Declarative RDF Construction from In-Memory Data Structures with RML

Ioannis Dasoulas[1,3,4,*], David Chaves-Fraga[1,2,3], Daniel Garijo[2] and Anastasia Dimou[1,3,4]

[1]KU Leuven, Department of Computer Science, B-2860, Sint-Katelijne-Waver, Belgium
[2]Universidad Politécnica de Madrid, Campus de Montegancedo, Boadilla del Monte, Spain
[3]Flanders Make@KULeuven, B-3000 Leuven, Belgium
[4]Leuven.AI – KU Leuven institute for AI, B-3000 Leuven, Belgium

Abstract

Knowledge graphs are often constructed from heterogeneous data sources using declarative mapping languages. Mapping languages define rules that apply ontology terms to raw data to describe how a knowledge graph should be constructed from these raw data. While most mapping languages and systems support knowledge graph construction from different data formats, e.g., CSV, XML or JSON, and different types of data sources, e.g., files, Web APIs or databases, there is still no support for mapping *in-memory data structures* to knowledge graphs, i.e. data which is temporarily stored in RAM. Currently, this data must first be stored in HDD, locally or in the cloud, for RDF construction systems to access them and construct a knowledge graph. However, writing these data to HDD and reading from HDD is a computationally expensive and redundant task. In this paper, we propose a method to construct RDF graphs from data produced by a software process and stored in RAM. We introduce an extension of RML's Logical Source to describe data structures produced by software, and exemplify our proposal with Python data structures. We extend a well-known RML system, Morph-KGC, to show the feasibility of our method and validate this extension with two use cases: OpenML, which translates machine learning executions into RDF, and SOMEF, which extracts software metadata from its documentation, converting them to triples. This proposal simplifies the construction of RDF graphs from in-memory data structures stored temporarily in RAM and enables the integration of data stored both in RAM and HDD.

Keywords

Knowledge Graphs, Mapping Languages, RML

1. Introduction

Graphs represented with the Resource Description Framework (RDF)[1] and knowledge graphs (KGs), in general, have become increasingly popular as a means of representing and analyzing

Hersonissos'23: Fourth International Workshop On Knowledge Graph Construction, May 28, 2023, Hersonissos, GR
*Corresponding author.
ioannis.dasoulas@kuleuven.be (I. Dasoulas); david.chaves@upm.es (D. Chaves-Fraga); daniel.garijo@upm.es (D. Garijo); anastasia.dimou@kuleuven.be (A. Dimou)
https://orcid.org/0000-0002-8803-1244 (I. Dasoulas); http://davidchavesfraga.com (D. Chaves-Fraga); https://dgarijo.com (D. Garijo); https://orcid.org/0000-0003-2138-7972 (A. Dimou)
0000-0002-8803-1244 (I. Dasoulas); 0000-0003-3236-2789 (D. Chaves-Fraga); 0000-0003-0454-7145 (D. Garijo); 0000-0003-2138-7972 (A. Dimou)

[1]https://www.w3.org/RDF/

complex data [1]. Mapping languages (e.g., the Relational to RDF mapping language (R2RML) [2] and its extension, e.g., the RDF mapping language (RML) [3]) construct RDF graphs based on a set of declarative rules, defined by the languages' syntax. These rules define how the data should be represented as RDF graphs using terms from ontologies [4] and are executed by RDF construction systems that construct the RDF graph.

While existing mapping languages and systems support KG construction from heterogeneous data sources, *in-memory* data structures are not fully supported. In-memory data structures are produced by a software program or application and are temporarily stored in Random Access Memory (RAM) (e.g., SPARK DataFrames, C++ linked lists, in-memory databases, etc.) [5]. As mapping languages assume that data sources are stored in the Hard Disk Drive (HDD), current KG construction system cannot construct RDF graphs from in-memory data structures. As a result, in-memory data structures are firstly stored locally or in the cloud and, then the KG construction systems load the data, either sequentially, e.g., RMLMapper [3] or ShExML [6], or in parallel, e.g., RMLStreamer [7] or Morph-KGC [8]. This process can be computationally expensive, as data is converted from one structure to another, read from and written to HDD [4]. While supporting in-memory Relational Databases (RDBs) (e.g., SQlite[2] or Redis[3]) is barely a matter of configuration, as they use the same connectivity interfaces as regular RDBs, the same does not hold for NoSQL Databases (DBs). In-memory NoSQL DBs (e.g., TinyDB[4] or ZODB[5]) access in-memory data structures in a different way than regular NoSQL DBs.

In this paper, we propose a method for constructing RDF graphs from in-memory data structures by extending RML to describe data produced by a software, independently of their internal structure. We use the Software Description Ontology (SDO) [9] to annotate the metadata about the software and the data that the software produces and exemplify this extension for Python dictionaries and Pandas[6] DataFrames. Our solution leverages data generated in run-time to construct RDF graphs without first having to store them and combines in-memory and locally stored heterogeneous data which was not possible so far.

We extend Morph-KGC [8], a scalable Python KG construction system that outperforms state-of-the-art systems in terms of execution time. Morph-KGC-RAM [10] our extension, is a Python package that adds the functionality of in-memory data structures integration, using the proposed RML syntax extension. Our approach requires the software that generates the data and the KG construction systems to run within the same process and share physical resources. After the data structures are generated and processed, the KG construction package can use them in combination with declarative mapping rules to transform them into RDF.

We validate our extension with two use cases from the data mining domain where an RDF graph is constructed from Python-based software. The OpenML [11] use case extracts metadata about machine learning experiments and translates them into RDF, while the SOMEF [12] use case extracts software metadata from its documentation and converts them to triples.

The contributions of this paper are the following: (i) an extension of RML's Logical Source to describe data sources produced by a software; (ii) a preliminary proof-of-concept of our

[2] https://sqlite.org/index.html
[3] https://redis.io
[4] https://tinydb.readthedocs.io/en/latest/
[5] https://zodb.org/en/latest/index.html
[6] https://pandas.pydata.org

extension for Python data structures; and (iii) a demonstration of two use cases for in-memory data integration to RDF. The remaining paper is structured as follows: Section 2 describes related work. Section 3 presents our extension of RML using the Software Description Ontology (SDO). Section 4 describes the Morph-KGC extension, while Section 5 discusses the use cases that the extension was used. Section 6 outlines our conclusions and plans for future work.

2. Related Work

Different software packages provide an abstraction layer for users to construct RDF graphs from in-memory data structures using ad-hoc scripts. KGLab[7], for example, is a software package based on RDFLib[8] to construct KGs from Python data structures. Nevertheless, these packages do not support the transformation of in-memory data structures to RDF graphs using mapping languages, requiring users to specify their own rules for integrating data into RDF graphs. While these solution works well for small-scale projects, it is harder to maintain and scale [13].

Mapping languages provide a standard formalization for KG construction, they are more interoperable than ad-hoc solutions and highly reusable for different types of data [3]. Over the past decades, several mapping languages have been proposed to describe the construction of KGs from heterogeneous data sources in a declarative way [4]. R2RML [2] and RML [3] are two of the most popular languages, supported by a large amount of KG construction systems [4]. KG construction systems are software implementations that process data sources and mapping rules to generate RDF graphs. Despite the large amount of KG construction systems [4], none of them supports in-memory data structures so far.

To describe in-memory data structures, we investigated ontologies which describe different aspects of a software that produces these data structures, e.g., projects, components, and processes. The **Description of a Project Ontology (DOAP)**[9] is a popular ontology for describing software projects. DOAP focuses on software organization and version control, without providing detailed annotations about data. Thus, it cannot be used to describe information about data structures and the characteristics of the software that produces them. The **Software Ontology (SWO)**[10] is a comprehensive ontology for software artifacts, covering topics such as software applications, libraries, and operating systems. Originally created for bioinformatics-related software, SWO reuses high-level Open Biomedical Ontologies to annotate software components, but it does not describe the expected content of data. In addition, SWO contains thorough taxonomies about data transformation techniques and programming languages, but defines them as classes, which reduces its flexibility and poses difficulties for reusing these concepts as instances[10]. The **Core Software Ontology (CSO)**[11] and its extension **Core Ontology of Software Components (COSC)** [14] provide concepts for describing software services and components. However, they are not being actively documented, while they require users to adapt to some unique formalizations they provide, thus making the ontologies hard to reuse. **OntoSoft** [15] and its extension **OntoSoft-VFF** [16] capture scientific software metadata from a

[7] https://derwen.ai/docs/kgl/
[8] https://rdflib.readthedocs.io/en/stable/
[9] http://usefulinc.com/ns/doap
[10] http://purl.obolibrary.org/obo/swo.owl
[11] http://km.aifb.kit.edu/sites/cos/

scientist's perspective, focusing mostly on provenance and ease of use, by integrating scientific questions in the descriptions of the ontologies' properties. The software description ontology **SD** [9] provides a lightweight set of concepts and relationships to describe software data, their metadata, and their parameters and features. SD provides a number of classes to describe software data, their features and variable presentations. SD favors the reuse of existing vocabularies such Schema.org[12], Codemeta[13] and W3C Data Cubes standard [17], which favors interoperability. Furthermore, it is a well documented ontology, recently tested in RDF systems, such as MODFLOW[14], whereas other well known software ontologies (e.g., Software Ontology (SWO) and Core Software Ontology (CSO)) are not being actively supported. Hence, we concluded that SD is suitable for annotating in-memory data structures and the software that produces them, due to its lightweight nature and preference for reusing existing core ontologies.

3. RML Extension

RML [3] describes customized mapping rules that define how RDF graphs are constructed from heterogeneous data formats (e.g., data in relational databases, data in CSV or JSON format, etc.) and sources, (e.g., databases, files or Web APIs). This data is described in the Logical Source (`rml:LogicalSource`) and the reference formulation (`rml:ReferenceFormulation`) defines how to access the data retrieved from the data source (e.g., `rr:SQL2008` for relational databases, `ql:CSV` for CSV files, `ql:JSONPath` for JSON files).

In this work, we extend RML's Logical Source to provide a way of defining RML mapping rules for in-memory data structures. We introduce a new type of source, the in-memory data structures, as well as new reference formulations, e.g., Python dictionaries, Pandas[16] DataFrames, to express how the retrieved data structures should be accessed.

Our proposed approach provides a uniform way to annotate data structures for different types of software, built in different programming languages. We do not only annotate the data structure, but also the software that produces this data. KG construction systems can leverage information about the software and its dependencies, to identify possible compatibility issues with the software that produces the data structures. For example, systems can evaluate if they support data structures from the programming language version "Python3.9" or "Java20", since there might be small variations for a data structure between different versions of a programming language. Software descriptions help systems detect possible conflicts between their dependencies and the software that produces the data structures, not only regarding programming languages but also software versions. For example, systems can evaluate whether they support data structures from the software package version that satisfies the requirement "pandas>=1.1.0" in Python or "org.apache.commons,commons-lang3,3.12.0" in Java.

Furthermore, software descriptions, such as software dependencies, also increase the reusability and interoperability of RML mapping rules with other software. Using this information, third parties can better understand how the data structures are generated and and the requirements of the software used, e.g., understand what programming language and dependencies to

[12] https://schema.org
[13] https://codemeta.github.io/
[14] https://models.mint.isi.edu/models/explore/MODFLOW/

Listing 1: SD description of Pandas data structures.

```
1  <#Logical-Source> a rml:LogicalSource;
2    rml:source <#In-memory-Source>;
3    rml:referenceFormulation ql:DataFrame;
4  <#In-memory-Source> a sd:DatasetSpecification;
5    sd:name "output_dataframe";
6    sd:hasDataTransformation <#Data-Transformation-Software>;
7  <#Data-Transformation-Software> a sd:DataTransformation;
8    sd:name "DataFrame creation application";
9    sd:hasSourceCode <#Software-Source-Code>;
10   sd:softwareRequirements "pandas>=1.1.0";
11   sd:license "MIT";
12 <#Software-Source-Code> a sd:SourceCode;
13   sd:programmingLanguage "Python3.9";
14 ql:DataFrame a rml:ReferenceFormulation; kg4di:definedBy "Pandas".
```

use, and replicate the complete pipeline of KG construction from in-memory data structures, having more information about the setup they need to accomplish to construct the KG.

In the remaining of the section, we describe how we use the Software Description Ontology as Logical Source and the Reference Formulations that we introduce for the data structures.

Software Description Ontology as Logical Source. We leverage the SD ontology[15] [9] to describe the software that produces as output the in-memory data structures.

In-memory data structures are described as `sd:DatasetSpecification` (Listing 1: lines 4-5), a SD class designed to describe inputs and outputs types of software applications and models. A name representing the data structure is defined (`sd:name`). This name can be used from the KG construction system to identify which data structure should be mapped to RDF (section 4.1). The software that produces the data structure is described as `sd:DataTransformation` (Listing 1: lines 6-11). This can be a software application, a function or a concrete software script. We describe common information about the software such as its name, its dependencies, the source code it uses and its license. Depending on the implementation, more software annotations can be added using SD (e.g., software inputs, parameters, versions). The source code of the software is described as `sd:SourceCode` (Listing 1: lines 12-13).

Reference Formulation for in-memory data structures. We introduce new reference formulations for Python data structures: `ql:Dictionary` for Python dictionaries and `ql:DataFrame` (Listing 1: line 3) for Pandas[16] DataFrames. For in-memory JSON data (Python strings in JSON format) we used the already existing `ql:JSONPath` reference formulation, since this data structure shares the structure of JSON files. Using this information, a system can decide how it can process and parse the data structure.

[15] https://w3id.org/okn/o/sd/
[16] https://pandas.pydata.org

Listing 2: SD description of Java data structures.

```
<#Logical-Source> a rml:LogicalSource;
  rml:source <#In-memory-Source>;
  rml:referenceFormulation ql:LinkedList;
<#In-memory-Source> a sd:DatasetSpecification;
  sd:name "output_linked_list";
  sd:hasDataTransformation <#Data-Transformation-Software>;
<#Data-Transformation-Software> a sd:DataTransformation;
  sd:name "Linked list creation application";
  sd:hasSourceCode <#Software-Source-Code>;
  sd:softwareRequirements "org.apache.commons,commons-lang3,3.12.0";
  sd:license "MIT";
<#Software-Source-Code> a sd:SourceCode;
  sd:programmingLanguage "Java20";
ql:LinkedList a rml:ReferenceFormulation; kg4di:definedBy "Java";
```

The reference formulation contains the name of the data structure (e.g., dictionary). The programming language (e.g., C#) or the software library (e.g., PySpark[17]) that defines the data structure is also specified[18] (Listing 1: line 16). Different programming languages and software libraries provide different ways of organizing and storing data in computer memory, even though they may refer to the data structures in the same way. For example, both Python and C# provide data structures that they refer to as 'dictionary'. Still, the data structures are different as each programming language has a unique way of organizing and storing data. Even if the programming language is common, there might be different software packages that produce different data structures but refer to them using the same name. For example, Pandas[16] and PySpark[17] are Python libraries that produce data structures which they refer to as 'DataFrames'. While both data structures share some similarities in their basic structure and operations (such as indexing and filtering), they have different underlying implementations and are designed to handle different types of data. As a result, there should be a discrete description for them.

Discussion. We showcase how our approach can construct RDF from in-memory data structures for Python using compatible dependencies, but it can be extended for other data structures of other programming languages. Our approach assumes that the software that generates the data is in the same programming language as the KG construction system and they run within the same process and share compatible software dependencies and physical resources. The SD ontology allows describing any software or package by different programming languages, ultimately, yielding any data structure. As long as a reference formulation indicates how to refer to these data structures, and corresponding parsers exist to interpret these references, any in-memory data structure can be addressed. For example, in Listing 2, a Java linked list and the software that produced it are declaratively described. This description can be used by a Java-based KG construction system.

[17] https://spark.apache.org/docs/latest/api/python/
[18] https://w3id.org/kg4di/definedBy

4. Validation

To validate our approach, we implemented our proposed solution in Morph-KGC[19], creating the Morph-KGC-RAM [10] extension (Section 4.1). We applied our approach to two use cases: the Open Machine Learning (OpenML)[20] [11] on the extraction of machine learning experiment metadata and their transformation to RDF and, the Software Metadata Extraction Framework (SOMEF)[21] [18] on the extraction of scientific software metadata from files.

4.1. Implementation

We extended Morph-KGC[19] with the extension Morph-KGC-RAM, which implements our proposed solution. Morph-KGC is a Python software library that can be used in the run-time of other Python-based software and outperforms state-of-the-art systems in execution time, in most cases [8].

Morph-KGC expects a configuration file (.ini) that specifies which mapping rules will be used. We extended Morph-KGC, to also expect Python objects that represent input data sources for the construction of KGs. The Python data structures currently supported are Python dictionaries, Pandas DataFrames and Python strings with the JSON format. Listing 3 shows how the extension can be used to map a Python data structure into RDF, using the RML rules from Listing 1.

Listing 3: Morph-KGC extension that uses the RML file from Listing 1 to generate a knowledge graph from the Python data structure 'my_data'

```
1 import morph_kgc
2 my_data = pd.DataFrame({'Id': [1,2,3],'Username': ["@jude","@emily","@wayne"]})
3 graph = morph_kgc.materialize('./config.ini', {"output_dataframe": my_data})
```

Using this extension, KGs can be constructed from multiple heterogeneous data sources stored in HDD, multiple in-memory data structures stored in RAM or their combination. Following the CI/CD development of Morph-KGC we integrated 72 test cases[22] for KG construction with Python data structures and examples[23] for constructing KGs using Python data structures.

4.2. Use Cases

We demonstrate two use cases from the data mining domain for which we used Morph-KGC-RAM: OpenML and SOMEF. We discuss each one in more details:

OpenML KG: Constructing a Machine Learning Experiment Knowledge Graph. OpenML[20] is an open platform for sharing datasets, algorithms and experiments related to machine learning. In this work, we used OpenML's Python API[24] to connect to the platform and download machine learning experiment data and related metadata, storing them as Python dictionaries and Pandas DataFrames. We used Morph-KGC-RAM to map data from thousands

[19] https://github.com/morph-kgc/morph-kgc
[20] https://www.openml.org
[21] https://somef.readthedocs.io/en/latest/
[22] https://github.com/morph-kgc/morph-kgc/tree/main/test/rml-in-memory
[23] https://github.com/morph-kgc/morph-kgc/tree/main/examples
[24] https://openml.github.io/openml-python/main/

Listing 4: Mapping OpenML dataset metadata into RDF using Morph-KGC-RAM

```
import openml, pandas as pd, morph_kgc
dataset_list = openml.datasets.list_datasets(size=10)
dataset_df = pd.DataFrame.from_dict(dataset_list, orient="index").reset_index()
g_rdflib = morph_kgc.materialize('./config.ini', {"df1": dataset_df})
```

Listing 5: RML Mapping rules for KG construction using OpenML Pandas DataFrames.

```
<Datasets_Map> a rr:TriplesMap;
  rml:logicalSource [ rml:source [
    a sd:DatasetSpecification; sd:name "df1"; sd:hasDataTransformation [
    sd:hasSoftwareRequirements "Pandas>=1.1.0";
    sd:hasSourceCode[ sd:programmingLanguage "Python3.9"; ]; ]; ];
    rml:referenceFormulation ql:DataFrame; ];
  rr:subjectMap [ rr:class mls:Dataset;
    rr:template "http://mldata.com/resource/openml/dataset{did}"; ].
ql:DataFrame a rml:ReferenceFormulation; kg4di:definedBy "Pandas".
```

of experiments into RDF, without having to store them first in the form of a database or a local file. An indicative example of the work on the use case can be seen in Listings 4 and 5. In Listing 4, OpenML's Python API is used to load some metadata about datasets of machine learning experiments in a Pandas DataFrame. Then, with Morph-KGC-RAM, the DataFrame is mapped into RDF, leveraging the mapping rules from Listing 5.[23]

SOMEF: Creating structured metadata from software repositories The SOftware Metadata Extraction Framework (SOMEF) is a Python engine designed to process software code repositories and represent their metadata as an RDF graph. SOMEF extracts these metadata properties using different techniques (regular expressions, supervised classification) and APIs (GitHub API, GitLab API), conflating them in a single, homogeneous record. Internally, the data is stored in a dictionary which is then translated to RDF. In Listing 6, a Python dictionary containing software metadata is translated into RDF using Morph-KGC-RAM. In Listing 7, a simple example of RML rules for the SOMEF use case is demonstrated.

Listing 6: Mapping SOMEF software metadata into RDF using Morph-KGC-RAM

```
import json, pandas as pd, morph_kgc
#somef_dictionary being a SOMEF metadata dictionary
g_rdflib = morph_kgc.materialize('./config.ini', {"dict1": somef_dictioanry})
```

5. Conclusions and Future Work

In this paper, we present an extension for RML's Logical Source, to describe in-memory data structures. This extension enables the definition of mapping rules for in-memory data structures stored in RAM. Moreover, we implement our proposal for Python dictionaries and Pandas

Listing 7: Mapping rules for KG construction using JSON strings of SOMEF software metadata.

```
<Software_Map> a rr:TriplesMap;
  rml:logicalSource [ rml:source [
    a sd:DatasetSpecification; sd:name "dict1";
    sd:hasDataTransformation [ sd:hasSourceCode[
      sd:programmingLanguage "Python3.9"; ]; ]; ];
    rml:referenceFormulation ql:Dictionary; rml:iterator "$"; ];
  rr:subjectMap [
    rr:template "https://www.w3id.org/okn/i/Repo/{full_name.*.result.value}";];
  rr:predicateObjectMap [ rr:predicate emi:applicationDomain;
    rr:objectMap [rml:reference "application_domain.*.result.value"; ]; ].
ql:Dictionary a rml:ReferenceFormulation; kg4di:definedBy "Python".
```

DataFrames, extending Morph-KGC, creating Morph-KGC-RAM, a system to construct RDF from heterogeneous data and Python data structures without having to store them first. We validate our approach over two use cases from the data mining domain, confirming that our approach constitutes a simple setup for KG construction from in-memory data structures.

The performance of our approach in terms of speed and efficiency is yet to be tested, compared to current workflows, where in-memory data structures are first stored for the KG construction system to access them. In the future, we plan to perform evaluations for systems that use this extension to map in-memory data structures into RDF, compared to systems that map data structures to RDF after first storing them in HDD, to measure the efficiency of our approach. Furthermore, we plan to evaluate our approach with more use cases and explore additional data structures, from different programming languages that can be used for KG construction. Finally, we plan to evaluate our approach to other KG construction systems, based on other well known programming languages, such as JAVA and JavaScript.

Acknowledgement

This research was partially supported by Flanders Make, the strategic research centre for the manufacturing industry and the Flanders innovation and entrepreneurship (VLAIO). David Chaves-Fraga and Daniel Garijo are supported by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with Universidad Politécnica de Madrid in the line Support for R&D projects for Beatriz Galindo researchers, in the context of the V PRICIT.

References

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

[2] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, W3C Recommendation, World Wide Web Consortium (W3C), 2012. URL: http://www.w3.org/TR/r2rml/.
[3] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, R. Van de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: Proceedings of the 7th Workshop on Linked Data on the Web, volume 1184, CEUR Workshop Proceedings, 2014. URL: http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.
[4] D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester, A. Dimou, Declarative rdf graph generation from heterogeneous (semi-) structured data: A systematic literature review, Journal of Web Semantics (2022) 100753.
[5] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, et al., Apache spark: a unified engine for big data processing, Communications of the ACM 59 (2016) 56–65.
[6] H. García-González, D. Fernández-Álvarez, J. Labra Gayo, ShExML: An heterogeneous data mapping language based on ShEx, in: European Knowledge Acquisition Workshop, EKAW, volume 2262, 2018. URL: https://ceur-ws.org/Vol-2262/ekaw-poster-08.pdf.
[7] G. Haesendonck, W. Maroy, P. Heyvaert, R. Verborgh, A. Dimou, Parallel RDF generation from heterogeneous big data, in: Proceedings of the International Workshop on Semantic Big Data, 2019, pp. 1–6.
[8] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M. S. Pérez, O. Corcho, Morph-KGC: Scalable knowledge graph materialization with mapping partitions, Semantic Web (2022). doi:10.3233/SW-223135.
[9] D. Garijo, M. Osorio, D. Khider, V. Ratnakar, Y. Gil, OKG-Soft: An Open Knowledge Graph with Machine Readable Scientific Software Metadata, in: 15th International Conference on eScience (eScience), 2019, pp. 349–358. doi:10.1109/eScience.2019.00046.
[10] I. Dasoulas, morph-kgc/morph-kgc: 2.5.0, 2023. URL: https://doi.org/10.5281/zenodo.7829223. doi:10.5281/zenodo.7829223.
[11] J. Vanschoren, J. N. Van Rijn, B. Bischl, L. Torgo, OpenML: networked science in machine learning, ACM SIGKDD Explorations Newsletter 15 (2014) 49–60.
[12] A. Mao, D. Garijo, S. Fakhraei, SoMEF: A framework for capturing scientific software metadata from its documentation, in: 2019 IEEE International Conference on Big Data (Big Data), IEEE, 2019, pp. 3032–3037.
[13] A. Iglesias-Molina, D. Chaves-Fraga, F. Priyatna, O. Corcho, Enhancing the Maintainability of the Bio2RDF Project Using Declarative Mappings., in: SWAT4HCLS, 2019, pp. 1–10.
[14] D. Oberle, S. Lamparter, S. Grimm, D. Vrandečić, S. Staab, A. Gangemi, Towards ontologies for formalizing modularization and communication in large software systems, Applied Ontology 1 (2006) 163–202.
[15] Y. Gil, V. Ratnakar, D. Garijo, OntoSoft: Capturing Scientific Software Metadata, in: Proceedings of the 8th International Conference on Knowledge Capture, K-CAP 2015, Association for Computing Machinery, NY, USA, 2015. doi:10.1145/2815833.2816955.
[16] L. A. M. C. Carvalho, D. Garijo, C. Bauzer Medeiros, Y. Gil, Semantic software metadata for workflow exploration and evolution, in: 2018 IEEE 14th International Conference on e-Science (e-Science), 2018, pp. 431–441. doi:10.1109/eScience.2018.00132.
[17] R. Cyganiak, D. Reynolds, The RDF Data Cube Vocabulary, W3C Recommendation, World Wide Web Consortium (W3C), 2014. URL: https://www.w3.org/TR/vocab-data-cube/.

[18] A. Kelley, D. Garijo, A Framework for Creating Knowledge Graphs of Scientific Software Metadata, Quantitative Science Studies (2021) 1–37. doi:10.1162/qss_a_00167.