valid template dataset is a template dataset where its template library is valid, and its set of instances is consistently typed and has referential integrity with respect to the template library.*

The difference between a semi-valid and a valid library is whether all template objects are properly defined or not.

► **Example 62.** Below are examples of violations of correctness of instances and templates as defined above, based on the templates from Example 44.

```
ex:Person(ex:bob, "Bob Green") .
ex:Person(_:bob, ex:bob_green, none) .
ex:Person(_:b, _:b, none) .
ex:HasFamilyRelation(ex:bob, ex:mary, _:someProp) .
```

The errors in the above instances are:

1. The instance has two arguments, but the signature requires three.
2. The IRI `ex:bob_green` is given as an argument to a parameter of the incompatible type `xsd:string`.
3. The blank node `_:b` is used inconsistently; it is used as an argument to two parameters with the types `owl:NamedIndividual` and a `xsd:string` that have no common subtype unequal to  $\perp$ .
4. The blank node `_:someProp` is used as an argument to a parameter with a nonBlank modifier.

```

ex:ErrTemplate1 [ owl:ObjectProperty ?r ] :: {
  ex:HasFamilyRelation(ex:bob, ex:mary, ?r) } .

ex:ErrTemplate2 [ ottr:IRI ?p ] :: {
  ex:Person(?p, "Mr. P", none) } .

ex:ErrTemplate3 [ owl:NamedIndividual ?p, xsd:string ?n ] :: {
  ex:MakePerson(?p, ?n) } .

ex:ErrTemplate4 [ ottr:IRI ?p ] :: {
  ex:ErrTemplate4(?p) } .

```

The errors in the above templates are:

- In `ex:ErrTemplate1`, the parameter `?r` has no `nonBlank` modifier and is used as argument to a template parameter with a `nonBlank` modifier.
- In `ex:ErrTemplate2`, the parameter `?p` has the type `ottr:IRI` and is used as an argument to a template parameter with the incompatible type `owl:NamedIndividual`.
- `ex:ErrTemplate3` depends on an undefined template `ex:MakePerson`.
- `ex:ErrTemplate4` has a cyclic definition.

## 4 Serialization Formats

The OTTR framework offers two serialization formats for representing templates and instances, a special-purpose format called `stOTTR`, and an RDF-based format specified using the `wOTTR` vocabulary.

### 4.1 stOTTR: Terse OTTR Syntax

The `stOTTR` serialization format is designed to be a terse and easy to read and write syntax for representing OTTR templates and instances following the abstract model and syntax as defined in Section 3. The `stOTTR` grammar takes the Turtle RDF grammar [2] as starting point and expands this to support expressing templates and instances. Formally, `stOTTR` is specified in Antlr<sup>6</sup> Extended Backus-Naur form (EBNF) that extends relevant parts of the formal Turtle grammar which is used for the representation of terms, i.e., IRIs, blank nodes and literals. The `stOTTR` grammar specification is developed in GitLab,<sup>7</sup> and published at `ottr.xyz`<sup>8</sup> and Zenodo.<sup>9</sup> Figure 5 on page 7 demonstrates the `stOTTR` format on the `o-p:NamedPizza` template.

### 4.2 wOTTR: RDF Vocabulary

`wOTTR` is an RDF vocabulary for expressing OTTR templates and instances in an RDF format. The motivation for an RDF-based serialization format for OTTR is to support a development and management environment for OTTR based only on semantic web standards, using, e.g., triple stores, SPARQL, OWL, and rule languages to manipulate and manage templates and their instances. The vocabulary is designed to result in a compact and readable representation of templates and instances in Turtle format exploiting in particular Turtle's compact RDF list

<sup>6</sup> <https://www.antlr.org/>

<sup>7</sup> <https://gitlab.com/ottr/spec/stOTTR>

<sup>8</sup> <https://spec.ottr.xyz/stOTTR/0.1.4/>

<sup>9</sup> <https://zenodo.org/records/12568905>

```

1  o-p:NamedPizza rdf:type ottr:Template ;
2  ottr:parameters
3    ( [ ottr:type owl:Class ; ottr:variable _:pizza ]
4      [ ottr:modifier ottr:optional ; ottr:type owl:NamedIndividual ;
5        ottr:variable _:country ]
6      [ ottr:type ( ottr:NEList owl:Class ) ; ottr:variable _:toppings ] ) ;
7  ottr:pattern
8    [ ottr:of o-owl-ax:SubObjectHasValue ;
9      ottr:values ( _:pizza pz:hasCountryOfOrigin _:country ) ] ,
10   [ ottr:of o-owl-ax:SubObjectSomeValuesFrom ;
11     ottr:modifier ottr:cross ;
12     ottr:arguments
13       ( [ ottr:value _:pizza ]
14         [ ottr:value pz:hasTopping ]
15         [ ottr:modifier ottr:listExpand ; ottr:value _:toppings ] ) ] ,
16   [ ottr:of o-owl-ax:SubClassOf ;
17     ottr:values ( _:pizza pz:NamedPizza ) ] ,
18   [ ottr:of o-owl-re:ObjectUnionOf ;
19     ottr:values ( _:b0 _:toppings ) ] ,
20   [ ottr:of o-owl-ax:SubObjectAllValuesFrom ;
21     ottr:values ( _:pizza pz:hasTopping _:b0 ) ] .

```

■ **Figure 11** The `o-p:NamedPizza` template in wOTTR syntax.

representation for expressing parameter lists, argument lists and complex type specifications. The entities defined in the wOTTR vocabulary lie close to the formal vocabulary established in Section 3; the classes, properties and named individuals of the vocabulary are listed in Table 3, Table 4, and Table 5, respectively. The wOTTR vocabulary is developed in GitLab,<sup>10</sup> and published at `ottr.xyz`<sup>11</sup> and Zenodo.<sup>12</sup>

Although the mapping from the wOTTR vocabulary to the formally defined concepts of OTTR should be immediate, there are some design choices and peculiarities that are due to the wish for a compact and readable representation, and the constraints of the RDF and OWL standards. We will illustrate these by using the `o-p:NamedPizza` template represented in the wOTTR vocabulary, which is listed in Figure 11.

**Variables** As RDF does not include variables we have chosen to use blank nodes for representing a template’s parameters. The variables of a template are specified as a list of parameters using the predicate `ottr:variable`, see, e.g., line 3 in Figure 11. Care must then be taken to not use the same blank nodes as constants.

**Lists** The wOTTR language makes frequent use of RDF lists as a means to represent an ordering of resources. We do this since RDF lists have a succinct serialization in RDF Turtle and as it is syntactically similar to ordinary function calls and predicates. In Figure 11, lists are used for parameter lists (starting on line 3), instance argument lists (e.g., on line 13), instance argument value lists (e.g., on line 9), and complex type specifications; line 6 specifies the type `NEList<owl:Class>`.

**Annotation properties** All properties of the wOTTR vocabulary are annotation properties. This is to indicate that the vocabulary is not intended to be used for reasoning over templates and instances. The use of RDF lists, as explained above, also places the vocabulary outside the OWL 2 DL fragment, as RDF lists are used in the serialization of OWL.

<sup>10</sup><https://gitlab.com/ottr/spec/wOTTR>

<sup>11</sup><https://spec.ottr.xyz/wOTTR/0.4.5/>

<sup>12</sup><https://zenodo.org/records/12581215>

■ **Table 3** wOTTR vocabulary classes and their definition. The `ottr:` prefix is omitted.

Class	Definition
<code>:Signature</code>	A <b>signature</b> specifies the permissible input for instances. It does this through its list of parameters. The IRI of the signature is a unique name that its instances must reference.
<code>:Template</code>	A <b>template</b> is a signature that additionally specifies a pattern. The pattern, which is a set of instances, determines the result of the direct expansion (1-step expansion) of an instance of the template.
<code>:BaseTemplate</code>	A <b>base template</b> is a signature with no pattern. The expansion of an instance of a base template is the instance itself.
<code>:Parameter</code>	A <b>parameter</b> specifies the variable terms or resources of a pattern and the permissible values for the corresponding instance arguments.
<code>:ParameterModifier</code>	A <b>parameter modifier</b> is a flag or marker that is set on a parameter to alter the permissible corresponding argument values and/or the behaviour of expanding instances.
<code>:Instance</code>	An <b>instance</b> is an instantiation of a signature, template or base template. The instance must refer to a signature and provide arguments that match the corresponding parameters of the signature.
<code>:ExpansionModifier</code>	An <b>expansion modifier</b> is a flag or marker that is used to alter the behaviour of expanding the marked instance.
<code>:Argument</code>	An <b>argument</b> specifies an input value for a given instance.
<code>:ArgumentModifier</code>	An <b>argument modifier</b> is a flag or marker that is used to identify that the argument plays a special role in modified expansions. See also <code>ExpansionModifier</code> .

**Two instance shapes** Template instances are stated using the property `ottr:of` which specifies the template. Instances may be specified using two different shapes, called *compact* and *canonical*. The compact shape uses the property `ottr:values` and an RDF list to directly give the argument values of the instance, very similar to how instances are written in stOTTR; line 9 shows an example. The canonical shape uses the property `ottr:arguments` and an RDF list of arguments, where each argument, usually represented by blank node, has a `ottr:value` property that sets the argument value; line 15 shows an example. The canonical shape can be used in all cases, but must be used when more data than just the argument value is required, as line 15 exemplifies by marking the argument for list expansion.

## 5 Template Libraries

A template library is a collection of templates developed and curated for a particular purpose, such as representing patterns for a given vocabulary, domain, or project. The ability to share and reuse templates for common modelling patterns is a core feature of the OTTR framework. This section gives an overview of the vision behind OTTR template libraries and the support and developments made towards the vision. Large parts of this section are taken from previous publications [52, 34] and are included here to give a complete presentation of the OTTR framework.

The vision of template libraries is similar to the role APIs and repositories of API source code play in software engineering. Just as software projects rely on stable access to APIs and documentation to work and be understood and used, ontology engineering projects using OTTR must be able to rely on the availability and documentation of templates. For this reason, it is critical that a published template does not change in any way that may affect the expansion of

■ **Table 4** wOTTR vocabulary properties, indicating their domain and range. The `ottr:` prefix is omitted.

Property	Domain	Range	Definition
<code>:parameters</code>	<code>:Signature</code>	List of <code>:Parameter</code>	Associates a signature with one required list of parameters.
<code>:annotation</code>	<code>:Signature</code>	<code>:Instance</code>	Associates a signature with an optional set of annotation instances.
<code>:variable</code>	<code>:Parameter</code>	<code>rdfs:Resource</code>	Sets the required variable of a parameter.
<code>:type</code>	<code>:Parameter</code>	(List of) <code>rdfs:Resource</code>	Sets an optional type of a parameter. A missing type implicitly sets the type to the most general type.
<code>:default</code>	<code>:Parameter</code>	<code>rdfs:Resource</code>	Sets an optional default value of a parameter. The default value is used in case an argument value is unspecified or is <code>ottr:none</code> .
<code>:pattern</code>	<code>:Template</code>	<code>:Instance</code>	Associates a template with an optional set of pattern instances.
<code>:name</code>		<code>xsd:token</code>	A human readable name or label.
<code>:of</code>	<code>:Instance</code>	<code>:Signature</code>	Associates an instance with its required signature.
<code>:arguments</code>	<code>:Instance</code>	List of <code>:Argument</code>	Associates an instance with a list of arguments.
<code>:values</code>	<code>:Instance</code>	List of <code>rdfs:Resource</code>	Associates an instance with a list of argument values
<code>:value</code>	<code>:Argument</code>	<code>rdfs:Resource</code>	Associates an argument with its argument value.
<code>:modifier</code>			

■ **Table 5** wOTTR vocabulary individuals and their type. The `ottr:` prefix is omitted.

Named Individual	Class	Definition
<code>:optional</code>	<code>:ParameterModifier</code>	<b>optional</b> is a parameter modifier which makes the value none a permissible instance argument value for this parameter.
<code>:nonBlank</code>	<code>:ParameterModifier</code>	<b>nonBlank</b> is a parameter modifier which makes blank nodes illegal instance argument values for this parameter.
<code>:cross</code>	<code>:ExpansionModifier</code>	<b>cross</b> is an expansion modifier which sets the list expansion operation to cross product.
<code>:zipMax</code>	<code>:ExpansionModifier</code>	<b>zipMax</b> is an expansion modifier which sets the list expansion operation to zip, extending smaller list to the length of the longest list by appending empty values.
<code>:zipMin</code>	<code>:ExpansionModifier</code>	<b>zipMin</b> is an expansion modifier which sets the list expansion operation to zip with the shortest list as length.
<code>:listExpand</code>	<code>:ArgumentModifier</code>	<b>listExpand</b> is an argument modifier that selects arguments for list expansion.
<code>:none</code>	<code>rdfs:Resource</code>	<b>none</b> is an individual which is used to designate a missing argument value.
<code>:Triple</code>	<code>:BaseTemplate</code>	<b>Triple</b> is a base template that represents an RDF triple.
<code>:NullableTriple</code>	<code>:BaseTemplate</code>	<b>NullableTriple</b> is a base template that represents an RDF triple and permits none value arguments.

its instances, and that the expansion can be performed at any time. The meaning of a template instance must stay constant; an instance should be considered as semantically equivalent to its expansion. This places strong requirements on the availability and versioning of templates.

To support the quality of template libraries and the management of these, concepts and procedures for library governance, together with methods for library maintenance and methodologies for library construction have been developed. A documentation system together with a purpose-built set of documentation templates is available for annotating templates to generate user-friendly documentation pages for publishing template libraries. This is presented in more detail below.

### 5.1 Template Life-cycle Management

To aid the life-cycle management of templates in the library, a set of template statuses and an interpretation of versioning categories for templates has been proposed [52].

#### 5.1.1 Status

A template's status indicates the maturity of the template and its level of support and endorsement. Each template has exactly one of the following statuses, here ordered from low to high:

*incomplete < draft < candidate < proposed recommendation < recommendation*

A template should not depend on templates of lower status than the template's own status. A template may additionally have the status of *deprecated*. The statuses are described in more detail below.

An **incomplete** template is of the lowest status, which is the default if no status is stated for a template. The only requirement for an incomplete template is that it must be syntactically correct according to its serialization format, but need not otherwise be a valid template. This means it is permissible for an incomplete template to, for example, depend on templates that do not (yet) exist or that contain type errors. An incomplete template is typically a placeholder for future work and should not be published for public use.

A **draft** template must be a syntactically correct and well-founded template, i.e., the template is completely defined and does not contain any formal errors. A draft template should not be considered mature or stable. In terms of its life-cycle, it is published in order to be available to others, both for use and for further development.

A **candidate** template is a draft template which additionally contains a complete set of metadata and is endorsed by a named individual or organization that aims to promote the template to recommendation status. The endorser is expected to actively participate in the support and promotion of the template; failure to do so may result in the deprecation of the template. A candidate template should be considered stable.

After a period in candidate status, a candidate template may be proposed as a recommendation and given the status of **proposed recommendation**. This triggers a public vote to promote the template to recommended status. Relevant comments and issues collected during the voting phase must be addressed if the template can be given the status of recommended.

A **recommended** template is of the highest status. This means that the template is of high quality and is well-integrated into the library.

A **deprecated** template is discouraged from use. An explanation for why a template is deprecated should be given.



### 5.1.2 Versioning

Each time a template is published, a new version number must be assigned to the template using semantic versioning<sup>13</sup> and the numbering scheme *major.minor.patch*, e.g., 0.2.3. In our translation of these types of updates to OTTR templates the notions of the expansion of a template and its signature are central. Any changes to the signature of a template which make existing instances incompatible with the updated template are backwards *incompatible* changes. For OTTR templates we use the following definitions:

**Patch updates** are backwards compatible changes that do not affect the expansion of a template.

These changes will neither affect existing instances of the updated template nor their expansion.

**Minor updates** are backwards compatible changes that may alter the expansion of the template.

An example is adding or removing body instances. This means that existing instances of the template remain legal and valid for the new version of the template, but their expansion will be different.

**Major updates** are backwards incompatible changes. An example is adding or removing a parameter to/from the template signature, or restricting the type of a parameter. This means that existing instances of the template will not be legal instances of the updated template.

In the case that the update greatly changes the perceived meaning of the template, the update should be categorized as a greater change than according to the above descriptions.

Draft templates are exempted from the above rules. Following the semantic versioning specification we require that draft templates have a version number with the major version 0, which signals that anything may change at any time and the public API should not be considered stable. When a template reaches the next status of candidate, the major version must be set to 1.

## 5.2 Metadata

Template metadata, like the status and version of a template, but also textual explanations, examples and provenance information, is captured by annotation instances placed on templates. The `doctr` package of the core template library contains templates created specifically for this purpose. The following lists the templates in the package and the information they are designed to capture:

**Signature:** label, description, scope and editorial notes, related resources

**Provenance:** time of creation and update, authors and contributors

**Version:** status, version number, references to previous and next versions

**ChangeNote:** change description, including author and timestamp of the change

**Example:** explanatory examples of the template

**Deprecated:** mark the template as deprecated, including an explanation of why

**Parameter:** parameter description, example, notes

The documentation of the template package is found at <http://tpl.ottr.xyz/p/doctr/>.

## 5.3 docTTR: Template Documentation System

The documentation system for OTTR templates is called *docTTR*. `docTTR` reads as input all the templates that constitute a library and produces a set of interlinked HTML pages. All pages are presented in an HTML frameset layout, inspired by Javadoc,<sup>14</sup> a documentation system for Java.

<sup>13</sup><https://semver.org/spec/v2.0.0.html>

<sup>14</sup><https://www.oracle.com/java/technologies/javase/javadoc-tool.html>

Each template documentation page contains the metadata captured by annotation instances, a list describing its parameters, the template rendered in different serialization formats, a generated sample instance of the template in all formats with its resulting expansion, both visualized and in RDF Turtle format; an expansion of the sample instance, a dependency graph showing all the templates that the template, and a list of the vocabulary elements that the template introduces.

## 5.4 Template Relations for Library Maintenance

Using the formal fundamentals of the OTTR language it is possible to define logical relationships between templates that describe characteristics that concern the quality of template libraries. We focus in particular on removing redundancy within a library, where we distinguish two different types of redundancy: a lack of reuse of existing templates, as well as recurring patterns not captured by templates within the library. To this end, we use the following template relations.

► **Definition 63.** Let  $T_1 = ((t_1, P_1, A_1), B_1)$  and  $T_2 = ((t_2, P_2, A_2), B_2)$  be two templates (that is, template  $T_i$  has name  $t_i$ , parameters  $P_i$ , annotations  $A_i$  and pattern  $B_i$ ). We say that  $T_1$  overlaps  $T_2$  if there exist sets of template instances  $I_1 \subseteq B_1$  and  $I_2 \subseteq B_2$  and substitutions  $\rho_1$  and  $\rho_2$  of respectively  $P_1$  and  $P_2$  such that  $\rho_1(I_1) = I_2$  and  $\rho_2(I_2) = I_1$ .

Furthermore, we say that  $T_1$  contains  $T_2$  if there exists a substitution  $\rho$  of the parameters of  $T_2$  such that  $\rho(B_2) \subseteq B_1$ .

Using these relations, we can now formally define notions of redundancy in template libraries.

**Lack of reuse** is a redundancy where a template  $S$  has a contains relationship to another template  $T$ , instead of a dependency relationship to  $T$ . That is,  $S$  duplicates the pattern represented by  $T$ , rather than instantiating  $T$ . This can be removed by replacing the offending portion of the pattern of  $S$  with a suitable instance of  $T$ .

**Uncaptured pattern** is a redundancy where a pattern of template instances is used by multiple templates, but this pattern is not represented by a template. In order to find uncaptured patterns one must analyse in what manner multiple templates depend on the same set of templates. If multiple templates *overlap* as defined above, this is a good candidate for an uncaptured pattern. However, an overlap does not necessarily need to occur for an uncaptured pattern to be present.

We have implemented an algorithm for efficiently identifying uncaptured patterns in large template libraries, and used it to successfully refactor a real-world template library [50].

## 5.5 Template Development Methodologies

The recursive structure of OTTR templates supports a top-down modelling approach to template library development, where complex modelling patterns are iteratively broken down into less complex patterns.

Lupp et al. [34] use this approach, and identify different informal layers to a template library: **Base templates** are at the lowest level and specify patterns over an underlying data representation language.

**Utility templates** are used to improve template formulation by grouping base template instances to avoid unnecessary repetition. Utility templates represent patterns that are typically only meaningful for users familiar with the language that the base templates abstract over, e.g., RDF.

**Logical templates** represent ontological axioms and convenient combinations of axioms, such as subclass relationship and concept partitioning.

**Domain templates** represent modelling patterns in a specific domain. They are independent of specific input formats and represent common domain conceptualizations.

**System/User-facing templates** record patterns that represent end-user or system formats which typically involve complex combinations of domain statements. A system template is hence only required if the system representation differs from the domain conceptualization and if the format is useful to represent as a template. This may be the case when the format is common, e.g., if the system is a de facto standard in the domain.

Each layer provides an “interface” for the layers above it to use, hence hiding the complexity of lower layers. The layering helps with the construction and maintenance of a template library in that the various layers typically fit different competencies and may be managed by people with different expertise – possibly with the assistance of ontology experts.

Blum et al. [5] give insights from an OTTR-specific ontology engineering methodology similar to that of Lupp et al. [34]. They characterize their methodology as both bottom-up, in the sense that existing data is taken as the starting point for developing templates, and top-down, as the methodology exploits the recursive structure of OTTR templates to incrementally break down the more complex data-facing templates to OWL and RDF templates. In their work, they found that OTTR templates helped simplify the communication between subject-matter experts and ontology experts in that it allows them to focus on *what* to model – the information content, represented by template signatures – without the need to consider *how* to model it, which is later represented by template bodies.

## 5.6 Public Template Libraries

There are a handful of publicly available OTTR template libraries:

**Core OTTR Library [52]** This library is developed and maintained by the OTTR team. It contains about 200 templates that predominantly represent common patterns over the OWL, RDFS and RDF vocabularies. It is intended to be a central resource for all OTTR users and other template libraries. The library is available at <https://tpl.ottr.xyz>.

**POSC Caesar Association** The POSC Caesar Association (PCA) has published the Product Life-cycle Management Reference Data Library (PLM-RDL) OTTR Library at <https://rds.posccaesar.org/ontology/plm/tpl/0.1/>:

*The PLM-RDL OTTR library is specifically intended to cover the process industry domain and the templates will, in general, make direct reference to resources from the PLM-RDL. If you wish to build ontologies that extend on the PLM-RDL, using these templates will help ensure that your modelling patterns are fully consistent with those of the PCA ISO 15926-14 compliant ontologies.*

**DiProMag** The DiProMag project has published an OTTR template library for modelling certain characteristics of chemical elements at [https://www.dipromag.de/dipromag\\_onto/](https://www.dipromag.de/dipromag_onto/). (See also Section 8.4.)

**Aspect OWL** The ontology design patterns for representing context in ontologies using AspectOWL [45] are encoded by an OTTR template library available at <https://odp.aspectowl.xyz/>.

## 6 Instantiation Tools

The tabular format specified by OTTR template signatures is well-suited for consuming and converting tabular data to RDF by way of OTTR’s expansion mechanism. Assuming the format of the input table matches the template signature, each row in the input table is naturally mapped

to a template instance. To this end, the OTTR framework specifies two methods for selecting and converting tabular data to OTTR template instances: *tabOTTR* specifies a small markup language for mapping tabular data files to OTTR template, and *bOTTR* is an RDF vocabulary for specifying mappings from database query results to OTTR template instances.

## 6.1 tabOTTR: Tabular OTTR Template Instances

The intended use of tabOTTR is to annotate existing tabular datafiles, such as CSV files and spreadsheets, with instructions that specify the data to select and how to transform it to the correct datatypes for OTTR template instantiation.

In order to generalize over different tabular file formats we use the terms *file*, *table*, *column*, *row* and *cell*. Tables, columns and rows within a file have a unique index which is a positive integer number assumed to be numbered in a straight-forward ordered consecutive manner. A cell has a 3-dimensional coordinate given by the indices: (table, column, row). For CSV files, we call the entire contents of the file a table and assign this the index 1. For spreadsheets, such as in MS Excel files, each sheet in a file represents a table.

The results of processing a file according to the tabOTTR specification is a set of template instances. Processing instructions are inserted directly into tables with the token #OTTR. This token must appear in the first column of the table, and the cell to the right of the token must contain a processing instruction name with possible instruction arguments given in consecutive cells immediately to the right of this cell. Each table can contain multiple processing instructions, and when processing a file all tables in the file are processed. There are three such processing instructions: **end**, **prefix** and **template**, which will be discussed below. An instruction dictates the interpretation of subsequent rows, and the scope of the instruction is the following rows until a row containing a new instruction or until the end of the table. Rows which are not in the scope of an instruction are not processed. All processed cells are trimmed for leading and trailing whitespace. All IRIs may be given using a QName, using prefixes set with the **prefix** instruction.

The instructions are defined as follows:

**end instruction** The **end** instruction takes no arguments. It is a no-operation instruction that takes no arguments and whose purpose is simply to terminate the previous instruction. We recommend to always use the **end** instruction to terminate an instruction, so that the scope of an instruction is made explicit.

**prefix instruction** The **prefix** instruction declares namespace prefixes that may be used in other instructions and that will be included in the processed output. The **prefix** instruction takes no arguments. Each following row declares a namespace prefix, where the

- 1. column contains the prefix name, and the
- 2. column contains the namespace name.

All other columns are ignored. The scope of the defined prefixes is the whole input file.

Conflicting declarations of the same prefix name must raise an error.

**template instruction** The result of processing a **template** instruction is a set of template instances.

A **template** instruction takes one argument: a template IRI. Subsequent rows have special meaning; assume the **template** instruction is on a row with index  $r$ , then:

- Row  $r + 1$  contains cells that specify the template argument index of the template instances. Cell values must be a non-negative integer or the empty string.
- Row  $r + 2$  contains cells that specify the type instruction of the template instance arguments. Cell values must have a legal type instruction; they are listed in Table 6.
- Row  $r + 3$  is ignored. This row can be used for informative content such as column headers.
- The following rows, rows  $> r + 3$ , in the scope of the **template** instruction specify each one instance of the template whose IRI is specified by the **template** instruction. The cells in these rows contain instance argument values. Each argument value is together with its

■ **Table 6** Permissible type instructions and their RDF resource interpretation.

Type instruction	RDF value interpretation
<code>iri</code>	IRI
<code>blank</code>	blank node
<code>text</code>	untyped literal
an IRI	typed literal
<code>auto</code>	determined by value, see below
$X+$ , where $X$ is one of the above type instructions	RDF list with list items determined by $X$

corresponding type instruction (as specified in row  $r + 2$ ) translated into an RDF resource following Algorithm 1. Each row then represents an argument list to a template instance ordered according to the positive integer indices of row  $r + 1$ .

► **Example 64.** This table contains a `prefix` instruction that declares two prefixes: `ex` and `foaf`. The `prefix` instruction is terminated by an `end` instruction.

```
#OTTR    prefix
ex       http://example.net#
foaf     http://xmlns.com/foaf/0.1/#
#OTTR    end
```

The result of processing the above table is equivalent to the following RDF Turtle.

```
@prefix ex:    <http://example.net#> .
@prefix foaf:  <http://xmlns.com/foaf/0.1/#> .
```

Type instructions are required for describing how values in tabular data files, which may not have any typing information, are to be translated into RDF resources. This makes it possible to for instance create an IRI or an RDF literal from the string value `http://example.com/Bob`. The values from columns that are `auto` typed are translated based on their string value and may hence end up as RDF resources of different RDF types.

Example 65 gives a simple demonstration of the `template` instruction. Table 7 demonstrates the translation of `auto` typed values. Figure 10 on page 13 shows a spreadsheet that specify 22 instances of the `o-p:NamedPizza` template listed in Figure 5.

► **Example 65.** The table below contains one `template` instruction, declaring instances of the `ottr:Triple` template, as prescribed by the first row. The next row selects which columns to use as arguments to the template (which takes 3 arguments). Columns with a value 0 will be ignored and can be, e.g., used for comments or spreadsheet formula calculations of intermediate values. The next row sets the type instruction for the columns which contain arguments, in this example, all columns have the type instruction `iri` which means that argument values are interpreted as IRIs. The next row is ignored and may be used for a descriptive label. The following rows specify instances of the templates. The `end` instruction closes the scope of the `template` instruction.

```
#OTTR    template    ottr:Triple
2        1          3          0          0
iri      iri         iri
Predicate Subject    Object
foaf:knows ex:Ann    ex:Bob
foaf:knows ex:Bob    ex:Carl
#OTTR    end
```

■ **Algorithm 1** Pseudo algorithm for translating tabular file argument values to RDF resources.

---

```

function translate(typeInstruction, value)
  Output : RDF resource
  if value = empty string then
    | return ottr:none
  else if typeInstruction is of the form X+ then
    | items ← split value by | ;
    | foreach item in items do item ← translate(X, item);
    | return RDF list of items
  else if typeInstruction = iri then
    | return IRI of value, accepting QNames and full IRIs
  else if typeInstruction = blank then
    | if value = * then
    | | return Fresh blank node
    | else
    | | return Blank node with label value
    | end
  else if typeInstruction is of the form of a QName or full IRI then
    | return Typed literal with lexical value value and datatype typeInstruction
  else if typeInstruction = text then
    | return Untyped literal with lexical value value
  else if typeInstruction = auto then
    | if value is of the form text^^type then
    | | return Typed literal with the lexical value text and type
    | else if value is of the form text@@lang then
    | | return Language tagged literal with the lexical value text and language tag lang
    | else if value = true, TRUE, false or FALSE then
    | | return Literal with boolean value of value and datatype xsd:boolean
    | else if value is of the form of a decimal number then
    | | return Literal with lexical value value and datatype xsd:decimal
    | else if value is of the form of an integer number then
    | | return Literal with lexical value value and datatype xsd:integer
    | else if value is of the form of an RDF Turtle labelled blank node then
    | | return Blank node with label value
    | else if value is of the form of a QName or full IRI then
    | | return IRI of value
    | else
    | | return Untyped literal with lexical value value
    | end
  else
  | return Untyped literal with lexical value value
  end

```

---

■ **Table 7** Examples of the translation using `auto` type instruction.

value with <code>auto</code> instruction	RDF resource	RDF type
<code>*</code>	<code>[]</code>	fresh blank node
<code>_:myBlank</code>	<code>_:myBlank</code>	labelled blank node
<code>myBlank</code>	<code>"myBlank"</code>	untyped literal
<code>ex:Ann</code>	<code>ex:Ann</code>	IRI
<code>http://other-example#Bob</code>	<code>&lt;http://other-example#Bob&gt;</code>	IRI
<code>Carl</code>	<code>"Carl"</code>	untyped literal
	<code>ottr:none</code>	IRI, special <i>none</i> value
<code>true</code>	<code>"true"^^xsd:boolean</code>	<code>xsd:boolean</code>
<code>True</code>	<code>"True"</code>	untyped literal
<code>1</code>	<code>"1"^^xsd:integer</code>	<code>xsd:integer</code>
<code>-1.2</code>	<code>"-1.2"^^xsd:decimal</code>	<code>xsd:decimal</code>

The table specifies the following template instances:

```
ottr:Triple(ex:Ann, foaf:knows, ex:Bob) .
ottr:Triple(ex:Bob, foaf:knows, ex:Carl) .
```

## 6.2 bOTTR: Batch Instantiation of OTTR Templates

*bOTTR* is an RDF vocabulary for specifying mappings between queries over the sources to given templates. *bOTTR* hence allows multiple data sources on different formats to be integrated via OTTR templates into a single RDF/OWL representation. The *bOTTR* vocabulary is specified by an OWL ontology that extends the *wOTTR* vocabulary (See Section 4.2), and provides terms for specifying sources, queries over these sources, and how to map the query results into instances of a specified OTTR template. The *bOTTR* specification is developed in GitLab,<sup>15</sup> and published at `ottr.xyz`<sup>16</sup> and Zenodo.<sup>17</sup>

The classes `ottr:InstanceMap`, `ottr:Source` and its subclasses, and `ottr:ArgumentMap`, which specify respectively the mappings, the source and the translation of source values to template instance arguments, are described below.

### 6.2.1 InstanceMap

The central notion of *bOTTR* is `ottr:InstanceMap`. An `ottr:InstanceMap` specifies a mapping between one `ottr:query` over a given `ottr:Source` and one `ottr:Template`. The result of applying the mapping is that each record in the query result set becomes an `ottr:Instance` of the specified `ottr:Template`. The `ottr:argumentMaps` specify how source values are translated to instance arguments.

An `ottr:InstanceMap` must specify a `ottr:template`, a `ottr:source`, a `ottr:query`, which must be a valid query over the specified `ottr:Source`, and optionally a list of `ottr:argumentMaps`. If set, the size of the argument map list must match the size of the query result records. These specify how query result values are translated to instance argument values. The result of processing a set of `ottr:InstanceMaps` is the set of instances resulting from each of the `ottr:InstanceMaps`.

<sup>15</sup> <https://gitlab.com/ottr/spec/bOTTR>

<sup>16</sup> <https://spec.ottr.xyz/bOTTR/0.1.2/>

<sup>17</sup> <https://zenodo.org/records/12607264>

### 6.2.2 Sources

A `ottr:Source` defines a source of data and how it can be accessed. The location of the source is specified with `ottr:sourceURL`, this can be either a URL or a file path. There are two main types of `ottr:Sources`: `ottr:StringSource` and `ottr:RDFSSource`. These source types again have subclasses. All query result values from a `ottr:StringSource` are assumed to be strings, and if no `ottr:argumentMaps` are set, then these source values are represented as untyped literals when transforming them into instance arguments. A `ottr:StringSource` makes no assumption on the type of its `ottr:query`, this must be specified by using a subclass of `ottr:StringSource`. For an `ottr:RDFSSource`, all values are assumed to be RDF resources. If no `ottr:argumentMaps` are set then the values are left as they are when used as instance arguments. An `ottr:RDFSSource` accepts only SPARQL queries.

There are four types of sources:

`ottr:RDFFileSource` is an `ottr:RDFSSource` which is specified by one or more RDF files using the `ottr:sourceURL` property.

`ottr:SPARQLEndpointSource` is an `ottr:RDFSSource` which is specified by a single SPARQL endpoint address using the `ottr:sourceURL` property.

`ottr:JDBCSource` is a `ottr:StringSource` which is specified using a single JDBC connection string using the `ottr:sourceURL` property. Additionally, a `ottr:JDBCSource` can specify a `ottr:jdbcDriver`, `ottr:username`, and `ottr:password` for connecting to the database, and set a `ottr:fetchSize` to indicate the number of query result rows retrieved on each fetch.

`ottr:H2Source` specifies a temporary H2 database source which is useful for loading and querying a CSV file using functionality supported by the H2 SQL query language. The path given to `CSVREAD` may use the special constant `@THIS_DIR@@` within the path to denote the directory of the `BOTTR` input file.

An example of each source can be seen in Example 66.

► **Example 66.** Below are examples of all the four sources. For the `ottr:H2Source`, we give a full instance map to illustrate the use of the `CSVREAD` function in the query.

```
[ ] a ottr:RDFFileSource ;
    ottr:sourceURL <http://example.com/file1.ttl>, <http://example.com/file2.ttl> .

[ ] a ottr:SPARQLEndpointSource ;
    ottr:sourceURL <http://example.com/sparql/> .

[ ] a ottr:JDBCSource ;
    ottr:sourceURL "jdbc:mysql://localhost/mydb" ;
    ottr:jdbcDriver "com.mysql.jdbc.Driver" ;
    ottr:username "Ann" ;
    ottr:password "password123" .

[ ] a ottr:InstanceMap ;
    ottr:source [ a ottr:H2Source ] ;
    ottr:query """
        SELECT 'First Name', 'Age', 'City', 'Homepage'
        FROM CSVREAD('@THIS_DIR@@/data/address.csv');
    """ ;
    ottr:template ex:Person .
```



### 6.2.3 ArgumentMap

An `ottr:ArgumentMap` specifies how source values are mapped to OTTR template instance arguments, i.e., RDF terms. In case no argument maps are specified, defaults apply from the choice of `ottr:Source` defined above. When applying an argument map to a source value, we sometimes refer to the **string value** of the source value. In case the source is a `ottr:StringSource`, then the string value of a source value  $x$  is  $x$ . In case the source is an `ottr:RDFSsource`, the string value is of an RDF resource  $x$  is:

- the lexical value of  $x$ , if  $x$  is a literal
- the IRI of  $x$ , if  $x$  is a IRI,
- the blank node label of  $x$ , if  $x$  is a blank node

An `ottr:ArgumentMap` is defined by the following properties, each specifies how values are transformed to RDF terms. The properties are applied in the order they are presented below.

`ottr:nullValue` Specifies the argument value to be used in case the source value is unspecified or NULL (as defined in the source's format). This value may be any RDF resource.

`ottr:labelledBlankPrefix` Selects which values to translate to labelled blank nodes. The default value is `_:`. If the string value of the source value starts with and is longer than the `ottr:labelledBlankPrefix`, then the argument value becomes a labelled blank node where the string value of the source value following the `ottr:labelledBlankPrefix` becomes the label. Example: if the `ottr:labelledBlankPrefix` is "ABC" and the source value is "ABCDEF", then a labelled blank node `_:DEF` is created.

`ottr:languageTag` Specifies the language tag of the argument value. If this value is set, then the source value becomes a language tagged literal (and automatically gets `ottr:type rdf:langString`) where the lexical value is the string value of the source value.

`ottr:listStart`, `ottr:listEnd`, `ottr:listSep` Specifies how to translate source values into lists using the string value of the source value. The default values are respectively: `( , )` and `,.` A value may represent a nested list of arbitrary depth. `ottr:listStart` and `ottr:listEnd` must be a one-character string that specifies respectively the start and end of a list. `ottr:listSep` is a string that specifies how the list elements are separated. The list element values are trimmed for white space. These properties may only be used if the `ottr:type` is a list type. Example: if the source value is `"(( a , b ), ( c , d ))"`, and the `ottr:type` is `(rdf:List rdf:List xsd:string)`, then the value becomes the RDF list `(( "a" "b" )("c" "d"))`.

`ottr:type` Specifies the `ottr:Type` of the argument value (using the wOTTR encoding of our type system). If the source is an `ottr:RDFSsource` and the type of the source value is compatible with the specified `ottr:type`, then the argument value is equal to the source value (the argument map leaves the value unchanged). If the value is not compatible or the source is a `ottr:StringSource`, then the string value of the source value is cast to the specified `ottr:type`. This may result in an error if the cast is not possible. This property may not be used if `ottr:labelledBlankPrefix` is used.

► **Example 67.** This example is an adaption of Example 66 to illustrate the functionality of argument maps.

```
[ ] a ottr:InstanceMap ;
  ottr:source [ a ottr:H2Source ] ;
  ottr:query """
    SELECT 'Homepage', 'First Name' || ' ' || 'Last Name' AS fullName, 'Bdate'
    FROM CSVREAD('people.csv');
  """ ;
  ottr:template ex:Person ;
```

## 5:42 The Reasonable Ontology Templates Framework

```
ottr:argumentMaps (  
  [ ottr:type ottr:IRI ]  
  [ ] ## empty Argument map  
  [ ottr:type xsd:date; ottr:nullValue ottr:none ]  
) .
```

Assuming that the contents of `people.csv` are

First Name	Last Name	Bdate	City	Homepage
Ann	Annsen	1990-12-01	Amsterdam	<a href="http://example.com/Ann">http://example.com/Ann</a>
Bob	Bobson	1987-03-23	Berlin	<a href="http://example.com/Bob">http://example.com/Bob</a>
Carl	Carlson	NULL	Cairo	<a href="http://example.com/Carl">http://example.com/Carl</a>

then the instance map will produce the following instances:

```
ex:Person( <http://example.com/Ann>, "Ann Annsen", "1990-12-01"^^xsd:date ) .  
ex:Person( <http://example.com/Bob>, "Bob Bobson", "1987-03-23"^^xsd:date ) .  
ex:Person( <http://example.com/Carl>, "Carl Carlson", none ) .
```

## 7 Implementations

There exist multiple implementations of the OTTR framework. One of these is a reference implementation that is developed and maintained by the OTTR project and which supports all the OTTR specifications. The other implementations are developed independently of the OTTR project and are motivated in part by the need to make the OTTR framework available in other platforms and programming languages than that of the reference implementation.

### 7.1 Lutra: The Reference Implementation of OTTR

*Lutra* is the reference implementation for the OTTR framework, written and actively maintained by the developers of OTTR. It is an open-source project under an LGPL licence, available at GitLab,<sup>18</sup> at a mirror repository at GitHub,<sup>19</sup> and with released sources also stored in Zenodo.<sup>20</sup> *Lutra* is written in Java and is built with Maven; the Maven artefacts are available through the Sonatype repository under the Java namespace `ottr.lutra`.<sup>21</sup> The codebase is developed following various established best practices: semantic versioning,<sup>22</sup> code style profile, static code analysis, unit testing, continuous integration and continuous deployment (CI/CD),<sup>23</sup> and the git-flow branching model.<sup>24</sup>

*Lutra* supports all the specifications presented in this paper:

- Expanding instances and templates
- Type checking templates and instances according to the type hierarchy of Table 1
- Reading and writing templates and instances in the `stOTTR` and `wOTTR` serialization formats
- Generating `docOTTR` documentation for template libraries
- Processing `tabOTTR` spreadsheets, supporting only Excel spreadsheet input
- Processing `bOTTR` specifications, supporting all types of sources mentioned in Section 6.2

<sup>18</sup><https://gitlab.com/ottr/lutra/lutra>

<sup>19</sup><https://github.com/rtto/lutra-mirror>

<sup>20</sup><https://zenodo.org/records/10954639>

<sup>21</sup><https://central.sonatype.com/search?q=ottr.lutra>

<sup>22</sup><https://semver.org/>

<sup>23</sup><https://about.gitlab.com/topics/ci-cd/>

<sup>24</sup><https://nvie.com/posts/a-successful-git-branching-model/>

```
Usage: lutra [-fhV] [--debugStackTrace] [--quiet] [--stdout] [-F=<fetchFormat>]
           [--haltOn=<haltOn>] [-I=<inputFormat>] [-L=<libraryFormat>]
           [-m=<mode>] [--messageLinePrefix=<linePrefix>] [-o=<out>]
           [-O=<outputFormat>] [-p=<prefixes>] [-e=<extensions>[,
           <extensions>...]]... [-E=<ignoreExtensions>[,
           <ignoreExtensions>...]]... [-l=<library>]... [<inputs>...]
```

■ **Figure 12** Lutra’s command line interface options and flags as reported by its `-help` option. (The help output also gives an explanation of the options and flags which is not shown here.)

There are three available modes of using Lutra: (1) as a command line interface (CLI) serviced by a Java executable JAR file, (2) as a Java API, or (3) as a web application. The available options and flags of CLI are listed in Figure 12. Possible modes of operation (set with option `-m`) are:

`expand` expands the input instances according to the specified template libraries and fetched templates, and writes the expansion result to the specified output format.

`format (re)`formats the input instances to the specified output format.

`expandLibrary` expands the specified template libraries.

`formatLibrary (re)`formats the specified template libraries to the specified output format.

`docttrLibrary` generates documentation pages for the specified template libraries.

`lint` checks the input instances or templates for errors.

`checkSyntax` runs a syntax check of the input, typically for use as an external service for editors.

The API provided by the Java project is documented using Javadoc.<sup>25</sup> The central class is `TemplateManager`<sup>26</sup> that is responsible for orchestrating the reading and writing of templates and instances. It contains methods for reading a library of templates from a file or folder, checking its correctness, reading instances from files, folders or mappings (both `bOTTR` and `tabOTTR`), expanding the instances and checking that the instances are correct with respect to a template library, translate instances or templates from one serialization to another, and more.

A web application variant of the CLI, called *WebLutra*, is available at <https://weblutra.ottr.xyz>. The intention of WebLutra is to make the OTTR framework available for simple use and experimentation without the need for installing and running software locally. WebLutra also serves the interactive examples that are part of the online primer for OTTR available at <https://primer.ottr.xyz>. It is implemented as a simple “wrapper” over the CLI interface exposing certain operations through a simple web form. WebLutra is packaged as a Docker image that is available from GitLab’s container registry.<sup>27</sup>

## 7.2 maplib: Support for Data Frame Mappings with OTTR

*maplib* [1] is a library written in Rust using Apache Arrow,<sup>28</sup> Polars<sup>29</sup> DataFrames and with Python bindings, which allows data frames, a popular data structure used in data analytics tools, to be mapped to RDF by way of OTTR templates.<sup>30</sup> By exploiting libraries and formats built for

<sup>25</sup> <https://javadoc.io/doc/xyz.ottr.lutra>

<sup>26</sup> <https://javadoc.io/doc/xyz.ottr.lutra/lutra-core/latest/xyz/ottr/lutra/TemplateManager.html>

<sup>27</sup> [https://gitlab.com/ottr/lutra/lutra/container\\_registry/1986127](https://gitlab.com/ottr/lutra/lutra/container_registry/1986127)

<sup>28</sup> <https://arrow.apache.org/>

<sup>29</sup> <https://pola.rs/>

<sup>30</sup> <https://pypi.org/project/maplib/>

## 5:44 The Reasonable Ontology Templates Framework

data processing, `maplib` is able to perform OTTR instance expansion with high performance. It also brings OTTR closer to use for data analytics and in platforms such as Jupyter.<sup>31</sup> From the documentation the library appears to only support templates in `stOTTR` syntax and it is not clear if the library supports all language features of OTTR. `maplib` is open source<sup>32</sup> and in active development.

### 7.3 OTTR Extension: Semantic MediaWiki Extension

*OTTR Extension*<sup>33</sup> is an extension for Semantic MediaWiki (SMW) [30] which enables some of OTTR's functionality within SMW [5]. The OTTR Extension allows templates and instances in `stOTTR` format to be entered directly into SMW pages. Taking an OTTR template as input, the OTTR Extension can generate an input web form in SMW that matches the signature of the template and with which instances of the template can be created by filling in the form. These instances expand to triples, as per its template definition, that are available in the page of the created template instance. The latest release is dated March 2023.

### 7.4 pyOTTR: Python Packages

There are two packages that implement OTTR functionality for Python, one developed by GitHub user Callidon<sup>34</sup> and one developed by GitHub user `michalporeba`.<sup>35</sup> Both packages are called *pyOTTR*, although they appear as two separate projects.

Callidon's `pyOTTR` package supports reading templates and instances in `stOTTR` format and expanding instances to RDF. The package is limited to supporting only the `stOTTR` format and does not support the complete functionality of the OTTR language, in particular, list expansion modes are listed as in development. However, the project appears abandoned as the latest code update was five years ago.

`michalporeba`'s `pyOTTR` package appears to support reading templates in `stOTTR` format, and expanding instance data read from CSV files to RDF. Whether the package supports the full OTTR language is not clear from the documentation. The package appears to be in a state of early and active development.

### 7.5 emacs-ottr-toolkit

The `emacs-ottr-toolkit`<sup>36</sup> is a collection of Emacs scripts to facilitate OTTR operations in GNU Emacs org-mode.<sup>37</sup> It uses existing Emacs extensions like `Flycheck`<sup>38</sup> and the Lutra CLI executable (Section 7.1) to offer shortcut commands and code snippet functionality that simplifies operations such as writing templates, syntax checking, and template instantiation from tables and query result sets.

---

<sup>31</sup><https://jupyter.org/>

<sup>32</sup><https://github.com/DataTreehouse/maplib>

<sup>33</sup><https://www.mediawiki.org/wiki/Extension:OtrrParser>

<sup>34</sup><https://github.com/Callidon/pyOTTR>

<sup>35</sup><https://github.com/michalporeba/pyOTTR>

<sup>36</sup><https://ottr.xyz/event/2021-06-18-user-forum/ottr-toolkit-20210618.html>

<sup>37</sup><https://orgmode.org/>

<sup>38</sup><https://www.flycheck.org>

## 8 Uses

The OTTR project has nurtured a healthy collaborative environment with its users particularly through the organization of several *OTTR user forums*<sup>39</sup> where OTTR developers and users have met to exchange experiences and future directions. These events were hosted by SIRIUS, a Norwegian Centre for Research-driven Innovation to address the problems of scalable data access in the oil & gas industry, and therefore attracted participants specifically from this and supporting industries. The findings from these forums are that industrial users of OTTR share similar requirements to the construction and management of knowledge graphs that fit well with the features provided by OTTR. Summarized they need support for:

- Large-scale knowledge graph development, since ontologies can reach sizes of 100,000's of classes.
- Uniform modelling: this is necessary to ensure that access to and use of the constructed knowledge base can be performed in a uniform and predictive manner, and to reduce management and maintenance costs.
- Transformation of data sourced from different formats and schemas: the input data to create the knowledge graph that typically comes from different domain standard representations that represent similar types of data differently needs harmonization.
- Subject-matter expert (SME) involvement: SMEs need to partake in the design of the information in the knowledge graph and be able to verify its quality.
- Collaborative development environments: Development is often performed in teams, with additional stakeholders that need to interact with the results in different ways.
- Automated mechanisms for quality assessment and verification are required due to the size and complexity of the modelling and representation task.

The current tools on offer for large-scale ontology development are few. The participants of the OTTR user forums report that editors like Protégé do not meet their requirements since this mode modelling with a desktop/client application is unsuited for uniform and collaborative modelling at the scale required. Also, the fact that these tools typically operate on low-level constructs, such as OWL axioms, makes them difficult for SMEs to understand and use.

In the following, we report from selected prominent industrial and academic uses of the OTTR framework to serve as examples of its utility.

### 8.1 Grundfos' Industrial Ontology Engineering Platform

At Grundfos, Brynildsen et al. have built what they call the *Industrial Ontology Engineering Platform (IOEP)*, where OTTR is an integral part, “to support domain experts in using and implementing ontologies while reducing the workload for ontology engineering specialists” [6]. The tool provides web-based tabular input user interfaces that are aligned with signatures of OTTR templates prepared by ontology experts. Subject-matter experts build ontologies by populating the user interface, supported by helpful editor features such as auto-generated IRIs and metadata, label lookups and searches. The tabular data is transformed using the OTTR templates to an ontology with a build service that relies on Lutra (See Section 7.1). The deployment of the tool is reported as successful in that improves ontology engineering scalability: the reuse of OTTR templates across different ontology development projects reduces ontology design efforts and

---

<sup>39</sup> <https://www.ottr.xyz/event/2021-01-28-user-forum/>,  
<https://www.ottr.xyz/event/2021-06-18-user-forum/>,  
<https://www.ottr.xyz/event/2022-05-11-user-forum/>

hence the workload for ontology engineering practitioners. Also, OTTR templates have helped to position SMEs as the primary owners and contributors of new ontological models by hiding the complexity of upper-level ontology away from the SME [6].

## 8.2 Bosch’s Ontology-Enhanced Machine Learning System

A similar approach to that of Grundfos’ is followed at Bosch [55]. In their *Ontology-Enhanced Machine Learning (SemML)* system, OTTR templates are part of the *Ontology extender* component “that allows domain experts to describe domains in terms of an upper-level ontology by filling in templates. Data scientists then also use templates to annotate domain terms with quality-related information” [55]. Domain experts describe the domain by filling in simple forms that are generated to match the signature of OTTR templates. The forms produce template instances that are expanded to OWL axioms. “Templates guarantee uniformity of the updates and the consistency of the updated ontology, as well as the relative simplicity of the ontology extension process.” [55] The ontology extender component is evaluated in a user study giving positive results with respect to correctness of use.

## 8.3 CapGemini and Norwegian Maritime Authority: Modelling Regulatory Requirements as SHACL Shapes

CapGemini and Norwegian Maritime Authority use the OTTR framework as part of a tool-chain where regulatory requirements are expressed as SHACL shapes [14]. Requirements are extracted from text using natural language processing and named entity recognition techniques. The resulting data is then transformed to RDF and SHACL using an OTTR template library for SHACL shapes.

## 8.4 DiProMag: Ontology of Magnetocaloric Materials

The goal of the DiProMag project is the digitization of a process chain for the production, characterization and prototypical application of magnetocaloric alloys.<sup>40</sup> To support this work, Blum et al. [5] have created OTTR Extension (see Section 7.3) and a template library to allow domain experts to build an ontology of magnetocaloric materials by filling in generated forms.

## 8.5 Aibel: Identifying Redundancies in a Large Template Library

Aibel, a global engineering company, has developed the “Material Master Data ontology”, a comprehensive representation of engineering requirements and specifications that contains in the range of 100,000 classes across a hierarchy of OWL ontologies.<sup>41</sup> Aibel applies automated reasoning and querying to support the selection of designs, and matching designs with products [49]. The ontology is predominately generated from numerous spreadsheets, prepared by ontology experts and populated by domain experts. A custom-built pipeline translates the spreadsheets into template instances, and then into ontologies following template specifications

To evaluate the feasibility of the OTTR approach to ontology engineering and maintenance, the complete set of spreadsheet specifications was translated into a library of 1185 OTTR templates, some of which contain more than 30 parameters. Analysis of the generated template

<sup>40</sup><https://www.dipromag.de/>

<sup>41</sup>An obfuscated version of the ontology is published at <https://github.com/Sirius-sfi/aibel-mmd-ontology> with the “intent of providing researchers and software developers with free access to a large-scale real industrial ontology”.

library revealed many redundancies, largely attributable to the limited expressive power of the spreadsheet template language. Using a semi-automated approach, we were able to identify uncaptured patterns representing relevant domain conceptualizations, dramatically reducing the number of redundancies [50]. This use case shows that the expressive power of OTTR templates can bring substantial benefits, in particular for industrial domains where there is a high degree of regularity in the patterns used. It also demonstrates the usefulness of maintenance techniques of template libraries.

## 9 Related Work

In this section, we introduce work that is related to the OTTR framework when considering it as a *framework for pattern-based ontology engineering supported by a shared library of reusable patterns and tools for practical pattern instantiation for large-scale knowledge graph construction and maintenance*. This makes for a very wide scope of related work, so our discussion will necessarily be limited to a selection of illustrative works. Yet, there are to our knowledge few works that fall into the same description as the OTTR framework, so our presentation will cover the most relevant ones.

Pattern-based ontology engineering is closely connected to ontology design patterns (ODPs) [11, 15]. Similar to software design patterns, ODP is a modelling solution to solve a recurrent ontology design problem that is recorded and shared to simplify ontology development and use. ODPs serve the purpose of alleviating some of the difficulties involved with creating ontologies by offering reusable, best-practice building blocks and structures for ontology construction, commonly implemented and published as small OWL ontologies. Methods for combining and instantiating ODPs are described [43, 13], and a methodology for building ontologies using patterns exists [3]. ODPs are documented and published in open repositories, such as the *OntologyDesignPatterns.org* portal,<sup>42</sup> *MODL* [47], and the *Manchester Catalogue*.<sup>43</sup> There are tools, such as *XDP* [12] and *CoModIDE* [46], which are built on top of *WebProtégé* [57], as convenient tools for modelling with and instantiating ODPs. However, while ODPs are often presented as “practical building blocks” [43], we argue that ODPs in their current form, i.e., as found at *OntologyDesignPatterns.org*, featuring a graphical representation, a description and a “reusable OWL building block”, are not practical enough, especially for the development of large ontologies. Using and adapting ODPs to a particular modelling task will normally require considerable manual work and ODPs do not describe a precise manner in which instances of the pattern may be created at scale. Frameworks that offer this functionality, in addition to the OTTR framework, are *GDOL* [29], *Dead simple OWL design patterns (DOS-DPs)* [39] and the *Ontology Pre-Processor Language OPPL* [20].

*Generic DOL (GDOL)* [29] is an extension of the *Distributed Ontology, Modelling, and Specification Language (DOL)* that supports a parametrization mechanism for ontologies. *GDOL/DOL* is a metalanguage for combining theories from a wide range of logics under one formalism while supporting pattern definition, instantiation, and nesting. Thus, it provides a broad formalism for defining ontology templates along similar lines as OTTR. The syntax for OWL in *GDOL* is the *Manchester OWL Syntax*, extended by parameters. *DOL* is supported by *Ontohub* [7] (an online ontology and specification repository) and *Hets* [35] (parsing and inference back-end of *DOL*). The difference between OTTR and *GDOL* is that the foundations of *GDOL* build on the more abstract and powerful concept of generics, rather than the simpler concept of macros. Also, *GDOL* is based on the comprehensive framework of *DOL* which supports expressions in different

---

<sup>42</sup><http://ontologydesignpatterns.org>

<sup>43</sup><http://odps.sourceforge.net/odp/html/>

logics, while OTTR is developed specifically to be used for semantic web technologies, e.g., with a type system developed for semantic web data. GDOL is more powerful than OTTR and is also therefore arguably more complex in use for the tasks that OTTR is developed for.

*Dead Simple OWL Design Patterns (DOS-DP)* [39] provides a simple pattern mechanism where patterns are expressed in template pattern files following YAML syntax and where property values refer to named variables. Pattern instances are created by providing filler values that correspond to the template pattern's named variables. Technically, instantiation is simply performed by a `printf` function that replaces the variable placeholder with the filler value. DOS-DP is designed to be simple and easy to use and does therefore not support inheritance or composition of patterns. The DOS-DP framework appears to be in active use; the Mondo Disease Ontology<sup>44</sup> is a user of DOS-DP.

*The Ontology Pre-Processor Language (OPPL)* [20] was originally developed as a language for manipulating OWL ontologies. Hence, it supports functions for adding and removing patterns of OWL axioms to/from an ontology. It relies heavily on its foundations in OWL DL and as such can only be used in the context of OWL ontologies. OPPL patterns are parameterized expressions which can be nested and can specify pattern instances and patterns directly in OWL ontologies. The syntax of OPPL is however distinct from that of RDF and OWL and requires separate tools for viewing and editing such patterns. A Protégé plugin for version 4.x exists, in addition to a tool called *Populous* [22] which allows OPPL patterns to be instantiated via spreadsheets. By allowing patterns to return a single element (e.g., a class) OPPL supports a rather restricted form of pattern nesting as compared to OTTR. The application focus of OPPL is somewhat different from that of OTTR: OPPL patterns are intended to be fully expanded once they are used in the ontology. In contrast, we believe that OTTR template instances can appear as instances lifted or lowered to the abstraction level suited for the given user. For instance, an ontology expert may prefer to examine an ontology formatted as a set of OWL axiom OTTR templates, while domain experts might prefer to see only the user-facing template instances. Additionally, OPPL patterns are limited to OWL expressions in Manchester syntax [17], while OTTR supports abstractions over any underlying language, although it is designed specifically for RDF. OPPL appears to no longer be actively maintained; the last release of the project was in 2013<sup>45</sup> the last update of its Git repository<sup>46</sup> was in 2020.

There are many tools for generating RDF and OWL knowledge bases from different sources using various mechanisms and mapping languages; here we mention a few that have served as inspiration for the OTTR framework. *SPARQL-generate* [32] is a template-based language and an extension of SPARQL with which one can generate RDF streams or text streams from RDF datasets and document streams in arbitrary formats. *Tawny OWL* [33] introduces a Manchester-like syntax for writing ontology axioms from within the programming language Clojure, and allows abstractions and extensions to be written as normal Clojure code alongside the ontology. Thus, the process of constructing an ontology is transformed into a form of programming, where existing tools for program development, such as versioning and testing frameworks can be used. The *M<sup>2</sup> mapping language* [38] allows spreadsheet references to be used in ontology axiom patterns. Finally, *XLWrap* [31] defines a mapping language to convert spreadsheets into RDF.

<sup>44</sup><https://github.com/monarch-initiative/mondo>

<sup>45</sup><https://oppl2.sourceforge.net/>

<sup>46</sup><https://github.com/owlcs/OPPL2>



## 10 Future Work

In this section, we list interesting ideas for extensions and further developments of the OTTR framework.

**Support for call-by-name** We wish to support *named parameters* or *call-by-name* in OTTR. This feature would increase the readability of template instances and simplify template instantiation slightly by allowing arguments to be given in arbitrary order and missing arguments to be omitted. Instances of the `o-p:NamedPizza` template using named parameters could then look like the following:

```
o-p:NamedPizza(
  pizza = ex:Margherita,
  country = ex:Italy,
  toppings = (ex:Mozzarella, ex:Tomato)
) .
o-p:NamedPizza(
  toppings = (ex:Potato, ex:Rosemary),
  pizza = ex:PotatoPizza
).
```

**Support for tuple or struct types** In Example 44, we see that when one needs to create a template that creates multiple instances describing objects with multiple attributes (such as the nuclear family of persons with IRIs and names), we need to zip multiple lists, each list with values for one attribute. It would be preferable to rather group the attributes of each object together in one struct or tuple, e.g., `(ex:bob, 'Bob Green', none)` with type `Tuple<owl:NamedIndividual, xsd:string, xsd:date>`, combined with functionality for extracting the tuple's elements. The `ex:NuclearFamily` template could then simply accept two lists of such tuples (instead of four lists, as defined above). Thus, extending OTTR with tuple/struct terms and types would make expressing such complex templates more convenient.

**Improved dependency management** The organization of templates into collections is currently done informally by placing closely related templates under the same namespace and in the same file or folder. We wish to strengthen the management of template libraries by formally characterizing the notion of template modules and allowing for explicit dependencies between modules to be expressed. The aim is to support systems to handle such template modules much like software project management systems like Maven<sup>47</sup> do and improve the robustness and stability of template instance expansion.

**Customizable expansions** A main requirement in the design of the OTTR framework is that an instance should be considered as semantically equivalent to its expansion. An enhancement to the framework would be to support different expansions of the same instance while still supporting the above requirement. A motivation for this enhancement is for instance to experiment with different representations of the same statement, or to support multiple equivalent representations of the same statement. Examples of this could be to expand instances that represent an ontology into different OWL profiles [36], or to expand a knowledge graph not only into RDF, but also other knowledge graph formats [16]. A way to support this is to introduce a more complex notion of a dataset that includes somehow instructions on what template libraries and base templates to use for expansion, and hence possibly allow a template signature to be associated with multiple different template bodies.

<sup>47</sup><https://maven.apache.org/>

**Test suite and benchmark** The availability of multiple implementations of the OTTR framework (Section 7) raises the question of what features of the OTTR framework they support and their relative performance. We wish to establish a test suite to be able to measure the conformance of the implementations against the specifications of the OTTR framework, and a benchmark to measure their performance.

**Term manipulation support** The OTTR framework does not naively provide a facility for manipulating terms, such as generating IRIs by concatenating strings, or mathematical calculations. Some of this functionality is covered by the use of the query languages within bOTTR, but this is arguably not optimal. FROG [21] is a declarative term manipulation language introducing pure functions that can be applied to terms within template definitions, and leverages the OTTR type system and extends it by introducing type generics. Updating the OTTR framework to support FROG remains future work.

**OTTR as query language** In this paper we have seen OTTR templates being used to transform complex statements in the form of template instances into expressions over a different data format such as RDF. An interesting approach is to also allow OTTR templates to be used “in reverse” as queries and use them to extract and assemble lower-level statements to more statements at a higher level of abstraction. Using a well-designed template library to both generate and extract information would be a clear benefit in terms of usability and maintainability. OTTR as a query language has been characterized under the name Reverse OTTR [53] and a prototypical implementation exists, yet proper integration into the OTTR framework remains.

**OTTR vs. Ontology-based data access (OBDA)** bOTTR provides a mapping mechanism from databases to ontologies via templates, similar to what is known as ontology-based data access (OBDA) [42]. The similarities and differences between bOTTR and OBDA warrant investigation, both from a materialization and a querying perspective. Furthermore, when bOTTR is used to construct a large knowledge graph updates to the mapped sources may change, thus requiring a corresponding update to the knowledge graph, preferably without needing to reconstruct the entire graph. This is not currently supported by the OTTR framework, but preliminary work has been done by Eckhoff and Zahl [8], and incorporating this into OTTR would be desirable.

**Template authoring support** Developing a template library is currently manual work using text editors. Improved tool support for the existing methodology and library maintenance techniques, as well as a graphical language and graphical user interface or integrated development environment (IDE) for template authoring, would arguably increase the efficiency and quality of creating and managing template libraries and lower the bar for new users.

**Static analysis of template libraries** The OTTR framework supports static analysis of template libraries in the form checking type correctness. Such analysis could be extended to also consider the semantics of the vocabularies used, and for instance, identify templates or combinations of templates that can result in inconsistent OWL ontologies or unsatisfiable concepts.

**Template bootstrapping** To advance the development of template libraries, pattern recognition and discovery methods could be applied to identify patterns in existing knowledge bases and databases and to capture these as well-designed template libraries.

**User evaluation of OTTR** Many of the benefits we claim about OTTR are justified through accepted benefits of user-defined abstraction from software engineering and information modeling, yet no formal user evaluation exists. Such an evaluation could examine what parts of OTTR are actually used – and by whom, what the learning curve of OTTR is, and identify possible new enhancements to the framework.

**Complexity analysis of OTTR** Snilsberg et al. [54] have developed a mathematical formalization of a subset of the OTTR language, and give a preliminary report on the characterization of the theoretical size of instance expansions and decision problems associated with the language

and its fragments. A rigorous analysis of the current expressivity of OTTR, versus desired or optimal expressivity, and comparison with other formalism is a clear candidate for future work, providing direction on what features one might include or omit.

**What is a good template mechanism?** The OTTR framework, GDOL, DOS-DP and OPPL provide different and partially overlapping functionality. There is also a proposal for powerful extensions to the OTTR language, called *Generators* and *GBoxes* [9], that are rules formulated over the OTTR templates. Kindermann et al. [24] identify characteristics that are deemed necessary for an ontology template mechanism which OTTR implements. Kindermann et al. [25] also investigate the use of ontology design patterns in practice. However, more investigation into the similarities and differences between the available pattern frameworks and experience into what is practically useful is needed to identify the correct balance between expressivity, complexity and usability, and to formulate metrics to characterize “good” template mechanisms and libraries.

## 11 Conclusion

This paper has given a complete overview of the current state of the OTTR framework, detailing its formal foundation, RDF and OWL adaptations, different serializations, template library management and support, mapping languages, multiple (independent) implementations, and real-world use and impact. We have also presented related and future work. We believe this paper illustrates the maturity of OTTR as a framework, a framework that has moved beyond the phase of prototypes and purely academic study, into a useful and practical framework that has a strong theoretical foundation and is based on sound engineering principles that are suitable for real-world, large-scale, maintainable ontology and knowledge graph construction.

---

## References

- Magnus Bakken. maplib: Interactive, literal RDF model mapping for industry. *IEEE Access*, 11:39990–40005, 2023. doi:10.1109/ACCESS.2023.3269093.
- David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. RDF 1.1 turtle: Terse RDF triple language. Technical report, W3C, 2014. URL: <https://www.w3.org/TR/turtle/>.
- Eva Blomqvist, Karl Hammar, and Valentina Presutti. Engineering ontologies with patterns - the extreme design methodology. In Pascal Hitzler, Aldo Gangemi, Krzysztof Janowicz, Adila Krisnadhi, and Valentina Presutti, editors, *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, volume 25 of *Studies on the Semantic Web*, pages 23–50. IOS Press, 2016. doi:10.3233/978-1-61499-676-7-23.
- Eva Blomqvist, Pascal Hitzler, Krzysztof Janowicz, Adila Krisnadhi, Tom Narock, and Monika Solanki. Considerations regarding ontology design patterns. *Semantic Web*, 7(1):1–7, 2016. doi:10.3233/SW-150202.
- Moritz Blum, Basil Ell, and Philipp Cimiano. Insights from an OTTR-centric ontology engineering methodology. In Raghava Mutharaju, Agnieszka Ławrynowicz, Primit Bhattacharyya, Eva Blomqvist, Luigi Asprino, and Gunjan Singh, editors, *Proceedings of the 14th Workshop on Ontology Design and Patterns (WOP 2023)*, volume 3636. CEUR-WS.org, 2023. URL: <https://ceur-ws.org/Vol-3636/paper8.pdf>.
- Mikkel Haggren Brynildsen, Claus Jakobsen, Nicolaj Abildgaard, and Caitlin Woods. Building an Industrial Ontology Engineering Platform. In *Posters, Demos, and Industry Tracks at ISWC 2023*. CEUR-WS.org, 2023. URL: [https://ceur-ws.org/Vol-3632/ISWC2023\\_paper\\_502.pdf](https://ceur-ws.org/Vol-3632/ISWC2023_paper_502.pdf).
- Mihai Codescu, Eugen Kuksa, Oliver Kutz, Till Mossakowski, and Fabian Neuhaus. Ontohub: A semantic repository engine for heterogeneous ontologies. *Appl. Ontology*, 12(3-4):275–298, 2017. doi:10.3233/A0-170190.
- Magnus Wiik Eckhoff and Preben Zahl. Efficient update of OTTR-constructed triplestores. Master's thesis, University of Oslo, 2023.
- Henrik Forssell, Christian Kindermann, Daniel P. Lupp, Uli Sattler, and Evgenij Thorstensen. Generating ontologies from templates: A rule-based approach for capturing regularity. *CoRR*, abs/1809.10436, 2018. arXiv:1809.10436, doi:10.48550/arXiv.1809.10436.
- Henrik Forssell, Daniel P. Lupp, Martin G. Skjæveland, and Evgenij Thorstensen. Reasonable macros for ontology construction and maintenance. In Alessandro Artale, Birte Glimm, and Roman Kontchakov, editors, *Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017*, volume 1879

- of *CEUR Workshop Proceedings*, 2017. URL: <https://ceur-ws.org/Vol-1879/paper34.pdf>.
- 11 Aldo Gangemi and Valentina Presutti. *Ontology Design Patterns*, pages 221–243. Springer, 2009. doi:10.1007/978-3-540-92673-3\_10.
  - 12 Karl Hammar. Ontology design patterns in web-protege. In Serena Villata, Jeff Z. Pan, and Mauro Dragoni, editors, *Proceedings of the ISWC 2015 Posters & Demonstrations Track, 2015*, volume 1486 of *CEUR Workshop Proceedings*, 2015. URL: [https://ceur-ws.org/Vol-1486/paper\\_50.pdf](https://ceur-ws.org/Vol-1486/paper_50.pdf).
  - 13 Karl Hammar and Valentina Presutti. *Template-Based Content ODP Instantiation*, volume 32 of *Studies on the Semantic Web*, pages 1–13. IOS Press, 2016. doi:10.3233/978-1-61499-826-6-1.
  - 14 Veronika Heimsbakk and Kristian Torkelsen. Using the shapes constraint language for modelling regulatory requirements, 2023. doi:10.48550/arXiv.2309.02723.
  - 15 Pascal Hitzler, Aldo Gangemi, Krzysztof Janowicz, Adila Krisnadhi, and Valentina Presutti, editors. *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, volume 25 of *Studies on the Semantic Web*. IOS Press, 2016.
  - 16 Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d’Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4):71:1–71:37, 2022. doi:10.1145/3447772.
  - 17 Matthew Horridge and Peter F. Patel-Schneider. OWL 2 Web Ontology Language Manchester Syntax. W3c working group note, W3C, 2012. URL: <https://www.w3.org/TR/owl2-manchester-syntax/>.
  - 18 Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosf, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3c member submission, W3C, 2004. URL: <http://www.w3.org/Submission/SWRL/>.
  - 19 Bernadette Hyland, Ghislain Ateazing, and Boris Villazón-Terrazas. Best practices for publishing linked data. W3c working group note, W3C, 2014. URL: <https://www.w3.org/TR/ld-bp/>.
  - 20 Luigi Iannone, Alan L. Rector, and Robert Stevens. Embedding Knowledge Patterns into OWL. In *ESWC*, pages 218–232, 2009. doi:10.1007/978-3-642-02121-3\_19.
  - 21 Marlen Jarholt. Frog: Functions for ontologies—an extension for the OTTR-framework. Master’s thesis, University of Oslo, 2022.
  - 22 Simon Jupp et al. Populous: a tool for building OWL ontologies from templates. *BMC Bioinformatics*, 13(S-1):S5, 2012. doi:10.1186/1471-2105-13-S1-S5.
  - 23 C. M. Keet. *An Introduction to Ontology Engineering*. College Publications, 2018.
  - 24 Christian Kindermann, Daniel P. Lupp, Martin G. Skjæveland, and Leif Harald Karlsen. Formal relations over ontology patterns in templating frameworks. In Eva Blomqvist, Torsten Hahmann, Karl Hammar, Pascal Hitzler, Rinke Hoekstra, Raghava Mutharaju, María Poveda-Villalón, Cogan Shimizu, Martin G. Skjæveland, Monika Solanki, Vojtech Svátek, and Lu Zhou, editors, *Advances in Pattern-Based Ontology Engineering, extended versions of the papers published at the Workshop on Ontology Design and Patterns (WOP)*, volume 51 of *Studies on the Semantic Web*, pages 120–133. IOS Press, 2021. doi:10.3233/SSW210010.
  - 25 Christian Kindermann, Bijan Parsia, and Uli Sattler. Detecting influences of ontology design patterns in biomedical ontologies. In Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtech Svátek, Isabel F. Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon, editors, *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part I*, volume 11778 of *Lecture Notes in Computer Science*, pages 311–328. Springer, 2019. doi:10.1007/978-3-030-30793-6\_18.
  - 26 Johan W. Klüwer, Martin G. Skjæveland, and Magne Valen-Sendstad. ISO 15926 templates and the Semantic Web. Technical report, W3C, 2008. W3C Workshop on Semantic Web in Oil & Gas Industry.
  - 27 Graham Klyne, Jeremy J. Carroll, and Brian McBride. RDF 1.1 Concepts and Abstract Syntax. W3c recommendation, W3C, 2014. URL: <https://www.w3.org/TR/rdf11-concepts/>.
  - 28 Holger Knublauch and Dimitris Kontokostas. Shapes Constraint Language (SHACL). W3c recommendation, W3C, 2017. URL: <https://www.w3.org/TR/shacl/>.
  - 29 Bernd Krieg-Brückner and Till Mossakowski. Generic ontologies and generic ontology design patterns. In Eva Blomqvist, Óscar Corcho, Matthew Horridge, David Carral, and Rinke Hoekstra, editors, *Proceedings of the 8th Workshop on Ontology Design and Patterns (WOP), 2017*, volume 2043 of *CEUR Workshop Proceedings*, 2017. URL: <https://ceur-ws.org/Vol-2043/paper-02.pdf>.
  - 30 Markus Krötzsch, Denny Vrandečić, and Max Völkel. Semantic mediawiki. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, volume 4273 of *Lecture Notes in Computer Science*, pages 935–942. Springer, 2006. doi:10.1007/11926078\_68.
  - 31 Andreas Langegger and Wolfram Wöb. XI-wrap - querying and integrating arbitrary spreadsheets with SPARQL. In Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, volume 5823 of *Lecture Notes in Computer Science*, pages 359–374. Springer, 2009. doi:10.1007/978-3-642-04930-9\_23.


- 32 Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. A SPARQL extension for generating RDF from heterogeneous formats. In Eva Blomqvist, Diana Maynard, Aldo Gangemi, Rinke Hoekstra, Pascal Hitzler, and Olaf Hartig, editors, *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part I*, volume 10249 of *Lecture Notes in Computer Science*, pages 35–50, Portoroz, Slovenia, May 2017. doi:10.1007/978-3-319-58068-5\_3.
- 33 Phillip Lord. The semantic web takes wing: Programming ontologies with tawny-owl. In Mariano Rodriguez-Muro, Simon Jupp, and Kavitha Srinivas, editors, *Proceedings of the 10th International Workshop on OWL: Experiences and Directions (OWLED), 2013*, volume 1080 of *CEUR Workshop Proceedings*, 2013. URL: [https://ceur-ws.org/Vol-1080/owled2013\\_16.pdf](https://ceur-ws.org/Vol-1080/owled2013_16.pdf).
- 34 Daniel P. Lupp, Melinda Hodkiewicz, and Martin G. Skjæveland. Template libraries for industrial asset maintenance: A methodology for scalable and maintainable ontologies. In Thorsten Liebig, Achille Fokoue, and Zhe Wu, editors, *Proceedings of the 12th International Workshop on Scalable Semantic Web Knowledge Base Systems co-located with 19th International Semantic Web Conference (ISWC 2020), Athens, Greece, November 2, 2020*, volume 2757 of *CEUR Workshop Proceedings*, pages 49–64. CEUR-WS.org, 2020. URL: [https://ceur-ws.org/Vol-2757/SSWS2020\\_paper4.pdf](https://ceur-ws.org/Vol-2757/SSWS2020_paper4.pdf).
- 35 Till Mossakowski, Christian Maeder, and Klaus Lüttich. The heterogeneous tool set, hets. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522. Springer, 2007. doi:10.1007/978-3-540-71209-1\_40.
- 36 Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language Profiles (Second Edition). W3c recommendation, W3C, 2012. URL: <http://www.w3.org/TR/owl-profiles>.
- 37 Mark A. Musen. The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12, 2015. doi:10.1145/2757001.2757003.
- 38 Martin J. O’Connor, Christian Halaschek-Wiener, and Mark A. Musen. M<sup>2</sup>: A language for mapping spreadsheets to OWL. In Evren Sirin and Kendall Clark, editors, *Proceedings of the 7th International Workshop on OWL: Experiences and Directions (OWLED), 2010*, volume 614 of *CEUR Workshop Proceedings*, 2010. URL: [https://ceur-ws.org/Vol-614/owled2010\\_submission\\_17.pdf](https://ceur-ws.org/Vol-614/owled2010_submission_17.pdf).
- 39 David Osumi-Sutherland, Mélanie Courtot, James P. Balhoff, and Christopher J. Mungall. Dead simple OWL design patterns. *J. Biomed. Semant.*, 8(1):18:1–18:7, 2017. doi:10.1186/s13326-017-0126-0.
- 40 Bijan Parsia, Peter Patel-Schneider, and Boris Motik. Owl 2 web ontology language structural specification and functional-style syntax. W3c recommendation, W3C, 2012. URL: <https://www.w3.org/TR/owl2-syntax/>.
- 41 Peter F. Patel-Schneider and Boris Motik. OWL 2 Web Ontology Language Mapping to RDF Graphs. W3c recommendation, W3C, 2012. URL: <https://www.w3.org/TR/owl-mapping-to-rdf/>.
- 42 Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008. doi:10.1007/978-3-540-77688-8\_5.
- 43 Valentina Presutti and Aldo Gangemi. Content ontology design patterns as practical building blocks for web ontologies. In Qing Li, Stefano Spaccapetra, Eric S. K. Yu, and Antoni Olivé, editors, *Conceptual Modeling - ER 2008, 27th International Conference on Conceptual Modeling, Barcelona, Spain, October 20-24, 2008. Proceedings*, volume 5231 of *LNCIS*, pages 128–141. Springer, 2008. doi:10.1007/978-3-540-87877-3\_11.
- 44 Leo Sauermaann and Richard Cyganiak. Cool uris for the semantic web. Technical report, W3C, 2008. URL: <http://www.w3.org/TR/cooluris/>.
- 45 Ralph Schäfermeier, Adrian Paschke, and Heinrich Herre. Ontology design patterns for representing context in ontologies using aspect orientation. In Krzysztof Janowicz, Adila Alfa Krisnadhi, María Poveda Villalón, Karl Hammar, and Cogan Shimizu, editors, *Proceedings of the 10th Workshop on Ontology Design and Patterns (WOP 2019) co-located with 18th International Semantic Web Conference (ISWC 2019), Auckland, New Zealand, October 27, 2019*, volume 2459 of *CEUR Workshop Proceedings*, pages 32–46. CEUR-WS.org, 2019. URL: <https://ceur-ws.org/Vol-2459/paper3.pdf>.
- 46 Cogan Shimizu and Karl Hammar. Comodide - the comprehensive modular ontology engineering IDE. In Mari Carmen Suárez-Figueroa, Gong Cheng, Anna Lisa Gentile, Christophe Guéret, C. Maria Keet, and Abraham Bernstein, editors, *Proceedings of the ISWC 2019 Satellite Tracks (Posters & Demonstrations, Industry, and Outrageous Ideas) co-located with 18th International Semantic Web Conference (ISWC 2019), Auckland, New Zealand, October 26-30, 2019*, volume 2456 of *CEUR Workshop Proceedings*, pages 249–252. CEUR-WS.org, 2019. URL: <https://ceur-ws.org/Vol-2456/paper65.pdf>.
- 47 Cogan Shimizu, Quinn Hirt, and Pascal Hitzler. MODL: A modular ontology design library. In Krzysztof Janowicz, Adila Alfa Krisnadhi, María Poveda-Villalón, Karl Hammar, and Cogan Shimizu, editors, *Proceedings of the 10th Workshop on Ontology Design and Patterns (WOP 2019) co-located with 18th International Semantic Web Conference (ISWC 2019), Auckland, New Zealand, October 27, 2019*, volume 2459 of *CEUR Workshop Proceedings*, pages 47–58. CEUR-WS.org, 2019. URL: <https://ceur-ws.org/Vol-2459/paper4.pdf>.
- 48 Martin G. Skjæveland, Henrik Forssell, Johan W. Klüwer, Daniel P. Lupp, Evgenij Thorstensen, and Arild Waaler. Pattern-based ontology design

- and instantiation with reasonable ontology templates. In Eva Blomqvist, Óscar Corcho, Matthew Horridge, David Carral, and Rinke Hoekstra, editors, *Proceedings of the 8th Workshop on Ontology Design and Patterns (WOP 2017) co-located with the 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21, 2017*, volume 2043 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017. URL: <https://ceur-ws.org/Vol-2043/paper-04.pdf>.
- 49 Martin G. Skjæveland, Anders Gjerver, Christian M. Hansen, Johan Wilhelm Klüwer, Morten R. Strand, Arild Waaler, and Per Øyvind Øverli. Semantic Material Master Data Management at Aibel. In *Proceedings of the ISWC 2018 Industry Track*, volume 2180 of *CEUR Workshop Proceedings*, 2018. URL: <https://ceur-ws.org/Vol-2180/paper-90.pdf>.
- 50 Martin G. Skjæveland, Daniel P. Lupp, Leif Harald Karlsen, and Henrik Forssell. Practical ontology pattern instantiation, discovery, and maintenance with reasonable ontology templates. In Denny Vrandečić, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl, editors, *The Semantic Web - ISWC 2018*, volume 11136 of *LNCS*, pages 477–494. Springer, 2018. doi:10.1007/978-3-030-00671-6\_28.
- 51 Martin G. Skjæveland, Daniel P. Lupp, Leif Harald Karlsen, and Johan W. Klüwer. OTTR: formal templates for pattern-based ontology engineering. In Eva Blomqvist, Torsten Hahmann, Karl Hammar, Pascal Hitzler, Rinke Hoekstra, Raghava Mutharaju, María Poveda-Villalón, Cogan Shimizu, Martin G. Skjæveland, Monika Solanki, Vojtech Svátek, and Lu Zhou, editors, *Advances in Pattern-Based Ontology Engineering, extended versions of the papers published at the Workshop on Ontology Design and Patterns (WOP)*, volume 51 of *Studies on the Semantic Web*, pages 349–377. IOS Press, 2021. doi:10.3233/SSW210025.
- 52 Martin G. Skjæveland. *The Core OTTR Template Library*, volume 51 of *Studies on the Semantic Web*, chapter 23, pages 378–393. IOS Press, 2021. doi:10.3233/SSW210026.
- 53 Erik Snilsberg. Reverse OTTR: A query language for RDF. Master’s thesis, University of Oslo, 2022.
- 54 Erik Snilsberg, Leif Harald Karlsen, Egor V. Kostylev, and Martin G. Skjæveland. Foundations of ontology template language OTTR (extended abstract). In Laura Giordano, Jean Christoph Jung, and Ana Ozaki, editors, *Proceedings of the 37th International Workshop on Description Logics (DL 2024), Bergen, Norway, June 18-21, 2024*, volume 3739 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2024. URL: <https://ceur-ws.org/Vol-3739/abstract-24.pdf>.
- 55 Yulia Svetashova, Baifan Zhou, Tim Pychynski, Stefan Schmidt, York Sure-Vetter, Ralf Mikut, and Evgeny Kharlamov. Ontology-enhanced machine learning: A bosch use case of welding quality monitoring. In Jeff Z. Pan, Valentina A. M. Tamma, Claudia d’Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, editors, *The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part II*, volume 12507 of *Lecture Notes in Computer Science*, pages 531–550. Berlin, Heidelberg, 2020. Springer. doi:10.1007/978-3-030-62466-8\_33.
- 56 Tania Tudorache. Ontology engineering: Current state, challenges, and future directions. *Semantic Web*, 11(1):125–138, December 2020. doi:10.3233/SW-190382.
- 57 Tania Tudorache, Csongor Nyulas, Natalya Fridman Noy, and Mark A. Musen. Webprotégé: A collaborative ontology editor and knowledge acquisition tool for the web. *Semantic Web*, 4(1):89–99, 2013. doi:10.3233/SW-2012-0057.
- 58 Denny Vrandečić. Explicit knowledge engineering patterns with macros. In *Proceedings of the Ontology Patterns for the Semantic Web Workshop at the 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005*. Ed.: Chris Welty. Galway, 2005.

# TØIRoads: A Road Data Model Generation Tool

Grunde Haraldsson Wesenberg  

Department of Informatics, University of Bergen, Norway  
Institute of Transport Economics, Oslo, Norway

Ana Ozaki  

Department of Informatics, University of Oslo, Norway  
Department of Informatics, University of Bergen, Norway

## Abstract

We describe road data models which can represent high level features of a road network such as population, points of interest, and road length/cost and capacity, while abstracting from time and geographic location. Such abstraction allows for a simplified traffic usage and congestion analysis that focus on the high level features. We provide theoretical results regarding mass conservation and sufficient conditions for avoiding congestion within the model. We describe a road data model gener-

ation tool, which we call “TØI Roads”. We also describe several parameters that can be specified by a TØI Roads user to create graph data that can serve as input for training graph neural networks (or another learning approach that receives graph data as input) for predicting congestion within the model. The road data model generation tool allows, for instance, the study of the effects of population growth and how changes in road capacity can mitigate traffic congestion.

**2012 ACM Subject Classification** General and reference → Evaluation

**Keywords and phrases** Road Data, Transportation, Graph Neural Networks, Synthetic Dataset Generation

**Digital Object Identifier** 10.4230/TGDK.2.2.6

**Category** Resource Paper

**Supplementary Material** *Software (Source Code)*: <https://github.com/gruwesen/TOIROADS> [19] archived at `swh:1:dir:a86388944844fcc00f4cad67e1ec75a998f36eae`

**Funding** *Grunde Haraldsson Wesenberg*: The author is supported by the Norwegian Research Council, project 322480.

*Ana Ozaki*: The author is supported by the Norwegian Research Council, projects 322480, 316022. This work was partly supported by the Research Council of Norway through its Centre of Excellence Integreat - The Norwegian Centre for knowledge-driven machine learning, project number 332645.

**Received** 2024-07-02 **Accepted** 2024-11-08 **Published** 2024-12-18

**Editors** Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler

**Special Issue** Resources for Graph Data and Knowledge

## 1 Introduction

Modelling and predicting traffic is fundamental for the administration of a city and its surroundings as it directly impacts the economy and everyday lives of the population [11, 2, 9, 1]. While works on traffic prediction tend to focus on short-term prediction (e.g., 12 steps of 5 minutes in the future) [14, 18, 5], a high-level view of the main properties of a road network is crucial for long-term city planning. For instance, one may want to plan for the increase of road capacity by enlarging road segments or to plan for the development of new links in the road network if a systematic congestion scenario is foreseen due to an increase of the population or significant changes in traffic flow resulting from the construction of new points of interest in the city.



© Grunde Wesenberg and Ana Ozaki;  
licensed under Creative Commons License CC-BY 4.0

*Transactions on Graph Data and Knowledge*, Vol. 2, Issue 2, Article No. 6, pp. 6:1–6:12



Transactions on Graph Data and Knowledge

TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The solutions to the high-level setting are classically explored in the context of agent-based traffic simulators [16]. However, agent-based traffic simulators, such as MATSIM [10], are co-evolutionary tools which inherently cannot be parallelized. The complete configuration of features and parameters used in the real world traffic simulators include the specification of a number of agent-based traffic routines. This complexity makes their format hard to use in machine learning approaches, which are parallelizable. Machine learning approaches, in particular those based on graph neural networks, are useful for predicting what would happen in these scenarios [12, 20, 6], however, they need datasets with the relevant features.

In this work we present a road data model generation tool, which we call “TØIRoads”, that can create datasets synthetically, based on user defined parameters. We describe road data models which can represent high level features of a road network such as population, points of interest, and road length/cost and capacity. Such abstraction allows for a simplified traffic usage and congestion analysis that focus on the high level features. We provide theoretical results regarding mass conservation and sufficient conditions for avoiding congestion within the model.

A main benefit of generating road data synthetically is that the data is independent of any particular city structure. Datasets of varying sizes can be created, simulating the fact that road networks also vary in size. Moreover our road data generator has the interesting feature that the user can give a particular structure (which could realistically represent the information from a real world road network) and use our road data model generation tool to create variants (“mutations”) of this network, e.g., with increasing population, which could indicate possible future congestion issues if the population grows without an increase of capacity in the road network.

In Section 3, we describe our road data models and establish some theoretical properties related to these data models. In Section 4, we describe several parameters that can be specified by a TØIRoads user to create graph data that can serve as input for training graph neural networks (or another learning approach that receives graph data as input) for predicting congestion within the model. In Section 5 we present some statistics associated with the datasets generated by TØIRoads. Finally, we conclude in Section 6.

## 2 Related Work

Traffic prediction is a broad research field that has been heavily studied in the literature [3, 11, 13]. Most works focus on short term or even real time traffic prediction, with the goal of supporting drivers to find an optimal path. Indeed, as many cars and drivers today are sending their information to the internet, some researchers have been very successful in predicting traffic based on real time driver resolution data [7].

Jiang and Luo (2022) describe some datasets used for traffic prediction with graph neural networks as well as what variants of graph neural networks are used for the prediction tasks [11]. They categorize studies by scale and domain, and include passenger flow for bus and metro systems, as well as various studies focusing on road traffic flow, speed, congestion and other variables. What is common among the road traffic datasets is the usage of counting stations in city road systems. Counting stations can have different time resolution as well as various measurements in addition to simple counting of vehicles. Olug et al. (2024) [17] show for a dataset like this that adding features, such as population density and districts, to the counting station data points can greatly improve predictions. These datasets related to various modes of transportation are based on available data and as such are quite relevant to traffic prediction research.

Bui et al writes about using spatial temporal graph neural networks for traffic forecasting [3], and points to how to construct the adjacency matrix of a graph based on traffic counting station nodes is an open problem, in terms of effectively capturing information streams. For instance, in



the urban vehicle detection system dataset [4] that they use, there is no single way to make the adjacency matrix, as there are many road links between each counting station. The TØIRoads dataset generator can take as input a graph structure based on real world data and make datasets with variants of the original data, and so, it might be used as a tool for studying this graph construction problem.

Here we consider traffic from the point of view of road administrators, also called road planners, who have the goal of predicting the need e.g., of improving road capacity in certain road segments to avoid congestion in the city. Our model is made to work as an aggregate of traffic data, representing high level features of a road network, such as population, points of interest, and road length/cost and capacity, while representing the graph structure in detail, at a road segment level. Our model is not restricted to a particular graph structure and it can generate graph structures synthetically. There are various works in the literature on generating graph data synthetically [8, 15], however, here we use features and structural constraints that are meaningful for road traffic prediction. We prove theoretical results related to traffic conservation and congestion for our road data models.

### 3 Modelling Road Traffic

In this section we describe our road data models. The model is an abstraction of the road traffic conditions found in real world scenarios, useful to estimate road usage and congestion. The road network is represented with a graph, where nodes are road segments and edges connect road segments which would be connected in the road network. High population contributes to increase the traffic of a road segment, while high cost contributes to decreasing the traffic. Road capacity is a feature that combines with the measure of traffic to determine if a road segment is congested or not. See Section 4.3 for a detailed example containing the notions presented here.

#### 3.1 Formal Definition

We formally define a road traffic model based on the components related to road usage and congestion described above. We call this model RoadGNN.

► **Definition 1** (RoadGNN). *A RoadGNN road data model is a labelled graph  $G = (V, E, L)$ , where  $V$  is a finite set of nodes representing road segments,  $E$  is a (finite) set of edges representing connections between road segments, and  $L$  is a labelling function mapping nodes to their respective attributes. Each node  $v \in V$  represents a road segment and  $v$  is associated via the labelling function  $L$  with the following attributes:*

- road weight  $w_v \in \mathbb{R}$  with  $w_v > 0$ ,
- capacity  $c_v \in \mathbb{R}$  with  $c_v > 0$ ,
- population  $p_{v,t} \in \mathbb{N}$  from node  $v$  to each node  $t \in V$ ,

where  $w_v$  is a weight value associated with node  $v$ , depending on its length/cost;  $c_v$  quantifies the road capacity associated with node  $v$ , and  $p_{v,t}$  represents the flow with origin in  $v$  and destination in  $t$ . In symbols,  $L(v) = (w_v, c_v, p_{v,t_1}, \dots, p_{v,t_n})$  with  $t_1, \dots, t_n \in V$ .

In the following we consider a fixed but arbitrary labelled graph  $G$ . We omit  $G$  from the notation to simplify it since there is no risk of confusion. A *path* from  $s \in V$  to  $t \in V$  is a sequence of nodes  $v_1, \dots, v_k \in V$  with  $s = v_1$ ,  $t = v_k$ , and  $(v_i, v_{i+1}) \in E$  for all  $1 \leq i < k$ . We define a total order on paths where path  $p_1$  is shorter than  $p_2$ , in symbols  $p_1 < p_2$ , if the sum of the weights  $w_{v_1}, \dots, w_{v_k}$  of the nodes  $v_1, \dots, v_k$  in  $p_1$  is smaller than that sum in  $p_2$ .

► **Remark 2.** All labelled graphs we consider in this work are *strongly connected*, that is, they have the property that for every pair of nodes  $(s, t) \in V \times V$  there is a path from  $s$  to  $t$ .

For each pair of nodes  $(s, t)$ , define  $S_{s,t}$  as the set of shortest paths between  $s$  and  $t$ . Also, we denote by  $S_{s,t}^v$  the subset of paths in  $S_{s,t}$  where  $v \in V$  occurs. Given a finite set  $S$ , we write  $|S|$  for the number of elements in  $S$ . By definition, sets of shortest paths in the graph are finite. Given a node  $v \in V$ , we write  $\text{in\_pop}(v)$  as a shorthand for  $\sum_{s \in V} p_{s,v}$  and  $\text{out\_pop}(v)$  as a shorthand for  $\sum_{t \in V} p_{v,t}$ . Let  $\text{in}(v)$  and  $\text{out}(v)$  denote the sets of incoming and out going nodes from node  $v$ .

We now describe how we estimate the values associated with road usage and congestion. We assume that agents aim at trips that minimize road usage, by sticking with shortest paths when calculating a route.

► **Definition 3 (Road Usage).** *The road usage of node  $v \in V$ , denoted  $U_v$ , is defined as follows:*

$$\sum_{(s,t) \in V^2} p_{s,t} \cdot \frac{|S_{s,t}^v|}{|S_{s,t}|}.$$

In real traffic, a road is not congested when the traffic is below capacity, and congested when the traffic is above capacity. We define our congestion coefficient as the quotient between road usage and capacity, and call the node congested if this is 1 or more.

► **Definition 4 (Congestion).** *Congestion is the ratio between road usage and capacity, in symbols,*

$$C_v := \frac{U_v}{c_v}.$$

*We say that a node  $v$  is congested if  $C_v \geq 1$ .*

► **Remark 5.** By definition of  $U_v$ , we have that  $U_v \leq \text{in\_pop}(v) + \text{out\_pop}(v)$ .

We now establish theoretical bounds regarding traffic conservation and congestion.

► **Theorem 6 (Conservation).** *For all  $v \in V$ ,*

1.  $U_v \leq \sum_{u \in \text{in}(v)} (U_u - \text{in\_pop}(u)) + \text{out\_pop}(v)$
2.  $U_v \leq \sum_{u \in \text{out}(v)} (U_u - \text{out\_pop}(u)) + \text{in\_pop}(v)$ .

**Proof.** We start proving Point 1. Let  $v$  be a node in  $V$ . By definition of  $U_v$ , we can write  $U_v$ , as follows:

$$\sum_{(s,t) \in (V \setminus \{v\}) \times V} p_{s,t} \cdot \frac{|S_{s,t}^v|}{|S_{s,t}|} + \sum_{(v,t) \in \{v\} \times V} p_{v,t} \cdot \frac{|S_{v,t}^v|}{|S_{v,t}|}.$$

By definition,  $S_{v,t}^v$  is the set of shortest paths from  $s$  to  $v$  that passes through  $v$ , which coincides with the set of shortest paths from  $s$  to  $v$ , denoted  $S_{v,t}$ . In other words,  $S_{v,t}^v = S_{v,t}$ . So  $U_v$  is

$$\sum_{(s,t) \in (V \setminus \{v\}) \times V} p_{s,t} \cdot \frac{|S_{s,t}^v|}{|S_{s,t}|} + \text{out\_pop}(v).$$

It remains to show that

$$\sum_{(s,t) \in (V \setminus \{v\}) \times V} p_{s,t} \cdot \frac{|S_{s,t}^v|}{|S_{s,t}|} \leq \sum_{u \in \text{in}(v)} (U_u - \text{in\_pop}(u)).$$

Given a node  $v$  and pair of nodes  $(s, t)$ , if a path  $p$  is in  $S_{s,t}^v$  and  $s \neq v$  then  $p$  is in  $S_{s,t}^u$  for some  $u \in \text{in}(v)$ . Then,

$$\sum_{(s,t) \in (V \setminus \{v\}) \times V} p_{s,t} \cdot \frac{|S_{s,t}^v|}{|S_{s,t}|} \leq \sum_{u \in \text{in}(v)} \sum_{(s,t) \in V^2} p_{s,t} \cdot \frac{|S_{s,t}^u|}{|S_{s,t}|}.$$

In other words,

$$\sum_{(s,t) \in (V \setminus \{v\}) \times V} p_{s,t} \cdot \frac{|S_{s,t}^v|}{|S_{s,t}|} \leq \sum_{u \in \text{in}(v)} U_u.$$

Moreover, we have that  $u \neq t$  since by assumption there is a path in  $S_{s,t}^v$  (in other words,  $v$  belongs to a shortest path between  $s$  and  $t$ , which would not be the case if  $u = t$  since  $u \in \text{in}(v)$ ). So we can subtract the population associated with those paths:

$$\sum_{(s,t) \in (V \setminus \{v\}) \times V} p_{s,t} \cdot \frac{|S_{s,t}^v|}{|S_{s,t}|} \leq \sum_{u \in \text{in}(v)} (U_u - \sum_{s \in V} p_{s,u}).$$

This means that

$$\sum_{(s,t) \in (V \setminus \{v\}) \times V} p_{s,t} \cdot \frac{|S_{s,t}^v|}{|S_{s,t}|} \leq \sum_{u \in \text{in}(v)} (U_u - \text{in\_pop}(u))$$

as required. The proof of Point 2 is analogous.  $\blacktriangleleft$

We now provide sufficient (but not necessary) conditions for preventing node congestion.

► **Theorem 7 (Outgoing Congestion).** *For all  $v \in V$ , if*

1.  $c_v \geq \sum_{u \in \text{out}(v)} c_u + \text{in\_pop}(v)$ , and
2.  $\forall u \in \text{out}(v)$ ,  $C_u < 1$ ,

*then  $v$  is not congested.*

**Proof.** We can rewrite Assumption 1 as

$$c_v \geq \Delta_v + \sum_{u \in \text{out}(v)} U_u + \text{in\_pop}(v) \tag{1}$$

where

$$\Delta_v := \sum_{u \in \text{out}(v)} (c_u - U_u).$$

By Point 2 of Theorem 6,

$$U_v \leq \sum_{u \in \text{out}(v)} (U_u - \text{out\_pop}(u)) + \text{in\_pop}(v).$$

Adding  $\Delta_v + \sum_{u \in \text{out}(v)} \text{out\_pop}(u)$  on both sides,

$$U_v + \Delta_v + \sum_{u \in \text{out}(v)} \text{out\_pop}(u) \leq \Delta_v + \sum_{u \in \text{out}(v)} U_u + \text{in\_pop}(v).$$

By Eq. 1,

$$U_v + \Delta_v + \sum_{u \in \text{out}(v)} \text{out\_pop}(u) \leq c_v.$$

By Assumption 2,  $\forall u \in \text{out}(v)$ ,  $c_u > U_u$ . Thus,  $\Delta_v > 0$ . Since  $\sum_{u \in \text{out}(v)} \text{out\_pop}(u) \geq 0$ , we have that  $c_v > U_v$ . In other words,  $v$  is not congested.  $\blacktriangleleft$

We also have an analogous theorem for the incoming congestion, which can be proved in a similar way as for outgoing congestion.

► **Theorem 8 (Incoming Congestion).** *For all  $v \in V$ , if*

1.  $c_v \geq \sum_{u \in \text{in}(v)} c_u + \text{out\_pop}(v)$ , and
2.  $\forall u \in \text{in}(v)$ ,  $C_u < 1$ ,

*then  $v$  is not congested.*

### 3.2 A Practical Special Case

In the model just described each node  $v$  has a feature  $p_{v,t}$  associated with each other node  $t$  in the graph, which quantifies the population going from  $v$  to  $t$ . This means that the number of features in a node grows depending on the size of the graph (in particular, in the number of nodes), which is not ideal in practice. So here we consider a simplified model, called S-RoadGNN, which associates a fixed number of features (4 features, namely, road weight, capacity, population, and points of interest) to each node. The idea is that people live along roads (population), and they travel to points of interest (POI) that are along the roads. The traffic associated with each node is given by the amount of transport demand density and it depends on its close and far neighbours. Although this simplification takes away some of the expressivity of our road data models (see Remark 12), it is flexible enough to create a range of road data models which are useful to study the effects of population growth and how changes in road capacity can mitigate traffic congestion.

► **Definition 9** (S-RoadGNN). *A S-RoadGNN road data model is a labelled graph  $G = (V, E, L)$ , where  $V$  is a finite set of nodes representing road segments,  $E$  is a (finite) set of edges representing connections between road segments, and  $L$  is a labelling function mapping nodes to their respective attributes. Each node  $v \in V$  has the following attributes:*

- road weight  $w_v \in \mathbb{R}$  with  $w_v > 0$ ,
- capacity  $c_v \in \mathbb{R}$  with  $c_v > 0$ ,
- population  $p_v \in \mathbb{N}$ ,
- points of interest  $i_v \in \mathbb{N}$ ,

where  $w_v$  is a weight value associated with node  $v$ , representing a road segment, depending on its cost/length;  $c_v$  quantifies the road capacity associated with node  $v$ ; and finally,  $p_v$  and  $i_v$  are associated with  $v$  depending on the population and on the points of interest in the vicinity of the road segment represented by  $v$ . In symbols,  $L(v) = (w_v, c_v, p_v, i_v)$ .

► **Definition 10** (Road Usage). *The road usage of node  $v \in V$ , denoted  $U_v$ , is defined as follows:*

$$\sum_{(s,t) \in V^2} p_s \cdot i_t \cdot \frac{|S_{s,t}^v|}{|S_{s,t}|}.$$

► **Definition 11** (Congestion). *Congestion is as in Definition 4: the ratio between road usage and capacity. A node  $v$  is congested if  $C_v \geq 1$ .*

► **Remark 12.** S-RoadGNN can be seen as a special case of RoadGNN where  $p_{s,t} = p_s \cdot i_t$ . In RoadGNN, we can have models where e.g., there is traffic from a node  $s$  to a node  $t$ , but no traffic from another node  $u$  that has some population to  $t$ . This is not possible in S-RoadGNN because in this model whenever a node  $u$  has some population and another node  $t$  has some points of interest, there is some traffic from  $u$  to  $t$ , namely  $p_u \cdot i_t$ .

We formalize this remark with the following theorem.

► **Theorem 13.** *Let  $G$  be an S-RoadGNN road data model and let  $G'$  be the RoadGNN road data model that is defined in the same way as  $G$ , except that  $p_{s,t} := p_s \cdot i_t$  for all  $s, t \in V$ . For all  $v \in V$ ,  $U_v = U'_v$ , where  $U_v$  and  $U'_v$  correspond to the road usage of  $v$  in  $G$  and  $G'$ , respectively.*

It follows from Theorem 13 that the conservation and congestion bounds established for RoadGNN also hold for S-RoadGNN. We are now ready to describe TØIRoads, which is a tool for generating S-RoadGNN road data models.

## 4 TØIRoads: Road Data Model Generation

Here we present our road data model generator, named TØIRoads, for simulating traffic conditions in a simplified road network. The road data models generated by TØIRoads are instances of the model described in Section 3.2. We first describe the parameters that the user can give and then the procedure for generating road data models according to the parameters given by the user.

### 4.1 Parameters for Road Data Model Generation

The user can give as input the *number of road data models* to be created, together with the *minimum and maximum number of nodes* allowed in each road data model. The user can also influence the *graph density*, which is the ratio between the number of edges in the graph and the total number of possible edges it could have. This is a number between 0 and 1 that quantifies the amount of edges that should be added after a graph  $G_N$  with  $N$  nodes is constructed.

We now describe further parameters that the user can give as input to TØIRoads, related to the features outlined in Definition 1.

- *Road weight* (length): the user can give as input the minimum and maximum values for the cost/length of a node representing a road segment.
- *Capacity*: the user can give as input the minimum and maximum values for the capacity of a road segment. In addition, the user can specify a parameter called *capacity factor* that is used to multiply the capacity values, in this way, the user can create road data models that tend to be more or less congested, depending on how the capacity factor is defined.
- *Population*: the user can give as input the maximum value for the population associated with a node representing a road segment, with the minimum value being 0. There is also a rate parameter where the user can specify the proportion of nodes with non-zero value.
- *Points of interest*: the user can give as input the maximum value for the points of interest associated with a node representing a road segment, with the minimum value being 0. There is also a rate parameter where the user can specify the proportion of nodes with non-zero value.

For the minimum and maximum values above, TØIRoads chooses a value in the corresponding interval, using the uniform distribution, and generates the road data models.

### 4.2 The Algorithm for Road Data Model Generation

In addition to the specification in Section 3.2, the graph data has some constraints, motivated by how road networks are designed in practice. The tool is made to generate graphs of arbitrary size, with a few control elements. There are no self-loops and the graph is strongly connected, which makes sense considering the structure of a road network. We formally state these properties in Propositions 14 and 15.

► **Proposition 14.** *Every road data model returned by Algorithm 1 is strongly connected.*

**Sketch.** The algorithm starts creating a graph  $G_1$  with just one node, which by definition, forms a strongly connected graph. Then, given  $G_i$ , with  $i \geq 1$ , the algorithm creates  $G_{i+1}$  by adding to  $G_i$  a new node  $n$  and two edges: one going from  $n$  to a random node in  $G_i$  and one coming from a random node to  $n$ . In this way, if  $G_i$  is strongly connected, the same will hold to  $G_{i+1}$ . ◀

If  $N$  is the number of nodes in a graph  $G_N$  then this procedure creates  $2N - 2$  edges since at each iteration 2 edges are added and the initial graph has no edges. Although this procedure creates a strongly connected graph, its density is only  $(2N - 2)/(N(N - 1))$ , since a fully connected (directed) graph has  $N(N - 1)$  edges. In practice, road networks can be much more dense. As mentioned in Section 4.1, TØIRoads allows the user to adjust this by including a parameter called

*density*. In fact, the implementation can add a bit more variation to this, which is useful to create multiple similar but different graphs in the dataset, using a *variation flag*. If true then the density value given by the user is multiplied by a random number between 0.9 and 1.1. The choice of the edges to be added follows a uniform distribution. We state this property in Proposition 15.

► **Proposition 15.** *Every road data model returned by Algorithm 1 has density  $d$ , provided that  $d > (2N - 2)/(N(N - 1))$  and the variation flag is set to false.*

**Sketch.** If the density parameter given by the user is lower than  $(2N - 2)/(N(N - 1))$  then there is no change in density (since we want to ensure the graph is strongly connected). Otherwise, the algorithm adds edges randomly so as to reach the desired density (which is the density value  $d$  given as input by the user if the variation flag is false, otherwise, a value close to  $d$ ). In detail, the algorithm first selects the  $N^2 - 3N + 2$  possible edges that could be added (that is, not those  $2N - 2$  that were included in the creation of the strongly connected graph). Then, it randomly chooses a desired number among these possible edges to reach density  $d$ . ◀

The description of the algorithm for generating road data models is given by Algorithm 1, assuming for simplicity that the variation flag is set to false.

■ **Algorithm 1** Road Data Model Generation.

---

```

1: input: number  $n$  of road data models, interval  $I$  for the number of nodes in each data model,
   density  $d$ , and intervals for road weight, capacity, population, and points of interest (Sec 4.1).
2: output:  $n$  road data models satisfying Proposition 14 and Proposition 15
3: for  $i = 1$  to  $n$  do
4:   Randomly choose a number of nodes  $m$  in  $I$ .
5:   Set road weight, capacity, population, and points of interest values to each node
6:   Create a strongly connected graph  $G^i$  with  $m$  nodes (as in Proposition 14)
7:   if density  $> (2N - 2)/(N^2 - N)$  then
8:     Add random edges so as to reach the desired density  $d$  (Proposition 15)
9:   end if
10:  Calculate road usage as in Definition 10
11:  Calculate congestion as in Definition 11
12: end for
13: return  $G^1, \dots, G^n$ 

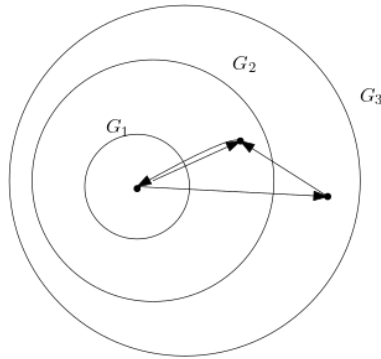
```

---

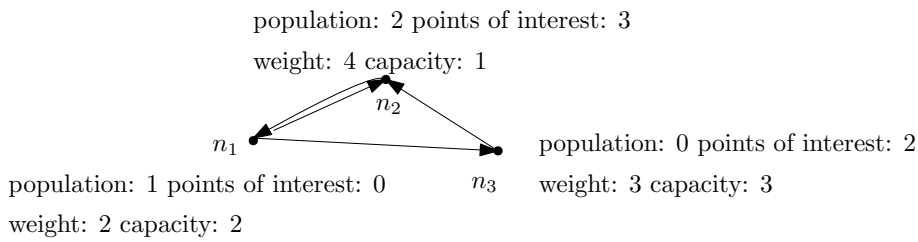
### 4.3 Example Run

For the convenience of the reader, we present an example run illustrating our definitions in Section 3 and the strategy in Sections 4.1 and 4.2. We choose small numbers for conciseness. Suppose the user wants to create a simple dataset with just one road data model where the graph has 3 nodes. Assume TØIRoads creates  $G_1, G_2, G_3$  as in Figure 1. Now suppose the user sets the following parameters for the features:

- road weight is a number between 1 and 5;
- road capacity is a value between 1 and 3;
- the capacity factor is 1;
- the rates for points of interest and population are both 0.7; and
- the maximum population and points of interest (per node representing a node segment) is 3.



■ **Figure 1** Graph Structure of  $G_1, G_2, G_3$ .



■ **Figure 2** Road Data Model.

Given the parameters above and the graph structure illustrated in Figure 1, suppose TØIRoads creates the road data model illustrated in Figure 2. In the road data model of Figure 2, there are two paths from  $n_1$  to  $n_2$ . The shortest one  $p_1$  corresponds to the sequence  $n_1, n_2$  (sum of weights 6) and the longest path  $p_2$  is the sequence  $n_1, n_3, n_2$  (sum of weights 9), by the order relation defined in Section 3.1. Considering  $V = \{n_1, n_2, n_3\}$ , the road usage of  $n_2$  is

$$\sum_{(s,t) \in V^2} p_s \cdot i_t \cdot \frac{|S_{s,t}^{n_2}|}{|S_{s,t}|} = 1 \cdot 3 + 2 \cdot 3 + 2 \cdot 0 + 0 \cdot 0 + 0 \cdot 3 = 9.$$

Since the capacity of this node is only 1, in this road data model, we have congestion in node  $n_2$ .

### 4.4 Graph Mutator

The graph mutator is an additional component added to the implementation to facilitate the creation of variants of a particular road data model given as input. The idea of this component is to give the user the possibility to train a machine learning model with enough data to simulate “what if” scenarios starting from a specific road data model, that could have realistic values from a city related to population, points of interest, etc. The parameters that the user can set are:

- the number of mutations of the original road data model that are to be generated,
- the percentage of nodes that should change
- the maximum values for the population, points of interest, and capacity.

The graph mutator component of TØIRoads chooses a value in the corresponding interval, according to the uniform distribution and generates the variations of the features values of the original graph (keeping the structure of the original graph).

## 5 Datasets generated by TØIRoads

In this section we provide some experimental results using TØIRoads for generating datasets. The experiments were performed on an Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz with 4 cores (x86\_64 architecture) with 11 GB of memory. The code used for the experiments is available at <https://github.com/gruwesen/TOIROADS>. Our hypotheses are that (i) the run time is heavily affected by density and number of nodes, (ii) congestion is affected by the number of nodes and the maximum number of population per node, and (iii) congestion is affected by the ratio of nodes that have a population value. From the mathematical point of view, points of interest and population are interchangeable variables in S-RoadGNN. So the same behaviour for population would hold for points of interest, meaning that there would be no need to perform experiments varying the points of interest instead of population. For this reason, runtime and congestion assessments are only done varying population. We first present some results regarding the time needed to create the datasets.

■ **Table 1** Runtime in seconds for generating 10 graphs varying the number of nodes and density.

Nodes	50	50	100	100	200	200	300	300	500	500
Density	0.04	0.5	0.04	0.5	0.04	0.5	0.04	0.5	0.04	0.5
Time	0.257	0.697	1.450	7.244	12.949	98.942	53.141	487.046	351.428	3732.413

The results in Table 1 show how increasing the number of nodes and density affects the runtime. For this experiment, the values of the remaining parameters were kept fix and set as follows: the road weight (indicating cost/length) was a number between 1 and 10, the population and points of interest rate was 0.4, the maximum for population and points of interest (per node representing a road segment) was 3, finally, road capacity was a number between 10 and 20.

We now present results showing how increasing the population (and not road capacity) can lead to more congested nodes in the road data model.

■ **Table 2** Congestion increasing according to population increase at 0.4 population rate.

Nodes	50	50	50	100	100	100	200	200	200
Total nodes	1000	1000	1000	2000	2000	2000	4000	4000	4000
Max population	4	10	40	4	10	40	4	10	40
Congested nodes	807	898	934	1624	1681	1829	3051	3250	3428
Congestion %	80.7%	89.8%	93.4%	82.1%	84.1%	91.5%	76.3%	81.3%	85.7%

Table 2 describes how max population increases the number of congested nodes when considering three sets of 20 graphs, each with 50, 100, and 200 nodes. The capacity is set to a random number between  $[3\sqrt{n}, 3n]$ , where  $n$  is the number of nodes, as this interval has been found to often yield a medium amount of congestion in S-RoadGNN. As can be seen from the table, increasing the max population increases congestion at every node number.

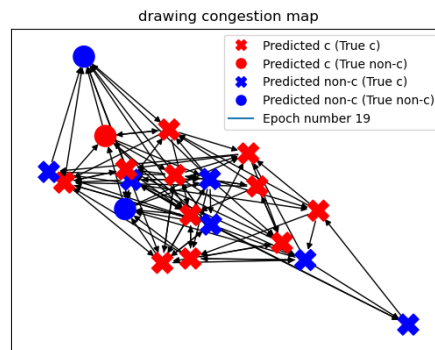
Table 3 describes how the population rate increases the congestion level in a similar manner to Table 2. Again, we see an increase in the number of congested nodes as we increase the population rate, similar to when we increase the maximum population. To reiterate, the max population is the max value of a node that has population, while the population rate describes the part of nodes that contain a population. Decreasing the population rate to 0.2 did more to lessen congestion than decreasing the max population.

To illustrate how this dataset can be used in the context of traffic prediction, we have made an experiment using a graph neural network (GNN) that predicts node congestion. In this experiment



■ **Table 3** Congestion increasing according to population rate increase at 40 max population.

Nodes	50	50	50	100	100	100	200	200	200
Total nodes	1000	1000	1000	2000	2000	2000	4000	4000	4000
Population rate	0.2	0.4	0.8	0.2	0.4	0.8	0.2	0.4	0.8
Congested nodes	608	778	957	1244	1575	1904	2240	2975	3749
Congestion %	60.8%	77.8%	95.7%	62.2%	78.75%	95.2%	56.0%	74.4%	93.73%



■ **Figure 3** Example of Congestion Prediction with a Graph Neural Network.

we trained a GNN model on a dataset generated by S-RoadGNN to predict whether a node is congested or not. We included in Figure 3 a graphical representation of the prediction results of the GNN. Figure 3 shows the resulting predictions for a single example graph after twenty epochs of training. This experiment was run using a Graph Attention Network (GAT) with three layers. The training dataset contained 100 graphs of 20 nodes, while the validation set contained 40 graphs of 20 nodes. The specific graph that is drawn in Figure 3 was drawn randomly from the validation set. In this figure, the model correctly predicted congestion in 13 of the 20 nodes.

## 6 Conclusion

We motivate and present road data models, some of their theoretical properties and a tool, called TØIRoads, for generating datasets within these models. While one may be tempted to assume that preventing congestion is always desirable, this may not be the case if congestion of non-environmental friendly means of transportation, such as private cars, serve as incentive for the use of public transportation. Nevertheless, being able to study the scenarios related to changes in the population and traffic flow depending on points of interest and to predict congestion (whether to prevent it or not) is important for long-term city planning. As future work, we plan to include a few more parameters such as a sensible strategy for adding location information (currently only represented in a relative way, based on paths and their weights) to our road data models. A potential improvement would be to allow different densities per region of the graph, to simulate more dense districts, as commonly happens to districts that are close to the center of a city.

## References

- 1 Azzedine Boukerche, Yanjie Tao, and Peng Sun. Artificial intelligence-based vehicular traffic flow prediction methods for supporting intelligent transportation systems. *Comput. Networks*, 182:107484, 2020. doi:10.1016/j.comnet.2020.107484.
- 2 Azzedine Boukerche and Jiahao Wang. Machine learning-based traffic prediction models for intelligent transportation systems. *Comput. Networks*, 181:107530, 2020. doi:10.1016/j.comnet.2020.107530.
- 3 Khac-Hoai Nam Bui, Jiho Cho, and Hongsuk Yi. Spatial-temporal graph neural network for traffic forecasting: An overview and open research issues. *Appl. Intell.*, 52(3):2763–2774, 2022. doi:10.1007/s10489-021-02587-w.
- 4 Khac-Hoai Nam Bui, Hongsuk Yi, and Jiho Cho. UVDS: A new dataset for traffic forecasting with spatial-temporal correlation. In Ngoc Thanh Nguyen, Suphamit Chittayasothorn, Dusit Niyato, and Bogdan Trawinski, editors, *Intelligent Information and Database Systems - 13th Asian Conference, ACIIDS 2021, Phuket, Thailand, April 7-10, 2021, Proceedings*, volume 12672 of *Lecture Notes in Computer Science*, pages 66–77. Springer, 2021. doi:10.1007/978-3-030-73280-6\_6.
- 5 Jeongwhan Choi and Noseong Park. Graph neural rough differential equations for traffic forecasting. *ACM Trans. Intell. Syst. Technol.*, 14(4):74:1–74:27, 2023. doi:10.1145/3604808.
- 6 Zhiyong Cui, Ruimin Ke, Ziyuan Pu, Xiaolei Ma, and Yin Hai Wang. Learning traffic as a graph: A gated graph wavelet recurrent neural network for network-scale traffic prediction. *Transportation Research Part C: Emerging Technologies*, 115:102620, 2020. doi:10.1016/j.trc.2020.102620.
- 7 Austin Darrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, Peter W. Battaglia, Vishal Gupta, Ang Li, Zhongwen Xu, Alvaro Sanchez-Gonzalez, Yujia Li, and Petar Velickovic. ETA prediction with graph neural networks in google maps. In Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 3767–3776. ACM, 2021. doi:10.1145/3459637.3481916.
- 8 Alessio Gravina and Danilo Numeroso. *NumGraph*. <https://numgraph.readthedocs.io>.
- 9 Xiao Han, Guojiang Shen, Xi Yang, and Xiangjie Kong. Congestion recognition for hybrid urban road systems via digraph convolutional network. *Transportation Research Part C: Emerging Technologies*, 121:102877, 2020. doi:10.1016/j.trc.2020.102877.
- 10 Andreas Horni, Kai Nagel, and Kay Axhausen. *Introducing MATSim*, pages 3–8. Ubiquity Press, August 2016. doi:10.5334/baw.1.
- 11 Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *Expert Syst. Appl.*, 207:117921, 2022. doi:10.1016/j.eswa.2022.117921.
- 12 Weiwei Jiang, Jiayun Luo, Miao He, and Weixi Gu. Graph neural network for traffic forecasting: The research progress. *ISPRS Int. J. Geo Inf.*, 12(3):100, 2023. doi:10.3390/ijgi12030100.
- 13 Alexandra Kapp, Julia Hansmeyer, and Helena Mihaljevic. Generative models for synthetic urban mobility data: A systematic literature review. *ACM Comput. Surv.*, 56(4):93:1–93:37, 2024. doi:10.1145/3610224.
- 14 Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: <https://openreview.net/forum?id=SJiHXGWAZ>.
- 15 Jiaqi Ma, Jiong Zhu, Yuxiao Dong, Danaï Koutra, Jingrui He, Qiaozhu Mei, Anton Tsitsulin, Xingjian Zhang, and Marinka Zitnik. The 3rd workshop on graph learning benchmarks (GLB 2023). In Ambuj K. Singh, Yizhou Sun, Leman Akoglu, Dimitrios Gunopulos, Xifeng Yan, Ravi Kumar, Fatma Ozcan, and Jieping Ye, editors, *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD*, pages 5870–5871. ACM, 2023. doi:10.1145/3580305.3599224.
- 16 Johannes Nguyen, Simon T. Powers, Neil Urquhart, Thomas Farrenkopf, and Michael Guckert. An overview of agent-based traffic simulators. *Transportation Research Interdisciplinary Perspectives*, 12:100486, 2021. doi:10.1016/j.trip.2021.100486.
- 17 Eren Olug, Kiyem Kaya, Resul Tugay, and Sule Gündüz Ögüdücü. IBB traffic graph data: Benchmarking and road traffic prediction model. *CoRR*, abs/2408.01016, 2024. doi:10.48550/arXiv.2408.01016.
- 18 Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *AAAI*, pages 914–921. AAAI Press, 2020. doi:10.1609/aaai.v34i01.5438.
- 19 Grunde Haraldsson Wesenberg. gruwesen/TOIROADS. Software, version 1.0., Norwegian Research Council, project 322480, swhId: swh:1:dir:a86388944844fcc00f4cad67e1ec75a998f36eae (visited on 2024-12-11). URL: <https://github.com/gruwesen/TOIROADS>, doi:10.4230/artifacts.22621.
- 20 Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 3634–3640. ijcai.org, 2018. doi:10.24963/ijcai.2018/505.

# Whelk: An OWL EL+RL Reasoner Enabling New Use Cases

James P. Balhoff<sup>1</sup> ✉ 

Renaissance Computing Institute, University of North Carolina, Chapel Hill, NC, USA

Christopher J. Mungall ✉ 

Lawrence Berkeley National Laboratory, Berkeley, CA, USA

## Abstract

Many tasks in the biosciences rely on reasoning with large OWL terminologies (Tboxes), often combined with even larger databases. In particular, a common task is retrieval queries that utilize relational expressions; for example, “find all genes expressed in the brain or any part of the brain”. Automated reasoning on these ontologies typically relies on scalable reasoners targeting the EL subset of OWL, such as ELK. While the introduction of ELK has been transformative in the incorporation of reasoning into bio-ontology quality control and production pipelines, we have encountered limitations when applying it to use cases involving high throughput query answering or reasoning about datasets describing instances (Aboxes).

Whelk is a fast OWL reasoner for combined EL+RL reasoning. As such, it is particularly useful for many biological ontology tasks, particularly those characterized by large Tboxes using the EL subset of OWL, combined with Aboxes targeting the RL subset of OWL. Whelk is implemented in Scala and utilizes immutable functional data structures, which provides advantages when performing incremental or dynamic reasoning tasks. Whelk supports querying complex class expressions at a substantially greater rate than ELK, and can answer queries or perform incremental reasoning tasks in parallel, enabling novel applications of OWL reasoning.

**2012 ACM Subject Classification** Information systems → Web Ontology Language (OWL); Software and its engineering → Software libraries and repositories; Applied computing → Life and medical sciences

**Keywords and phrases** Web Ontology Language, OWL, Semantic Web, ontology, reasoner

**Digital Object Identifier** 10.4230/TGDK.2.2.7

**Category** Resource Paper

**Supplementary Material** *Software (Source code)*: <https://github.com/INCATools/whelk> [9]  
archived at `swh:1:dir:f8919707e053212b2c74bc63988a06ffe03fb796`

*Software (Evaluation scripts and input ontologies)*: <https://doi.org/10.5281/zenodo.13891879> [7]

*InteractiveResource (Whelk on the Web)*: <https://balhoff.github.io/whelk-web/>

**Acknowledgements** We would like to thank N.L. Harris for helpful comments on a draft of the paper.

**Received** 2024-07-01 **Accepted** 2024-11-10 **Published** 2024-12-18

**Editors** Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler

**Special Issue** Resources for Graph Data and Knowledge

## 1 Introduction

OWL (Web Ontology Language) is heavily used in the biosciences as a framework for constructing widely used ontologies, for example the Gene Ontology [17], Uberon [30], Human Phenotype Ontology [16], SNOMED-CT [22], and the NCI Thesaurus [35]. These ontologies are characterized by a number of features: (1) their large size relative to other ontologies; (2) the use of existential

<sup>1</sup> Corresponding author



restrictions to encode graph-oriented information such as paronomies and developmental lineages (we call this the Relation Graph); and (3) the fact that the majority of axioms are encoded using the EL subset of OWL [42]. As such, they are amenable to automated reasoning using engines that are tuned for this profile, the most notable of which is the ELK Reasoner, introduced in 2010 [24]. In fact the introduction of ELK has been instrumental in making reasoning scalable for bio-ontologies [28]. ELK is the default reasoner in the ROBOT tool [23], and is deployed as a part of the production pipelines for dozens of ontologies [27].

However, two limitations of ELK constrain its applications: (1) It has limited support for instance graphs (Aboxes) in that, while it infers class membership, it does not materialize inferred object property assertions. Nor does it implement SWRL [21], a language for extending OWL with custom reasoning rules. Additionally, the expressivity of the OWL EL profile excludes key types of object property axioms, such as inverse and functional properties. Nonetheless, most available OWL reasoners with support for rich Abox reasoning do not scale to the size and complexity of terminologies easily handled by ELK. (2) Classification and query answering are blocking operations on a mutable state. That is, although ELK supports incremental reasoning, after new axioms are added and classified, querying the previous state of the ontology requires removing those just added. Likewise, because answering complex queries typically requires some incremental classification, ELK processes a single query at a time. Reasoner queries are frequently used to explore biomedical terminologies, e.g., find all classes that are a *MuscleOrgan* and (*partOf* some *Head*).

In order to support various use cases hindered by those limitations, we implemented the Whelk reasoner, based on the ELK algorithm described by Kazakov et al [24]. In addition to the OWL EL reasoning rules defined for ELK, Whelk supports the OWL RL profile [42] as well as reasoning with SWRL rules. Whelk’s implementation is based on immutable functional data structures [32], so that each time axioms are added, a new reasoner state is created; references to the previous state remain unchanged. This allows Whelk to answer queries concurrently, and also allows concurrent incremental classification of unrelated sets of axioms – for example, various Abox ontologies which build upon the same ontology Tbox. The Tbox can be classified ahead of time and reused. We evaluated Whelk on varied benchmarks taken from real-world ontology reasoning scenarios, comparing Whelk to two releases of ELK, the state of the art open source OWL EL reasoner.

## 2 OWL ontologies

OWL is an ontology language for the Semantic Web [40]. An ontology is a formal representation of concepts within a domain and the logical relationships between those concepts, supporting knowledge representation with explicit semantics. The semantics of OWL are based in Description Logics [2]. The format we use for OWL examples in this paper follows the user-friendly Manchester syntax [41]. An OWL ontology models a domain using classes, properties, and individuals, and consists of axioms making statements about these entities. Individuals are specific objects within the model (e.g., *Bob*, *Alice*, *NewYorkCity*), while classes are used to define categories of individuals (e.g., *Person*, *City*). Individuals are said to be instances of classes. Properties denote relationships between individuals in the model (e.g., *Bob livesIn NewYorkCity*). In addition to simple named classes, complex class expressions can be constructed that define categories based on combinations of other class expressions and properties. For example, the concept “all entities that live in some City” can be constructed using an expression known as an existential restriction, which describes a relationship which all members of the class must have (in Manchester syntax, *livesIn some City*). By constructing the intersection of that expression with the class *Person*, we can create an expression representing the class of city dwellers: *Person and (livesIn some City)*. Named classes,

e.g., *CityDweller*, can be linked to expressions defining them using an equivalent class axiom: *CityDweller EquivalentTo (Person and (livesIn some City))*. Classes can also be related hierarchically to one another via a subclass axiom (e.g., *City SubClassOf GeographicalRegion*), and declared to have no instances in common (e.g., *GeographicalRegion DisjointWith Person*). Likewise, OWL supports hierarchical relationships among properties, as well as property features such as transitivity and property chains, which allow us to infer new relationships between individuals that are linked by a sequence of intervening relationships. A familiar property chain example would be inference of *hasUncle* from a path: *hasParent*  $\circ$  *hasBrother*  $\rightarrow$  *hasUncle*.

An OWL reasoner can be used to perform a number of tasks with regard to an OWL ontology, by computing the implied consequences of the asserted axioms. One of the most common tasks is Tbox (terminology) *classification*, that is, computing the hierarchical relationships (subsumptions) among all classes in the ontology. Checking if any classes are inferred to be equivalent to `owl:Nothing` (the empty class) is a common quality control task for OWL-based terminologies. Another common task is Abox (assertions) *materialization*: computing all inferred relationships between individuals, as well as the inferred types of the individuals (their class membership). *Consistency checking* evaluates whether the ontology contains a logical impossibility, based on the provided axioms. Different OWL applications may focus only on a subset of reasoning tasks. Development of large bio-ontologies is often solely focused on classification. Developers of an anatomical terminology may make heavy use of class expressions and inferred subsumptions to ensure consistent modeling and automatic calculation of a complex hierarchy, but never create instances of the terms. On the other hand, other ontology use cases may be more focused on instance graphs, computing inferred relationships between individuals, possibly using a less complex schema-like terminology. An OWL reasoner may also be used to answer queries over the inferred knowledge, returning all subclasses known for a given class, or all individuals which are instances of a given class. Queries using complex class expressions are commonly referred to as DL (description logic) queries.

Even though the complete OWL DL language is decidable, in practice for many tasks DL reasoning over ontologies of non-trivial size is time and compute-intensive, and often infeasible. For this reason OWL provides a number of profiles (language subsets), each of which limits the language in specific ways in order to allow more efficient reasoning. The profiles are designed to provide adequate expressivity for particular use cases. For example, the OWL EL profile provides logical features commonly used in the development of large complex terminologies, such as bio-ontologies, where the focus is ontology classification based on existential restrictions and intersections, and quality control using disjoint classes axioms. The OWL RL profile is more targeted to inference of relationships among large numbers of individuals, providing additional property features such as inverse properties and functional properties, while at the same time having fewer features for inferring class hierarchies as compared to EL.

### 3 Features and implementation

Whelk is implemented in Scala, a programming language for the Java Virtual Machine (JVM), which is fully interoperable with Java and has strong support for functional programming, plus language constructs that encourage immutable data structures [44]. Whelk provides an implementation of the `OWLReasoner` interface defined by the Java OWL API [20], making it readily usable within popular software for working with OWL such as Protégé [31] and ROBOT [23]. Whelk can also be used within pure Scala programs without reliance on the OWL API. Via Scala.js [45], it can be used as part of browser-based JavaScript applications, and can also be compiled to native code using Scala Native [43].

Whelk supports all axiom types within the OWL EL and RL profiles [42], with the limitation that reasoning about data property values (concrete values such as strings or numbers) is not supported. Additionally, `HasKey` axioms are not supported. Whelk also extends OWL EL and RL reasoning with SWRL rules (again with the exception of data property values). SWRL rules allow matching arbitrary patterns of class assertions and object property assertions to generate new inferred class and object property assertions about individuals.

### **3.1 Parallel extension of reasoning state**

As described above, Whelk is built on immutable data structures which return a new instance when modified, rather than allowing mutation. Implementations based on shared structure allow reasonable performance and efficient use of memory [32]. A Whelk reasoner instance is initialized with a starting set of axioms. All reasoning rules are applied, and a reasoning state object is returned containing the classification derived up to that point. This reasoning state can be extended with additional asserted axioms. Reasoning continues until classification is complete, and a new reasoning state is returned. Any references to the earlier reasoning state are still valid, and can be queried without reflecting conclusions derived from extension with the second set of axioms. Thus any number of independent extensions to the reasoning state can be created. This is much like programming with a singly linked list in a language like Lisp or Haskell; prepending a new item to a list of size 2 results in a new list of size 3, but doesn't affect references to the first list, which still consists of 2 elements.

When answering a DL query, the reasoning state is extended with an equivalent class axiom representing the query expression. Once additional reasoning is completed and the query is answered, the new reasoning state can simply be discarded in order to roll back to the initial state. This approach provides Whelk with very fast DL query performance. Since the reasoning state is immutable, any number of queries can be processed simultaneously without interference. Currently, while any kind of axiom can be provided in the starting set, only class (Tbox) and individual (Abox) axioms are supported when extending reasoning states. Adding new property (Rbox) axioms or SWRL rules after the initial classification is complete requires a complete reclassification (a limitation shared with ELK). The approach described for processing DL queries works equally well for other use cases, such as extending a reasoning state representing a large terminology with various sets of Abox axioms describing individuals instantiating that terminology, or applying sets of additional Tbox axioms representing alternative conceptions of a domain.

### **3.2 Supported reasoning tasks**

Like ELK, Whelk supports standard OWL reasoning tasks such as classification, coherency and consistency checking, and queries for subclasses, superclasses, or instances of arbitrary class expressions. In addition, Whelk is also able to materialize all inferred relationships between individuals (object property assertions), a feature not directly supported by ELK.

### **3.3 Reasoning implementation**

#### **3.3.1 OWL EL**

Whelk's EL reasoning is based on the inference rules detailed in figure 3 of Kazakov et al.[24]. Although the concrete implementation looks quite different from the source code of ELK due to the choice of language and use of immutable data structures, Whelk's implementation closely follows Algorithm 2 of Kazakov et al., taking an expression from the queue and applying each rule in turn, adding any generated expressions to the queue. Whelk's Scala code is compact and

attempts to transform the ELK rules to programming code as directly as possible. As one example, ELK's rule  $R_{\sqcap}$  handles the case that a concept that is a subclass of an intersection expression should be inferred to be a subclass of each of the concepts in the intersection:

$$\frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}$$

The Scala version of this rule is a function that accepts a concept inclusion (subclass inference) from the queue, along with the current reasoner state (which holds various indices providing fast lookup into the ontology and computed inferences) and the queue collecting produced expressions to which rules will be applied. If the superclass in the concept inclusion is an intersection (called a `Conjunction` in the data model), then two new concept inclusions are added to the queue:

```
def ruleMinConj(
  ci: ConceptInclusion,
  reasoner: ReasonerState,
  todo: Stack[QueueExpression]): ReasonerState =
  ci match {
    case ConceptInclusion(sub, Conjunction(left, right)) =>
      todo.push(ConceptInclusion(sub, left))
      todo.push(ConceptInclusion(sub, right))
      reasoner
    case _ => reasoner
  }
```

ELK 0.6.0 improved its coverage of OWL EL by adding support for property range axioms. The latest release of *Whelk* also supports the use of property ranges. ELK currently lacks complete support for “self restrictions”, that is, class expressions stating that all instances of that class have a specific property relation to themselves. These expressions are fully supported by *Whelk*, which enables the use of a useful pattern known as *rolification* [25], in which a property acts as a sort of marker for a class. As one example, the OBO Relation Ontology (RO) includes the following axioms defining a *rolification* property for the class *KinaseActivity* and using it in a property chain:

*KinaseActivity SubClassOf (isKinaseActivity some Self)*

*capableOf*  $\circ$  *isKinaseActivity*  $\circ$  *hasDirectInput*  $\rightarrow$  *phosphorylates*

This in effect defines a property chain *capableOf*  $\circ$  *hasDirectInput* that is only matched when the intermediate node has the type *KinaseActivity*.

### 3.3.2 OWL RL

As stated above, *Whelk* extends the ELK design with support for the OWL RL profile. Handling of OWL RL axiom types that are not included within the OWL EL profile is accomplished via three approaches. First, certain OWL RL axioms, while not explicitly included in OWL EL, can be transformed to equivalent EL constructs. For example, OWL EL does not include union class expressions (e.g., *Animal or Plant*). These expressions are allowable within OWL RL, but only when used on the subclass side of subclass axioms. By asserting a subclass for such expressions for each of the union operands when loading the ontology, we can obtain the inferences supported by OWL RL using the ELK reasoning rules. Thus, if an expression such as *C or D* appears in the ontology, we need only to inject these axioms:

*C SubClassOf (C or D)*

$D \text{ SubClassOf } (C \text{ or } D)$

For OWL RL complement expressions, e.g., *not C*, we inject:

$(C \text{ and } (\text{not } C)) \text{ SubClassOf } \text{owl:Nothing}$

And for OWL RL cardinality restrictions of cardinality 0, e.g., *p max 0 C*, we inject:

$(p \text{ max } 0 C) \text{ and } (p \text{ some } C) \text{ SubClassOf } \text{owl:Nothing}$

The first two transformations are also supported by ELK [14].

Secondly, support for the remaining OWL RL class expression constructs is provided by additional rules implemented similarly to the standard ELK rules. These include “all values from” restrictions and cardinality restrictions of cardinality 1. For example, the rule for “all values from” can be written in the style used in Kazakov et al., where *i* and *j* are individuals:

$$\frac{i \sqsubseteq \forall R.C \quad i \underline{R} j}{j \sqsubseteq C}$$

In this rule,  $i \underline{R} j$  is a *link*, a type of conclusion representing existential restrictions used in the ELK reasoning rules; a link between two individuals is equivalent to an object property assertion. As part of Abox materialization, Whelk generates all inferred links between individuals, so that there is no need for this rule to consider the property hierarchy. This rule is implemented by two Scala functions similar to the above example: one to handle newly generated concept inclusions, and one to handle newly generated links.

Lastly, other OWL RL axiom types are transformed to equivalent SWRL rules, and handled by Whelk’s SWRL rule engine. For example, the inverse properties axiom *p inverseOf r* is transformed to these SWRL rules, where leading question marks indicate variables:

$$p(?x, ?y) \rightarrow r(?y, ?x)$$

$$r(?x, ?y) \rightarrow p(?y, ?x)$$

### 3.3.3 SWRL rule engine

The Whelk SWRL rule engine is implemented as an extension of the EL reasoner. It supports inference based on user-defined rules which can match patterns of individual types and relationships (as above, reasoning with datatype property values is not currently supported). An example of such a rule included in the OBO Relation Ontology is one that infers “phosphorylates” relationships between gene product instances:

$$\begin{aligned} & \text{directlyRegulates}(?a1, ?a2) \wedge \text{KinaseActivity}(?a1) \wedge \text{enabledBy}(?a1, ?g1) \\ & \wedge \text{enabledBy}(?a2, ?g2) \rightarrow \text{phosphorylates}(?g1, ?g2) \end{aligned}$$

SWRL rules perform instance-level reasoning; that is, they match and produce class assertions and property assertions for individuals. As stated above, Whelk generates SWRL rules for certain OWL RL axioms; these ensure that all inferred relations between individuals are materialized. Like the EL reasoner, the SWRL rule engine works on a queue of produced expressions. When an expression is taken from the EL reasoner queue, if it is a concept inclusion involving an individual, or a link where the subject and object are individuals, it is also added to the SWRL engine queue. The SWRL rule engine is an implementation of the widely used Rete pattern-matching algorithm first developed by Forgy [15] and described in detail by Doorenbos [13]. Its design is an adaptation



of our earlier work on an RDF rule engine [4]. When the rule engine is constructed, it builds a tree of join nodes, where each node represents a pattern occurring in the body of a rule, linked in such a way that a path from the root to a leaf represents a complete rule body. As concept inclusions and links are taken from the queue, they are sent to any join nodes which use the same class or property predicate. Join nodes in turn check their predecessor join nodes (if any) for compatible partial solutions, and if found, activate successor join nodes. The final node in a tree branch is a production node representing a rule head, which may generate a new concept inclusion or link (representing class assertions and object property assertions), which is added to the EL reasoner queue.

This integration does result in some redundant derivation of inferences. For example, while the ELK reasoning algorithm handles transitive properties and property chains, it attempts to derive only the links required for complete classification of named classes in the ontology [24]. In order to compute a complete set of OWL RL inferences and to materialize all inferred object property assertions, we additionally create SWRL rules for transitive properties and property chains for use in the rule engine. While there may exist a more efficient approach, the integration nonetheless provides the capability for computing Abox inferences while working with terminologies requiring an OWL EL reasoner for scalability, a feature not available from most EL reasoners.

## 4 Evaluation

### 4.1 Testing

The Whelk codebase includes a suite of unit tests to verify that it derives identical subsumptions to ELK for OWL EL axioms. Inferences for OWL RL are compared to the output of the OWL reasoner Hermit [18] (which supports all OWL features but is not scalable for large ontologies) using test cases targeting the reasoning rules outlined in the OWL RL profile spec [42]. The test suite can be easily extended by adding new test ontologies to the repository.

### 4.2 Performance

We compared Whelk 1.2.1 to two versions of ELK: the long-established ELK 0.4.3, and the very recently released ELK 0.6.0, which adds support for object property range axioms. In the performance evaluations we make use of three ontologies (Table 1):

- *UNIV-BENCH-OWL2EL* – a benchmark Tbox covering the OWL EL profile [34].
- *uberon-go-cl-ro* – a merged set of ontologies from the OBO Foundry containing mutually referential axioms: Uberon anatomy ontology, Gene Ontology (GO), Cell Ontology (CL), Relation Ontology (RO) [30, 17, 12, 36]. Because the published versions of OBO ontologies typically include the precomputed classification, our test ontology was derived from the source version of each component ontology.
- *nci-thesaurus* – the NCI Thesaurus reference terminology from the National Cancer Institute [35].

Each performance evaluation was implemented as a custom Scala script. All testing scripts and input ontologies are available from the whelk-paper GitHub repository archived at <https://doi.org/10.5281/zenodo.13891879>. All performance tests were executed on an Apple MacBook Pro with an M2 Max chip with 12 cores, and 64 GB RAM. We set the maximum heap size for the Java Virtual Machine to 16 GB.

■ **Table 1** Characteristics of test ontologies.

Ontology	Classes	Object properties	Logical axioms
UNIV-BENCH-OWL2EL	131	82	398
uberon-go-cl-ro	70,057	656	108,610
nci-thesaurus	188,034	97	258,241

### 4.2.1 Computed subsumptions

Because the three reasoners differ slightly in support for axiom types, we filtered out axioms making use of `ObjectUnionOf` and `ObjectHasSelf` expressions as well as `ObjectPropertyRange` axioms. We programmatically verified that all three reasoners derived the same class subsumptions for each test ontology.

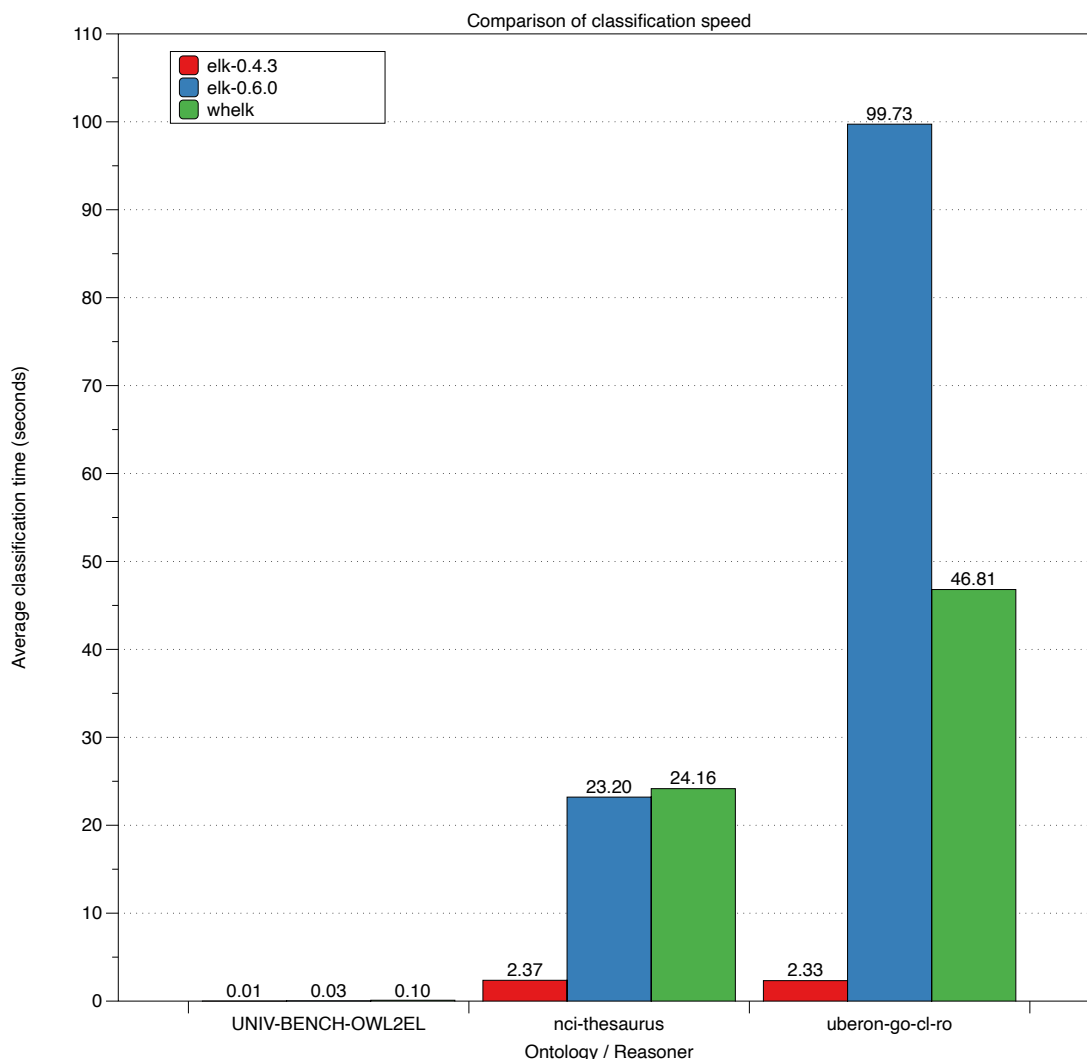
### 4.2.2 Ontology classification speed

We measured the time required for each reasoner to classify each of the input ontologies and answer a query to check the coherency of the ontology (“list any classes equivalent to `owl:Nothing`”). Within a single process for each reasoner and ontology combination, we computed the classification of each ontology 12 times. We discarded the first two runs (to allow warmup of the JVM) and computed the average of the remaining 10 runs. ELK’s classification algorithm is multi-threaded, while Whelk’s is single-threaded only; we allowed ELK to use all available CPUs. The UNIV-BENCH-OWL2EL ontology is so tiny that each reasoner takes only a fraction of a second, and so we exclude it from further performance tests. For classification of `nci-thesaurus`, ELK 0.4.3 is ~10 times faster than Whelk, completing the task in 2.4 seconds vs. 24.2 seconds for Whelk (Figure 1). On the other hand, ELK 0.6.0 is significantly slower than its previous release, comparable to Whelk with a time of 23.2 seconds. For `uberon-go-cl-ro`, ELK 0.4.3 outperforms Whelk by a larger margin (~20 times faster), at 2.3 seconds vs. 46.8 seconds. ELK 0.6.0 is the slowest for `uberon-go-cl-ro`, averaging 99.7 seconds. The decrease in performance between ELK 0.4.3 and 0.6.0 is unexpected, and we plan to investigate this issue with the developers.

### 4.2.3 DL query speed

In order to measure how quickly each reasoner can answer successive DL queries, for each ontology (`uberon-go-cl-ro` and `nci-thesaurus`) we extracted all complex class expressions used, at all levels of nesting. This procedure provides us with class expressions likely to represent relevant queries, rather than generating random combinations of classes and properties. After classifying the ontology, we queried the reasoner for subclasses of each class expression, using the ‘`getSubclasses`’ method of the `OWLReasoner` interface. As assurance that each reasoner returned the same subclasses, we collected the number of subclasses returned for each class expression and reported the sum. The query script submitted queries to the reasoner at each of three levels of parallelism: 1 (query all expressions sequentially), 4, or 8 workers. We measured the time required to answer all queries for all combinations of ontology, reasoner, and parallelism, averaging three runs following a warmup run.

For sequential queries, ELK 0.4.3 and 0.6.0 perform similarly, executing 46,685 queries against `nci-thesaurus` at 51 and 47 queries per second, respectively (Figure 2). In comparison, Whelk is able to execute 2306 queries per second. As expected, at higher levels of concurrent queries, ELK’s performance remains the same. Whelk’s query answering speed scales with the number of workers: 8159 queries per second using 4 workers; 13,679 queries per second using 8 workers.

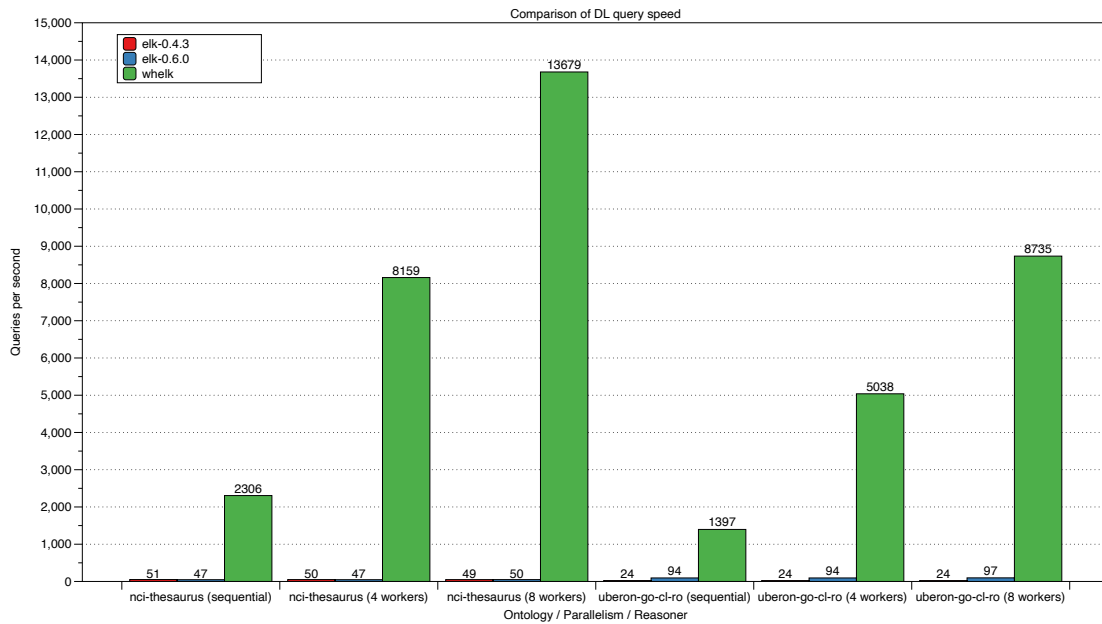


**Figure 1** Time required for each reasoner to classify and check coherency of three input ontologies (lower is better). ELK 0.4.3 is 10-20 times faster than Whelk for the two large ontologies that we tested (nci-thesaurus and uberon-go-cl-ro).

Results for executing 66,169 queries against uberon-go-cl-ro follow the same pattern, although in this case ELK 0.6.0 outperforms 0.4.3 (94 queries per second vs. 24, for sequential queries). Again Whelk is much faster at 1397 queries per second (sequentially) and 8735 queries per second (8 workers).

#### 4.2.4 Abox consistency checking speed

We next tested how quickly each reasoner could perform incremental reasoning when extending an ontology with multiple independent sets of Abox axioms, after having previously classified the Tbox. We used the uberon-go-cl-ro ontology as a Tbox for a collection of 4590 Gene Ontology Causal Activity models (GO-CAMs) [37]. This ontology contains axioms defining most of the biological concepts and relations used in the GO-CAMs. The GO-CAMs are small Abox ontologies of class assertions and object property assertions, containing an average of 43 logical axioms, with



**Figure 2** Query answering rate at different levels of parallelism of DL query submission (higher is better). ELK processes a single query at a time, while Whelk responds to parallel requests. Whelk’s sequential DL query speed is ~15-50 times greater than ELK for these two ontologies, and up to 279 times faster when handling 8 queries at a time.

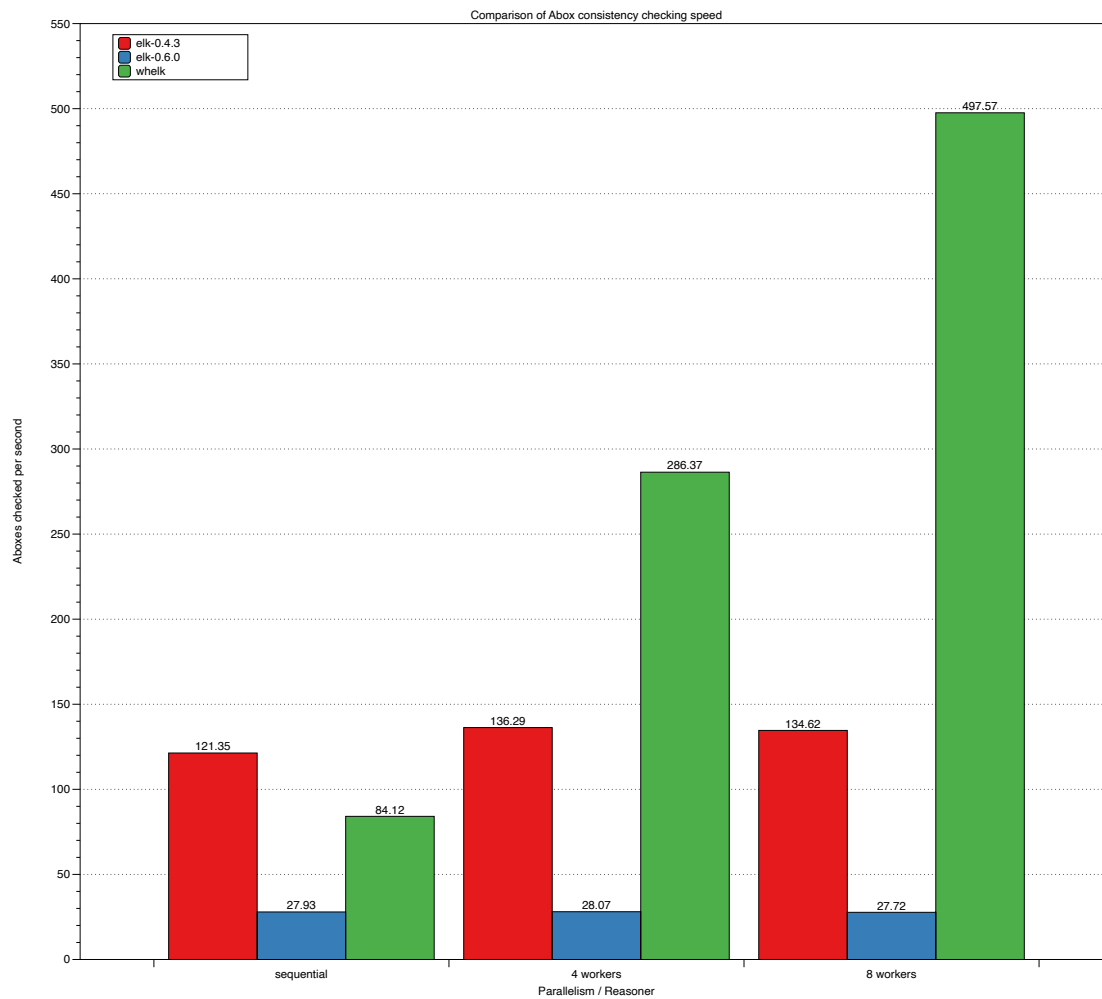
the smallest containing 4 and the largest containing 2590. Similarly to the DL query test, we used each reasoner to first classify the ontology, then added each Abox either sequentially or in parallel using 4 or 8 workers. We measured the time to classify and then query the consistency of all 4590 Aboxes, averaging three runs following a warmup run. For the two ELK reasoners, for each Abox we added its axioms to the base ontology using the OWL API and used the `OWLReasoner` “flush” method to trigger classification. After querying the consistency of the result, we removed the Abox axioms from the ontology. Because Whelk is designed particularly for this scenario, we used its Scala API directly rather than going through OWL API, which does not allow adding axioms in parallel.

Both ELK reasoners found 36 inconsistent Aboxes. Whelk detected 56 inconsistent Aboxes; this difference is expected since Whelk supports OWL RL constructs that ELK does not. In the sequential scenario, ELK 0.4.3’s greater classification speed allowed it to outperform the other reasoners, checking 121 Aboxes per second compared to 84 for Whelk and 28 for ELK 0.6.0 (Figure 3). As in the DL query test, performing the tasks in parallel did not provide appreciable benefit for the ELK reasoners, but allowed much higher throughput using Whelk, checking 286 Aboxes per second with 4 workers, and 498 Aboxes per second with 8 workers.

## 5 Applications

### 5.1 Relation graph materialization

A substantial number of ontology use cases in the biosciences translate to what we call “relation graph” questions, such as “what are the parts of the nucleus” or “where is this gene localized”. These can be translated to OWL subclass queries involving existential restrictions, e.g.  $?c \text{ SubClassOf } R \text{ some } D$  or  $C \text{ SubClassOf } R \text{ some } ?d$ . The latter is particularly challenging,



■ **Figure 3** Abox consistency checking rate at different levels of parallelism of axiom addition (higher is better). ELK processes a single incremental reasoning task at a time, while Whelk can extend its reasoning state in parallel.

as the way to answer this with standard OWL query interfaces is to test subsumption for different values of  $?d$ , rather than executing a single query. A relation graph in the sense we describe is illustrated in Figure 4, which shows a small subset of the Gene Ontology TBox depicting subclass axioms and subclass existential axioms as edges.

We previously implemented a system for computing such entailed existential relation edges using the OWL API and the ELK reasoner, but we found the performance did not scale to the level we needed. Constructing and classifying named versions of all combinations of properties and classes quickly generates a prohibitively large ontology. Performing successive DL queries can be done with a manageable, constant memory size, at the cost of a possibly long runtime. Using Whelk, we created a tool called “relation-graph”, which efficiently materializes every inferred relationship  $C \text{ SubClassOf } R \text{ some } D$  for all properties and classes from an input ontology. Relation-graph relies on Whelk’s much faster query answering performance, and also performs queries in parallel. Further, it makes use of the class and property hierarchies to avoid unnecessary queries. Entailed  $C \text{ SubClassOf } R \text{ some } D$  relationships are output as simple RDF triples  $C R D$ , which lend themselves to straightforward SPARQL queries that are logically complete after the relation-graph materialization.

Relation-graph is a crucial component of the Ubergraph construction pipeline, which generates an RDF knowledge graph combining many OBO library ontologies along with the full set of materialized relation graph edges [3]. The Ubergraph relation closure has proved to be a valuable resource for conveniently harnessing the logical semantics provided by its component ontologies [10, 19], and relation-graph itself is used to support a number of other applications [33, 38]. As part of the relation-graph tool, Whelk has been of critical value in making the Ubergraph reasoning precomputation feasible. Ubergraph is based on a large merged ontology, presently consisting of more than 5 million logical axioms, with almost 4 million named classes and more than 1000 object properties. It takes Whelk approximately 40 minutes to classify the ontology, while ELK 0.4.3 can classify this ontology in 99 seconds. However, in the course of building Ubergraph, the relation-graph tool first classifies the ontology and then uses Whelk to execute more than 90 million DL queries. In our tests, on this ontology ELK completes ~2.5 queries per second; at that rate it would take more than 400 days to complete this task, while this phase of the Ubergraph build completes in less than 8 hours using relation-graph with Whelk, making the extended classification time a worthwhile trade-off.

## 5.2 Reasoning with Aboxes and biomedical terminologies in Protégé and ROBOT

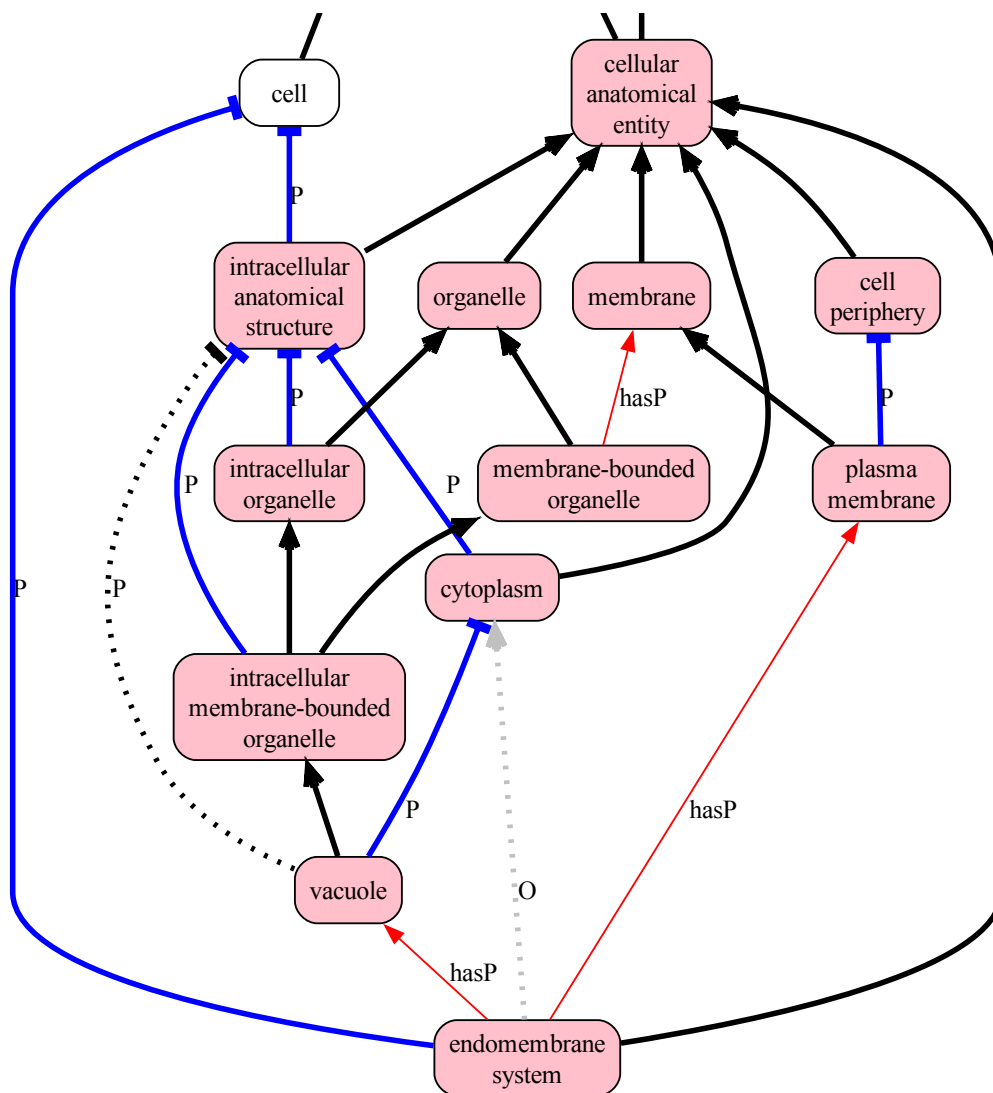
Whelk is the only reasoner available for the OWL API we are aware of which can efficiently classify large biomedical ontologies such as Uberon, Gene Ontology (GO), and NCI Thesaurus and also materialize inferred object property assertions. It therefore fills a valuable niche for those who are using such ontologies to reason over instance models, for example within the Gene Ontology GO-CAM project [37]. The GO Consortium uses GO-CAMs to describe the activity of gene products within cellular processes, using OWL to provide a much more expressive modeling capability than traditional flat gene-to-term associations. The modeling in GO-CAMs relies on the rich property axiomatization in the OBO Relation Ontology, including inverse property axioms and SWRL rules. The core reference terminology, combining GO, Uberon, CL, RO, and several other ontologies, consists of nearly 1 million logical axioms. Being able to load this ontology into Protégé and classify a GO-CAM is invaluable for exploring modeling consequences or debugging unexpected inferences found in a particular model. Whelk can also be used to materialize Abox inferences within the command-line OWL tool ROBOT. This can be done for one of the GO-CAM files described above with a command such as the following:

```
robot merge -i uberon-go-cl-ro.ofn -i gocam-5b91dbd100000506.ttl \  
reason --reasoner whelk --axiom-generators "ClassAssertion PropertyAssertion" \  
-o inferred.ttl
```

Unexpected inferences can also be debugged within ROBOT, using the “explain” feature it shares in common with Protégé.

## 5.3 Reasoner-driven web services

We have integrated Whelk as a reasoner option within Owlery [5], an application for exposing OWL reasoner functionalities via a set of web services. Like Protégé and ROBOT, Owlery is built upon the Java OWL API and thus integration of any reasoner supporting the OWLReasoner interface is trivial. Owlery supports standard OWLReasoner queries such as subclasses, superclasses, and equivalent classes of submitted class expressions, returning the results in a JSON-LD format. A distinct advantage to using Whelk within such a web services application is that, while the initial



■ **Figure 4** Subgraph of the GO Tbox focused on “endomembrane system”, rendered as a relation graph. Solid lines indicate asserted axioms and dashed lines entailed. Black: *SubClassOf*; Blue (P): *C SubClassOf partOf some D*; Red (hasP): *C SubClassOf hasPart some D*. Grey (O): *C SubClassOf overlaps some D*. The entailed dashed lines follow from the indicated axioms plus: *partOf* and *hasPart* are transitive, and there is a property chain  $hasPart \circ partOf \rightarrow overlaps$ .

ontology classification at startup may be slower, such a service can then run indefinitely, and subsequent queries to the reasoner are non-blocking, allowing scaling to a much higher level of concurrent traffic as demonstrated by the DL queries tests above (figure 2).

In addition to server-side web services, as noted above Whelk can be compiled to JavaScript using Scala.js. As far as we are aware, Whelk is the only OWL reasoner available for use within web browser client-side code. We provide a demonstration at <https://balhoff.github.io/whelk-web/>.

## 5.4 Testing hypothetical axioms

We previously implemented a system (k-BOOM [29]) for converting ontology term mappings into precise logical relationships; k-BOOM generates hypothetical axioms representing possible interpretations of a set of mappings, and attempts to find the set of interpretations which is both logically coherent and has the highest joint probability. This system was used to construct the initial version of the Mondo, a disease ontology which provides a unified logical view over several different source terminologies [39]. The original version of k-BOOM, based on ELK, required more than a day of runtime to analyze the term mapping inputs for Mondo. We have implemented a new tool, boomer [8], which is based on Whelk and can perform the same task in a matter of minutes.

## 6 Discussion

While ELK 0.4.3 provides much higher performance for ontology classification, Whelk’s design allows it to target use cases for which ELK does not perform as well. As shown above, these use cases primarily involve concurrent extension of existing reasoner states, although even for sequential DL queries without parallelism, Whelk’s design proves to result in very high performance. While our work directly reuses the rules for inference defined in the ELK publication, the success of Whelk in supporting the particular scenarios described here brings to light the value in exploring alternative reasoner implementations targeting different software ecosystems or performance use cases. Unfortunately, a recent study shows that only 25 of 73 tested OWL reasoner implementations are usable and actively maintained [1]. Many OWL reasoners have begun life as research prototypes providing a single implementation, and very few have grown into community-developed open source projects (although there are exceptions, for example Openllet [11]).

While Whelk is primarily maintained by a single developer, it is now used within a number of different applications supporting life sciences research projects making use of ontology-based knowledge graphs, which support its continued development. The Whelk codebase is fairly compact (the core EL reasoning rules comprise less than 500 lines of Scala), and we have created preliminary ports to other languages, including Rust [6], allowing it to be used with the recently developed horned-owl package [26].

Whelk’s development and extensions to the ELK inference rules have been driven by a pragmatic approach. For example, the EL and RL reasoning engines derive some duplicate inferences. While we would welcome input on a more efficient integration, the current implementation returns sound results and provides useful Abox inference features as an add-on to OWL EL.

In this paper we have explored the utility of the Whelk reasoner almost entirely in contrast to its precursor, ELK. This is a testament to how universally important the ELK reasoner has been to the bio-ontology ecosystem, allowing reasoner-driven quality control pipelines to become the norm for many widely used ontologies. ELK still remains vitally important for those workflows, as well as for most interactive terminology development within Protégé. But for the additional types of use cases described here, Whelk provides distinct advantages.

---

## References

- 1 Konrad Abicht. Owl reasoners still useable in 2023, 2023. doi:10.48550/arXiv.2309.06888.
- 2 Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2 edition, 2007.
- 3 J Balhoff, Ugur Bayindir, Anita R Caron, N Matentzoglou, David Osumi-Sutherland, and C Mungall. Ubergraph: integrating OBO ontologies into a unified semantic graph. In *ICBO 2022: International Conference on Biomedical Ontology (ICBO)*, 2022. doi:10.5281/zenodo.7249759.



- 4 J Balhoff, Benjamin M Good, S Carbon, and C Mungall. Arachne: An OWL RL reasoner applied to gene ontology causal activity models (and beyond). In *17th International Semantic Web Conference (ISWC 2018)*, October 2018. doi:10.5281/zenodo.2397192.
- 5 James P Balhoff. owlery: Owlery is a set of REST web services which allow querying of an OWL reasoner containing a configured set of ontologies. Accessed: 2024-6-29. URL: <https://github.com/phenoscape/owlery>.
- 6 James P Balhoff. whelk-rs: Whelk is an OWL EL reasoner. Accessed: 2024-6-29. URL: <https://github.com/INCATools/whelk-rs>.
- 7 James P. Balhoff. Incatools/whelk-paper, October 2024. Software, v1.1.0. doi:10.5281/zenodo.13891879.
- 8 James P Balhoff and Christopher J Mungall. boomer: Bayesian OWL ontology merging. Accessed: 2024-6-29. URL: <https://github.com/INCATools/boomer>.
- 9 James P. Balhoff and Christopher J. Mungall. Whelk reasoner. Software, version 1.2.1., swhdl: swh:1:dir:f8919707e053212b2c74bc63988a06ffe03fb796 (visited on 2024-12-10). URL: <https://github.com/INCATools/whelk>, doi:10.4230/artifacts.22615.
- 10 Katy Börner, Sarah A Teichmann, Ellen M Quardokus, James C Gee, Kristen Browne, David Osumi-Sutherland, Bruce W Herr, 2nd, Andreas Bueckle, Hrishikesh Paul, Muzlifah Haniffa, Laura Jardine, Amy Bernard, Song-Lin Ding, Jeremy A Miller, Shin Lin, Marc K Halushka, Avinash Boppana, Teri A Longacre, John Hickey, Yiing Lin, M Todd Valerius, Yongqun He, Gloria Pryhuber, Xin Sun, Marda Jorgensen, Andrea J Radtke, Clive Wasserfall, Fiona Ginty, Jonhan Ho, Joel Sunshine, Rebecca T Beuschel, Maigan Brusko, Sujin Lee, Rajeev Malhotra, Sanjay Jain, and Griffin Weber. Anatomical structures, cell types and biomarkers of the human reference atlas. *Nature cell biology*, 23(11):1117–1128, November 2021. doi:10.1038/s41556-021-00788-6.
- 11 Openlet code repository. openlet: Openlet is an OWL 2 reasoner in java, build on top of pellet. Accessed: 2024-6-29. URL: <https://github.com/Galigator/openlet>.
- 12 Alexander D Diehl, Terrence F Meehan, Yvonne M Bradford, Matthew H Brush, Wasila M Dahdul, David S Dougall, Yongqun He, David Osumi-Sutherland, Alan Ruttenberg, Sirarat Sarntivijai, Ceri E Van Slyke, Nicole A Vasilevsky, Melissa A Haendel, Judith A Blake, and Christopher J Mungall. The cell ontology 2016: enhanced content, modularization, and ontology interoperability. *Journal of biomedical semantics*, 7(1):44, July 2016. doi:10.1186/s13326-016-0088-7.
- 13 Robert B Doorenbos. *Production Matching for Large Learning Systems*. PhD thesis, Carnegie Mellon University, 1995. URL: <http://reports-archive.adm.cs.cmu.edu/anon/1995/CMU-CS-95-113.pdf>.
- 14 ELK code repository. Elk: Owl features. Accessed: 2024-10-04. URL: <https://github.com/liveontologies/elk-reasoner/wiki/OwlFeatures>.
- 15 Charles L Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence*, 19(1):17–37, September 1982. doi:10.1016/0004-3702(82)90020-0.
- 16 Michael A Gargano, Nicolas Matentzoglou, Ben Coleman, Eunice B Addo-Lartey, Anna V Anagnostopoulos, Joel Anderton, Paul Avillach, Anita M Bagley, Eduard Bakštein, James P Balhoff, Gareth Baynam, Susan M Bello, Michael Berk, Holli Bertram, Somer Bishop, Hannah Blau, David F Bodenstern, Pablo Botas, Kaan Boztug, Jolana Čady, Tiffany J Callahan, Rhiannon Cameron, Seth J Carbon, Francisco Castellanos, J Harry Caulfield, Lauren E Chan, Christopher G Chute, Jaime Cruz-Rojo, Noémi Dahan-Oliel, Jon R Davids, Maud de Dieuleveult, Vinicius de Souza, Bert B A de Vries, Esther de Vries, J Raymond DePaulo, Beata Derfalvi, Ferdinand Dhombres, Claudia Diaz-Byrd, Alexander J M Dingemans, Bruno Donadille, Michael Duyzend, Reem Elfeky, Shahim Essaid, Carolina Fabrizzi, Giovanna Fico, Helen V Firth, Yun Freudenberg-Hua, Janice M Fullerton, Davera L Gabriel, Kimberly Gilmour, Jessica Giordano, Fernando S Goes, Rachel Gore Moses, Ian Green, Matthias Griese, Tudor Groza, Weihong Gu, Julia Guthrie, Benjamin Gyori, Ada Hamosh, Marc Hanauer, Kateřina Hanušová, Yongqun Oliver He, Harshad Hegde, Ingo Helbig, Kateřina Holasová, Charles Tapley Hoyt, Shangzhi Huang, Eric Hurwitz, Julius O B Jacobsen, Xiaofeng Jiang, Lisa Joseph, Kamyar Keramatian, Bryan King, Katrin Knoflach, David A Koolen, Megan L Kraus, Carlo Kroll, Maaike Kusters, Markus S Ladewig, David Lagorce, Meng-Chuan Lai, Pablo Lapunzina, Bryan Laraway, David Lewis-Smith, Xiarong Li, Caterina Lucano, Marzieh Majd, Mary L Marazita, Victor Martinez-Glez, Toby H McHenry, Melvin G McInnis, Julie A McMurry, Michaela Mihulová, Caitlin E Millett, Philip B Mitchell, Veronika Moslerová, Kenji Narutomi, Shahrzad Nematollahi, Julian Nevado, Andrew A Nierenberg, Nikola Novák Čajbiková, John I Nurnberger, Jr, Soichi Ogishima, Daniel Olson, Abigail Ortiz, Harry Pachajoa, Guiomar Perez de Nanclares, Amy Peters, Tim Putman, Christina K Rapp, Ana Rath, Justin Reese, Lauren Rekerle, Angharad M Roberts, Suzy Roy, Stephan J Sanders, Catharina Schuetz, Eva C Schulte, Thomas G Schulze, Martin Schwarz, Katie Scott, Dominik Seelow, Berthold Seitz, Yiping Shen, Morgan N Similuk, Eric S Simon, Balwinder Singh, Damian Smedley, Cynthia L Smith, Jake T Smolinsky, Sarah Sperry, Elizabeth Stafford, Ray Stefancsik, Robin Steinhaus, Rebecca Strawbridge, Jagadish Chandrabose Sundaramurthi, Polina Talapova, Jair A Tenorio Castano, Pavel Tesner, Rhys H Thomas, Audrey Thurm, Marek Turnovec, Marielle E van Gijn, Nicole A Vasilevsky, Markéta Vlčková, Anita Walden, Kai Wang, Ron Wapner, James S Ware, Addo A Wiafe, Samuel A Wiafe, Lisa D Wiggins, Andrew E Williams, Chen Wu, Margot J Wyrwoll, Hui Xiong, Nefize Yalin, Yasunori Yamamoto, Lakshmi N Yatham, Anastasia K Yocum, Allan H Young, Zafer Yüksel, Peter P Zandi, Andreas Zankl, Ignacio Zarante, Miroslav Zvolský, Sabrina Toro, Leigh C Car-

- mody, Nomi L Harris, Monica C Munoz-Torres, Daniel Danis, Christopher J Mungall, Sebastian Köhler, Melissa A Haendel, and Peter N Robinson. The human phenotype ontology in 2024: phenotypes around the world. *Nucleic acids research*, 52(D1):D1333–D1346, January 2024. doi:10.1093/nar/gkad1005.
- 17 Gene Ontology Consortium, Suzi A Aleksander, James Balhoff, Seth Carbon, J Michael Cherry, Harold J Drabkin, Dustin Ebert, Marc Feuermann, Pascale Gaudet, Nomi L Harris, David P Hill, Raymond Lee, Huaiyu Mi, Sierra Moxon, Christopher J Mungall, Anushya Muruganugan, Tremayne Mushayahama, Paul W Sternberg, Paul D Thomas, Kimberly Van Auken, Jolene Ramsey, Deborah A Siegele, Rex L Chisholm, Petra Fey, Maria Cristina Aspromonte, Maria Victoria Nugnes, Federica Quaglia, Silvio Tosatto, Michelle Giglio, Suvarna Nadendla, Giulia Antonazzo, Helen Attrill, Gil Dos Santos, Steven Marygold, Victor Strelets, Christopher J Tabone, Jim Thurmond, Pinglei Zhou, Saadullah H Ahmed, Praoparn Asanitthong, Diana Luna Buitrago, Meltem N Erdol, Matthew C Gage, Mohamed Ali Kadhum, Kan Yan Chloe Li, Miao Long, Aleksandra Michalak, Angeline Pesala, Armalya Pritazahra, Shirin C C Saverimuttu, Renzhi Su, Kate E Thurlow, Ruth C Lovering, Colin Logie, Snezhana Oliferenko, Judith Blake, Karen Christie, Lori Corbani, Mary E Dolan, Harold J Drabkin, David P Hill, Li Ni, Dmitry Sitnikov, Cynthia Smith, Alayne Cuzick, James Seager, Laurel Cooper, Justin Elser, Pankaj Jaiswal, Parul Gupta, Pankaj Jaiswal, Sushma Naithani, Manuel Lera-Ramirez, Kim Rutherford, Valerie Wood, Jeffrey L De Pons, Melinda R Dwinell, G Thomas Hayman, Mary L Kaldunski, Anne E Kwitek, Stanley J F Laulederkind, Marek A Tutaj, Mahima VEDI, Shur-Jen Wang, Peter D'Eustachio, Lucila Aimo, Kristian Axelsen, Alan Bridge, Nevila Hyka-Nouspikel, Anne Morgat, Suzi A Aleksander, J Michael Cherry, Stacia R Engel, Kalpana Karra, Stuart R Miyasato, Robert S Nash, Marek S Skrzypek, Shuai Weng, Edith D Wong, Erika Bakker, Tanya Z Berardini, Leonore Reiser, Andrea Auchincloss, Kristian Axelsen, Ghislaine Argoud-Puy, Marie-Claude Blatter, Emmanuel Boutet, Lionel Breuza, Alan Bridge, Cristina Casals-Casas, Elisabeth Coudert, Anne Estreicher, Maria Livia Famiglietti, Marc Feuermann, Arnaud Gos, Nadine Gruaz-Gumowski, Chantal Hulo, Nevila Hyka-Nouspikel, Florence Jungo, Philippe Le Mercier, Damien Lieberherr, Patrick Masson, Anne Morgat, Ivo Pedruzzi, Lucille Pourcel, Sylvain Poux, Catherine Rivoire, Shyamala Sundaram, Alex Bateman, Emily Bowler-Barnett, Hema Bye-A-Jee, Paul Denny, Alexandr Ignatchenko, Rizwan Ishtiaq, Antonia Lock, Yvonne Lussi, Michele Magrane, Maria J Martin, Sandra Orchard, Pedro Raposo, Elena Speretta, Nidhi Tyagi, Kate Warner, Rossana Zaru, Alexander D Diehl, Raymond Lee, Juancarlos Chan, Stavros Diamantakis, Daniela Raciti, Magdalena Zarowiecki, Malcolm Fisher, Christina James-Zorn, Virgilio Ponzferrada, Aaron Zorn, Sridhar Ramachandran, Leyla Ruzicka, and Monte Westerfield. The gene ontology knowledgebase in 2023. *Genetics*, 224(1), May 2023. doi:10.1093/genetics/iyad031.
  - 18 Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: An OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, October 2014. doi:10.1007/s10817-014-9305-1.
  - 19 Bruce W Herr, 2nd, Josef Hardi, Ellen M Quardokus, Andreas Bueckle, Lu Chen, Fusheng Wang, Anita R Caron, David Osumi-Sutherland, Mark A Musen, and Katy Börner. Specimen, biological structure, and spatial ontologies in support of a human reference atlas. *Scientific data*, 10(1):171, March 2023. doi:10.1038/s41597-023-01993-8.
  - 20 Matthew Horridge and Sean Bechhofer. The OWL API: A java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011. doi:10.3233/SW-2011-0025.
  - 21 Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosz, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. URL: <https://www.w3.org/Submission/SWRL/>.
  - 22 SNOMED International. SNOMED international. Accessed: 2024-6-27. URL: <https://www.snomed.org/>.
  - 23 Rebecca C Jackson, James P Balhoff, Eric Douglass, Nomi L Harris, Christopher J Mungall, and James A Overton. ROBOT: A tool for automating ontology workflows. *BMC bioinformatics*, 20(1):407, July 2019. doi:10.1186/s12859-019-3002-3.
  - 24 Yevgeny Kazakov, Markus Krötzsch, and František Simančík. The incredible ELK. *Journal of Automated Reasoning*, 53(1):1–61, November 2013. doi:10.1007/s10817-013-9296-3.
  - 25 Adila Krisnadhi, Frederick Maier, and Pascal Hitzler. OWL and rules. In Axel Polleres, Claudia d'Amato, Marcelo Arenas, Siegfried Handschuh, Paula Kroner, Sascha Ossowski, and Peter Patel-Schneider, editors, *Reasoning Web. Semantic Technologies for the Web of Data: 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*, pages 382–415. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-23032-5\_7.
  - 26 Phil Lord. horned-owl. Accessed: 2024-6-30. URL: <https://github.com/philord/horned-owl>.
  - 27 Nicolas Matentzoglou, Damien Goutte-Gattat, Shawn Zheng Kai Tan, James P Balhoff, Seth Carbon, Anita R Caron, William D Duncan, Joe E Flack, Melissa Haendel, Nomi L Harris, William R Hogan, Charles Tapley Hoyt, Rebecca C Jackson, Hyeongsik Kim, Huseyin Kir, Martin Laralde, Julie A McMurry, James A Overton, Bjoern Peters, Clare Pilgrim, Ray Stefancsik, Sofia Mc Robb, Sabrina Toro, Nicole A Vasilevsky, Ramona Walls, Christopher J Mungall, and David Osumi-Sutherland. Ontology development kit: a toolkit for building, maintaining and standardizing biomedical ontologies. *Database: the journal of biological databases and curation*, 2022, October 2022. doi:10.1093/database/baac087.
  - 28 C J Mungall, H Dietze, and D Osumi-Sutherland. Use of OWL within the gene ontology.

- Keet and Valentina Tamma, editors, *Proceedings of the 11th International Workshop on OWL: Experiences and Directions (OWLED 2014)*, pages 25–36, Riva del Garda, Italy, October 17–18, 2014, October 2014. doi:10.1101/010090.
- 29 Christopher J Mungall, Sebastian Koehler, Peter Robinson, Ian Holmes, and Melissa Haendel. k-BOOM: A bayesian approach to ontology structure inference, with applications in disease ontology construction, May 2016. doi:10.1101/048843.
  - 30 Christopher J Mungall, Carlo Torniai, Georgios V Gkoutos, Suzanna E Lewis, and Melissa A Haendel. Uberon, an integrative multi-species anatomy ontology. *Genome biology*, 13(1):R5, January 2012. doi:10.1186/gb-2012-13-1-r5.
  - 31 Mark A Musen and Protégé Team. The protégé project: A look back and a look forward. *AI matters*, 1(4):4–12, June 2015. doi:10.1145/2757001.2757003.
  - 32 Chris Okasaki. *Purely Functional Data Structures*. PhD thesis, Carnegie Mellon University, 1996. URL: <https://www.cs.cmu.edu/~rwh/students/okasaki.pdf>.
  - 33 Tim E Putman, Kevin Schaper, Nicolas Matentzoglou, Vincent P Rubineti, Faisal S Alquaddoomi, Corey Cox, J Harry Caufield, Glass Elsarboukh, Sarah Gehrke, Harshad Hegde, Justin T Reese, Ian Braun, Richard M Bruskiwich, Luca Capelletti, Seth Carbon, Anita R Caron, Lauren E Chan, Christopher G Chute, Katherina G Cortes, Vinicius De Souza, Tommaso Fontana, Nomi L Harris, Emily L Hartley, Eric Hurwitz, Julius O B Jacobsen, Madan Krishnamurthy, Bryan J Laraway, James A McLaughlin, Julie A McMurry, Sierra A T Moxon, Kathleen R Mullen, Shawn T O’Neil, Kent A Shefchek, Ray Stefancsik, Sabrina Toro, Nicole A Vasilevsky, Ramona L Walls, Patricia L Whetzel, David Osumi-Sutherland, Damian Smedley, Peter N Robinson, Christopher J Mungall, Melissa A Haendel, and Monica C Munoz-Torres. The monarch initiative in 2024: an analytic platform integrating phenotypes, genes and diseases across species. *Nucleic acids research*, 52(D1):D938–D949, January 2024. doi:10.1093/nar/gkad1082.
  - 34 Gunjan Singh, Sumit Bhatia, and Raghava Mutharaju. OWL2Bench: A benchmark for OWL 2 reasoners. In *The Semantic Web – ISWC 2020*, pages 81–96. Springer International Publishing, 2020. doi:10.1007/978-3-030-62466-8\_6.
  - 35 Nicholas Sioutos, Sherri de Coronado, Margaret W Haber, Frank W Hartel, Wen-Ling Shaiu, and Lawrence W Wright. NCI thesaurus: a semantic model integrating cancer-related clinical and molecular information. *Journal of biomedical informatics*, 40(1):30–43, February 2007. doi:10.1016/j.jbi.2006.02.013.
  - 36 Barry Smith, Werner Ceusters, Bert Klagges, Jacob Köhler, Anand Kumar, Jane Lomax, Chris Mungall, Fabian Neuhaus, Alan L Rector, and Cornelius Rosse. Relations in biomedical ontologies. *Genome biology*, 6(5):R46, April 2005. doi:10.1186/gb-2005-6-5-r46.
  - 37 Paul D Thomas, David P Hill, Huaiyu Mi, David Osumi-Sutherland, Kimberly Van Auken, Seth Carbon, James P Balhoff, Laurent-Philippe Albou, Benjamin Good, Pascale Gaudet, Suzanna E Lewis, and Christopher J Mungall. Gene ontology causal activity modeling (GO-CAM) moves beyond GO annotations to structured descriptions of biological functions and systems. *Nature genetics*, 51(10):1429–1433, October 2019. doi:10.1038/s41588-019-0500-1.
  - 38 Santiago Timón-Reina, Mariano Rincón, Rafael Martínez-Tomás, Bjørn-Eivind Kirsebom, and Tormod Fladby. A knowledge graph framework for dementia research data. *NATO Advanced Science Institutes series E: Applied sciences*, 13(18):10497, September 2023. doi:10.3390/app131810497.
  - 39 Nicole A Vasilevsky, Nicolas A Matentzoglou, Sabrina Toro, Joseph E Flack, IV, Harshad Hegde, Deepak R Unni, Gioconda F Alyea, Joanna S Amberger, Larry Babb, James P Balhoff, Taylor I Bingaman, Gully A Burns, Orion J Buske, Tiffany J Callahan, Leigh C Carmody, Paula Carrio Cordo, Lauren E Chan, George S Chang, Sean L Christiaens, Michel Dumontier, Laura E Failla, May J Flowers, H Alpha Garrett, Jr, Jennifer L Goldstein, Dylan Gration, Tudor Groza, Marc Hanauer, Nomi L Harris, Jason A Hilton, Daniel S Himmelstein, Charles Tapley Hoyt, Megan S Kane, Sebastian Köhler, David Lagorce, Abbe Lai, Martin Larralde, Antonia Lock, Irene López Santiago, Donna R Maglott, Adriana J Malheiro, Birgit H M Meldal, Monica C Munoz-Torres, Tristan H Nelson, Frank W Nicholas, David Ochoa, Daniel P Olson, Tudor I Oprea, David Osumi-Sutherland, Helen Parkinson, Zoë May Pendlington, Ana Rath, Heidi L Rehm, Lyubov Remenik, Erin R Riggs, Paola Roncaglia, Justyne E Ross, Marion F Shadbolt, Kent A Shefchek, Morgan N Similuk, Nicholas Sioutos, Damian Smedley, Rachel Sparks, Ray Stefancsik, Ralf Stephan, Andrea L Storm, Doron Stupp, Gregory S Stupp, Jagadish Chandrabose Sundaramurthi, Imke Tammen, Darin Tay, Courtney L Thaxton, Eloise Valasek, Jordi Valls-Margarit, Alex H Wagner, Danielle Welter, Patricia L Whetzel, Lori L Whiteman, Valerie Wood, Colleen H Xu, Andreas Zankl, Xingmin Aaron Zhang, Christopher G Chute, Peter N Robinson, Christopher J Mungall, Ada Hamosh, and Melissa A Haendel. Mondo: Unifying diseases for the world, by the world, April 2022. doi:10.1101/2022.04.13.22273750.
  - 40 W3C OWL Working Group. Owl 2 web ontology language document overview (second edition). Accessed: 2024-10-04. URL: <https://www.w3.org/TR/owl2-overview/>.
  - 41 W3C OWL Working Group. Owl 2 web ontology language manchester syntax (second edition). Accessed: 2024-10-04. URL: <https://www.w3.org/TR/owl2-manchester-syntax/>.
  - 42 W3C OWL Working Group. OWL 2 web ontology language profiles (second edition). URL: <https://www.w3.org/TR/owl2-profiles/>.
  - 43 École Polytechnique Fédérale Lausanne (EPFL). Scala native. Accessed: 2024-6-25. URL: <https://scala-native.org/>.
  - 44 École Polytechnique Fédérale Lausanne (EPFL). The scala programming language. Accessed: 2024-6-30. URL: <https://www.scala-lang.org/>.
  - 45 École Polytechnique Fédérale Lausanne (EPFL). Scala.js. Accessed: 2024-6-25. URL: <https://www.scala-js.org/>.



# MELArt: A Multimodal Entity Linking Dataset for Art

Alejandro Sierra-Múnera   

Hasso Plattner Institute, University of Potsdam, Potsdam, Germany

Linh Le   

The University of Queensland, Brisbane, Australia

Gianluca Demartini   

The University of Queensland, Brisbane, Australia

Ralf Krestel   

ZBW – Leibniz Information Centre for Economics, Kiel, Germany

Kiel University, Kiel, Germany

## Abstract

Traditional named entity linking (NEL) tools have largely employed a general-domain approach, spanning across various entity types such as persons, organizations, locations, and events in a multitude of contexts. While multimodal entity linking datasets exist (e.g., disambiguation of person names with the help of photographs), there is a need to develop domain-specific resources that represent the unique challenges present in domains like cultural heritage (e.g., stylistic changes through time, diversity of social and political context). To address this gap, our work presents a novel multimodal entity linking benchmark dataset for the art domain together with a comprehensive experimental evaluation of existing NEL methods on this new dataset. The dataset encapsulates various entities unique to the art domain. During the dataset creation

process, we also adopt manual human evaluation, providing high-quality labels for our dataset. We introduce an automated process that facilitates the generation of this art dataset, harnessing data from multiple sources (Artpedia, Wikidata and Wikimedia Commons) to ensure its reliability and comprehensiveness. Furthermore, our paper delineates best practices for the integration of art datasets, and presents a detailed performance analysis of general-domain entity linking systems, when applied to domain-specific datasets. Through our research, we aim to address the lack of datasets for NEL in the art domain, providing resources for the development of new, more nuanced, and contextually rich entity linking methods in the realm of art and cultural heritage.

**2012 ACM Subject Classification** Computing methodologies → Information extraction

**Keywords and phrases** A Multimodal Entity Linking Dataset, Named Entity Linking, Art Domain, Wikidata, Wikimedia, Artpedia

**Digital Object Identifier** 10.4230/TGDK.2.2.8

**Category** Resource Paper

**Supplementary Material** The source code for generating MELArt is published on Github under an MIT license and the code for the experiments described in Section 4.3 is also published on Github. The annotations included in MELArt and the candidates' information except for the image files, are available on the UQ Research Data Manager under an CC-BY license. The image files can be extracted from Wikimedia using a file contained in the same repository and a script described in the Github repository.

*Dataset (Dataset):* <https://doi.org/10.48610/8a1ccdf> [10]

*Software (Dataset generation code):* <https://github.com/HPI-Information-Systems/MELArt> [11]  
archived at [swh:1:dir:ec4380448f4087c011040d0e3dca7832baa11182](https://swh.1.dir.ec4380448f4087c011040d0e3dca7832baa11182)

*Software (Experiments code):* [https://github.com/HPI-Information-Systems/MELArt\\_experiments](https://github.com/HPI-Information-Systems/MELArt_experiments) [12]  
archived at [swh:1:dir:203f2a69c5bc9064db3873a0160ca52f62095c25](https://swh.1.dir.203f2a69c5bc9064db3873a0160ca52f62095c25)



© Alejandro Sierra-Múnera, Linh Le, Gianluca Demartini, and Ralf Krestel;  
licensed under Creative Commons License CC-BY 4.0

Transactions on Graph Data and Knowledge, Vol. 2, Issue 2, Article No. 8, pp. 8:1–8:22



Transactions on Graph Data and Knowledge

TGDK Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**Funding** The article is based upon work conducted as part of the Australia–Germany Joint Research Cooperation Scheme (Universities Australia – DAAD). Project number 57600378.

*Alejandro Sierra-Múnera*: Funded by the HPI Research School on Data Science and Engineering

**Received** 2024-07-02 **Accepted** 2024-11-15 **Published** 2024-12-18

**Editors** Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler

**Special Issue** Resources for Graph Data and Knowledge

## 1 Introduction

The art world, rich in historical and cultural value, is facing a noticeable challenge in the application of computational techniques due to the limited availability of comprehensive datasets for machine learning tasks, despite the existence of some notable datasets, like, for example, The Metropolitan Museum of Art’s Open Access dataset [17], which offers over 397,121 images of public-domain artworks with rich metadata including 224,208 classes. Similarly, the Rijksmuseum dataset includes extensive collections of Dutch art, pivotal for studies in European art history. WikiArt [7] and Google’s Art & Culture datasets<sup>1</sup> provide broader scopes, encompassing diverse global art pieces. Additionally, Artemis [1] stands out as a unique dataset focusing on the emotional responses to art, offering a different dimension for analysis. Artpedia [13], further enriches the available resources for computational art research by associating paintings with corresponding visual and contextual sentences from Wikipedia. IICONGRAPH [9], defines a knowledge graph focused on iconographic and iconological statements for the Italian cultural-heritage landscape.

Despite these advancements, none of these resources are tailored for the task of entity linking or entity disambiguation, in which connections between art subject mentions and their corresponding entities in a knowledge graph can be automatically discovered.

Artpedia, as a comprehensive art resource, offers vast potential for developing an art entity linking dataset. It features a diverse collection of artworks, detailed metadata, high-quality images, and rich contextual information, making it a reliable source for art research [13]. The manual selection of text related to the artworks in Artpedia is its strength, enhanced by the manual classification between *visual* (describing what is depicted in the artwork) and *contextual* (describing other aspects of the artwork) sentences. The detailed descriptions and contextual information provided by Artpedia are vital for textual analysis, which, when combined with visual data, can significantly enhance entity recognition and linking accuracy. Utilizing Artpedia’s resources to develop an entity linking dataset would not only leverage the reliability of Artpedia’s textual descriptions, but also offer new insights into the complexity between visual elements and contextual information in artworks, advancing both the technological and cultural understanding of art.

Wikidata, although being a general purpose knowledge graph, contains specific relations that connect artworks with their corresponding subjects, and, compared to other domain-specific resources, covers more artworks with a good density of subjects per artwork [3]. This coverage, plus the tight connection between Artpedia, Wikipedia and Wikidata, suggests the potential to combine these resources and integrate textual and structured connections between artworks and subjects.

In this paper, we present a new dataset for multimodal (i.e., containing textual mentions and images) named entity linking in art that addresses the limitations of current datasets by: i) integrating Artpedia’s detailed textual descriptions of artworks to enhance the entity recognition

---

<sup>1</sup> <https://artsandculture.google.com/>

and linking; and ii) meticulously integrating Artpedia with Wikidata; iii) retrieving structured information about paintings from Wikidata; vi) focusing on named entities in the depicted art and expanding their reference labels; vii) implementing named entity recognition and linking using these expanded labels.

## 2 Entity Linking Challenges in the Art Domain

Entity linking, involves the identification, disambiguation, and connection of entities mentioned in text, to their corresponding entities in a knowledge base. In the specific realm of entity linking, the art field presents unique challenges. In the art domain, it is especially important to not only consider text, but also the images associated with the artworks.

However, the scarcity of datasets tailored to entity linking in artwork is a significant barrier to developing effective techniques. This domain is filled with subjective interpretations and ambiguities, complicating the task of entity linking [5]. For example, a single symbol or motif might have varying meanings in different cultures and historical periods, requiring extensive contextual knowledge for precise entity identification. The inconsistency in image quality, influenced by the age of the artwork or the materials used in its creation, often results in detail or color fidelity loss. This variability adds complexity to the processes of entity recognition and linking, needing algorithms to handle these idiosyncrasies. Therefore, developing efficient entity linking methods in the art domain involves addressing these challenges and highlights the need for custom art-related entity linking datasets, particularly those that are multimodal. Multimodal entity linking datasets for the general domain have been proposed. However, in contrast to datasets like WikiMEL [14], TwitterMEL [2] and WikiDiverse [15], where the images are photographs and the entities are typically persons, for the majority of art subjects the visual representations in MELArt, the dataset we propose in this work, come from paintings. These artistic representations of the subjects come with a high variety of styles and contexts, making it challenging for an automated model to deal with. We adhere to the multimodal entity linking task defined in these datasets, treating the input as a sentence with a mention accompanied by an image, and the objective is to link the correct entity to the mention. An example of the task, specific to the art domain, can be found in Figure 1. Here the input is the sentence “Mary, sitting on a throne . . .” with the corresponding mention “Mary”, with the additional image from the painting. The output should be a score or a ranking of candidate entities, like the candidates on the right in Figure 1, in which the ground truth entity (Q345 in this example) should be at the top of the ranking. Including the image of the painting and potential images for the candidate entities is essential for the art domain, given the visual nature of artwork.

The art domain presents distinctive challenges due to the rich historical context, variability in artistic descriptions, and the interplay of various figures and symbols. We want to illustrate the unique challenges of entity linking within the art domain through specific example sentences with links to Wikidata entities.

**Complex Scene Composition.** The example sentence “Here the Virgin lifts the veil over the sleeping Child, who is turned toward the audience, with her other arm around the young John, who has a reed across his shoulder” contains the NEL challenge of identifying and linking multiple figures present in a single artwork. The entities involved in this example are the Virgin Mary (Q345), the Christ Child (Q942467), and the Child Saint John (Q1698874). The NEL task complexity arises in distinguishing each figure and associating them with their respective symbolic attributes, like the veil and the reed.

**Annunciation with St. Margaret and St. Ansanus (Q979440)**  
painting by Simone Martini and Lippo Memmi



"Mary, sitting on a throne, is portrayed at the moment that she is startled out of her reading, reacting with a graceful and composed reluctance, looking with surprise at the celestial messenger."



**Virgin Mary (Q345)**  
human biblical figure, human. mother of Jesus



**Mary, Queen of Scots (Q131412)**  
human. Queen of Scotland from 1542 to 1567



**Mary Magdalene (Q63070)**  
human biblical figure, human. follower of Jesus

■ **Figure 1** Multimodal entity linking example in the art domain. On the right, we show 3 potential entity candidates for the mention *Mary* marking *Virgin Mary (Q345)* as the correct entity for the artwork on the left. The text corresponds to a *visual sentence* from the Artpedia dataset.

**High Disambiguation.** The sentence “Queen Charlotte dropped behind Montagne due to the loss of this mast and thus failed to capture her, and led to criticism of the painting” presents a sophisticated entity linking challenge. The primary task is to accurately identify and link “Queen Charlotte” to the entity HMS Queen Charlotte (the ship - Q680379), a task complicated by the potential misinterpretation of “Queen Charlotte” as a person entity, especially given the use of the pronoun “her” in the sentence. This situation underscores a notable aspect of entity disambiguation, where the capacity for multimodal entity linking to discern between human and non-human entities becomes crucial. Such a capability is particularly significant given that many existing datasets exhibit a pronounced bias towards the presence of person entities, highlighting the need for more nuanced and context-aware approaches in entity linking methods for the art domain.

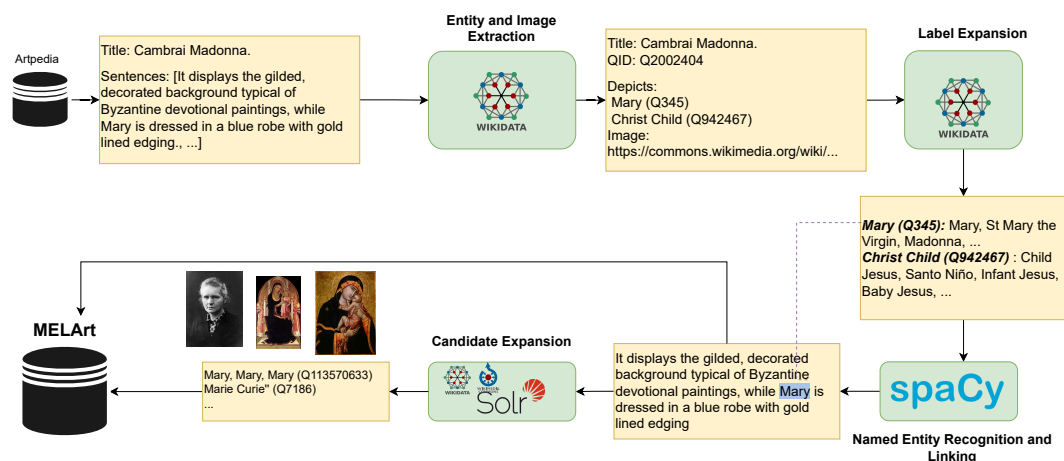
**Artistic Historical Linkage.** The sentence “The central panel shows the Crucifixion of Christ with John the Apostle and the Virgin Mary” necessitates a precise entity linking process, wherein the name mention “Christ” must be correctly linked to the entity Jesus (Q302) rather than Christ Child (Q942467). The accurate identification of this historical event hinges on the detection of the Cross in the image, which serves as a critical visual cue signifying the Crucifixion. This visual element is essential for distinguishing between the adult Jesus at the Crucifixion and other representations of Jesus, such as the Christ Child. The presence of other key figures in the narrative, such as John the Apostle and the Virgin Mary, further contextualizes the scene, reinforcing the correct linkage to the Crucifixion event. This example highlights the importance of contextual and iconographic cues in historical art for accurate entity linking, especially in complex religious narratives where different phases of a central figure’s life are depicted.

Through these examples, we can see how the art domain poses unique challenges for entity linking, including the need for disambiguation, dealing with variable descriptions, understanding the historical and artistic context, distinguishing between multiple entities, and recognizing artistic authorship. Each aspect addresses the multifaceted nature of entity linking in the realm of art history and criticism.

### 3 Dataset Construction

Our art entity linking dataset construction method, as depicted in Fig. 2, incorporates a multi-step process that is in alignment with the critical components of the dataset. The following details each step in the process:





■ **Figure 2** The MELArt automatic construction process.

- **Extraction of Painting Title and Visual/Contextual Sentences:** The Artpedia title of the painting, such as “Portrait of Mlle Rachel”, along with contextual sentences like “Portrait of Mlle Rachel is an oil painting on millboard”, are extracted from the Artpedia dataset.
- **Entity and Image Extraction (Multiple-art sources integration):** This step involves querying the unique identifier of the painting from Wikidata, like, for example, “Q979440” (see Fig. 1). Additionally, the image of the painting is retrieved from Wikimedia. This step is discussed in Section 3.1.
- **Label Expansion:** This phase entails expanding the set of surface forms used to refer to the depicted named entities, taking advantage of Wikidata’s alternative labels. This step is discussed in Section 3.2.
- **Named Entity Recognition and Linking:** This process identifies occurrences of names within texts. It utilizes the expanded set of labels to cover the different ways used to refer to the depicted entities. This step is discussed in Section 3.3
- **Candidate Expansion:** This step creates an extensive set of Wikidata entities that, besides the depicted entities, could be referenced from the text. This is a crucial step to make our dataset valuable for the evaluation of NEL models. This step is discussed in Section 3.4.

### 3.1 Entity and Image Extraction

Given the need to ensure precise matching and integration of data across data sources, to efficiently merge Artpedia with the two other online databases, Wikidata and Wikimedia Commons, a structured approach is necessary. Our process can be systematically broken down to:

1. **Locating the artwork’s Entity-ID by Title:** Identify and locate the specific entity in Wikidata using its Artpedia title (which is equivalent to its Wikipedia article and unique inside Artpedia). This step is implemented by querying Wikidata’s links to Wikipedia articles using the following SPARQL query<sup>2</sup>:

<sup>2</sup> All the Wikidata extraction is performed on a dump downloaded on 2024-09-13 and generated on 2024-09-06T23:44:15Z installed on QLever [4] as SPARQL engine.

```

PREFIX schema: <http://schema.org/>
SELECT ?wikipedia_title ?wikidata_id WHERE {{
  VALUES ?wikipedia_title {"@title"@en}.
  ?wikipedia_id schema:name ?wikipedia_title.
  ?wikipedia_id schema:about ?wikidata_id.
  ?wikipedia_id schema:isPartOf <https://en.wikipedia.org/>.
}}

```

where *@title* is the title from Artpedia. In this process we found that for a subset of 20 paintings, the match was not possible. The two main reasons for a missing match were: i) deleted Wikipedia articles like “Charles II in armor” and ii) Wikipedia articles that became disambiguation pages like “The Three Musicians (painting)”. For those cases we manually defined the matching between Artpedia’s title and Wikidata.

2. Extract from Wikidata, using SPARQL, the following predicates for each artwork:
  - P18: *image*. We extract the URL of the first image for each painting that is part of Wikimedia Commons.
  - P180: *depicts*. This corresponds to each “entity visually depicted in an image, literarily described in a work, or otherwise incorporated into an audiovisual or other medium”.

### 3.2 Label Expansion

In this step, we take advantage of the rich set of alternative labels defined in Wikidata entities to consider the multiple surface forms that might be present in Artpedia artwork description sentences to refer to depicted entities. To do so, we extract Wikidata’s alternative labels for the depicted entities. For each of the depicted entities, we list all the available alternative labels. For example, the entity *Virgin Mary (Q345)* can be referred to by multiple surface forms, which include among others: *Our Lady*, *The Virgin Mary*, *Madonna*, and *Holy Virgin*.

At the end of this process, the total number of labels for depicted named entities becomes 12,966, averaging 2.9 labels per depicted entity.

### 3.3 Named Entity Recognition and Linking

In this stage, we create a rich set of mention-entity pairs by finding the depicted entity labels inside Artpedia’s sentences. This pair construction is a critical step, as it forms the backbone of the MELArt dataset, linking the entities mentioned in the artwork descriptions to their corresponding entities within a structured and accessible knowledge base. This structure is critical for facilitating research and applications in art and cultural heritage data analysis. The steps we use are the following:

1. Select named entities. The depicted entities identified in the previous step are carefully filtered to preserve named entities and remove general entities. Given the set of all the entities depicted in an artwork, we decide to only keep named entities as iconographic statements. In our process, this is important because automatically matching non-named entities might create false positives since their surface forms are common words. Similar to [9], to determine if an entity is a named entity (i.e., an iconographic statement), we extract the English labels, and check if any of them contain an uppercase character. In this way we avoid entities like *horse* or *hand* and keep entities like *Christ* or *Paris*. An initial evaluation of using Wikidata entity types (i.e., using *instance\_of* predicates) as a filter for named entities, with a manually defined set of classes like *human* and *city*, showed smaller coverage of entities as compared to the capitalization heuristic. An inspection of the missed entities showed that the selection of the set

■ **Table 1** Compilation of the Name Mention “Gustave Courbet” and different entities sharing this label.

Label	Description	Instance of
Gustave Courbet	French painter (1819-1877)	Human
Gustave Courbet	Exhibition Fondation Beyeler	Art Exhibition
Gustave Courbet	2007–2008 Exhibition at Metropolitan Museum	Art Exhibition
Gustave Courbet	Print in the National Gallery of Art (NGA 162984)	Print
Gustave Courbet	(Temporary) Art exhibition	Art Exhibition
Gustave-Courbet-Straße	Street in Mitte district, Berlin, Germany	Street
Gustave-Courbet-Straße	Street in Taucha, Saxony, German	Street
Gustave Courbet’s Meeting: A Portrait of the Artist as a Wandering Jew	Journal article; published in 1967	Academic journal article
Gustave Courbet’s “The Sleepers”. The Lesbian Image in Nineteenth-Century French Art and Literature	Article	Scholarly Article
rue Gustave-Courbet	Street in Paris, France	Street
Allée Gustave-Courbet	Alley of Montreuil in Seine-Saint-Denis, France	Allée

of classes needed to cover all the depicted named entities in the dataset is a manually intensive and error-prone task. We thus refrain from manually defining and maintaining such an entity type set and instead follow the more automated process that looks at letter capitalization.

2. Match the labels to the artwork sentences. For each of the paintings, we match any of the labels of the depicted named entities to the visual and contextual sentences from Artpedia, using the Spacy’s Phrase Matcher<sup>3</sup>. The result is a set of spans in the sentences corresponding to the depicted entities. Given that the list of possible entities is limited by the depicted named entities, this approach focuses on high matching precision. The matching is case-sensitive, to avoid false positives, especially with names like *Our Lady* which are composed of common words.
3. Filter nested mentions. In the case of overlapping matches, where one match is completely contained in another match, we select the longer and more specific entity mention. For instance, if the entity *Madonna and Child* is recognized as an entity, but in the same span *Madonna* is also recognized, then the latter will not be considered.

### 3.4 Candidate Expansion

In this stage, given the recognized and linked mentions, we build an extensive set of Wikidata entities that can be potentially linked to the depicted entities. We base our expansion on the assumption that entities sharing labels are challenging to disambiguate, with a limited context.

<sup>3</sup> <https://spacy.io/api/phrasematcher>

Table 1 illustrates various entity candidates for the name “Gustave Courbet”. It exemplifies the multifaceted challenges inherent in entity disambiguation tasks within art datasets. The table shows the wide variety of meanings that can be connected to a single name, highlighting how one term can have many different interpretations or associations. The labels in the table range from the person “Gustave Courbet” himself to various exhibitions themed around him, and even streets named after him. This example shows how widely a single name can be interpreted, pointing to different entities or concepts each time it is mentioned. The descriptions confer granularity, distinguishing the historical individual from specific art expositions and location entities. Crucially, the “Instance of” category elucidates the ontological nature of each referent, spanning classifications like “Temporary Exhibition”, “Print”, and “Scholarly Article.” This typological diversity underscores the need for an advanced semantic analysis approach, incorporating label and feature extraction (from both text and image) to ensure precision in the mapping of nominal references to their corresponding entities within a knowledge base such as Wikidata. The steps we use are the following:

1. Wikidata labels full-text index. We first index all the English entity labels from Wikidata into an Apache Solr<sup>4</sup> core. This system is configured to index the text of the labels using an English analyzer composed of tokenization, stop word removal, possessive filter and stemmer. In addition to the labels, we include the number of Wikipedia articles for each entity to represent the popularity of the entity.
2. Expand the candidates list using the mentions’ text. Given the set of matched mentions in the sentences, we expand the set of candidate matching entities for those mentions using the indexed Wikidata labels. For each mention (e.g., “Mary”) we perform two queries to the Solr index. First, we find the 25 most similar labels to the mention, based on the BM25 algorithm ( $k_1=1.2, b=0.75$ ). This returns candidates with very similar labels like “Mary, Mary, Mary” (Q113570633). Additionally, we perform the same search but sorting the results according to the number of Wikipedia links in descending order and choosing the first 25 results. This query finds entities like “Marie Curie” (Q7186) with only a partial label match, but with a high popularity. We then include these two sets of candidates and the ground truth entity to the universe of MELArt candidates. This final set contains 53,901 entities.
3. Extract details from the candidates. For each of the candidate entities, we use SPARQL to extract the following predicates.
  - P18: *image*. For each candidate entity, we extract all Wikimedia Commons images, except those that correspond to the Artpedia artwork images. We filter those because for some candidate entities, the related images already correspond to the paintings in which they have been depicted. If we were to keep them, that would make the entity linking task trivial in those cases. For instance, the painting *Ophelia*(Q21192340) by *John William Waterhouse* depicts the character *Ophelia*(Q1800888) from Hamlet, but the image associated with the Wikidata entity of the character is Waterhouse’s painting.
  - P31: *instance of*. We extract the classes the entity belongs to, and their main labels.
  - We extract Wikidata’s English description of the entity.
4. Download images. Together with the painting image links, and all the candidate images, the image files are downloaded from Wikimedia Commons or Wikipedia in a few cases. Wikimedia Commons links are given priority in case there are multiple images for a painting.

---

<sup>4</sup> <https://solr.apache.org/>

## 4 Evaluation and Results

We perform two types of evaluation of the quality of the MELArt dataset. First, we evaluate the reliability of the automatically annotated dataset by employing human annotators who check its correctness. The second involves the use of MELArt with state-of-the-art (SOTA) entity linking models.

### 4.1 Data Quality Evaluation Using Human Annotations

To evaluate the quality of MELArt, we follow a multistep process with two human annotators that manually complete the entity recognition and linking tasks on a sample of the painting description sentences. The goal of this evaluation is to answer the following question: are the automatically annotated mentions precise and exhaustive? Ideally, all the annotations should correspond to correct entity mentions, and they should be linked to the right Wikidata entities.

We limit the scope of this evaluation to annotations of linked Wikidata entities, thus avoiding the identification of depicted entities not linked in Wikidata, even though there are hints in the text indicating their presence. For instance, for the painting *Rucellai Madonna* (Q948745) and the visual sentence “The painting depicts the Virgin and Child enthroned, surrounded by angels on a gold background.” an annotator could identify *Madonna and Child* (Q9309699) as a good fit for the “Virgin and Child” span, however, according to Wikidata’s triples, this artwork depicts *Virgin Mary* (Q345) and *Christ Child* (Q942467) as independent entities. In such cases, we assume Wikipedia’s links as the ground truth and refrain from adding extra links.

#### Step 1: Initial joint exploration of a small sample

The initial step involves the selection of a sample containing 30 paintings, which was presented to both annotators. The annotators jointly explored the sample, identifying and discussing annotation errors and agreeing on precise definitions to address borderline cases. From this initial data exploration step, the following coding rules were agreed upon:

- In the presence of nested entities, the more specific one must be chosen, and the nested annotations must be removed. For instance, the entity *Madonna and Child* is preferred to annotating *Madonna* if both entities are present in the artwork.
- The article preceding the entities should be included, only if it is capitalized. For instance, “The painting depicts the [Virgin] and . . .” should not include the article *the*, while “. . . from left to right, John the Baptist, [The Virgin Mary] with . . .” should include it.
- Only nominal mentions are considered, avoiding pronouns referring to the entities in a different span of text.

#### Step 2: Independent Annotation

In this step, a bigger random sample of 100 painting descriptions was drawn from the dataset. Annotators worked independently on their assigned tasks to mitigate bias. The annotators did not have access to the annotation made by the other annotator, but they had access to the automatic annotations. The annotators based their decision on their own understandings of the task and the available information about the paintings in Wikipedia and Wikidata.

After this step, we compared the annotations derived from each annotator and computed inter-annotator agreement using Krippendorff’s Alpha. The value of *alpha* after the second step was **0.89**, which can be interpreted as a high level of agreement.

## 8:10 MELArt: A Multimodal Entity Linking Dataset for Art

<b>A</b>	It depicts <span style="border: 1px solid green; padding: 2px;">Q33923</span> Peter denying Jesus after Jesus was arrested . <span style="border: 1px solid purple; padding: 2px;">Q130704</span> The Denial of Saint Peter
<b>B</b>	It depicts <span style="border: 1px solid green; padding: 2px;">Q33923</span> Peter denying Jesus after Jesus was arrested . The <span style="border: 1px solid green; padding: 2px;">Q130704</span> Denial of Saint Peter
<hr/>	
	It depicts <span style="border: 1px solid purple; padding: 2px;">Q33923</span> Peter denying Jesus after Jesus was arrested . <span style="border: 1px solid purple; padding: 2px;">Q130704</span> The Denial of Saint Peter

■ **Figure 3** Example of the curation of an annotation disagreement. The top annotations correspond to annotator A and B. The bottom annotation is the curated ground truth as determined by the discussion among annotators.

### Step 3: Aggregation and Definition of the Ground Truth

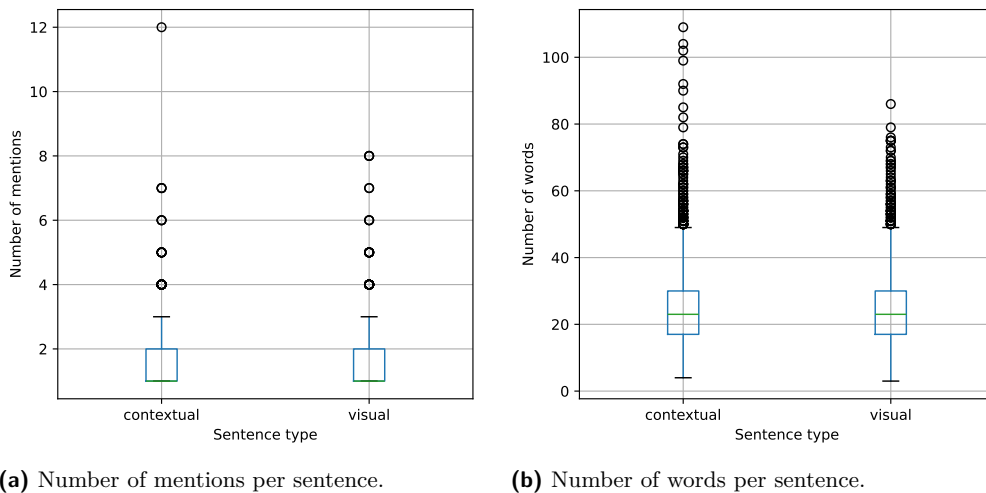
The final step combines the individual annotations into a unified dataset, resolving the few discrepancies through discussion among the human annotators. Both annotators explored the set of 100 documents, using the curation functionalities of the INCEpTION tool, and discussed disagreements until the set of annotations was considered curated and consistent. We consider this curated sample as ground truth entity linking annotations under the scope of the ‘depicted’ predicate of Wikidata. Figure 3 shows an example of the resolution of a disagreement. In this example the disagreement relates to the boundaries of the mention and whether the mention should include the article *The* for the second mention. As per the rule discussed in Section 4.1 the article should be included, but given that it is at the beginning of a sentence, and it is capitalized for that reason, and the name of the entity in Wikidata does not contain the article, the annotators agreed not to include it. In the same example, we show the cases in which named entities are not included in the dataset. Here “Jesus” is an entity mentioned in the text, but not depicted in the painting, thus not part of the multimodal annotations of MELArt.

### Step 4: Using Sequence Labeling Metrics to Measure MELArt Quality

In the last step, MELArt annotations are compared against the final version of the ground truth for the sample of 100 paintings. Specifically, we compute precision, recall and F1 for the presence of manually annotated mentions in the automatically generated annotations. To consider a mention as a true positive, the left or right boundaries of the span must match, as well as the assigned entity id.

### Human Evaluation Results

The evaluation metrics for MELArt data against the manually curated ground truth focus on the assessment of the dataset’s quality. We observed an F1-score of 0.79 with Precision being notably high (0.90). This reflects the dataset’s high level of reliability in terms of correctly identifying relevant items. The majority of false positives, correspond to mentions not considered as named entities by the human annotators, but included by MELArt heuristics. An example is several mentions to the entity “mother” (Q7560), which contains one label in uppercase. The recall score (0.70), indicates the dataset’s comprehensive nature, encompassing a broad range of relevant items. However, it also shows how automatically matching the labels of Wikidata to natural language does not cover all possible surface forms. This was also observed during the annotation process, in which multiple entities were identified as missing. One of the most common patterns missed by our automatic annotation approach are mentions only containing the first name, last name, or portions of the entity name. For example, the entity *Frida Kahlo* is mentioned in certain cases as



■ **Figure 4** Complexity of sentences per sentence type.

*Frida*, which, given the context, can be identified by a human annotator as being the artist, but may be missed by the automatic processes we used to generate MELArt. On the other hand, we believe that the annotations contained in MELArt serve the purpose of training and evaluating next-generation approaches for the multimodal entity linking task in the art domain, given its high precision. It is also important to note that the traditional NEL task starts with a set of pre-defined mentions, so NEL-specific models are not directly affected by missing mentions. Ideally, more mentions could be included in the dataset, including pronominal mentions, increasing the difficulty of the linking task. However, imprecise while exhaustive annotations would hinder the evaluation benchmarking potential of the dataset.

In general, the reported metrics suggest that the automatically generated dataset is of high quality, offering a reliable and accurate basis for the training of multimodal NEL models.

The published version of MELArt, used for training and evaluation, includes the manual curated ground truth as the test portion of the dataset, and all the paintings not included in the manual evaluation are part of the training or validation portions of the dataset. The division between training and validation portions is based on a random 80:20 split.

## 4.2 Resulting Dataset: Statistics and Analysis

In total, MELArt covers 1,616 artworks split in training (1,188), validation (328) and test (100) sets.

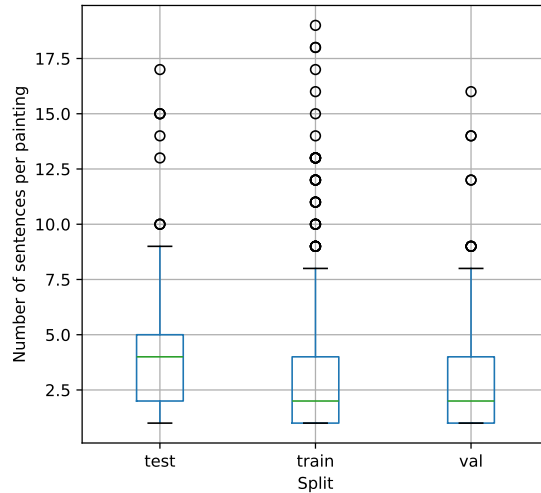
The dataset resulting from the process described above was constructed from 7,170 visual and 14,526 contextual sentences from Artpedia, contrasted with the matching of 2,118 visual and 2,716 contextual sentences in MELArt. The reduction in number of sentences is due to many sentences not directly mentioning the depicted entities (e.g., “The couple had been married for only a brief period”). The numbers also show a slightly stronger representation of visual sentences (44%) in MELArt as compared to Artpedia (33%). This is expected given that depicted entities are more likely to be mentioned in visual than contextual sentences.

The final MELArt statistics after introducing the manual annotations are:

- Number of mentions: 6,585
- The training, validation, and test splits correspond to: 4,632, 1,308, and 645 mentions respectively.
- 2,916 mentions are associated to visual sentences and 3,669 to contextual sentences.

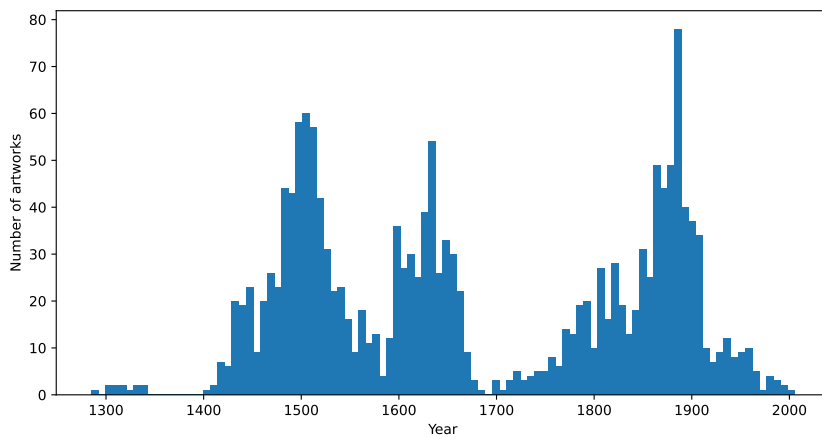
## 8:12 MELArt: A Multimodal Entity Linking Dataset for Art

In Figure 4, we can see that although there are especially long contextual sentences, the trend in terms of number of mentions and number of words per sentence remains similar among visual and contextual sentences.



■ **Figure 5** Number of sentences per artwork by dataset split.

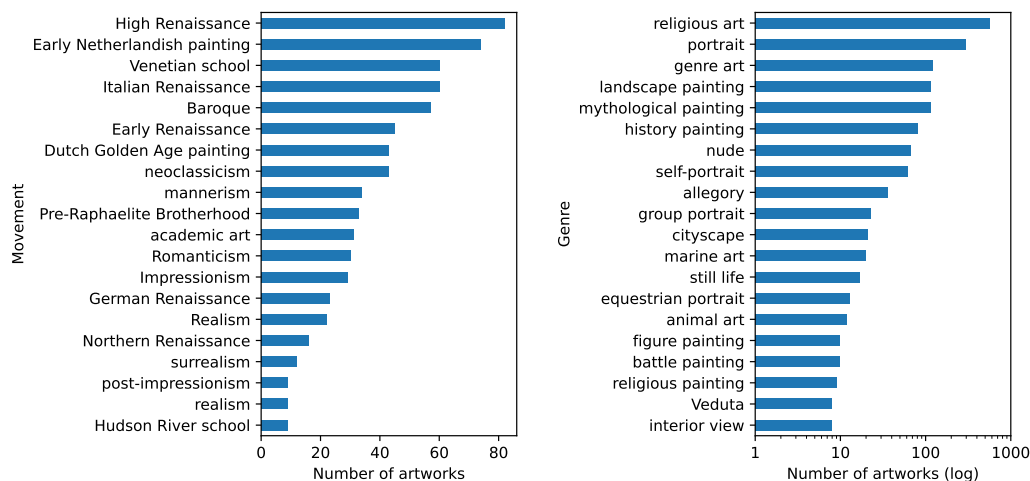
Considering the split of the dataset, we can see in Figure 5 that in terms of number of sentences for each artwork, the automatically annotated training and validation sets show similar trends. The manually annotated test set, however, has slightly more annotated sentences per artwork. This can be interpreted as a result of the lower recall of the automatic annotations, resulting in less mentions per artwork in the automatic annotations as compared to the manually annotated test set. When training a model with MELArt, this does not impose a problem in terms of the format of the training and validation data points, but raises an interesting challenge about linking potentially unseen surface forms that can be found in the test set.



■ **Figure 6** Artworks in MELArt by year of inception according to Wikidata.



In terms of the artworks in the dataset, we can leverage the information in Wikidata to understand potential biases in the dataset sources and generated annotations. Figure 6 shows a histogram representing the periods in which the included artworks were created<sup>5</sup>. We can see three strongly represented periods: one covering the 15th and 16th centuries, one for the 17th century, and lastly the 19th century.



(a) Number of artworks per movement.

(b) Number of artworks per genre.

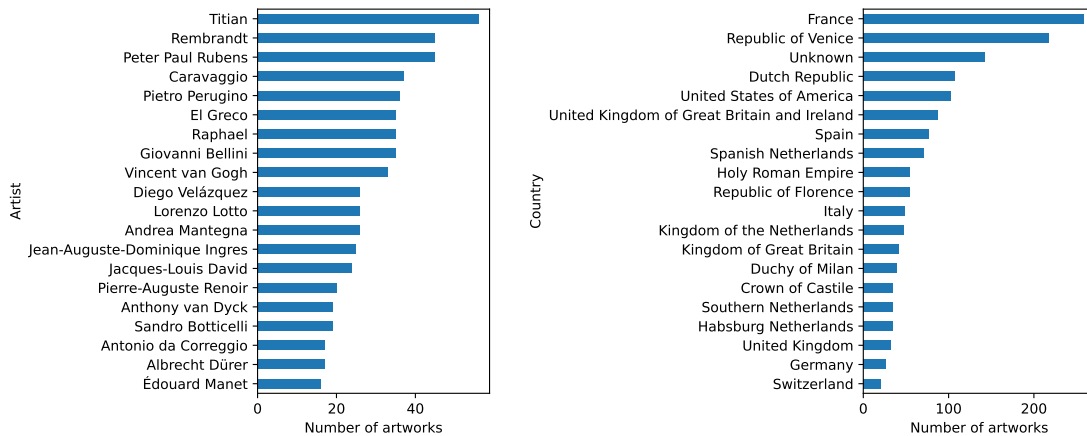
■ **Figure 7** Top 20 most represented movements and genres in MELArt.

By observing the movements and genres in Figure 7, it is clear that the *Renaissance* is the most common movement in the dataset, being also tightly related with the *religious art* genre. The genres show a stronger skewness towards *religious art* and *portrait*, which might be a result of focusing on named entities in MELArt. Analyzing the category distribution of Artpedia paintings before and after matching with the Wikidata “Depicts” predicate, reveals interesting distribution differences. In the Artpedia dataset, “religious art” (660 paintings), “portrait” (585), and “genre art” (259) are the most popular genres, while in the MELArt dataset, the dominance of “religious art” (571) and “portrait” (295) persists. Notably, “genre art” sees a significant drop from 259 to 121 in the matched dataset. Categories like “mythological painting” and “history painting” show a smaller decrease in representation. The matched dataset also exhibits a drastic reduction in categories like “landscape art” and “animal art”, highlighting a potential under-representation of these genres in MELArt due to the lack of named entities depicted in them. This comparison reveals a skew in MELArt’s representation of art, with a strong focus on religious and portrait art, while other genres like “landscape art” being less prominently featured due to the scarcity of named entities in them. This reflects possible biases and limited art genre diversity in both Artpedia and MELArt.

In Figure 8 we analyze the origin of the artworks according to the artists, where it is clear that the dataset is strongly skewed towards European art. Another clear bias in the dataset is related to the gender of the artists: out of 468, 413 are male, 27 are female and 1 is a non-binary/queer artist, according to gender information in Wikidata.

<sup>5</sup> Making use of Wikidata’s inception date (P571).

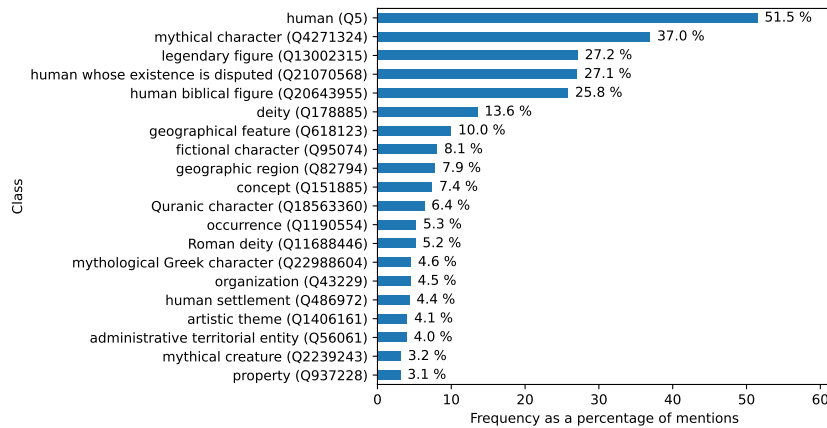
## 8:14 MELArt: A Multimodal Entity Linking Dataset for Art



(a) Number of artworks per artist.

(b) Number of artworks per artist's nationality.

■ **Figure 8** Top 20 most represented artists and their nationalities in MELArt.



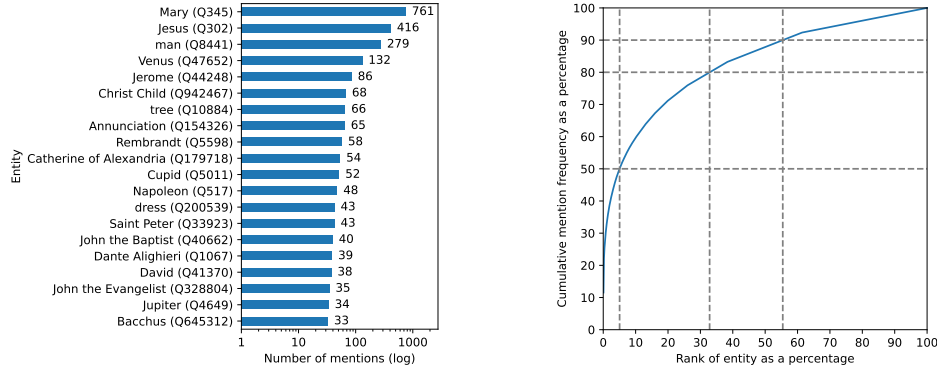
■ **Figure 9** Top 20 most represented classes in MELArt's mentions.

We also illustrate statistics about the depicted subjects in the annotations in Figures 9 and 10. We observe that although entities from different classes<sup>6</sup> are mentioned in the sentences it is more common to find *humans* and its subclasses than others, e.g., *geographical features*.

In terms of concrete entities, aligned with the *genres* seen in Figure 7, religious entities are very prominent, with *Mary (Q345)* being the most frequently mentioned entity in the dataset. We also see that the dataset is skewed towards a group of entities, having half of the mentions linked to the most frequent five percent of the 1,306 mentioned entities, and eighty percent of the mentions being covered by one third of all the entities.

All the artworks have an associated image in the dataset. Roughly half of the full candidate set contain at least one image. In the subset of mentioned candidates more than 90% contain at least one image.

<sup>6</sup> We restrict our analysis to Wikidata classes directly related to the mentions.



(a) Number of mentions by entity (Top 20).

(b) Skewness of the entity representation.

■ **Figure 10** Sparsity of the entity mentions.

### 4.3 Evaluation of Entity Linking Baselines using MELArt

In this section, we present the evaluation of tree entity linking methods using the MELArt dataset. To this end, each method ranks the candidates according to its specific scoring function, and we leverage on common ranking evaluation metrics to compare them. The resulting scores are sorted in descending order to compute Hits at  $k$  ( $H@k$ ), Mean Reciprocal Rank ( $MRR$ ), and Mean Rank ( $MR$ ). These metrics are computed as follows:

$$H@k = \frac{1}{N} \sum_i I(\text{rank}(i) < k) \quad (1)$$

$$MRR = \frac{1}{N} \sum_i \frac{1}{\text{rank}(i)} \quad (2)$$

$$MR = \frac{1}{N} \sum_i \text{rank}(i) \quad (3)$$

The metric  $H@k$  reflects the presence of the correct entity within the top  $k$  entities ranked by score.  $MRR$  denotes the average of the inverse of the rank of the correct entity, and  $MR$  represents the average rank of the correct entity relative to all entities. Thus, higher values of  $H@k$  and  $MRR$  correspond to better performance, whereas for  $MR$ , a lower value denotes superior performance. The measured ranks are computed per mention, meaning that if a sentence has two entities mentioned, each one is ranked independently.

In order to showcase the use of our dataset to evaluate multimodal entity linking method effectiveness on art data, we tested one entity and relation linking tool for Wikidata and trained and tested two state-of-the-art (SOTA) entity-linking methods using our MELArt dataset.

### MIMIC (KDD-2023)

In this work [6], the authors introduce a novel framework, called MIMIC, to improve multimodal entity linking which connects web content to a multimodal knowledge graph. Addressing the limitations of previous methods in handling abbreviated texts and implicit visual cues, MIMIC uses an input and feature encoding layers and three specialized interaction units: TGLU (Text-based Global-Local Interaction Unit) for textual context, VDLU (Vision-based Dual Interaction

## 8:16 MELArt: A Multimodal Entity Linking Dataset for Art

Unit) for visual cues, and CMFU (Cross-modal Fusion-based Interaction Unit) for cross-modal fusion. This approach, validated on three benchmark datasets, significantly outperforms existing models, showcasing its efficiency in extracting and integrating complex multimodal data. In our experiments, we evaluate three configurations for MELArt using this model: “MIMIC” refers to including the painting and candidate images when present for the multi-modal entity linking tasks, “MIMIC (no cand. image)” excludes the candidate images and uses only the paintings, and “MIMIC (no images)” where MIMIC only uses the textual information. The textual information that describes the candidates is the concatenation of the main label, the entity description, and its types. For instance, “Mary” (Q345) is described as the string “Mary. mother of Jesus. Types: human biblical figure”.

For training the bi-encoder, the following hyperparameters were used:

- Learning rate:  $1e^{-5}$
- CLIP model: openai/clip-vit-base-patch32
- Batch size: 128
- Maximum number of epochs: 20

### BLINK (EMNLP-2020)

This work [16] introduces a new methodology for text-based entity linking, particularly targeting zero-shot scenarios. This approach is notable for its ability to link text to entities in a knowledge base without having seen these entities during the training phase. The authors propose a two stage model. The first stage (bi-encoder) uses dense vector representations for entity retrieval, a method that significantly enhances scalability and efficiency. The second stage (cross-encoder), uses the top-k most similar entities discovered by the bi-encoder, and re-ranks them by jointly encoding the sentence (context) and candidate text. This innovative approach offers a more effective solution for zero-shot entity linking challenges, where traditional methods often struggle due to the lack of prior training data on new entities. In our experiments, we train and evaluate the bi-encoder, and use the best bi-encoder candidates to train a cross-encoder. It is important to note that there is a risk of propagating errors from the bi-encoder to the cross-encoder, especially during training. Thus, by using the best bi-encoder for training the cross-encoder we explore the upper bound of BLINK’s performance using MELArt. We train the bi-encoder using *bert-large-uncased*, extracting the top 30 candidates, and train the cross-encoder with those candidates using *bert-base-uncased*. For reporting MR and MRR we assume a rank of 500 for entities that have not been found in the top 30 candidates of a mention.

For training the bi-encoder the following hyperparameters were used

- Learning rate:  $3e^{-5}$
- BERT model: bert-large-uncased
- Batch size: 8
- Maximum number of epochs: 10

For training the cross-encoder:

- Learning rate:  $2e^{-5}$
- BERT model: bert-base-uncased
- Batch size: 2
- Maximum number of epochs: 5

## FALCON 2.0 (CIKM-2020)

This tool presented in [8], and available through an API<sup>7</sup>, allows users to submit portions of text, and it automatically recognizes and links entities to and relations in the Wikidata knowledge graph. The authors propose to jointly recognize and link entities and relations. The recognition phase is based on the concept of N-Gram tiling based on English morphological rules, and the linking phase ranks triples of recognized entities and relations based on the background knowledge from the knowledge graph. We acknowledge that FALCON does not necessarily solve the same task as MIMIC and BLINK, due to its more complex task of both recognition and linking. Additionally, FALCON does not use the candidate set of MELArt, but the whole set of Wikidata entities as candidates, making the ranking exercise harder. However, in our evaluation experiments, we want to include FALCON as an off-the-shelf alternative to specialized entity linking models.

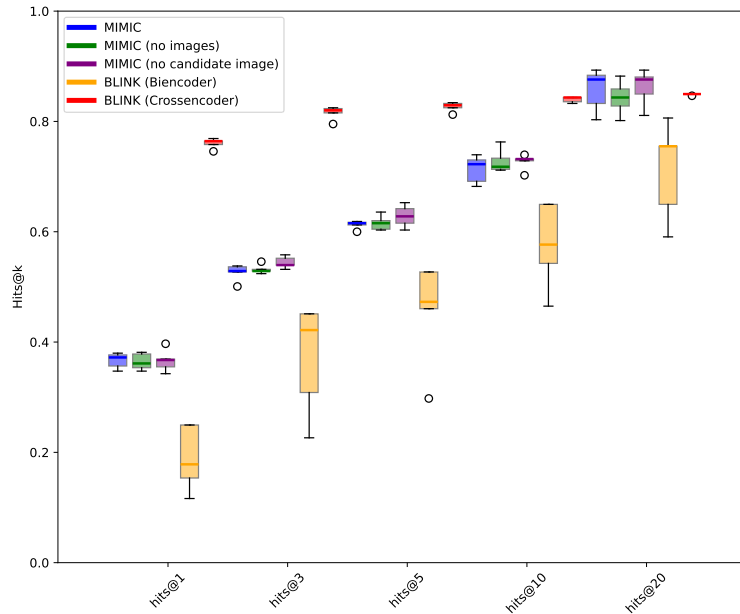
## Entity Linking Results

■ **Table 2** Results of SOTA Entity Linking baselines on our dataset. The numbers enclosed by brackets correspond to the standard deviation for multiple runs.

Baselines	H@1	H@3	H@5	H@10	H@20	MR	MRR
BLINK (Bi-encoder)	0.19 (±0.06)	0.37 (±0.1)	0.46 (±0.09)	0.58 (±0.08)	0.71 (±0.09)	128.51 (±41.02)	0.32 (±0.07)
BLINK (Cross-encoder)	<b>0.76</b> (±0.01)	<b>0.82</b> (±0.01)	<b>0.83</b> (±0.01)	<b>0.84</b> (±0.01)	0.85 (±0.00)	76.25 (±0.1)	<b>0.79</b> (±0.01)
MIMIC	0.37 (±0.01)	0.53 (±0.01)	0.61 (±0.01)	0.71 (±0.03)	<b>0.86</b> (±0.04)	<b>50.03</b> (±15.49)	0.48 (±0.01)
MIMIC (no cand. image)	0.37 (±0.02)	0.54 (±0.01)	0.63 (±0.02)	0.73 (±0.01)	0.86 (±0.03)	56.14 (±17.72)	0.49 (±0.01)
MIMIC (no images)	0.36 (±0.02)	0.53 (±0.01)	0.62 (±0.01)	0.73 (±0.02)	0.84 (±0.03)	71.81 (±24.68)	0.48 (±0.01)
FALCON 2.0	0.09	0.13	0.16	0.16	0.19	403.95	0.12

By executing these SOTA entity linking methods on our new dataset, we aim to assess the usefulness of MELArt for training these models, as well as understanding how challenging MELArt is as a test dataset. In our work, BLINK [16] and MIMIC [6] are chosen as the SOTA baselines. BLINK, a SOTA method for textual entity linking tasks, provides a rigorous standard against the text-processing capabilities of new systems to be evaluated. MIMIC, a SOTA multimodal entity linking, is employed alongside BLINK to test the integration and interpretation of both textual and visual features in the entity linking task. Additionally, FALCON 2.0 [8], serves as a text-based entity recognition and linking baseline. FALCON inference model is different from specialized entity-linking models like MIMIC and BLINK because instead of being provided with a mention intended to be linked, the input is only the raw text. FALCON recognizes the entities and relations in the texts and ranks candidates for each recognized span. Thus, in our evaluation we run FALCON with the test set of MELArt, and then for each mention in the evaluation set, we match the surface form with the mention text. If the surface form of the recognized entity overlaps with the text in MELArt’s mention, we evaluate the top k candidates of FALCON. We only consider a hit, if the mention can be matched to the surface form of an entity span from

<sup>7</sup> <https://labs.tib.eu/falcon/falcon2/api-use>



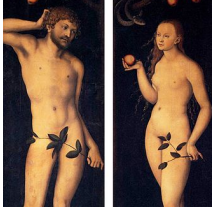
■ **Figure 11** Hits@K for different values of k in different runs.

FALCON. Given that not all mentions can be matched between FALCON and MELArt, and that we only consider the first 50 ranked candidates from FALCON, similar to BLINK we assume a rank of 500 when the ground truth could not be found in the first 50 results.

Based on Table 2, MIMIC and BLINK exhibit distinct performance characteristics as observed across the various evaluation metrics considered. In all measures except H@20, BLINK (Cross-encoder) outperforms the other baselines, suggesting that textual information remains the most important feature to disambiguate between candidates. However, comparing BLINK and FALCON, we observe that training and fine-tuning the model, which is the setup for BLINK experiments, strongly improves the linking results. However, comparing BLINK and FALCON in absolute numbers is not a fair comparison, given the fact that FALCON also performs the recognition phase that can propagate errors to the linking phase. In fact, in our experiments, only 20% of the MELArt mentions could be matched to FALCON recognized spans, thus hindering FALCON’s measures in general. What is very clear, is that the cross-encoder training has a strong positive impact on the distinct measures.

MIMIC is the only multi-modal baseline in our experiments, and from the three configurations that we used for MIMIC, we observe that including images does not significantly improve performance, suggesting that the difference in style between multiple representations of the same entity confuses the computer vision components of the linking model.

Finally, observing Figure 11, where we compare the variance in results with different seeds for each model, we note that the bi-encoder’s performance varies a lot depending on the initial parameter values. In fact, with one of the seeds that we tested, we obtained results close to zero for all measures, but we excluded that test from the compiled results. This variability of the bi-encoder is a high risk for the cross-encoder model because it relies on the bi-encoder predictions to perform the final re-ranking. Contrary to the bi-encoder, the BLINK cross-encoder has very low variety for a fixed set of candidates.

MELArt	<p><u>Adam</u> is shown scratching the right crown part of his scalp.</p>	
	<p>Answer:</p> <p>Adam (Q70899)</p> <p>first man according to the Abrahamic creation and religions such as Judaism, Christianity, and Islam Types: human biblical figure, protoplast, mythical character</p>	

**MIMIC (1)**

1. Adam. first man according to the Abrahamic creation and religions such as Judaism, Christianity, and Islam. Types: human biblical figure, mythical character, protoplast (Q70899)



2. Adam. Wikimedia disambiguation page. Types: Wikimedia disambiguation page (Q71568)

3. Adam Adam. . Types: human (Q94779082)

**FALCON (1)**

1. Adam. first man according to the Abrahamic creation and religions such as Judaism, Christianity, and Islam. Types: human biblical figure, mythical character, protoplast (Q70899)

2. Adam Adam. . Types: human (Q94779082)

3. *Ádám*. A novel by Zsigmond Justh. Types: literary work (Q72078825)

**BLINK Cross-enc.(1)**

1. Adam. first man according to the Abrahamic creation and religions such as Judaism, Christianity, and Islam. Types: human biblical figure, mythical character, protoplast (Q70899)

2. ildephonsus of toledo. scholar and theologian. types : human (Q456069)

3. john adam. british colonial governor of india ( 1779 - 1825 ). types : human (Q441081)

**BLINK Bi.enc. (8)**

1. Adam Adam. . Types: human (Q94779082)

2. Adam Baldwin. American actor. Types: human (Q312161)

3. Adam (Adam Goodes). painting by Alan Jones. Types: painting (Q104318681)

■ **Figure 12** Top-ranked entities for the mention *Adam*. On top the ground truth as defined in the manual annotations of MELArt, on the bottom the predictions and the ground truth rank in parentheses.


## Qualitative Analysis

By exploring the top-ranked entities of MIMIC and BLINK (Cross-encoder) we identify some challenges for NEL models. An example which most of the models, including FALCON, were able to correctly link is seen in Figure 12. Here, except for BLINK (Bi-encoder), all models confidently assigned *Adam* (Q70899) as the top-ranked entity.

Figure 13 shows a more challenging example. Both MIMIC and BLINK in their best versions, failed to include the right answer in their top results. We can see from the predicted rankings in fact that similar entities were chosen by the models. This example also shows the data quality challenges present in sources like Wikidata where the same painting has two QID and MIMIC accidentally detected the duplicate entity. Probably, the usage of the images by MIMIC resulted in a high similarity between the two image files, even though they are not the same file.

In the example portrayed in Figure 14, BLINK predicts the ground truth on top of the ranking, even without the visual information, but only using the textual information. MIMIC on the other hand, ranked the ground truth very low. From the examples that we explored in this study, it is noticeable that MIMIC tends to favor other artworks with similar subjects in the ranking. We can also observe here some noise in the candidate set represented by Wikidata entity disambiguation pages. Different from Figure 14, in Figure 15 MIMIC outperforms BLINK that does not even

## 8:20 MELArt: A Multimodal Entity Linking Dataset for Art

MELArt	<p>Dona Mariana (known as <b>Maria Anna</b>) (b. 1634) was the daughter of Emperor Ferdinand III and the Infanta Maria Anna of Spain, and then nineteen years old.</p> <p>Answer:</p> <p>Mariana of Austria (Q311469) Queen consort of Spain (1634-1696) Types: human</p>	
--------	---	--

### MIMIC (124)

Maria Anna, Queen of Spain, painting by Diego Velázquez and workshop. Types: painting. (Q27981333)



Maria Anna of Spain, Wife of Holy Roman Emperor Ferdinand III (1606-1646). Types: human. (Q158662)



Maria Anna of Spain, print in the National Gallery of Art (NGA 39748). Types: print (Q65358630)





### BLINK (>30)

Maria Anna of Spain, Wife of Holy Roman Emperor Ferdinand III (1606-1646). Types: human. (Q158662)

Infanta Maria of Spain. (1580-1583); daughter of Philip II of Spain and Anna of Austria. Types: human (Q3847598)

Anna, town in Valencia, Spain. Types: municipality of the Valencian Community, municipality of Spain (Q1824255)

■ **Figure 13** Top-ranked entities for the mention *Maria Anna*. On top the ground truth as defined in the manual annotations of MELArt, on the bottom the predictions and the ground truth rank in parentheses.

MELArt	<p>David's works also show Napoleon's journey through the <b>Great St. Bernard</b> Pass, but there are significant stylistic differences between the two conceptions.</p> <p>Answer:</p>  <p>Great St Bernard Pass (Q623424)</p> <p>mountain pass in the Western Alps Types: mountain pass, border crossing</p>	
--------	--	--

### MIMIC (475)

1. The Great St Bernard, painting by Joseph Mallord William Turner. Types: watercolor painting (Q18571394)

2. St. Bernard. Wikimedia disambiguation page. Types: Wikimedia disambiguation page (Q9340925)

3. St. Bernard, human settlement in Nova Scotia, Canada. Types: human settlement (Q7587291)

### BLINK Cross-enc.(1)


1. **Great St Bernard Pass, mountain pass in the Western Alps. Types: mountain pass, border crossing (Q623424)**

2. Bernard of Menthon. Priest and founder. Types: human (Q736265)

3. Bernard of Clairvaux. Burgundian saint, abbot and theologian (1090-1153). Types: human (Q188411)

■ **Figure 14** Top-ranked entities for the mention *Great St. Bernard*. On top the ground truth as defined in the manual annotations of MELArt, on the bottom the predictions and the ground truth rank in parentheses.



MELArt	<p>Finally, at the top left is <b>Simon of Cyrene</b>, his face upside upturned.</p>	
	<p>Answer: Simon of Cyrene (Q328739)</p> <p>human biblical figure in Matthew 27:32, man who was forced by the Romans to carry the cross of Jesus. Types: human biblical figure</p>	

**MIMIC (1)**

1. **Simon of Cyrene**. human biblical figure in Matthew 27:32, man who was forced by the Romans to carry the cross of Jesus. Types: human biblical figure (Q328739)



2. Simon of Cyrene carries the cross. fifth Station of the Cross. Types: statio (Q114315626)



3. Order of Simon of Cyrene. . Types: award (Q7100546)

**BLINK Cross-enc.(>30)**

1. Saint Catherine of Alexandria. sculpture by David Zürn. Types: sculpture (Q124993845)

2. St Catherine of Alexandria. painting by Bernhard Strigel. Types: painting (Q64789016)

3. Simon Simon. Swiss topographer (1857-1925). Types: human (Q28082980)

■ **Figure 15** Top-ranked entities for the mention *Simon of Cyrene*. On top the ground truth as defined in the manual annotations of MELArt, on the bottom the predictions and the ground truth rank in parentheses.

consider the ground truth in the top-30 candidates. This is a typical issue of error propagation in models like BLINK in which a lighter model (Bi-encoder) first filters a set of candidates for a stronger model to re-rank in the second stage.

## 5 Conclusion

In this paper, we have introduced a novel multimodal entity linking dataset, characterized by its reliability and thorough evaluation tested over various entity linking baselines, both in a multimodal and in a text-based setting. The dataset is automatically extracted from multiple sources and supplemented with meticulous human annotations. The MELArt dataset, carefully curated and rigorously tested, presents a novel resource that challenges traditional NEL approaches and offers rich opportunities for advancing the field of entity linking in the art domain.

The results obtained from our experimental evaluation reveal that our dataset poses a significant challenge to existing general-domain pretrained entity linking models, and that training and fine-tuning models enhance their in-domain linking performance. This makes our dataset MELArt a valuable asset for researchers and practitioners in this domain. The results also indicate that there is room for improvement in the usage of images which are particularly important for the art domain, and that specialized models might gain better performance.

## References

- 1 Panos Achlioptas, Maks Ovsjanikov, Kilichbek Haydarov, Mohamed Elhoseiny, and Leonidas J. Guibas. Artemis: Affective language for visual art. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 11569–11579. Computer Vision Foundation / IEEE, 2021. doi:10.1109/CVPR46437.2021.01140.
- 2 Omar Adjali, Romaric Besançon, Olivier Ferret, Hervé Le Borgne, and Brigitte Grau. Multimodal entity linking for tweets. In *Advances in Information Retrieval - 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14-17, 2020, Proceedings, Part I*, volume 12035 of *Lecture Notes in Computer Science*.

- Science*, pages 463–478. Springer, 2020. doi:10.1007/978-3-030-45439-5\_31.
- 3 Sofia Baroncini, Bruno Sartini, Marieke van Erp, Francesca Tomasi, and Aldo Gangemi. Is dc:subject enough? A landscape on iconography and iconology statements of knowledge graphs in the semantic web. *Journal of Documentation*, 79(7):115–136, March 2023. doi:10.1108/JD-09-2022-0207.
  - 4 Hannah Bast and Björn Buchhold. Qlever: A query engine for efficient sparql+text search. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, CIKM '17, pages 647–656, New York, NY, USA, 2017. ACM. doi:10.1145/3132847.3132921.
  - 5 Sabine Lang and Björn Ommer. Transforming information into knowledge: How computational methods reshape art history. *Digital Humanities Quarterly*, 15(3), 2021. URL: <http://www.digitalhumanities.org/dhq/vol/15/3/000560/000560.html>.
  - 6 Pengfei Luo, Tong Xu, Shiwei Wu, Chen Zhu, Linli Xu, and Enhong Chen. Multi-grained multimodal interaction network for entity linking. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, pages 1583–1594. ACM, 2023. doi:10.1145/3580305.3599439.
  - 7 Roberto Pirrone, Vincenzo Cannella, Orazio Gambino, Arianna Pipitone, and Giuseppe Russo. Wikiart: An ontology-based information retrieval system for arts. In *Ninth International Conference on Intelligent Systems Design and Applications, ISDA 2009, Pisa, Italy, November 30-December 2, 2009*, pages 913–918. IEEE Computer Society, 2009. doi:10.1109/ISDA.2009.219.
  - 8 Ahmad Sakor, Kuldeep Singh, Anery Patel, and Maria-Esther Vidal. Falcon 2.0: An entity and relation linking tool over wikidata. In *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, CIKM '20, pages 3141–3148, New York, NY, USA, 2020. ACM. doi:10.1145/3340531.3412777.
  - 9 Bruno Sartini. IICONGRAPH: improved iconographic and iconological statements in knowledge graphs. In *The Semantic Web - 21st International Conference, ESWC 2024, Hersonissos, Crete, Greece, May 26-30, 2024, Proceedings, Part II*, volume 14665 of *Lecture Notes in Computer Science*, pages 57–74, Cham, 2024. Springer. doi:10.1007/978-3-031-60635-9\_4.
  - 10 Alejandro Sierra-Múnera, Linh Le, Gianluca Demartini, and Ralf Krestel. MELArt Dataset. Dataset (visited on 2024-12-10). URL: <https://doi.org/10.48610/8a1ccdf>.
  - 11 Alejandro Sierra-Múnera, Linh Le, Gianluca Demartini, and Ralf Krestel. MELArt Dataset Generation. Software, swhId: swh:1:dir:ec4380448f4087c011040d0e3dca7832baa11182 (visited on 2024-12-10). URL: <https://github.com/HPI-Information-Systems/MELArt>, doi:10.4230/artifacts.22529.
  - 12 Alejandro Sierra-Múnera, Linh Le, Gianluca Demartini, and Ralf Krestel. MELArt Experiments. Software, swhId: swh:1:dir:203f2a69c5bc9064db3873a0160ca52f62095c25 (visited on 2024-12-10). URL: [https://github.com/HPI-Information-Systems/MELArt\\_experiments](https://github.com/HPI-Information-Systems/MELArt_experiments), doi:10.4230/artifacts.22616.
  - 13 Matteo Stefanini, Marcella Cornia, Lorenzo Baraldi, Massimiliano Corsini, and Rita Cucchiara. Artpedia: A new visual-semantic dataset with visual and contextual sentences in the artistic domain. In *Image Analysis and Processing - ICIAP 2019 - 20th International Conference, Trento, Italy, September 9-13, 2019, Proceedings, Part II*, volume 11752 of *Lecture Notes in Computer Science*, pages 729–740. Springer, 2019. doi:10.1007/978-3-030-30645-8\_66.
  - 14 Peng Wang, Jiangheng Wu, and Xiaohang Chen. Multimodal entity linking with gated hierarchical fusion and contrastive training. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, pages 938–948. ACM, 2022. doi:10.1145/3477495.3531867.
  - 15 Xuwu Wang, Junfeng Tian, Min Gui, Zhixu Li, Rui Wang, Ming Yan, Lihan Chen, and Yanghua Xiao. Wikidiverse: A multimodal entity linking dataset with diversified contextual topics and entity types. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 4785–4797. Association for Computational Linguistics, 2022. doi:10.18653/v1/2022.acl-long.328.
  - 16 Ledell Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. Scalable zero-shot entity linking with dense entity retrieval. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6397–6407. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.emnlp-main.519.
  - 17 Nikolaos-Antonios Ypsilantis, Noa Garcia, Guangxing Han, Sarah Ibrahim, Nanne van Noord, and Giorgos Tolias. The met dataset: Instance-level recognition for artworks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL: <http://cmp.felk.cvut.cz/met/>.

# Horned-OWL: Flying Further and Faster with Ontologies

Phillip Lord  

School of Computing, Newcastle University,  
United Kingdom

Martin Larralde  

Leiden University Medical Center,  
The Netherlands  
Structural and Computational Biology Unit,  
EMBL, Heidelberg, Germany

Filippo De Bortoli  

TU Dresden, Germany  
Center for Scalable Data Analytics and Artificial  
Intelligence (ScaDS.AI), Dresden/Leipzig, Germany

James P. Balhoff 

Renaissance Computing Institute,  
University of North Carolina, Chapel Hill, NC, USA

Björn Gehrke  

Institute for Implementation Science in Health Care,  
Faculty of Medicine, University of Zurich,  
Switzerland

Janna Hastings 

Institute for Implementation Science in Health Care,  
Faculty of Medicine, University of Zurich,  
Switzerland  
School of Medicine, University of St. Gallen,  
Switzerland  
Swiss Institute of Bioinformatics, Switzerland

James A. Overton  

Knocean Inc., Toronto, Canada

Jennifer Warrender  

School of Computing, Newcastle University,  
United Kingdom

## Abstract

Horned-OWL is a library implementing the OWL2 specification in the Rust language. As a library, it is aimed at processes and manipulation of ontologies, rather than supporting GUI development; this is reflected heavily in its design, which is for performance and pluggability; it builds on the Rust idiom, treating an ontology as a standard Rust collection, meaning it can take direct advantage of the data manipulation capabilities of the Rust standard library. The core library consists of a data model implementation as well as an IO framework supporting many common formats for OWL: RDF, XML and the OWL functional syntax; there

is an extensive test library to ensure compliance to the specification. In addition to the core library, Horned-OWL now supports a growing ecosystem: the `py-horned-owl` library provides a Python front-end for Horned-OWL, ideal for scripting ontology manipulation; `whelk-rs` provides reasoning services; and `horned-bin` provides a number of command line tools.

The library itself is now mature, supporting the entire OWL2 specification, in addition to SWRL rules, and the ecosystem is emerging into one of the most extensive for manipulation of OWL ontologies.

**2012 ACM Subject Classification** Applied computing → Life and medical sciences; Software and its engineering → Software libraries and repositories; Information systems → Web Ontology Language (OWL)

**Keywords and phrases** Web Ontology Language, OWL, Semantic Web

**Digital Object Identifier** 10.4230/TGDK.2.2.9

**Category** Resource Paper

**Supplementary Material** Version 1.0.0 of the source code for Horned-OWL was used for the performance results. Version 1.0.1 of the source code for `py-horned-owl` were used for the performance results.

*Software (Source Code):* <https://github.com/phillord/horned-owl> [12]

*Software (Documentation):* [https://docs.rs/horned-owl/1.0.0/horned\\_owl/](https://docs.rs/horned-owl/1.0.0/horned_owl/) [10]

*Software (Source Code):* <https://github.com/ontology-tools/py-horned-owl>

*Software (Source Code):* <https://github.com/INCATools/whelk-rs>



© Phillip Lord, Björn Gehrke, Martin Larralde, Janna Hastings, Filippo De Bortoli, James A. Overton, James P. Balhoff, and Jennifer Warrender;  
licensed under Creative Commons License CC-BY 4.0

*Transactions on Graph Data and Knowledge*, Vol. 2, Issue 2, Article No. 9, pp. 9:1–9:14



Transactions on Graph Data and Knowledge  
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**Funding** *Björn Gehrke*: Funded by the Wellcome Trust in the GALENOS project.

*Filippo De Bortoli*: Supported by the German Federal Ministry of Education and Research (BMBF, SCADS22B) and the Saxon State Ministry for Science, Culture and Tourism (SMWK) by funding the competence center for Big Data and AI “ScaDS.AI Dresden/Leipzig”.

**Acknowledgements** During the course of this work Ignazio Palmisano provided extensive help supporting our understanding of the OWL2 specification, and how the OWL API implemented it. We are very grateful; Horned-OWL would not have been possible without him.

**Received** 2024-06-27 **Accepted** 2024-11-19 **Published** 2024-12-18

**Editors** Aidan Hogan, Ian Horrocks, Andreas Hotho, Lalana Kagal, and Uli Sattler

**Special Issue** Resources for Graph Data and Knowledge

## 1 Introduction

The Web Ontology Language (OWL) has been in existence since 2003, moving to OWL2 in 2009 [5, 4]. It provides a specification for developing ontologies, which provides computationally amenable, logical descriptions of the world. These ontologies are applicable to any domain of knowledge, but have become embedded in biomedicine in particular, where ontologies have got both large individually (in biology, the widely used Gene Ontology - GO [14] - contains  $\sim 50,000$  terms, while in the medical domain SNOMED CT contains roughly 300,000 terms), and spread over an enormous range of biological domains, with the BioPortal [13] containing over 1000 ontologies, containing 14 million terms.

OWL consists of a set of different specifications that describe a data model, a formal semantics that can be used to determine computational entailment, and a number of different syntaxes for serialization, including a mapping to RDF which enables OWL to play its part in the semantic web.

The majority of the current infrastructure for OWL is built using Java; at the time of the inception of OWL, Java was in its prime, and was one of the most common languages in scientific computing. As a result, much of the infrastructure for OWL is implemented in Java, including the OWL API [6] and Apache Jena.

The use of Java brings with it a number of issues. While it is possible to write fast Java, neither the language nor the standard idioms for its use are designed for performance; for example, while the Gene Ontology is large in ontological terms it is only 50,000 terms and 500,000 axioms or when serialized as RDF 5,000,000 triples; in computational terms this is not large, but using the OWL API, GO can still take minutes to read into memory. Second, Java is less and less widely used in scientific computing having been largely displaced by Python; this limits Java’s practical utility, especially within Jupyter notebooks which have become a major tool for reproducible science. Finally, Java poorly integrates with machine learning and AI tooling which again is mainly implemented in Python, which limits the ease with which knowledge-rich OWL ontologies can be used with or by this massive growth area in computing.

In this paper, we introduce and describe **Horned-OWL**, a novel library that provides a Rust API to process OWL ontologies. Compared to the OWL API [6], this library is aimed at scalability and performance; analytical capabilities, rather than GUI development; and interoperability with Python.

Currently, **Horned-OWL** implements the core OWL data model, which we discuss in detail in Section 2.2, a pluggable system for indexing which is the focus of Section 2.3 and support for (de)serialization in RDF/XML as well as some other formats detailed by the W3C standard, detailed in Section 2.4.

Aside from the library, **Horned-OWL** provides a suite of tools to operate with ontologies akin to **ROBOT** [7] that showcase the usage of the library itself. This suite, named **horned-bin**, is presented in Section 3, where we additionally describe existing tools that rely on the library, such

as the `whelk-rs` reasoner (Section 3.2) and the `py-horned-owl` interface (Section 3.3). An evaluation of the performance of the library appears in Section 4 and shows that `Horned-OWL` substantially outperforms the `OWL API` particularly in memory usage.

## 2 Horned-OWL

### 2.1 Background on OWL2

We start with a brief discussion of the OWL2 specification; an overview is available directly from W3C [5] which we summarize here.

OWL2 is part of the Semantic Web stack which, as the name suggests, brings explicit semantics, while integrating with the Web. It is widely used as a language for the representation of ontologies, which are models of a domain shared between a community of users. In OWL2, the explicit semantics comes from an underlying description logic, which itself maps to first order logic; the link to the web comes from an alignment with the syntax and semantics of RDF/S which is the semantic web representation of a knowledge graph, the use of XML Schema datatypes and IRIs as its primary identifier.

Although at heart, OWL2 simply describes a set of individuals and the relationships between them, it is fairly complex: it has six different types of named entity and 37 different types of axiom. This complexity is such that OWL also supports profiles; simpler subsets with different computational properties. As well as this semantics, OWL2 has three different syntaxes, plus a mapping to RDF which itself has at least five different syntaxes in common use. On top of this, we also have SWRL, which, although not strictly part of OWL2, is sometimes used in OWL ontologies.

It is this complexity which makes OWL2 rather challenging to implement; it was a design aim to support all of OWL2 (including SWRL), however, because we wanted it to operate over the many biological ontologies that already exist; this is also a complex ecosystem and most of the parts of OWL2 are used somewhere in that ecosystem.

### 2.2 Design

The core of `Horned-OWL` is the `model` namespace. As the name suggests, this implements the core data model of OWL. As a part of the semantic web, OWL is built largely on top of the IRI (Internationalized Resource Identifiers). This is defined in `Horned-OWL` simply as follows:

```
1 pub struct IRI<A>(pub (crate) A);
```

The type `A` is generic and could be any type; we will cover the reason for introducing this genericity in a later section describing the Python implementation. While the type of `A` is not constrained on the `IRI` struct, in practice the vast majority of methods in `Horned-OWL` do constrain it to the `ForIRI` trait, which allows using the contents as a string. This design pattern of types unconstrained on struct, but constraints on methods is common in Rust and considered best practice. In practice, the most common type of `A` is `Rc<str>` which is a reference counted pointer, meaning that multiple instances of the same IRI do not each instantiate their own string.

```
1 impl<T: ?Sized> ForIRI for T where
2     T: AsRef<str>
3         + Borrow<str>
4         // Other traits omitted for length
5 {
6 }
```

## 9:4 Horned-OWL: Flying Further and Faster with Ontologies

The generation of new IRI instances is handled by a single entity, called `Build`. This also handles the generation of `AnonymousIndividual` instances which, by definition, do not have IRI identifiers. The `Build` object allows caching and sharing of IRI and, in addition, operates as an “Arena” object, meaning that all IRI instances in an ontology will share the same lifetime.

```
1 pub struct Build<A: ForIRI>(g
2     RefCell<BTreeSet<IRI<A>>>,
3     RefCell<BTreeSet<AnonymousIndividual<A>>>,
4 );
```

While OWL uses IRIs to identify most entities, in `Horned-OWL`, we use a struct for all OWL named entities; this is known as the newtype pattern in Rust and adds type safety to most of the methods in `Horned-OWL`; it is not possible to pass an IRI identifying a class to a function requiring an object property, for instance. The `Class` struct is defined as follows:

```
1 pub struct Class<A>(IRI<A>)
```

An OWL ontology consists of a number of components; in `Horned-OWL` these are all modelled as a single large `enum`. This enum includes all OWL axioms, the ontology IRI and version IRI; and, although not strictly part of OWL, we also support SWRL rules through this mechanism. This varies slightly from the formal definition of OWL; the main advantage of this approach is described later.

Any component in OWL can also support a set of annotations which we support through the use of the following struct:

```
1 pub struct AnnotatedComponent<A> {
2     pub component: Component<A>,
3     pub ann: BTreeSet<Annotation<A>>,
4 }
```

Ontology components themselves have individual representations. For example, a `DeclareClass` axiom is defined using a “tuple struct” as follows:

```
1 pub struct DeclareClass<A>(Class<A>)
```

Conversely, the `SubClassOf` axiom uses named fields, trading concision for readability.

```
1 pub struct SubClassOf<A> {
2     sup: ClassExpression<A>,
3     sub: ClassExpression<A>
4 }
```

The complexity of OWL is such that `Horned-OWL` uses a system of rules to determine naming, with as much consistency as possible, to ensure that in use it maintains the principle of least surprise; it should be possible to guess most field names for anyone familiar with OWL.

An ontology itself is represented by an empty “tagging” trait in `Horned-OWL` as follows:

```
1 pub trait Ontology<A> {
2 }
```

This seems rather perverse given that `Horned-OWL` is an API for ontologies, but the reason is because an OWL ontology is treated as a set of components; the ontology needs no methods because all information about that `Ontology` object is available from one of its components. In

practice, every type that implements `Ontology` also supports conversion to a Rust `Iterator`; this means that `Ontology` is simply a specialisation of a Rust collection<sup>1</sup>.

In addition, a `MutableOntology` type has been defined which provides a generic mechanism for adding and removing entities from an ontology.

## 2.3 Indexing

To make practical use of the ontology as a collection that `Horned-OWL` implements, we need to support one or more indexes – mechanisms for efficient look up and querying of ontology components. `Horned-OWL` does not take the route of providing sensible defaults for its “standard” ontology implementation. Instead, it enables a series of indexes which can be plugged in; this ensures that in use we pay only for the cost of indexes that we need to use.

Currently, `Horned-OWL` provides data structures for ontologies with four or less indexes; there is no fundamental limitation here, and higher numbers would be possible. Within `Horned-OWL` itself, three indexes are the most that are used at one time, for the RDF reader.

The simplest ontology index is the `SetIndex` with associated `SetOntology`. This simply stores all ontology components in a memory-backed set. It is useful on its own, but also in conjunction with other indexes, because it guarantees to record all components added to it. However, access to components of the ontology requires iteration.

Perhaps the most useful index is the `ComponentMappedOntology`; this allows rapid access to ontology components based on their type. For example, the following code, which is a unit test, shows how to access to all the `DeclareClass` axioms. As with the `SetOntology`, this also stores all components added to it.

```

1 let mut o = ComponentMappedOntology::new_rc();
2 let b = Build::new_rc();
3 o.declare(b.class("http://www.example.com/a"));
4 assert_eq!(o.i().declare_class().count(), 1);

```

Other ontology index types include:

**Declaration Mapped:** Look up the type of a named entity given an IRI

**IRI Mapped:** Look up components containing a given IRI

**Logically Mapped:** Look up components logically equal (i.e. ignoring annotations) to an existing component

The importance of these ontology indexes cannot be overstated. For instance, the RDF reader constructs and returns an `RDFOntology` which uses a `SetIndex`, `DeclarationMappedIndex` and a `LogicallyMappedIndex`. This is critical because RDF parsing requires regularly looking up triples that have been already parsed to understand and decode later triples; in particular, it must understand the declared type of many IRIs. Without use of the `DeclarationMappedIndex` this requires a full iteration of the ontology for each lookup which, in practice, means the RDF parser would operate in cubic or worse time. This would result in catastrophically poor performance; our first naive implementation of RDF parsing took well over an hour to parse the Gene Ontology for example. The equivalent XML parser does not need these indexes as the type of an IRI is always given at point of use; the plugging indexing system means, that is does not have to pay the cost of building them.

<sup>1</sup> Restrictions in its type system means that it was not possible to represent this notion directly in Rust. The recent addition of “Generic Associated Types” may make this representation possible; however, GATs are currently quite limited and their availability came well after `Horned-OWL` was developed.

## 2.4 Input/Output Framework

In its current version, Horned-OWL supports several different syntaxes of OWL: the OWL/XML syntax defined by part of the OWL 2 specification <sup>2</sup>, the OWL/RDF syntax and the OWL functional syntax. The `io` module contains a submodule for each syntax, which in turn consists of a `reader` and a `writer` module. Each of these different formats has some idiosyncrasies.

We first offered support for the OWL/XML syntax, as it was relatively straightforward to implement. The reader works on a single file at a time and returns both an ontology (currently, a `SetOntology`) and a `PrefixMapping` supplying the IRI prefixes. In this case, both the reader and writer heavily rely on the `quick-xml` crate.

Supporting the OWL/RDF syntax is rather more complex. One of the main challenges is that, to correctly parse an ontology written using this syntax, we must also recursively parse all the imported RDF graphs to determine the kind of entity associated to an IRI, which may be declared in one of the imported graphs or later in the ontology that is currently being parsed. In contrast for OWL/XML this is not needed, as the type of each IRI is given at the point of use. For this reason, the OWL/RDF reader, which reads a single ontology, comes with a companion `closure_reader` which attempts to parse the full import closure. This has been optimised so that the `closure_reader` only parses those parts of the ontology that are absolutely necessary – imported ontologies are only parsed until all the IRIs encountered previously are associated to a declared entity.

Another issue is that no ordering of the triples in the RDF graph is guaranteed, and so the parser may have to traverse the list of RDF triples multiple times; in some more perverse cases, this could result in poor performance; for example, an RDF list where the triples appear from the last item to the first would parse in quadratic time; in practice, all the ontologies we have found in RDF use a first to last appearance which parses in linear time. As was noted in Section 2.3, Horned-OWL indexes are used to avoid other cases which would result in quadratic or worse performance. The reader, therefore, returns a highly indexed ontology type. To read RDF triples, the reader relies on the `rio` crate.

The OWL/RDF writer in Horned-OWL uses the `pretty_rdf` crate, which was implemented specifically to support Horned-OWL. The `rio` crate, which is used in the reader, is focused on serialisation of RDF as a set of triples; `pretty_rdf`, conversely, supports many of the RDF shorthand syntaxes, something that the OWL API also supports. This makes the RDF output more readable and concise, at the cost of increased time and complexity in serialisation.

Support for the OWL functional syntax is a recent addition to Horned-OWL; this uses the `Pest` crate to generate the parser from a parsing expression grammar. The writer is implemented through a Rust trait and supports writing most types of the `horned-owl` crate without copying data or requiring a dedicated writer type.

Although not yet complete, we plan to fully support the Manchester syntax for ontologies, which will bring Horned-OWL into parity with the OWL API in terms of the syntaxes it supports <sup>3</sup>.

Furthermore, the fact that `rio` supports other RDF syntaxes such as N3 and Turtle means that Horned-OWL could be further expanded to offer readers for these formats; with a little more effort, we could also support writing in these formats as `pretty_rdf` uses a data model very similar to `rio`, and `rio` can write to these syntaxes. However, we lack a good use case for doing this work. One current limitation, however, is that `Rio` does not return an IRI prefix mapping for RDF which makes roundtripping hard.

---

<sup>2</sup> <https://www.w3.org/TR/owl2-xml-serialization/>

<sup>3</sup> We remain a little conflicted as to whether this is a good thing; while each of the syntaxes have their advantages, it is not so clear why so many syntaxes are needed



## 3 Ecosystem

### 3.1 Command Line

As well as providing a library, **Horned-OWL** provides an increasing number of command line tools. These support a git-style subcommand command line and provide tools for operating on OWL files in batch. For the current release, these tools are rather biased toward the sort of functionality needed for debugging **Horned-OWL**, but we expect these to expand into a full suite for OWL manipulation in batch in due course. Currently available tools include:

**horned-big**: Generates OWL files of arbitrary size, useful for performance testing

**horned-compare**: Compares the statistics of two ontologies

**horned-materialize**: Downloads the OWL import closure

**horned-parse**: Parses an OWL file for errors only

**horned-summary**: Provides summary statistics of an ontology

### 3.2 Reasoning interface and **whelk-rs**

**Horned-OWL** includes a preliminary reasoner interface defining functions for classifying ontologies, checking entailments and consistency, and retrieving inferred superclasses and subclasses. This interface is implemented by **whelk-rs**, a port to Rust of the **Whelk** reasoner, which targets the Java OWL API, which is an adaptation of the reasoning rules defined by Kazakov et al. [8]. **whelk-rs** supports the OWL EL profile, a subset of the OWL language targeted to scalable reasoning on large, structured terminologies such as biomedical ontologies. Initial testing suggests that **whelk-rs** is approximately twice as fast as the original Scala-based **Whelk** reasoner. Our expectation is that performance will improve further with plans to adopt a more idiomatic Rust coding style in the future.

### 3.3 Python bindings and **py-horned-owl**

One of the shortcomings of Rust is its learning curve, which is notoriously steeper than that of many other programming languages. In particular, the ownership and borrowing mechanisms of Rust and the fact that it is strongly typed result in this language being not very well suited for prototyping and short development cycles. On the other hand, a language like Python is rather easy to use and widely employed in many industrial and research areas, including scientific computing; however, it tends to be rather slow, especially for highly computational tasks.

The goal of the **py-horned-owl**<sup>4</sup> library is to make the power and functionality of **Horned-OWL** available to Python programmers. In this, we are following a frequent design pattern, used by libraries such as NumPy or SciPy, implementing a front-end for most use in Python with a backend written in a more performant native language (C, C++ or Fortran in the case of NumPy and SciPy).

Under the hood, **py-horned-owl** uses the PyO3 bindings to map the data structures, enums and functions defined in **Horned-OWL** to Python classes, union types and methods. This allows for the creation and manipulation of ontology components within the Python environment, similarly to what is done in **Horned-OWL** for Rust. For example, **py-horned-owl** provides the **PyIndexedOntology** class, representing an ontology with helper methods to query its components based on one or more indexes. One of these indexes is **IRIMappedIndex**, which allows quick access to components by their IRI.

---

<sup>4</sup> <https://github.com/ontology-tools/py-horned-owl>

## 9:8 Horned-OWL: Flying Further and Faster with Ontologies

We can load existing ontologies using the `open_ontology` method, which supports all OWL serialisations available in Horned-OWL. The axioms and components of the ontology can be queried. The following illustrates a typical scenario of usage:

```
1 import pyhornedowl
2 ontology = pyhornedowl.open_ontology("<path/to/ontology>")
3 # Get all components
4 components = ontology.get_components()
5 # Construct an axiom
6 from pyhornedowl.model import *
7 axiom = SubClassOf(
8     ontology.clazz(":Child"),
9     ObjectSomeValuesFrom(
10         ontology.object_property(":has_parent"),
11         ontology.clazz(":Human"))
12 )
13 # Add the axiom
14 ontology.add_axiom(axiom)
```

Since PyO3 outputs native Python modules, the static type information asserted in Rust is lost in the conversion process. However, `py-horned-owl` provides stubs that encode type information and provide hints which, for example, allow IDEs or other tools to do static type checking.

The development of `py-horned-owl` has directly influenced that of `Horned-OWL`. In earlier versions of the crate, IRIs were implemented as a newtype wrapper around `Rc<str>`; however, this type cannot be used across threads which is required for PyO3, necessitating instead the use of `Arc<str>`; this type is fully synchronized which, according to the documentation, comes at a 20-30% performance cost. To avoid paying unnecessary allocation and performance costs and retain flexibility, the types used in `Horned-OWL` have been then made fully generic, leading to the current version of the crate.

## 4 Evaluation

### 4.1 Testing

`Horned-OWL` contains an extensive series of tests to ensure consistency and compliance to the OWL2 specification: in total there are 928 unit tests, 46 doc tests (which are unit tests but visible as examples in code documentation) and 7 integration tests. The test set takes around 3s to run; the majority of this time comes from the doc tests, which is quite normal for Rust.

The bulk of these tests come from the IO framework. All of the readers and writers use a common series of tests. Test files are written using `Tawny-OWL` [11] which provides a clean, declarative and documentable representation of OWL, backed by the OWL API [11]. These files are then used to generate all the other required formats. For the OWL reader, tests are written by hand and compared to pre-determined semantics hard-coded in Rust; for example we generate ontology serialisations containing a single class declaration:

```
1 (defclass C)
```

The generated OWX file is parsed and resultant `Horned-OWL` ontology is checked as follows:

```
1 assert_eq!(ont.i().declare_class().count(), 1);
2 assert_eq!(
3     String::from(&ont.i().declare_class().next().unwrap().0),
4     "http://www.example.com/iri#C"
5 );
```

Other readers are checked by parsing and comparing the ontology to that generated by parsing the OWX files. Writers are tested by roundtripping: reading, writing and reading again. These tests are predominantly defined parametrically: addition of a new ontology defined in Tawny-OWL will automatically result in new tests for all serialisations; similarly, we will be able to test any new serialisations against these ontologies.

There are a few other forms of test: the RDF reader includes some tests with triples in differing orders which Tawny-OWL cannot directly produce; and some tests cannot directly compare RDF or OWX read ontologies as these two formats differ slightly in their expressive capability. The test set is extensive with 130 different ontologies defined in Tawny-OWL, defining 373 logical components, 13 annotations, and 126 meta components (Ontology IRIs). All 48 of the Horned-OWL component types are used in these ontologies (each type of OWL axiom, SWRL rule and meta component) are present in these files, excepting doc IRIs which are implicit.

We have recently adopted the pre-commit framework, meaning that our tests are defined declaratively and run before commit or push; additionally, standardized format and use of Rust idiom are enforced through the `rustfmt` and `clippy` tools respectively.

## 4.2 Performance

In order to test the performance of Horned-OWL we perform a series of tasks, using real world ontologies where possible. This performance testing can be found in the owl-performance repository and was conducted on a Ubuntu (64-bit) VM with 24GB of memory<sup>5</sup>.

In Figure 1, we show the results of generating a simple large ontology: in this case, a set of OWL class declarations. This predominately tests for in-memory performance at constructing an ontology, then serialization of a structurally simple ontology; this functionality is built into Horned-OWL directly; equivalent code was written for the OWL and `py-horned-owl`. Horned-OWL shows approximately linear performance and is faster than the OWL API. `py-horned-owl` scales similarly.

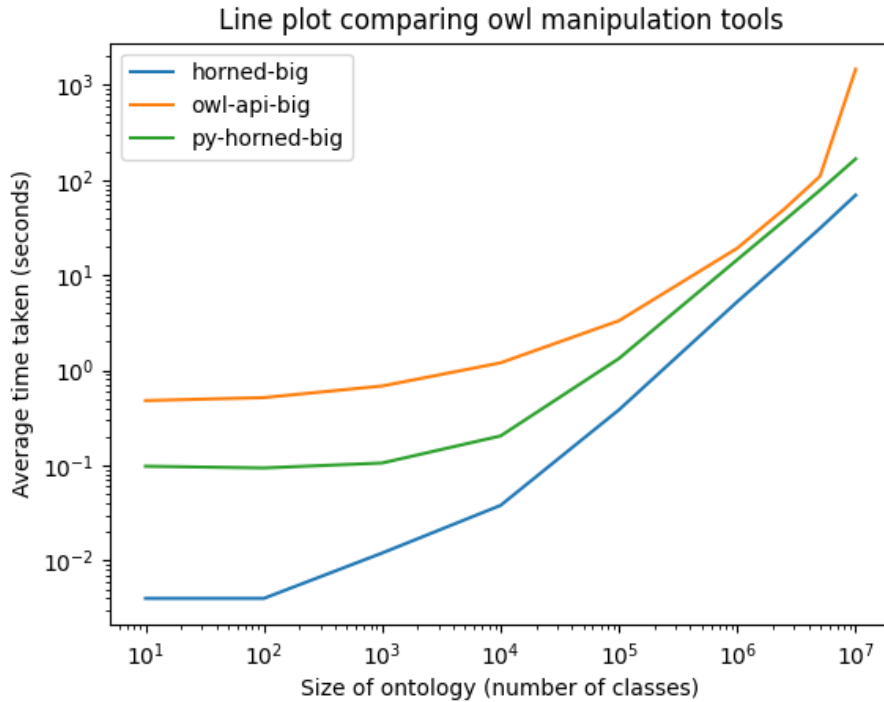
The use of an artificial form of ontology is useful for testing scalability, because we can generate ontologies with an arbitrary number of axioms. However, the performance against real world ontologies might be quite different, and for this reason, we test against these next. We have chosen a number of well-known ontologies present in BioPortal; these differ in their size, in terms of logical content, annotation and the complexity of their axiomatisation. The largest, the NCBI taxonomy, representing the species taxonomy, is the simplest consisting only of classes, annotations and subclass statements. Details are shown in Table 1.

■ **Table 1** Size and Complexity of Test Ontologies.

	No. Triples	No. Component	Size on Disk
BFO	1221	741	155k
GO	256778	752036	121M
ChEBI	7466140	4281507	772M
NCBI Taxonomy	17648344	16134154	1.5G

We test the parsing performance by simply parsing these ontologies and testing against time, as shown in Figure 2. Horned-OWL is faster in all cases. As can be seen, `py-horned-owl` shows a small performance penalty. Some of this may be because of the cost of the interface between Python and Rust, or the use of the synchronized `Arc` instead of the single-threaded `Rc`.

<sup>5</sup> The JVM was set to a max heap size of 10G.



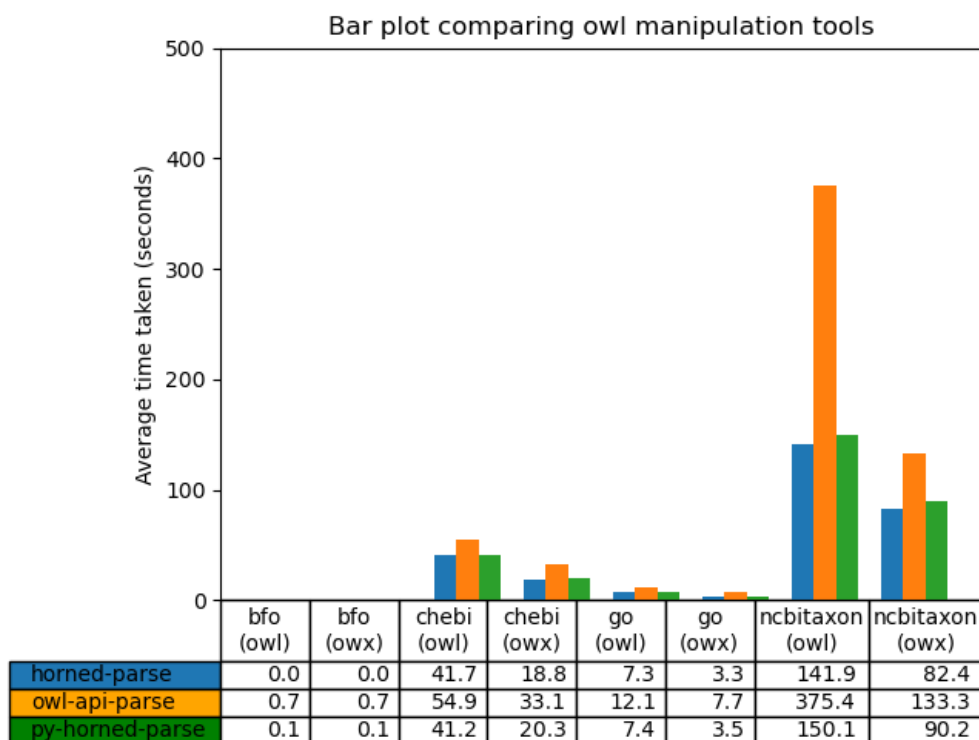
■ **Figure 1** Generating a big ontology.

Finally, we test memory usage. This is a somewhat complex task given the different nature of the environments we are testing. Rust has deterministic memory use, but both Java and Python are garbage collected and will tend to use the memory that is available to them. We took, therefore, the simple approach of running each in restricted memory, we achieved on Ubuntu through the use of the `systemd-run` command. We test only whether the parsing completed or not. As can be seen from Figure 3, Horned-OWL is capable of running in a memory constrained environment. We have additionally tested extreme memory constraints: Horned-OWL is capable of parsing `bfo.owl` in 2M of memory, which is 20x smaller than the OWL API.

Performance testing is always a difficult task: there are many factors and variables to control and the tasks carried out are often time-consuming making heavy use of CPU. The results are frequently insightful, however; as a result of the work for this paper, we uncovered a performance bug in the associated `pretty_rdf` crate that resulted in poorly scalable performance while writing RDF<sup>6</sup>; likewise our analysis of `py-horned-owl` has made us reconsider the use and representation of indexes which resulted in substantial performance improvements. However, we will always be limited in a capacity to make such improvements while the performance testing is hard; therefore, we have now re-implemented a formal benchmarking harness for Horned-OWL; Rust support has considerably advanced since our first effort in this area; this should make our performance test results less ephemeral and will make future decisions on optimisations more possible.

In short, it is clear that Horned-OWL is already highly performant in terms of both CPU and memory usage and has a clear path to becoming more so.

<sup>6</sup> A three-line bugfix restored linear rather than quadratic performance which meant writing NCBI taxonomy became practical; see [https://github.com/phillord/pretty\\_rdf/commit/66466737](https://github.com/phillord/pretty_rdf/commit/66466737)



■ **Figure 2** Parsing various well known ontologies.

### 4.3 Comparison to other libraries

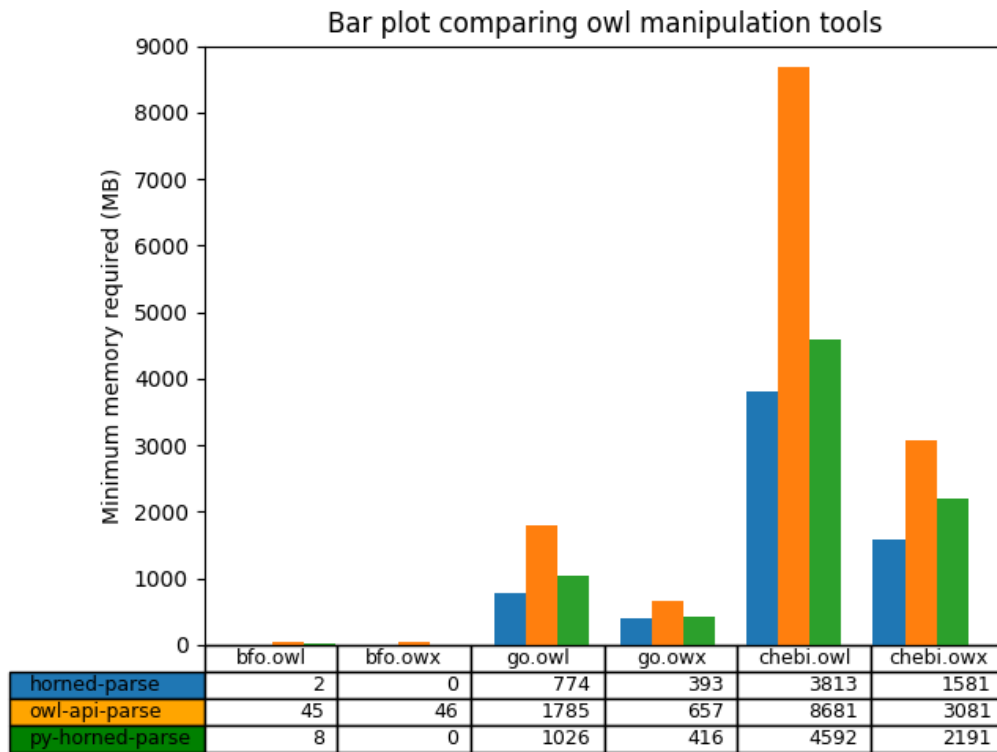
Since the Horned-OWL project started a number of other OWL libraries have been developed, which cover some of the same ground as Horned-OWL. Most notably here is OWLReady (now OWLReady2) which is a Python implementation of OWL2 [9]; its emphasis is on a high-level Object Oriented interface in Python for OWL, somewhat similar to Tawny-OWL [11] which takes a similar approach in Clojure. While this form of interface is very convenient for programming, our experience with both Tawny-OWL and the OWL API is that it is not as convenient for dynamic ontology manipulation and analyses. Additionally, it is likely to come with an overhead; our early analysis tends to confirm this, as Horned-OWL appears to be significantly more performant than OWLReady. A more recent addition to this space is COWL [2]; this is aimed at memory constrained embedded systems; it provides a data model and supports functional syntax only. Currently, Horned-OWL cannot perform well in this space, but we note that Rust does provide strong support for embedded systems through the `no_std` environment; this could be supported in later versions of Horned-OWL, providing a crate that, like COWL, supports only the data model and limited syntax options.

### 4.4 Limitations

Horned-OWL has a number of limitations.

We are currently testing Horned-OWL against all ontologies in BioPortal [13]. Unfortunately, the complexity of OWL means that comparing results to the OWL API is non-trivial; in addition, there are some known failures.

In the ideal world, given the predominance of the OWL API for the generation of OWL, we would like Horned-OWL to be identical with OWL API serializations. This is extremely challenging for a number of reasons; and this is particularly true for the RDF serialisation of OWL. First,



■ **Figure 3** Bar chart showing the affect restricting memory has on the parsing tools.

for a general OWL ontology it is neither possible to determine unambiguously what the RDF serialisation is nor, in reverse, determine the OWL ontology from a given RDF representation. Second, to add to this complexity, RDF provides a number of “shortcut” syntaxes which are both complex to implement and mean that a single RDF graph can be serialized in many different ways.

Similarly, while the XML representation of OWL is much less ambiguous and should roundtrip cleanly whether produced by the OWL API or Horned-OWL, the two are not currently lexically identical if for no other reason than for the use of whitespace. These lexical differences also impact on RDF/XML representation of OWL. This would be problematic for uses of Horned-OWL where ontologies are stored in tools such as git; switching regularly between the OWL API and Horned-OWL would result in a large number of misleading diffs. This could be circumvented by roundtripping the final ontology in a pipeline using either Horned-OWL or the OWL API. We note that a similar issue is currently caused by different versions of the OWL API which do not produce whitespace identical serialisation; we are sure the same issue will face Horned-OWL as it evolves.

We note that some differences in behaviour between Horned-OWL and the OWL API are pushing at the limitations of the OWL specification; it is not always clear which is correct. For example, the OWL API will produce empty IRI tags (`<IRI></IRI>`) which Horned-OWL refuses to parse; we see the impact of this in the performance testing, as Horned-OWL currently cannot parse `bfo.owlx`. Similarly the OWL API adds typing information for built in classes (`<owl:AnnotationProperty rdf:about="http://www.w3.org/2000/01/rdf-schema#comment"/>`), which are probably correct but unnecessary; Horned-OWL benignly reports these as unhandled.

## 5 Discussion

Horned-OWL itself is now feature complete for OWL2, as well as including SWRL rules. As these specifications are now stable and themselves unlikely to evolve, our hope is that Horned-OWL itself will now show a similarly slow evolution. The core library (i.e. the Horned-OWL crate) now contains the data model and serialisation: the other features that we have added (the indexing described earlier, plus a visitor and some normalisation functionality) were necessary for efficient implementation of this core.

This does not mean that the overall environment of Horned-OWL will not expand; however, we will do so by adding additional crates. We have already gone this route by removing the command line function to its own crate (`horned-bin`) albeit one managed in the same repository as Horned-OWL; again, this is for reasons of performance; users of the library should not need to bear to the cost of additional dependencies required by these binaries. In the case of the command line library, we already have a good idea of the functionality that it is likely to need: the ROBOT tool [7] which is built on the OWL API, provides a clear exemplar here. For `py-horned-owl`, there is no real equivalent capability for scripting OWL and it will be interesting to see what functionality will be developed there. Finally, we note a possibility raised by Horned-OWL that we have not yet fully explored: Rust has strong support for WebAssembly which raises the possibility that OWL might yet become usable on the web.

One key area limitation for the current Horned-OWL ecosystem is in reasoning. Currently, `whelk-rs` provides support for the EL profile, but there is no DL reasoner available. We note that history has not been kind to many OWL2 reasoners with most abandoned or no longer usable [1]. Nonetheless, it should be possible to either port one of these to Rust, as `whelk-rs` has been, or use them directly through the Rust C ABI interface. We have not yet begun exploring whether this would be possible, nor whether it would be sensible, since the high worst case complexity of DL means the reasoners might not scale to the size of ontology that Horned-OWL can otherwise handle.

The initial experiments on Horned-OWL started over seven years ago. At this time, Rust was relatively immature, making initial progress quite slow. It has been pleasing to see that both the language and ecosystem has advanced substantially since this time. This has included an increased support for Semantic Web technologies: Horned-OWL for example, makes use of the `rio`<sup>7</sup> crate which provides RDF parsing support. Other Semantic Web technologies that are supported in Rust include SPARQL through the `oxigraph`, SHACL and shape expressions through the `rudof` crate and Linked Data through the `sophia` framework [3], to mention a few. Horned-OWL fits cleanly into this ecosystem, by providing support for OWL.

We believe that Horned-OWL is essential to the future utility and importance of the OWL specification and ontologies more generally. We have already noted the practical reality that Python now has completed a virtual take over in scientific computing, and that without good support for OWL in this language, scientists will simply move to other technologies. More importantly, however, while ontologies have been enormously successful, particularly in biomedicine, they now feature in much of the same space as newer AI technologies; these have highlighted what has always been a fundamental limitation of scalability. Horned-OWL cannot fully resolve this problem, but with its focus on performance it is a step in the right direction.

---

<sup>7</sup> <https://github.com/oxigraph/rio>

## References

- 1 Konrad Abicht. OWL reasoners still useable in 2023, 2023. [arXiv:2309.06888](https://arxiv.org/abs/2309.06888), [doi:10.48550/arXiv.2309.06888](https://doi.org/10.48550/arXiv.2309.06888).
- 2 Ivano Bilenchi, Floriano Scioscia, and Michele Ruta. Cowl: A lightweight OWL library for the semantic web of everything. In Giuseppe Agapito, Anna Bernasconi, Cinzia Cappiello, Hasan Ali Khattak, In-Young Ko, Giuseppe Loseto, Michael Mrissa, Luca Nanni, Pietro Pinoli, Azzurra Ragone, Michele Ruta, Floriano Scioscia, and Abhishek Srivastava, editors, *Current Trends in Web Engineering - ICWE 2022 International Workshops, BECS, SWEET and WALIS, Bari, Italy, July 5-8, 2022, Revised Selected Papers*, volume 1668 of *Communications in Computer and Information Science*, pages 100–112. Springer, 2022. [doi:10.1007/978-3-031-25380-5\\_8](https://doi.org/10.1007/978-3-031-25380-5_8).
- 3 Pierre-Antoine Champin. Sophia: A Linked Data and Semantic Web toolkit for Rust. The Web Conference 2020: Developers Track, April 2020. URL: <https://www2020devtrack.github.io/site/schedule>.
- 4 Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Journal of Web Semantics*, 6(4):309–322, 2008. Semantic Web Challenge 2006/2007. [doi:10.1016/j.websem.2008.05.001](https://doi.org/10.1016/j.websem.2008.05.001).
- 5 W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition), 2012. URL: <https://www.w3.org/TR/owl2-overview/>.
- 6 Matthew Horridge and Sean Bechhofer. The OWL API: A java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011. [doi:10.3233/SW-2011-0025](https://doi.org/10.3233/SW-2011-0025).
- 7 Rebecca C. Jackson, James P. Balhoff, Eric Douglass, Nomi L. Harris, Christopher J. Mungall, and James A. Overton. ROBOT: A tool for automating ontology workflows. *BMC Bioinform.*, 20(1):407:1–407:10, 2019. [doi:10.1186/S12859-019-3002-3](https://doi.org/10.1186/S12859-019-3002-3).
- 8 Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. The Incredible ELK - From Polynomial Procedures to Efficient Reasoning with  $\mathcal{EL}$  ontologies. *J. Autom. Reason.*, 53(1):1–61, 2014. [doi:10.1007/S10817-013-9296-3](https://doi.org/10.1007/S10817-013-9296-3).
- 9 Jean-Baptiste Lamy. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial Intelligence in Medicine*, 80:11–28, 2017. [doi:10.1016/j.artmed.2017.07.002](https://doi.org/10.1016/j.artmed.2017.07.002).
- 10 Phillip Lord. Horned-OWL. Software (visited on 2024-11-29). [doi:10.4230/artifacts.22531](https://doi.org/10.4230/artifacts.22531).
- 11 Phillip Lord. The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL, 2013. [arXiv:1303.0213](https://arxiv.org/abs/1303.0213).
- 12 Phillip Lord, Martin Larralde Björn Gehrke, Janna Hastings, Filippo De Bortoli, James A. Overton, James P. Balhoff, and Jennifer Warrender. Horned-OWL. Software (visited on 2024-11-29). [doi:10.4230/artifacts.22530](https://doi.org/10.4230/artifacts.22530).
- 13 Natalya F. Noy, Nigam H. Shah, Patricia L. Whetzel, Benjamin Dai, Michael Dorf, Nicholas Griffith, Clement Jonquet, Daniel L. Rubin, Margaret-Anne Storey, Christopher G. Chute, and Mark A. Musen. BioPortal: ontologies and integrated data resources at the click of a mouse. *Nucleic Acids Research*, 2009. [doi:10.1093/nar/gkp440](https://doi.org/10.1093/nar/gkp440).
- 14 The Gene Ontology Consortium. The Gene Ontology Resource: 20 years and still GOing strong. *Nucleic Acids Research*, 47(D1):D330–D338, January 2019. [doi:10.1093/nar/gky1055](https://doi.org/10.1093/nar/gky1055).