Semantic Web -1 (2024) 1–28
DOI 10.3233/SW-243580
IOS Press

Empowering the SDM-RDFizer tool for scaling up to complex knowledge graph creation pipelines[1]

Enrique Iglesias [a,*], Maria-Esther Vidal [a,b,c], Diego Collarana [d] and David Chaves-Fraga [e]

[a] *L3S Research Center Germany, Hannover, Germany*
E-mail: iglesias@l3s.de
[b] *Leibniz University, Hannover, Germany*
[c] *TIB Leibniz Information Centre for Science and Technology, Hannover, Germany*
E-mail: maria.vidal@tib.eu
[d] *University of Bonn, Fraunhofer FIT, Germany*
E-mail: diego.collarana.vargas@fit.fraunhofer.de
[e] *Grupo de Sistemas Intelixentes, Universidade de Santiago de Compostela, Spain*
E-mail: david.chaves@usc.es

Editors: Mehwish Alam, Télécom Paris, Institut Polytechnique de Paris, France; Ruben Verborgh, Ghent University – imec, Belgium
Solicited reviews: Dominik Tomaszuk, University of Bialystok, Poland; Antoine Zimmermann, Mines Saint-Etienne, Univ. Clermont Auvergne, France; one anonymous reviewer

Abstract. The significant increase in data volume in recent years has prompted the adoption of knowledge graphs as valuable data structures for integrating diverse data and metadata. However, this surge in data availability has brought to light challenges related to standardization, interoperability, and data quality. Knowledge graph creation faces complexities from large data volumes, data heterogeneity, and high duplicate rates. This work addresses these challenges and proposes data management techniques to scale up the creation of knowledge graphs specified using the RDF Mapping Language (RML). These techniques are integrated into SDM-RDFizer, transforming it into a two-fold solution designed to address the complexities of generating knowledge graphs. Firstly, we introduce a reordering approach for RML triples maps, prioritizing the evaluation of the most selective maps first to reduce memory usage. Secondly, we employ an RDF compression strategy, along with optimized data structures and novel operators, to prevent the generation of duplicate RDF triples and optimize the execution of RML operators. We assess the performance of SDM-RDFizer through established benchmarks. The evaluation showcases the effectiveness of SDM-RDFizer compared to state-of-the-art RML engines, emphasizing the benefits of our techniques. Furthermore, the paper presents real-world projects where SDM-RDFizer has been utilized, providing insights into the advantages of declaratively defining knowledge graphs and efficiently executing these specifications using this engine.

Keywords: Data integration systems, knowledge graphs, RDF mapping languages

[1] E. Iglesias and M-E. Vidal equally contributed and are the first authors of this work.
[*] Corresponding author. E-mail: iglesias@l3s.de.

1. Introduction

Advancements in data collection devices and methods, such as sensors, wearables, and genomic tests, have resulted in the generation of vast amounts of heterogeneous data, including omics and patient health data. This data is available across various organizations and companies, such as hospitals, universities, and pharmaceuticals. However, the presence of data silos often hampers the combination, analysis, and reuse of this valuable data, which prevents the discovery of insights essential for decision-making. To address this challenge, knowledge graphs (KGs) have gained significant traction in both industrial and academic sectors [32]. KGs provide a unified representation of heterogeneous data, enabling the convergence of data and their meaning. KGs can be defined as data integration systems consisting of a unified schema, data sources, and mapping rules that establish correspondences between the data sources and the unified schema. These declarative definitions of KGs empower modularity and reusability while also allowing users to trace the process of KG creation. Therefore, KGs serve as expressive data structures for modeling integrated data and metadata, and their declarative specifications can be explored, validated, and traced, thus, enhancing transparency and maintenance.

The Semantic Web community has played a relevant role in addressing the challenges associated with integrating heterogeneous datasets into KGs. To tackle this complex task, the community has actively contributed with methodologies, formalisms, and engines aimed at facilitating the creation and maintenance of KGs [6,38,54]. Declarative mapping languages, such as R2RML [21], RML [24], and SPARQL-Generate [40], have emerged as powerful tools within this context. These languages allow knowledge engineers to define mapping rules or assertions [34,44] capable of generating KGs expressed in RDF[2] through systematic evaluations. Mapping rules enable the seamless definition of concepts within a unified schema, encompassing classes, properties, and attributes. This is achieved by harnessing data from diverse sources presented in various formats, including tabular, CSV, JSON, or RDF. The use of declarative mapping languages significantly enhances the flexibility and efficiency of the KG creation process. By providing a standardized approach to mapping data to RDF, these languages empower organizations to extract valuable insights from disparate sources, thereby driving informed decision-making. Furthermore, the Semantic Web community has actively contributed to the development of various engines designed to execute mapping rules [10,17,18,48].

Various RML engines are available such as Morph-KGC [3], RMLMapper,[3] RocketRML [57], and SDM-RDFizer v3.2 [33]. These engines have implemented techniques to execute RML mapping rules efficiently. However, given the variety of parameters that can affect the performance of the KG creation process [15], existing engines may not scale up to KG creation pipelines defined in terms of complex mapping rules or large data sources. Given the amount of available data, new methods are demanded to scale up to complex scenarios where heterogeneous data sources need to be integrated to provide the basis for knowledge analysis and decision-making.

In this paper, our primary research objective is to address the challenge of KG creation through declarative specifications using RML. We present data management techniques implemented in the latest version of SDM-RDFizer, denoted as SDM-RDFizer v4.5.6. These techniques play a pivotal role in satisfying the requirements of data collection and processing within complex data integration systems. Specifically, they enable the efficient scaling up of KG creation pipelines in real-world scenarios characterized by large and heterogeneous data sources. The significance of these data management techniques is underscored by their effectiveness in handling complex KG creation scenarios encountered in practical use cases. These scenarios are exemplified by the testbeds proposed in the Knowledge Graph Construction Workshop 2023 Challenge at ESWC 2023 [63] and are characterized by parameters reported by Chaves-Fraga et al. [15]. These complex use cases span different domains, such as biomedicine [2,52, 59] and energy [37], where the declarative definition of KGs facilitates the smooth integration of large amounts of heterogeneous data, potentially duplicated across different data sources. As a result, SDM-RDFizer v4.5.6 is empowered to scale up data collection, processing, and integration towards efficient KG creation.

This paper extends the work reported by Iglesias et al. [33]. Our new contributions are as follows:

- Data structures for RDF data compression and planning techniques for mapping rule execution.

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

[3] https://github.com/RMLio/rmlmapper-java.

- Physical operators that leverage these data structures to efficiently handle complex RML mappings, including multiple joins and large data sources with high duplicate rates.
- A new version of the SDM-RDFizer tool (named SDM-RDFizer v4.5.6) which incorporates these data structures and physical operators, enabling the execution of complex KG creation pipelines.
- An empirical evaluation using two state-of-the-art benchmarks, GTFS-Madrid-Bench [16] and SDM-Genomic-Datasets.[4] Our evaluation encompasses 416 testbeds and compares our new version with Morph-KGC, RMLMapper, RocketRML, and the previous version of SDM-RDFizer (v3.2). The results demonstrate the advantages of the data management techniques proposed in this paper and implemented in SDM-RDFizer v4.5.6. Specifically, the evaluation highlights the significance of data structures and physical operators in executing complex configurations like those found in the SDM-Genomic-Datasets benchmark.

The rest of the paper is structured as follows: Section 2 defines and illustrates a KG creation pipeline and the requirements to be satisfied for an RML engine. Section 3 summarizes previous approaches. Section 4 defines the data management methods implemented in SDM-RDFizer v4.5.6. Section 5 reports the results of our experimental studies. Section 6 describes the main characteristics of our tool and, finally, our conclusions and future work are outlined in Section 7.

2. Declarative specifications of pipelines for knowledge graph creation

This section provides the basis for understanding the problem of KG creation. First, we define some basic concepts related to KGs, data integration systems, KG creation pipelines, and RML (RDF Mapping Language). Next, we present the steps involved in declaratively specifying a KG using RML. To ensure the effectiveness of RML engines, we will elucidate the requirements that they need to satisfy, based on existing evaluation studies reported in the literature [15,34,39]. Finally, we will illustrate the problem of KG creation in a use case derived from the data integration challenges in the biomedical area. This example illustrates data integration issues reported by Chandak et al. [12] and observed in the data management tasks of the EU H2020 funded projects iASiS[5] and CLARIFY.[6]

2.1. Preliminaries

Data can be siloed and scattered across various data sources. As the volume of available data continues to expand, the prevalence of these data silos is expected to increase, consequently hindering interoperability. Data integration aims to gather information from heterogeneous sources and provide a unified view from which relationships and patterns hidden in isolated sources can be uncovered [11,28]. To illustrate, consider data related to genes collected from tissues from tumors and stored in three data sources. A relational database maintains gene-related information (e.g., created from Genecards[7]). Further, the results of tumor tissue analysis are maintained in a tabular format (e.g., CSV), and an XML file stores information about the tumors from which the tissues were sampled. When analyzing genes (e.g., BRCA1) related to tissues sampled from specific tumors (e.g., breast tumors), integrating these data sources becomes essential to establish the connections existing between these entities. However, each data source follows a different format and schema, creating interoperability conflicts. Data integration has the objective of providing data management methods to create these holistic views (e.g., integrated views of genes, tissues, and tumors). While these interoperability issues are readily apparent in the biomedical field [39], similar issues persist across industrial and scientific domains where data is autonomously generated. The challenge is to provide data management mechanisms that enable seamless data integration while also ensuring scalability and maintainability to guarantee long-term usability and value of the integrated data.

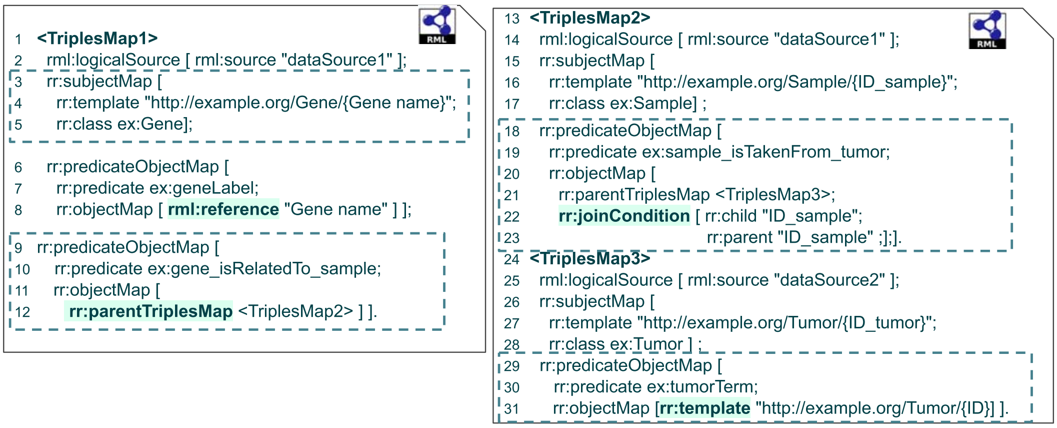
A data integration process can be specified as a data integration system $DIS_{\mathcal{G}} = \langle O, S, M \rangle$ [42]. Here, O represents a unified schema consisting of classes and properties, S denotes a set of data sources, and M corresponds

[4] https://figshare.com/articles/dataset/SDM-Genomic-Datasets/14838342

[5] https://cordis.europa.eu/project/id/727658.

[6] https://www.clarify2020.eu/.

[7] https://www.genecards.org/



```
<TriplesMap1>
  rml:logicalSource [ rml:source "dataSource1" ];
  rr:subjectMap [
    rr:template "http://example.org/Gene/{Gene name}";
    rr:class ex:Gene];

  rr:predicateObjectMap [
    rr:predicate ex:geneLabel;
    rr:objectMap [ rml:reference "Gene name" ] ];

  rr:predicateObjectMap [
    rr:predicate ex:gene_isRelatedTo_sample;
    rr:objectMap [
      rr:parentTriplesMap <TriplesMap2> ] ].
```

```
<TriplesMap2>
  rml:logicalSource [ rml:source "dataSource1" ];
  rr:subjectMap [
    rr:template "http://example.org/Sample/{ID_sample}";
    rr:class ex:Sample] ;
  rr:predicateObjectMap [
    rr:predicate ex:sample_isTakenFrom_tumor;
    rr:objectMap [
      rr:parentTriplesMap <TriplesMap3>;
      rr:joinCondition [ rr:child "ID_sample";
                         rr:parent "ID_sample" ;];].
<TriplesMap3>
  rml:logicalSource [ rml:source "dataSource2" ];
  rr:subjectMap [
    rr:template "http://example.org/Tumor/{ID_tumor}";
    rr:class ex:Tumor ] ;
  rr:predicateObjectMap [
    rr:predicate ex:tumorTerm;
    rr:objectMap [rr:template "http://example.org/Tumor/{ID}] ].
```

Fig. 1. **Example.** Main concepts of the RDF mapping language (RML). Triples maps <TriplesMap1>, <TriplesMap2>, <TriplesMap3> correspond to RML mapping rules that define entities in the classes `ex:Gene`, `ex:Sample`, and `ex:Tumor`, respectively.

to mapping rules establishing the correspondences between concepts in O and the attributes of the sources in S. Traditionally, the design of a data integration system is built over certain assumptions [28]: i) The unified schema models all concepts (i.e., classes, attributes, and relationships) of the entities present in the data sources. ii) Data sources are well-defined in terms of their signatures or schemas. iii) Interoperability issues arising from different representations of the same real-world entities have been previously addressed using entity alignment methods. iv) Collected data is assumed to be mostly correct and consistent. In case of inaccuracies, data curation methods are applied during a pre-processing stage.

KGs are directed edge-labeled graphs that represent statements as entities and their relationships as labeled edges (Gutierrez et al. [29]). Given a KG $\mathcal{G}$, the KG creation process can be defined as a data integration system $DIS_{\mathcal{G}} = \langle O, S, M \rangle$, such as the execution of mapping rules in M over the data sources in S generates nodes and labeled edges in $\mathcal{G}$. These rules can be represented as Horn clauses (Namici et al. [44]), such as $body(\overline{X}) : -head(\overline{Y})$, following the Global As View (GAV) approach [42], i.e., *body* corresponds to the conjunction of data source predicates in S, and the *head* is a predicate in O which corresponds to a class, attribute, or property. W3C-standard mapping languages like R2RML[8] and its extension, the RDF Mapping Language-RML [25,36], can be utilized to specify the mapping rules in M for KGs expressed using the Resource Description Framework (RDF).

R2RML mappings define how tables, columns, and rows in a relational database are declaratively translated into RDF triples. RML is a mapping language that represents mapping rules in RDF; it offers the features of R2RML by allowing the specification of mapping rules over heterogeneous data sources, e.g., JSON, CSV, and XML. RML provides a well-established and standardized approach to mapping heterogeneous data to RDF. The W3C Community Group on Knowledge Graph Construction has proposed the RML ontology [36] as a common agreement on how the mapping rules must be defined. This standardization ensures consistency and compatibility with other RDF-related tools and technologies. Moreover, RML mappings can be reused for different data sources, reducing redundancy in mapping definitions. This reusability streamlines the creation and maintenance of RDF KGs for various applications. Declarative RML mappings are represented in RDF, serving as metadata that can be queried to facilitate the maintainability and reusability of DISs. Consequently, these declarative specifications of KGs as RML DISs promote modularity, supporting maintenance, testing, debugging, and enabling collaborative definition and reusability. RML is currently being used by several companies and public organizations [19,20,50,60] to create their KGs, also including the Google Enterprise Knowledge Graph.[9]

Fig. 1 depicts RML mapping rules (a.k.a. triples maps). Triples map represents mapping rules where the resources (`rr:subjectMap`) of an RDF class and their properties (`rr:predicateObjectMap`) are assigned

[8] https://www.w3.org/TR/r2rml/

[9] https://cloud.google.com/enterprise-knowledge-graph/docs/entity-reconciliation-console

to values (`rr:objectMap`) based on a logical data source (`rml:logicalSource`). A `rr:objectMap` can be also defined as a reference or a join with the `rr:subjectMap` in another triples map called `rml:parentTriplesMap`; if the two triples maps are defined over different logical sources the term `rr:joinCondition` specifies the attributes in the logical sources whose values need to match. A *triples map* defines a group of mapping rules that specifies RDF triples with the same subject. For example, the triples map tagged <TriplesMap1> (lines 1 to 12) defines resources in the class `ex:Gene` and the RDF triples for these resources with the predicates, `ex:geneLabel` and `ex:gene_isRelatedTo_sample`. The data to populate the RDF triples are collected from the data sources defined with the term `rml:logicalSource` element (lines 2, 14, and 25).

2.2. *Knowledge graph creation pipelines*

The declarative definition of a KG $\mathcal{G}$ as a data integration system $DIS_{\mathcal{G}} = \langle O, S, M \rangle$ where mapping rules are specified in RML, requires the following steps [23]:

1. Description of the schema of the data sources S_i in S and the location source. If S_i is a relational database, S_i can be defined as a SQL query or a table. On the other hand, if S_i is a semi-structured source (e.g., in XML or JSON), S_i is defined as an iterator representing the iteration pattern to be followed to extract the data from S_i. In XML, the iteration is defined using XPath, while JSONPath is utilized for JSON.
2. Selection of the set C of classes from O whose entities are defined in terms of sources in S and whose properties are specified according to the attributes of these sources.
3. For each class C_j in C and source S_k in S, such as S_k defines the entities in C_j, create a triples map $t_{j,k}$ with logical source S_k and `rr:subjectMap` with the function `rr:template` over the attributes $A_{k,1}, \ldots, A_{k,p}$ from S_k that compose the Internationalized Resource Identifier (IRI) of the entities of C_j. Fig. 1 illustrates three triples maps defining the classes `ex:Gene`, `ex:Sample`, and `ex:Tumor`.
4. Data type properties dta of C_j defined as attribute $D_{k,1}$ in S_k, $t_{j,k}$ include a `rr:predicateObjectMap`, whose `rr:predicate` is dta, and `rr:objectMap` is $D_{k,1}$. Lines 6–8 in Fig. 1 show the term `rr:predicateObjectMap` specifying the terms `rr:predicateObjectMap`, `rr:predicate`, and `rr:objectMap` defining the instances of the RDF triples whose subject is defined by the `rr:subjectMap` in lines 3 and 5, the predicate is `ex:geneLabel` and the object value corresponds to the attribute `"Gene name"` from `"dataSource1"`; the term `rr:reference` indicates that the value is a literal.
5. If *op* is an object property of C_j defined with the attributes $B_{k,1}, \ldots, B_{k,q}$ in S_k, a `rr:predicateObjectMap` will be part of $t_{j,k}$ with `rr:predicate` is *op*, and `rr:objectMap` is the function `rr:template` over $B_{k,1}, \ldots, B_{k,q}$ from S_k that compose the Internationalized Resource Identifier (IRI) of the object values of *op*. Lines 29–31 in Fig. 1 illustrate the specification of `ex:tumorTerm` whose values are IRIs.
6. The object values of an object property *op* can be defined as the subject on a triples map defined over the same logical source. In this case, the term `rr:parentTriplesMap` is utilized to reference the triples map. In Fig. 1, the values of the predicate `ex:gene_isRelatedTo_sample` correspond to the resources of the class `ex:Sample` specified by `TriplesMap2`. Contrary, if both triples maps are defined over two different logical sources, the term `rr:joinCondition` specifies the attributes to be joined in the data sources of the triples maps to be merged. The term `rr:predicateObjectMap` in lines 18 and 23 defines the predicate `ex:sample_isTakenFrom_tumor` as the subject of `TriplesMap3` whose values for the attributes named `"ID_sample"` in `"dataSource1"` and `"dataSource2"` have the same values.

2.3. *Requirements of a knowledge graph creation pipeline*

Requirements for RML engines are defined based on: parameters analyzed by Chaves-Fraga et al. [15]; testbeds assessed by the KGCW 2023 Challenge [63]; and data integration requirements presented by Kinast et al. [39].

Chaves-Fraga et al. [15] have established that various parameters influence the efficiency of declaratively specified KG creation as a DIS. These parameters are grouped into five dimensions: mapping, data, platform, source, and output. The *data dimension* considers the characteristics of the data stored in a data source; it includes size, frequency distribution, data partitioning, and format. The impact of the hardware resources is represented in the *hardware dimension*, while the *source dimension* includes parameters such as data source transfer time and data initial delays. Finally, the *output dimension* groups parameters related to how the RDF triples are generated. They include duplicate removal, RDF triple generation at once, or in a streaming manner. Thus, these parameters encompass the complexity of the mapping rules (e.g., mapping shape, number of joins, and join selectivity).

The Extended Semantic Web Conference (ESWC) 2023 hosted a Knowledge Graph Creation Workshop (KGCW), which introduced a challenge dataset [63] aimed at evaluating the performance of existing KG creation RML engines. This dataset was designed to assess memory usage and execution time, considering various parameters that influence the KG creation process, as established by Chaves-Fraga et al. [15]. The aim was to create multiple test cases by generating RML triples maps covering a wide range of scenarios. They include: a) size of the data sources based on number of rows and columns; b) number of RML triples maps and their properties; c) complexity of joins among RML triples maps; and d) data diversity (e.g., duplicate rate and empty values). These scenarios each have unique effects on the KG creation process. Data source size and the complexity of joins primarily impact memory usage. Meanwhile, the number of properties, triples maps, and data diversity can influence execution time. How KG creation engines handle these variables determines the specific impact on execution time or memory usage due to data diversity. In summary, this challenge dataset offers a comprehensive examination of KG creation scenarios, making clear the requirements that need to be satisfied to enhance the performance of RML engines.

Kinast et al. [39] present the outcomes of a systematic literature analysis, showing the functional requirements for integrating medical data. These functional requirements include several categories, encompassing data acquisition, processing, analysis, metadata management, traceability, lineage, and security. While some requirements are specific to the medical domain, the following are domain-agnostic: i) the capability to collect data in various formats; ii) the use of standardized ontologies, vocabularies, rules, and processes; iii) the possibility of representing integrated data through multidimensional models (e.g., RDF KGs); and iv) the capacity of managing large volumes of data.

We have elucidated the following requirements; they are divided into two categories: data collection and processing.

Data collection: This category encompasses requirements for accessing data from various data sources:

- **RE1-Heterogeneous data**: Collect data from diverse sources in various formats (e.g., CSV, JSON, or relational databases) [15,39,63].
- **RE2-Large data**: An engine should be able to handle data of different sizes [15,39,63].
- **RE3-Fragmented data**: Gather all attributes of entities within a class [15,63].

Data processing: This category outlines requirements for the processes involved in integrating the collected data:

- **RE4-Duplicated data**: Efficiently process and integrate duplicated data [15,63].
- **RE5-Data diversity**: Ensure that variations in data frequency distributions across multiple sources do not impede the efficiency of data integration and processing [15,63].
- **RE6-Semi-structured data**: Enable the engine to convert semi-structured data (e.g., XML or JSON) into a standardized format (e.g., RDF) [39].
- **RE7-Standardized data integration**: Enable the engine to process data from diverse sources with correspondences expressed using standard mapping languages (e.g., R2RML or RML) [39].
- **RE8-Mapping complexity**: Ensure the engine can process any level of complexity, ranging from the number of properties and mappings to the diversity of joins between mappings [15,63].

2.4. A use case for knowledge graph creation

This section illustrates the problem of defining a KG using a declarative approach where mapping rules are specified in RML. In this example, we focus on a portion of a DIS that defines a biomedical KG [2,52,64,65], e.g., the one created in the context of the EU H2020 projects iASiS[5] and CLARIFY[6]. The data sources, referred to as SDM-Genomic-Datasets [33] vary in size, with 100k, 1M, and 5M rows, and each dataset has a different percentage of data duplicate rate (25% or 75%). The unified ontology[10] consists of classes like `iasis:Mutation`,

[10] https://github.com/SDM-TIB/iASiSOntology

`iasis:Gene`, `iasis:Sample`, `iasis:GenomePosition`, `iasis:Tumor`, and `iasis:SomanticStatus`; they are used to provide a harmonized definition of the concepts scattered across the collected data sources. The ontology and its properties are defined in six RML triples maps, as presented in Fig. 2a. This pipeline consists of one parent triples map (`TriplesMap1`) and five child triples maps (`TriplesMap2`, `TriplesMap3`, `TriplesMap4`, `TriplesMap5`, and `TriplesMap6`) that refer to the same triples map. Due to the shape of the triples maps, data source sizes, and percentage of duplicates, this use case is considered complex and challenging to execute.

Three state-of-the-art RML engines, i.e., RMLMapper v6.0,[11] Morph-KGC v2.1.1,[12] and SDM-RDFizer v3.2[28], are utilized to create this portion of the KG; following configurations reported in the literature [13,17,34,35], the engines timed out in five hours. The results of executing these engines in six testbeds are shown in Fig. 2. The execution time was significantly impacted by factors such as the size of the data sources, the percentage of duplicates, and the number and type of joins among triples maps. In fact, two out of the three engines timed out when processing a data source with 5M records. Although SDM-RDFizer exhibited relatively better execution time compared to RMLMapper and Morph-KGC, it still required considerable time to create the KG. In this paper, we present a new version of SDM-RDFizer (v4.5.6[13]) that incorporates data management techniques for planning the execution of triples maps and efficiently compressing intermediate results. These improvements have enabled SDM-RDFizer to scale up to complex scenarios, as reported in Section 5.

3. Related work

This section summarizes the key contributions from the existing literature regarding creating RDF KGs. First, an overview of the data models, formats, and frameworks to transform Web data is presented. Then, existing technologies for KG creation are described, while the following two sections present the main approaches for performing the KG integration process in a virtual or materialized manner [6].

3.1. Representing data on the web and transforming web data into RDF

The adoption of the Web as a framework for publishing electronic data [1] has driven the development of semi-structured data models, formats, and languages aimed at facilitating their curation, retrieval, and version control [1]. In this context, XML emerged as a standard proposed by the World Wide Web Consortium (W3C) for representing semi-structured data from diverse sources. XML employs a tag-based syntax that is easily readable by both humans and machines. Additionally, RDF graphs can be represented using XML syntax, with tools like XSPARQL [9] and Gloze [8] supporting the transformation between XML and RDF specifications. CSV, an abbreviation for "Comma Separated Value," is another commonly used format for representing and exchanging data on the Web. The CSV2RDF[14] framework offers standardized procedures for converting CSV data into RDF, while Tarql[15] relies on wrappers for CSV data sources. These wrappers enable the execution of CONSTRUCT SPARQL 1.1 queries, facilitating the creation of RDF graphs from CSV data. Relational or tabular data also serve as a prevalent data model for presenting information on the Web. Tools developed by Polfliet and Ryutaro [47], Sequeda and Miranker [56], as well as Auer et al. [7], exemplify solutions for transforming relational data into RDF format. In a broader context, Thakker [61] offers a comprehensive analysis of various data transformation techniques applied to data represented in different models (such as relational or semi-structured databases) and formats (e.g., XML[16] or JSON[17]). These

[11] https://github.com/RMLio/rmlmapper-java/releases/tag/v6.0.0

[12] https://doi.org/10.5281/zenodo.6524684

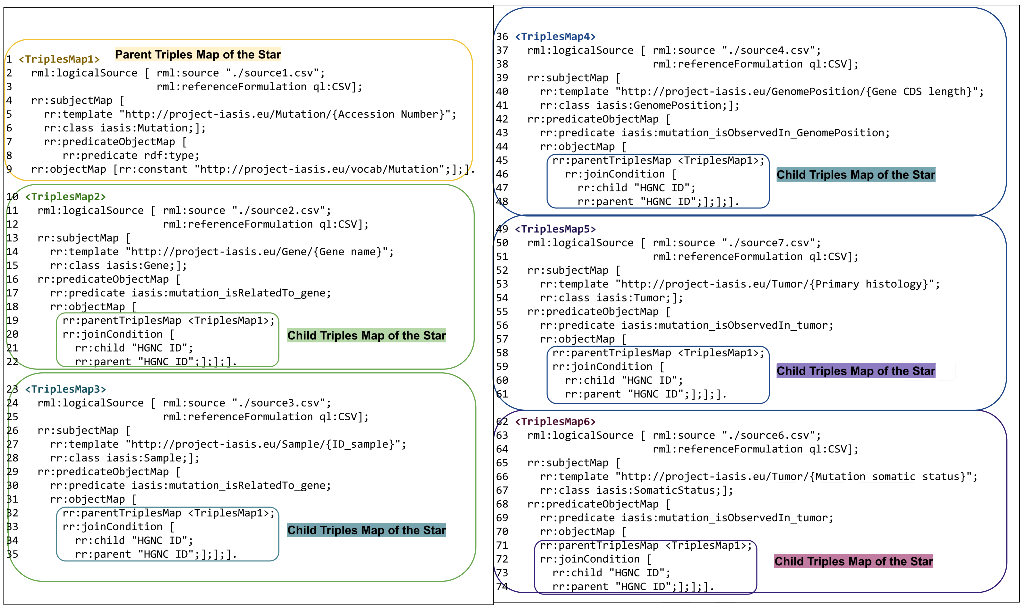
[13] https://doi.org/10.5281/zenodo.7027549

[14] https://www.w3.org/TR/csv2rdf/

[15] https://ld4pe.dublincore.org/learning_resource/tarql-sparql-for-tables/

[16] https://www.w3.org/TR/xml/

[17] https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf


Parent Triples Map of the Star
Child Triples Map of the Star
Child Triples Map of the Star
Child Triples Map of the Star
Child Triples Map of the Star
Child Triples Map of the Star

```
<TriplesMap1>
  rml:logicalSource [ rml:source "./source1.csv";
                      rml:referenceFormulation ql:CSV];
  rr:subjectMap [
    rr:template "http://project-iasis.eu/Mutation/{Accession Number}";
    rr:class iasis:Mutation;];
    rr:predicateObjectMap [
       rr:predicate rdf:type;
  rr:objectMap [rr:constant "http://project-iasis.eu/vocab/Mutation";];].

<TriplesMap2>
  rml:logicalSource [ rml:source "./source2.csv";
                      rml:referenceFormulation ql:CSV];
  rr:subjectMap [
    rr:template "http://project-iasis.eu/Gene/{Gene name}";
    rr:class iasis:Gene;];
  rr:predicateObjectMap [
    rr:predicate iasis:mutation_isRelatedTo_gene;
    rr:objectMap [
      rr:parentTriplesMap <TriplesMap1>;
      rr:joinCondition [
        rr:child "HGNC ID";
        rr:parent "HGNC ID";];];].

<TriplesMap3>
  rml:logicalSource [ rml:source "./source3.csv";
                      rml:referenceFormulation ql:CSV];
  rr:subjectMap [
    rr:template "http://project-iasis.eu/Sample/{ID_sample}";
    rr:class iasis:Sample;];
  rr:predicateObjectMap [
    rr:predicate iasis:mutation_isRelatedTo_gene;
    rr:objectMap [
      rr:parentTriplesMap <TriplesMap1>;
      rr:joinCondition [
        rr:child "HGNC ID";
        rr:parent "HGNC ID";];];].

<TriplesMap4>
  rml:logicalSource [ rml:source "./source4.csv";
                      rml:referenceFormulation ql:CSV];
  rr:subjectMap [
    rr:template "http://project-iasis.eu/GenomePosition/{Gene CDS length}";
    rr:class iasis:GenomePosition;];
  rr:predicateObjectMap [
    rr:predicate iasis:mutation_isObservedIn_GenomePosition;
    rr:objectMap [
      rr:parentTriplesMap <TriplesMap1>;
        rr:joinCondition [
          rr:child "HGNC ID";
          rr:parent "HGNC ID";];];].

<TriplesMap5>
  rml:logicalSource [ rml:source "./source7.csv";
                      rml:referenceFormulation ql:CSV];
  rr:subjectMap [
    rr:template "http://project-iasis.eu/Tumor/{Primary histology}";
    rr:class iasis:Tumor;];
  rr:predicateObjectMap [
    rr:predicate iasis:mutation_isObservedIn_tumor;
    rr:objectMap [
      rr:parentTriplesMap <TriplesMap1>;
      rr:joinCondition [
        rr:child "HGNC ID";
        rr:parent "HGNC ID";];];].

<TriplesMap6>
  rml:logicalSource [ rml:source "./source6.csv";
                      rml:referenceFormulation ql:CSV];
  rr:subjectMap [
    rr:template "http://project-iasis.eu/Tumor/{Mutation somatic status}";
    rr:class iasis:SomaticStatus;];
  rr:predicateObjectMap [
    rr:predicate iasis:mutation_isObservedIn_tumor;
    rr:objectMap [
    rr:parentTriplesMap <TriplesMap1>;
    rr:joinCondition [
      rr:child "HGNC ID";
      rr:parent "HGNC ID";];];].
```

(a) A Star Triple Map

RML Engine	Time (secs)	
	Percentage of Duplicates: 25%	Percentage of Duplicates: 75%
Data Size: 100K		
RMLMapper	11,961.81 sec	12,669.84 sec
Morph-KGC	23.24 sec	23.71 sec
SDM-RDFizer v3.2	79.54 sec	45.3 sec
Data Size: 1M		
RMLMapper	Timeout (five hours)	Timeout (five hours)
Morph-KGC	2,411.1 sec	2,013.16 sec
SDM-RDFizer v3.2	1,323.61 sec	756.24 sec
Data Size: 5M		
RMLMapper	Timeout (five hours)	Timeout (five hours)
Morph-KGC	Timeout (five hours)	Timeout (five hours)
SDM-RDFizer v3.2	8,092.01 sec	7,307.18 sec

(b) Performance of State-of-the-art RML Engines. Timeout of five hours.

Fig. 2. **Motivating example.** A star triples map and its impact on the performance of a KG creation pipeline.

transformations are executed using query languages like SPARQL[18] or Gremlin.[19] This analysis emphasizes the significance of this topic within both the database and semantic web communities. SDM-RDFizer also facilitates the transformation of data from diverse sources represented in various formats, including JSON, XML, CSV, or relational databases. However, SDM-RDFizer relies on declarative definitions using R2RML or RML to establish correspondences between data sources and RDF. Since different factors can influence the execution of these declarative mapping rules [15], SDM-RDFizer employs various data structures and physical operators to mitigate the impact of data size, mapping rule complexity, and duplicate records.

[18] https://www.w3.org/TR/sparql11-query/
[19] https://tinkerpop.apache.org/gremlin.html

3.2. Knowledge graph creation and existing technologies

The creation of a KG $\mathcal{G}$ can be defined as a data integration system [42], represented as $DIS_{\mathcal{G}} = \langle O, S, M \rangle$. In this system, O denotes a unified ontology consisting of classes and properties, S represents a set of data sources, and M corresponds to mapping rules or assertions that align with the concepts defined in O, expressed as conjunctive queries over the sources in S. By executing M over the data sources in S, the instances in $\mathcal{G}$ are generated. The input data sources are typically represented in a global view, commonly established as an ontology [46]. The connection between these input sources and the ontology is often established through mapping rules, such as the W3C recommendation R2RML [21] for relational databases or its main extension for heterogeneous data formats (e.g., CSV, XML, and JSON) called RML [24,36]. However, there are other solutions that adapt different languages (e.g., SPARQL) for integration purposes, such as SPARQL-Anything [5] and SPARQL-Generate [40]. The works by Hofer et al. [31] and Van Assche et al. [6] contribute to the understanding of the most suitable mapping languages, ontologies, data sources, and KG creation engines based on specific use cases. Hofer et al. report a comprehensive study to determine the state-of-the-art in KG construction, defining the requirements for popular KGs like DBpedia and WorldKG, evaluating existing tools and strategies, and identifying areas that require further attention. On the other hand, Van Assche et al. [6] survey mapping languages and engines to determine their appropriateness in different user situations. These works play a vital role in informing the community about the existing approaches, enabling researchers to identify areas that need further exploration and improve upon state-of-the-art.

3.3. Virtual data integration in knowledge graphs

The creation of a virtual KG involves generating it dynamically based on a request expressed as a query over a target ontology, with mapping rules used to transform the input query into an equivalent query for the source(s) in S [67]. Virtual KG creation approaches offer the advantage of integrating frequently updated data, such as streaming or evolving data. However, they face challenges in query writing, optimization, and execution to ensure the reliability and completeness of KG creation pipelines [68,69]. Several solutions have been proposed, with a focus on translating SPARQL queries into SQL queries. These solutions include Ontop [10], Ultrawrap [55], Morph [48], Squerall [43], Ontario [27], and Morph-CSV [17]. Ontop, Ultrawrap, and Morph create virtual RDF KGs while evaluating SPARQL queries against relational databases. Optimization techniques are employed to speed up query execution while maintaining the completeness of query answers. However, these approaches are limited to data sources accessible via relational databases. Morph-CSV follows a similar approach and introduces query rewriting techniques to efficiently apply domain-specific constraints on raw tabular data, such as CSV files, to enhance SPARQL2SQL engines. Ontario and Squerall tackle the problem of virtual KG creation by treating it as query execution over a federation of heterogeneous data sources, including JSON, CSV, XML, RDF, and relational databases. Although SDM-RDFizer is not specifically designed for generating virtual KGs, recent work by Rohde et al. [49] demonstrates that it can be easily extended to efficiently create a KG by rewriting an input query based on RML triples maps.

3.4. Materialized knowledge graph creation

A KG is constructed by integrating various sources in S using mapping rules defined in M. The community has actively contributed to data management techniques and ETL tools that facilitate the integration of large and diverse data sources into KGs through declarative mapping rules. One such tool is RMLMapper[3], it is an in-memory RML compliant engine designed to address source heterogeneity during KG creation. RMLMapper is capable of executing RML mapping rules (a.k.a. triples maps) defined over logical sources in different formats (e.g., CSV, JSON, XML, Excel files, LibreOffice) accessible through remote access (e.g., SPARQL endpoints or Web APIs) or management systems (e.g., Oracle, MySQL, PostgreSQL, or SQLServer). However, RMLMapper may not scale up in complex scenarios [4]. Morph-KGC [3] is a tool that processes R2RML, RML, and RML-star [22] and improves the scalability of KG creation by introducing the concept of mapping-partitions. This technique addresses several parameters that affect KG construction, such as duplicate elimination and parallel execution. However, it primarily focuses on mapping planning, and the authors acknowledge the need for new techniques or data structures to handle complex [R2]RML operators, including join conditions. The initial version of SDM-RDFizer [33] also emphasizes scalability and relies on a set of physical data structures and corresponding operators for efficient duplicate removal and join condition processing. While this speeds up KG creation, these data structures may impact scalability in terms of memory consumption. Stadler et al. [58] present data management methods that resort to the translation of RML triples maps into SPARQL queries over the Apache Spark Big Data framework to scale up the process of KG creation. RMLStreamer [45] also prioritizes scalability, and efficiently generates RDF triples continuously while eliminating duplicates at the end. Other RML-compliant engines, such as RocketRML [57], focus on heterogeneity rather than scalability. Meanwhile, tools like Chimera [53] and CARML[20] implement data management techniques for handling large JSON and XML files, reducing memory consumption, and incrementally generating KGs. Despite significant efforts in developing these solutions, there is still room for research and the implementation of more scalable approaches that ensure their adoption in industry and academia. In the Knowledge Graph Construction Workshop 2023 Challenge at ESWC 2023 [63], participants such as CARML, SANSA (Stadler et al. [58]), RMLStreamer, and SDM-RDFizer showcased their capabilities. The results demonstrated that no single engine can outperform others in all aspects. CARML showed better results in terms of memory usage, while SDM-RDFizer efficiently handled task-oriented test cases involving join execution, duplicate removal, and empty values. As expected, SANSA exhibited the lowest execution time, while RMLStreamer scaled well with larger data sources. The version of SDM-RDFizer presented in this paper aims to address these issues by empowering data management methods and structures, not only speeding up KG creation and reducing memory consumption but also enabling efficient execution of complex use cases involving duplicate removal, empty values, and expensive joins.

4. SDM-RDFizer: A tool for the materialized creation of knowledge graphs

This section presents the SDM-RDFizer tool, focusing on its architecture and data management techniques that enable the engine to meet the requirements outlined in Section 2.3. The development of SDM-RDFizer follows the Agile software development methodology, which ensures a flexible and iterative approach guided by the requirements for scaling up KG creation. These requirements are gathered from various sources, including the community (e.g., the KGC Challenge at ESWC 2023), ongoing projects, and fundamental findings from the data management field. By embracing the Agile methodology, the development team can effectively adapt to evolving needs and prioritize scalability. The iterative nature of Agile allows for continuous improvement and integration of valuable feedback throughout the development process. Moreover, by actively involving stakeholders and considering their input, SDM-RDFizer aims to address the specific challenges faced by users in KG creation. Through the Agile approach, SDM-RDFizer strives to deliver a more responsive, collaborative, and efficient tool that effectively tackles the demands of scaling up KG creation.

4.1. The SDM-RDFizer architecture

SDM-RDFizer implements multiple data structures that optimize different aspects of the KG creation process such as duplicate removal, join execution, and data compression, and operators that execute various types of triples maps efficiently. Additionally, SDM-RDFizer is able to plan the execution of the RML triples maps to reduce execution time and secondary memory consumption.

Fig. 3 depicts the SDM-RDFizer architecture in terms of its components. The SDM-RDFizer comprises two main modules: **Triples Map Planning** (TMP) and **Triples Map Execution** (TME). TMP determines the order of evaluation of the triples maps so that the amount of memory used is kept at a minimum. Next, TME evaluates the triples maps in the generated order. Each triples map is defined in terms of a physical RML operator (**Simple Object Map** (SOM), **Object Reference Map** (ORM), and **Object Join Map** (OJM)) so that RDF triples can be generated as the execution of these operators. Each generated triple is compared to the corresponding **Predicate Tuple Table** (PTT) to determine if the triple is a duplicate. If the triple is a duplicate, it is discarded. If the triple is not a duplicate, it is added to the corresponding PTT and the KG. The generated RDF resources and literal are represented in the

[20] https://github.com/carml/carml

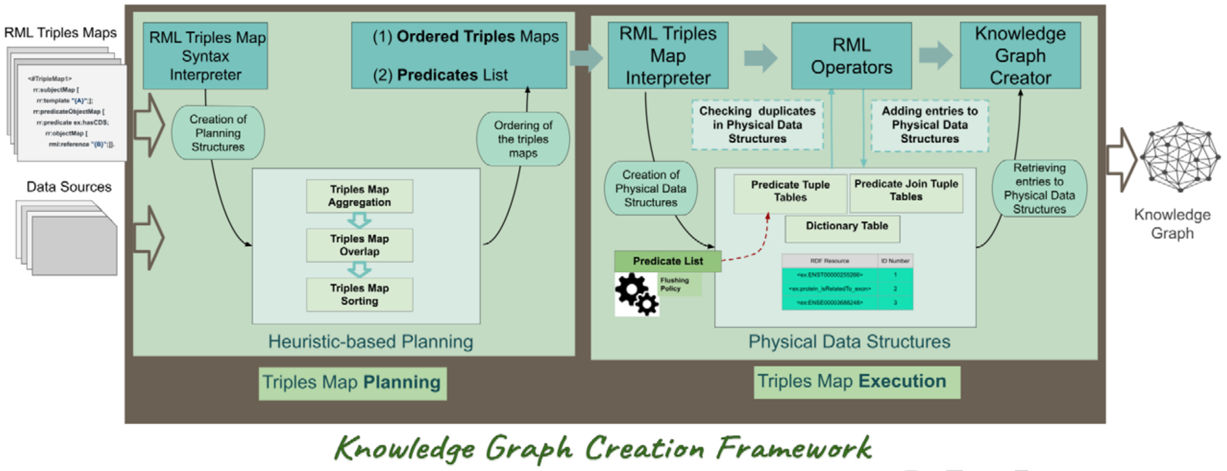


Fig. 3. **The SDM-RDFizer architecture.** SDM-RDFizer receives as an input RML triples maps with its corresponding data sources. The triples map planning component plans and orders the execution of the triples maps. The triples map execution component resorts to physical operators, data structures, and compression techniques to efficiently execute the triples maps in the order stated during planning.

Dictionary Table (DT). In case there is a join condition, the **Predicate Join Tuple Table** (PJTT) is used to store the results of the join condition.

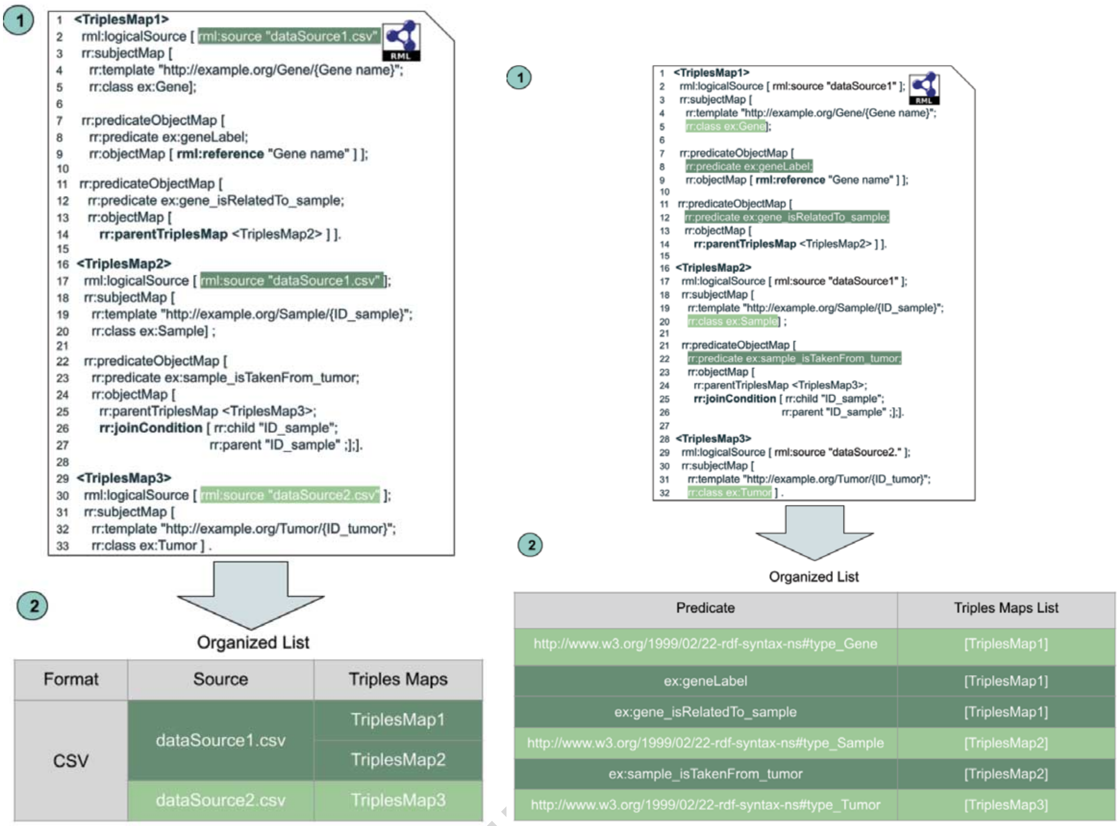
4.2. *The SDM-RDFizer planning phase*

The Triples Map Planning (TMP) module reorders RML triples maps so that the most selective ones are evaluated first, while non-selective rules are executed at the end. TMP organizes the triples maps and data sources so that the number of RDF triples kept in the main memory is reduced. As a result, the KG creation process consumes the minimum amount of memory and execution time. TMP defines two data structures, the **Organized Triples Maps List** (OTML) and the **Predicate List** (PL); they encapsulate the order in which the triples maps should be executed. OTML groups the triples maps by data source, while PL groups the triples maps by predicate and orders them. Triples maps are sorted to determine which of them define the same predicates.

Organized Triples Maps List (OTML) groups triples maps by their data source; see Fig. 4a. OTML is only used with triples maps with file data sources (e.g., CSV, JSON, and XML), i.e., it is not utilized for relational databases. This is because, when processing relational databases, the RML engine should collect the data indicated in the triples map using SQL queries.

During the TMP phase, triples maps are classified based on the logical data source format (i.e., CSV, JSON, and XML). Afterward, they are grouped by their data source; thus, a data source is opened once to execute all the triples maps. Implementing this data structure into the SDM-RDFizer causes the processor to adopt a hybrid approach. A data-driven approach is used for triples maps with file data sources, while a mapping-driven approach is used for triples maps with relational databases. Figure 4a depicts the OTML for the triples maps in Fig. 1. Since all these triples maps are over CSV files, only one group is created.

Predicate List (PL) groups triples maps by their predicates by creating a list of triples maps associated with a particular predicate; see Fig. 4b. PL has two purposes, the first is to determine when a PTT associated with a certain predicate is ready to be flushed from main memory, and the second is to organize OTML. Each time a triples map is processed, it is removed from the list of the corresponding predicates. If the list becomes empty, no triples maps require the corresponding predicate and the associated PTT is safe to be flushed. For a generic predicate, e.g., `rdf:type`, we add its range alongside the predicate to its entry in PL. In Fig. 4b (line 20), there is the predicate `rdf:type`, and its range is `ex:Sample`. The corresponding entry in the PL is `rdf:type_Sample`. PL also organizes OTML by determining which triples maps have overlapping predicates. By default, the execution order considers the triples maps with the least overlap first and the one with the most overlap last. Following each triples



```
<TriplesMap1>
  rml:logicalSource [ rml:source "dataSource1.csv" ];
  rr:subjectMap [
    rr:template "http://example.org/Gene/{Gene name}";
    rr:class ex:Gene];

  rr:predicateObjectMap [
    rr:predicate ex:geneLabel;
    rr:objectMap [ rml:reference "Gene name" ] ];

 rr:predicateObjectMap [
   rr:predicate ex:gene_isRelatedTo_sample;
   rr:objectMap [
     rr:parentTriplesMap <TriplesMap2> ] ].

<TriplesMap2>
  rml:logicalSource [ rml:source "dataSource1.csv" ];
  rr:subjectMap [
    rr:template "http://example.org/Sample/{ID_sample}";
    rr:class ex:Sample] ;

  rr:predicateObjectMap [
    rr:predicate ex:sample_isTakenFrom_tumor;
    rr:objectMap [
      rr:parentTriplesMap <TriplesMap3>;
      rr:joinCondition [ rr:child "ID_sample";
                         rr:parent "ID_sample" ;];].

<TriplesMap3>
  rml:logicalSource [ rml:source "dataSource2.csv" ];
  rr:subjectMap [
    rr:template "http://example.org/Tumor/{ID_tumor}";
    rr:class ex:Tumor ] .
```

Format	Source	Triples Maps
CSV	dataSource1.csv	TriplesMap1
		TriplesMap2
	dataSource2.csv	TriplesMap3

(a) Organized Triples Map List.

```
<TriplesMap1>
  rml:logicalSource [ rml:source "dataSource1" ];
  rr:subjectMap [
    rr:template "http://example.org/Gene/{Gene name}";
    rr:class ex:Gene];

  rr:predicateObjectMap [
    rr:predicate ex:geneLabel;
    rr:objectMap [ rml:reference "Gene name" ] ];

 rr:predicateObjectMap [
   rr:predicate ex:gene_isRelatedTo_sample;
   rr:objectMap [
     rr:parentTriplesMap <TriplesMap2> ] ].

<TriplesMap2>
  rml:logicalSource [ rml:source "dataSource1" ];
  rr:subjectMap [
    rr:template "http://example.org/Sample/{ID_sample}";
    rr:class ex:Sample] ;

  rr:predicateObjectMap [
    rr:predicate ex:sample_isTakenFrom_tumor;
    rr:objectMap [
      rr:parentTriplesMap <TriplesMap3>;
      rr:joinCondition [ rr:child "ID_sample";
                         rr:parent "ID_sample" ;];].

<TriplesMap3>
  rml:logicalSource [ rml:source "dataSource2." ];
  rr:subjectMap [
    rr:template "http://example.org/Tumor/{ID_tumor}";
    rr:class ex:Tumor ] .
```

Predicate	Triples Maps List
http://www.w3.org/1999/02/22-rdf-syntax-ns#type_Gene	[TriplesMap1]
ex:geneLabel	[TriplesMap1]
ex:gene_isRelatedTo_sample	[TriplesMap1]
http://www.w3.org/1999/02/22-rdf-syntax-ns#type_Sample	[TriplesMap2]
ex:sample_isTakenFrom_tumor	[TriplesMap2]
http://www.w3.org/1999/02/22-rdf-syntax-ns#type_Tumor	[TriplesMap3]

(b) Predicate List.

Fig. 4. **The data structures for the planning phase.** The organized triples map list groups triples maps (TMs) first by data source and then by data source format, thus reducing the need to open any particular data source more than once. Predicate list groups the TMs by predicate. Each time a TM is finished processing, it is removed from the predicate list, and if a predicate becomes empty, the corresponding PTT is flushed. PL also defines the order of execution of the organized triples map list by determining which TMs have overlapping predicates.

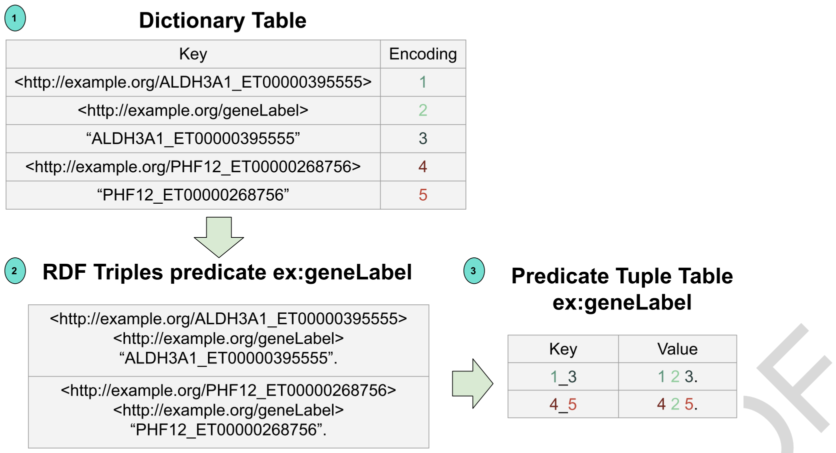
map is the triples map that overlaps with it so that when a triples map is finished executing, the most memory is released, and if any PTT remains, they will be flushed when the following triples map culminates.

4.3. The SDM-RDFizer triples map execution phase

This section explains the main data management methods implemented in SDM-RDFizer. The Triples Map Execution (TME) module generates the KG; it follows the order established by the TMP module when executing the RML triples maps. TME introduces multiple data structures, **Predicate Tuple Table** (PTT), **Predicate Join Tuple Table** (PJTT), and **Dictionary Table** (DT) that optimize different aspects of the KG creation process, like duplicate removal, join execution, and data compression respectively. TME resorts to three novel operators, **Simple Object Map** (SOM), **Object Reference Map** (ORM), and **Object Join Map** (OJM). Each operator covers a different type of triples map. The SOM operator executes a simple projection of the data to generate triples. ORM executes a parent reference between two triples maps with the same data source, and OJM is similar to ORM. Still, there must be a join condition, and the triples maps do not require the same data source. The following sub-sections define in more detail these operators and data structures.

4.3.1. The SDM-RDFizer data structures

SDM-RDFizer implements three data structures to efficiently manage and store the intermediate RDF triples generated during the execution of RML triples maps. These data structures avoid the generation of duplicated


1
Dictionary Table
2
RDF Triples predicate ex:geneLabel
3
Predicate Tuple Table ex:geneLabel

Key	Encoding
<http://example.org/ALDH3A1_ET00000395555>	1
<http://example.org/geneLabel>	2
"ALDH3A1_ET00000395555"	3
<http://example.org/PHF12_ET00000268756>	4
"PHF12_ET00000268756"	5

<http://example.org/ALDH3A1_ET00000395555> <http://example.org/geneLabel> "ALDH3A1_ET00000395555".

<http://example.org/PHF12_ET00000268756> <http://example.org/geneLabel> "PHF12_ET00000268756".

Key	Value
1_3	1 2 3.
4_5	4 2 5.

Fig. 5. **Dictionary table and predicate triple table.** Each generated RDF resource or literal is encoded in terms of an identification number in base 36. The dictionary table stores this encoding. RDF triples generated for a predicate, e.g., `ex:geneLabel`, are stored into its corresponding predicate tuple table using the identification number of each RDF resource and literal.

triples. Intermediate results are stored in these data structures independently of the format of the data sources. Thus, SDM-RDFizer exhibits a performance agnostic of the format of the data sources.

Dictionary Table (DT) encodes each RDF resource generated during the execution of a KG creation pipeline with an identification number. It is implemented as a hash table where the key is the IRI or Literal that represents the RDF resource, and the value is the identification number in base 36. An encoding function, *encode(.)*, is implemented to transform each RDF resource into its corresponding identification number. Fig. 5 illustrates an example of the DT that includes five entries with the identification numbers 1–5.

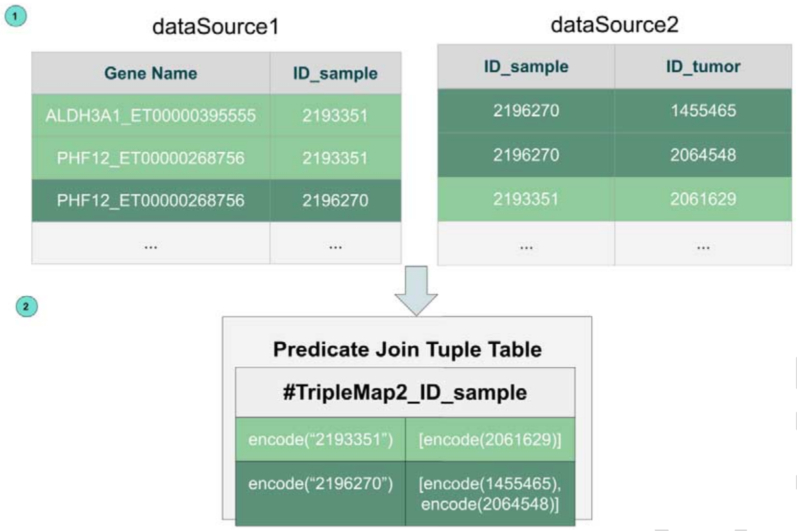
Predicate Tuple Table (PTT) is a table that stores all the RDF triples generated so far for a given predicate p. Fig. 5 presents a PTT for the predicate `ex:geneLabel`. PTTs correspond to hash tables where the hash key of an entry corresponds to the encoding of the subject and object of a generated RDF triple, and the value of the entry is the encoding of the RDF triple. As shown in Fig. 5, the identification number stored in DT for the corresponding RDF resources and literals, are used to encode the entries of a PTT. A PTT avoids the duplicate generation of an RDF triple, i.e., whenever an RDF triple is part of a PTT, this triple has been previously created, and the new triple should not be considered and added to the KG. However, when an RDF triple is not within PTT, then it is unique and must be added to PTT and the RDF KG. Whenever an RDF triple is generated during the execution of a triples map, the corresponding PTT is checked. PTTs bring significant savings not only in sources with high-duplicated rates but also when data sources create RDF triples of p also overlap.

Predicate Join Tuple Table (PJTT) is a table that stores the subjects of the triples generated by a join. It is implemented as an index hash table to the data source of the parent triples map in a join condition. A PJTT key corresponds to the encoding of each value(s) of the attributes in the join condition. The value of a key in a PJTT corresponds to the encoding of the subject values in the data source of the parent triples maps, which are associated with encoding the values of the attributes in the hash key. DT is used to retrieve the encodings of the RDF resources or literals. Thus, a PJTT minimizes the space to store intermediate results. Fig. 6 illustrates the PJTT for the join condition between triples maps `TriplesMap2` and `TriplesMap3` in Fig. 1 (lines 21–23). This PJTT corresponds to an index hash from the encoding values of the attribute `ID_sample` to the values of the attribute `ID_tumor` in `dataSource2`. The set of encoded values enables to directly access all the values of `ID_tumor` that join with a value of `ID_sample`. Thus, a PJTT enables direct access to the subjects associated with a join condition, and implements an index join corresponding to the most efficient implementation of a join [41].

4.3.2. *The SDM-RDFizer physical operators*

SDM-RDFizer resorts to three physical operators to efficiently execute triples maps in a KG creation pipeline.

Simple Object Map (SOM) generates an RDF triple by performing a simple predicate object map statement; Algorithm 1 sketches the SOM implementation. Given a triples map and its respective data source, SOM generates



dataSource1

Gene Name	ID_sample
ALDH3A1_ET00000395555	2193351
PHF12_ET00000268756	2193351
PHF12_ET00000268756	2196270
...	...

dataSource2

ID_sample	ID_tumor
2196270	1455465
2196270	2064548
2193351	2061629
...	...

Predicate Join Tuple Table

#TripleMap2_ID_sample	
encode("2193351")	[encode(2061629)]
encode("2196270")	[encode(1455465), encode(2064548)]

Fig. 6. **Predicate join triple table.** The PJTT for the join between triples maps `TriplesMap2` and `TriplesMap3` in Fig. 1 (lines 21–23). The encoding stored in the dictionary table is used to minimize the space to store intermediate results.

the RDF triples. The encoding of each generated RDF triple is checked against the corresponding PTT. If the RDF triple already exists in PTT, it is discarded. If not, it is added both to PTT and the KG.

Algorithm 1 Simple object map (SOM)

```
Input: Triples Map $tm1$ defines predicate $p$ on logical source $S$ and $tm1$ subjectMap is $f1(att1)$ and $tm1$ objectMap for $p$ is $f2(att2)$
Output: RDF Triples for $p$ generated from $tm1$
for each: row $\in S$
    Create RDF triple $t = (f1(row.att1), p, f2(row.att1))$
    Add $f1(row.att1)$ and $f2(row.att1))$ to the Dictionary Table
    if $encode(f1(row.att1), f2(row.att1))$ does not belong to the PTT for $p$ then
        Add $encode(f1(row.att1), f2(row.att1))$ to PTT for $p$
        Add $(f1(row.att1), p, f2(row.att1))$ to the KG
    end if
    return $KG$
```

Object Reference Map (ORM) implements the object reference between two triples maps defined over the same data source. It extends SOM by using the subject of the parent triples map as the object of another triple map. Thus, the same process as in SOM is applied to the generated RDF triples, i.e., the encoding of the triples is checked against PTT to determine if they will be added to the KG. Algorithm 2 illustrates the ORM implementation.

Algorithm 2 Object reference map (ORM)

```
Input: Triples Map $tm1$ refers to Triples Map $tm2$ to define predicate $p$ on logical source $S$ and $tm1$ subjectMap is $f1(att1)$ and $tm2$ subjectMap is $f2(att2)$
Output: RDF Triples for $p$ generated from $tm1$
for each: row $\in S$
    Create RDF triple $t = (f1(row.att1), p, f2(row.att1))$
    Add $f1(row.att1)$ and $f2(row.att1))$ to the Dictionary Table
    if $encode(f1(row.att1), f2(row.att1))$ does not belong to the PTT for $p$ then
        Add $encode(f1(row.att1), f2(row.att1))$ to PTT for $p$
        Add $(f1(row.att1), p, f2(row.att1))$ to the KG
    end if
    return $KG$
```

Object Join Map (OJM) implements an index join in executing a join condition between two triples maps defined over two different data sources. OJM resorts to the corresponding PJTT to access the encoded values in the child map associated with the encoded values of the data source of the parent triples map. Thus, if the encoded value *encode*(e) of a value in the data source of the child triples map exists in PJTT, then the set of values for *encode*(e) is used to generate the resulting RDF triples. Finally, similar to the last two operations, the generated RDF triples are checked against the corresponding PTT to avoid duplicate generation. Algorithm 3 sketches the OJM implementation for a join between triples maps $tm1$ and $tm2$.

Algorithm 3 Object join map (OJM)

```
Input: Triples Map $tm1$ refers to Triples Map $tm2$ to define predicate $p$ on logical sources $S1$ and $S2$ and join condition $B$ on attributes $S1Att$ and $S2Att$, respectively, and $tm1$ subjectMap is $f1(att1)$ and $tm2$ subjectMap is $f2(att2)$
Output: RDF Triples for $p$ generated from $tm1$
for row1 $\in S1$ do
    if $encode(row1.S1Att)$ belongs to the PJTT for $tm2$ then
        for v $\in$ valueSet($encode(row1.S1Att)$) in PJTT for $tm2$ do
            Create $t = (f1(row1.att1), p, f2(decode(v)))$
            Add $f1(row1.att1)$ and $f2(decode(v))$ to the Dictionary Table
            if $encode(f1(row1.att1), f2(decode(v)))$ does not belong to the PTT for $p$ then
                Add $encode(f1(row1.att1), f2(decode(v)))$ to PTT for $p$
                Add $(f1(row1.att1), p, f2(decode(v)))$ to the KG
            end if
        end for
    end if
end for
return $KG$
```

4.4. Requirements for data integration and SDM-RDFizer

RML allows for the definition of mapping rules over structured data (e.g., CSV and relational databases) and semi-structured data (e.g., XML and JSON). SDM-RDFizer is an RML-compliant engine that can process all data source formats that RML covers, fulfilling RE1-Heterogeneous data and RE6-Semi-structured data. Additionally, by being able to process RML mapping and their corresponding data sources, RE7-Standardized data integration has also been covered. SDM-RDFizer is developed on Python, and given the nature of the programming language, there is no hard cap on how much memory a process can consume (only being limited by the environment in which the engine is executed), which allows data sources of all sizes to be processed, thus fulfilling RE2-Large data. The engine can also upload all the corresponding attributes of the data sources, hence covering RE3-Fragmented data. SDM-RDFizer implements multiple data structures that optimize different aspects of the KG creation process, like duplicate removal and join execution. In the case of duplicate removal, the engine presents PTT, a hash table that stores all the triples generated by that point in time, and all new triples are compared to the corresponding PTT to determine if it is a duplicate. The triple is discarded if it is a duplicate, thus fulfilling RE4-Duplicated data. Finally, the engine introduces three operators (SOM, ORM, and OJM), each transforming a different type of mapping rule, therefore proving RE8-Mapping complexity. Furthermore, the OJM uses a data structure called PJTT; it stores the result of executing a join between two triples maps, so executing the same join multiple times is unnecessary, thus proving RE5-Data diversity.

5. Empirical evaluation

This section presents the main results of the experimental evaluation conducted on SDM-RDFizer, aiming to address the following research questions: **RQ1)** *What is the impact of the data duplicate rates in the execution time of a knowledge graph creation approach?* **RQ2)** *What is the impact of the input data size in the execution time of a knowledge graph creation approach?* **RQ3)** *How the types of a triples maps affect the existing engines?*

To provide a comprehensive overview of the empirical assessment and the observed results, this section includes the definition of the experimental configuration. This configuration encompasses the selection of benchmarks, metrics, engines, and the description of the experimental environment used to evaluate state-of-the-art RML engines. Each experimental configuration is repeated five times, and the average execution time is reported as the outcome. The obtained results are carefully analyzed to identify the strengths and weaknesses of SDM-RDFizer in comparison to other engines.

5.1. Experimental settings

Benchmarks: Experiments are performed over datasets from **GTFS-Madrid-Bench** and **SDM-Genomic-Datasets**. Therefore, our experiments cover a large range of parameters that affect the KG creation task, i.e., dataset size, triples map type and complexity, selectivity of the results, and types of join between the triples maps. In total, we consider three different mapping types: Simple Object Map (SOM), Object Reference Map (ORM), and Object Join Map (OJM). All resources and experimental settings used in this evaluation are publicly available[21] and Table 1 summarizes the used configurations.

GTFS-Madrid-Bench [16]: This benchmark enables the generation of different configurations that affect the characteristics of creating a KG. We generate four logical sources with the scaling factor 1-csv, 5-csv, 10-csv, and 50-csv. The scale value influences the size of the resulting KG. For example, the KG generated from 5-csv is five times bigger than the KG generated from 1-csv. We consider 13 triples maps with 73 SOMs, and 12 OJMs involving ten data sources.

SDM-Genomic-Datasets [33]: This benchmark is created by randomly sampling data records from somatic mutation data collected in COSMIC.[22] SDM-Genomic-Datasets include eight different logical sources of various sizes, including 10k, 100k, 1M, and 5M rows. For every pair of sources of the same size, they differ in the percentage of the data duplicate rate, which can be either 25% or 75%, where each duplicate value is repeated 20 times. For example, a 10k logical source with 75% data duplicate rate has 25% duplicate-free records (i.e., 2500 rows) and the rest of the 75% records (i.e., 7500 rows) correspond to 375 different records which are duplicated 20 times; in total there are 2,875 unique values. The SDM-Genomic-Datasets offers ten different configurations. **Conf1:** A triples map containing one SOM. **Conf2:** A triples map containing four SOMs. **Conf3:** Set of two triples maps with one ORM. **Conf4:** Set of five triples maps with four ORMs. **Conf5:** Set of two triples maps with one OJM. **Conf6:** Set of five triples maps with four OJMs. We group the aforementioned triples map configuration into a set named **AllTogether**. Furthermore, the benchmark includes three additional configurations to evaluate the impact of two other influential parameters on the performance of KG creation frameworks.[23] **Conf7** aims at evaluating the impact of defining the same predicates using different triples maps. It has a set of four triples maps with two OJMs. For each pair of triples maps, there is an OJM. The data sources of one pair of the triples maps are a subset of the other pair. Both pairs of triples maps share the same predicate. **Conf8** provides a triples map that is connected to five other triples maps with different logical sources through join, i.e., this triples map is connected via a five-star join with the other triples maps. It comprises a set of six triples maps with five OJMs; five child triples maps refer to the same parent triples map. **Conf9** combines the first two configurations in one testbed; it is composed of a set of ten triples maps with seven OJMs.

State-of-the-art RML Engines and Metrics.The following RML tools are included in the empirical study. RMLMapper v6.0,[24] RocketRML v1.11.3 [57],[25] Morph-KGC v2.1.1 [3],[26] and SDM-RDFizer v4.5.6;[27] it implements all the techniques described in this paper, i.e., planning techniques, physical operators, and data compression

[21] https://github.com/SDM-TIB/SDM-RDFizer-Experiments/tree/master/swj2022
[22] https://cancer.sanger.ac.uk/cosmic GRCh37, version90, released August 2019.
[23] https://doi.org/10.6084/m9.figshare.17142371
[24] https://github.com/RMLio/rmlmapper-java
[25] https://github.com/semantifyit/RocketRML/
[26] https://github.com/oeg-upm/Morph-KGC
[27] https://github.com/SDM-TIB/SDM-RDFizer.

Table 1

Datasets and configurations of triples maps. The table describes each data source and configuration of TMs used in the experiments and their corresponding benchmarks. Configuration of TMs in bold are considered complex cases. They include several types of TMs of various complexity and complex joins (e.g., five-start joins)

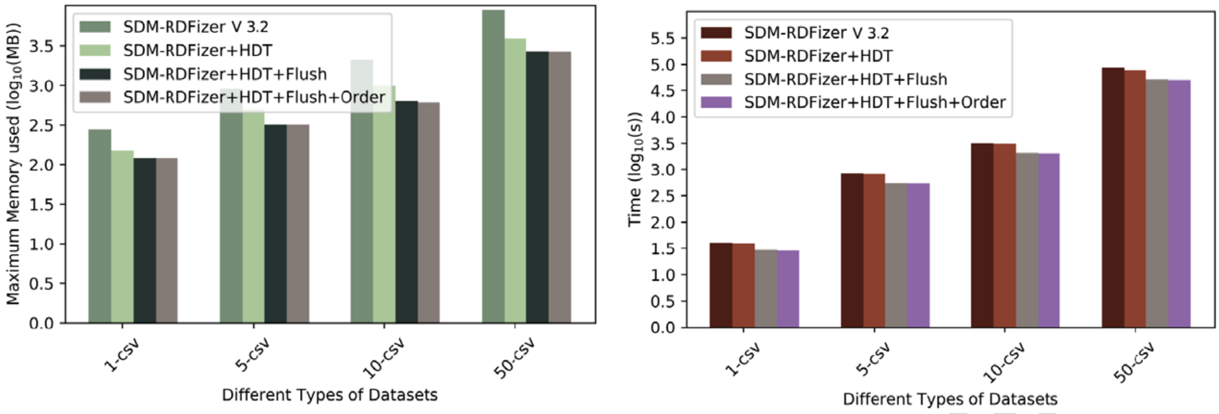
	Parameter: Dataset Size	
Benchmark	Size	Description
GTFS-Madrid-Bench	1-CSV	Ten different data sources are 4.8 Mb in total.
	5-CSV	Ten different data sources are 10 Mb in total. The generated KG is five times bigger than the KG generated from 1-CSV.
	10-CSV	Ten different data sources are 21 Mb in total. The generated KG is ten times bigger than the KG generated from 1-CSV.
	50-CSV	Ten different data sources are 102 Mb in total. The generated KG is fifty times bigger than the KG generated from 1-CSV.
SDM-Genomic-Datasets	10k	Each data source has 10,000 rows.
	100k	Each data source has 100,000 rows.
	1M	Each data source has 1,000,000 rows.
	5M	Each data source has 5,000,000 rows.
	Parameters: Mapping Assertion (MA) Type and Complexity, Selectivity of the Results, and Type of Joins	
Benchmark	Mapping Configuration	Description
GTFS-Madrid-Bench	Standard Config	13 TMs with 73 SOMs, and 12 OJMs.
SDM-Genomic-Datasets	Conf1	A TM containing one SOM.
	Conf2	A TM containing four SOMs.
	Conf3	Set of two TMs, with one ORM.
	Conf4	Set of five TMs with four ORMs.
	Conf5	Set of two TMs, with one OJM.
	Conf6	Set of five TMs, with four OJMs.
	AllTogether	Combines Conf1- Conf6.
	Conf7	Set of four TMs with two OJMs. Evaluates the impact of defining the same predicates using different TMs.
	Conf8	Set of six TMs with five OJMs. Recreates a five-star join where five TMs refer to the same parent TM.
	Conf9	Set of ten TMs with seven OJMs. Combines Conf7+Conf8

techniques to reduce the size of the main memory structures required to store intermediate results that were generated. The SDM-RDFizer v3.2[28] is also used to determine if there is an increase in performance between the older and newer version.

The other state-of-the-art engines were chosen because they excel in handling one of the requirements mentioned above. Morph-KGC [3] was chosen because it presented an extensive study that compares the execution time of Morph-KGC against several RML/R2RML engines in GTFS-Madrid-Bench and SDM-Genomic Datasets, outperforming all of them in most cases. Morph-KGC divides the triples maps into smaller triples maps and executes them in parallel (requirements RE3-Fragmented data and RE8-Mapping complexity). RMLMapper handles heterogeneous data well (requirements RE1-Heterogeneous data and RE6-Semi-structured data), and in the study conducted in Arenas-Guerrero et al. [4], it presents the best level of conformance with respect to test cases defined by RML [30]. Finally, RocketRML was chosen because it has multiple specialized implementations focused on improving join execution between mappings [57] (requirement RE8-Mapping complexity).

The performance of the RML engines is evaluated in terms of the following metrics. *Execution time*: Elapsed time spent to create a KG. The execution time is measured using the Python library `time`. The experiments are executed five times, and the average is reported. The timeout is five hours. *Maximum memory usage*: The most memory used to generate a KG. The memory usage is measured using the Python library `malloc`. The `malloc` library measures the memory usage in Kilobytes; we convert the result into Megabytes. The experiments are executed in

[28] https://doi.org/10.5281/zenodo.3872104.



(a) Maximum Memory used.

(b) Execution time.

Fig. 7. **Results of the execution of the GTFS-Madrid-bench benchmark.** Execution time and memory consumption of various version of the SDM-RDFizer when transforming the GTFS-Madrid-bench benchmark.

an Intel(R) Xeon(R) equipped with a CPU E5-2603 v3 @ 1.60 GHz 20 cores, 64 GB memory and with the O.S. Ubuntu 16.04LTS with a disk speed of 222.14 MB/sec.

5.2. *Performance of the RML engines in the GTFS-Madrid-bench*

This experiment aims to illustrate the performance increase regarding execution time and memory consumption the proposed data structures and operators will bring when implementing them into a KG creation engine. We evaluate the performance of different versions of the SDM-RDFizer. The previous version of SDM-RDFizer (i.e., SDM-RDFizer v3.2) only contains the operators for triples map transformation and the data structures for duplicate removal and join execution. In contrast, the much more complete version of the SDM-RDFizer (i.e., SDM-RDFizer v4.5.6, a.k.a. SDM-RDFizer+HDT+Flush+Order) contains all the proposed data structures and operators. We also include other combination of the proposed techniques: one only applies data compression (i.e., SDM-RDFizer+HDT), and the other has data compression and main memory flushing but no ordering (i.e., SDM-RDFizer+HDT+Flush).

It can be seen in Fig. 7b that SDM-RDFizer v3.2 and SDM-RDFizer+HDT do not have much difference in execution time; this is because compressing data requires time to be executed; thus, any savings that result from this step can only be appreciated in terms of memory consumption (Fig. 7a). On the other hand, SDM-RDFizer+HDT+Flush and SDM-RDFizer+HDT+Flush+Order reduce execution time compared to the two previous versions of the SDM-RDFizer. This reduction in execution time can be attributed to flushing unneeded data from main memory, thus making the duplicate removal process faster. Unfortunately, there are few savings between these last two configurations of the SDM-RDFizer. This testbed contains 13 triples maps, thus making the organization process take longer and negatively impacting the execution time.

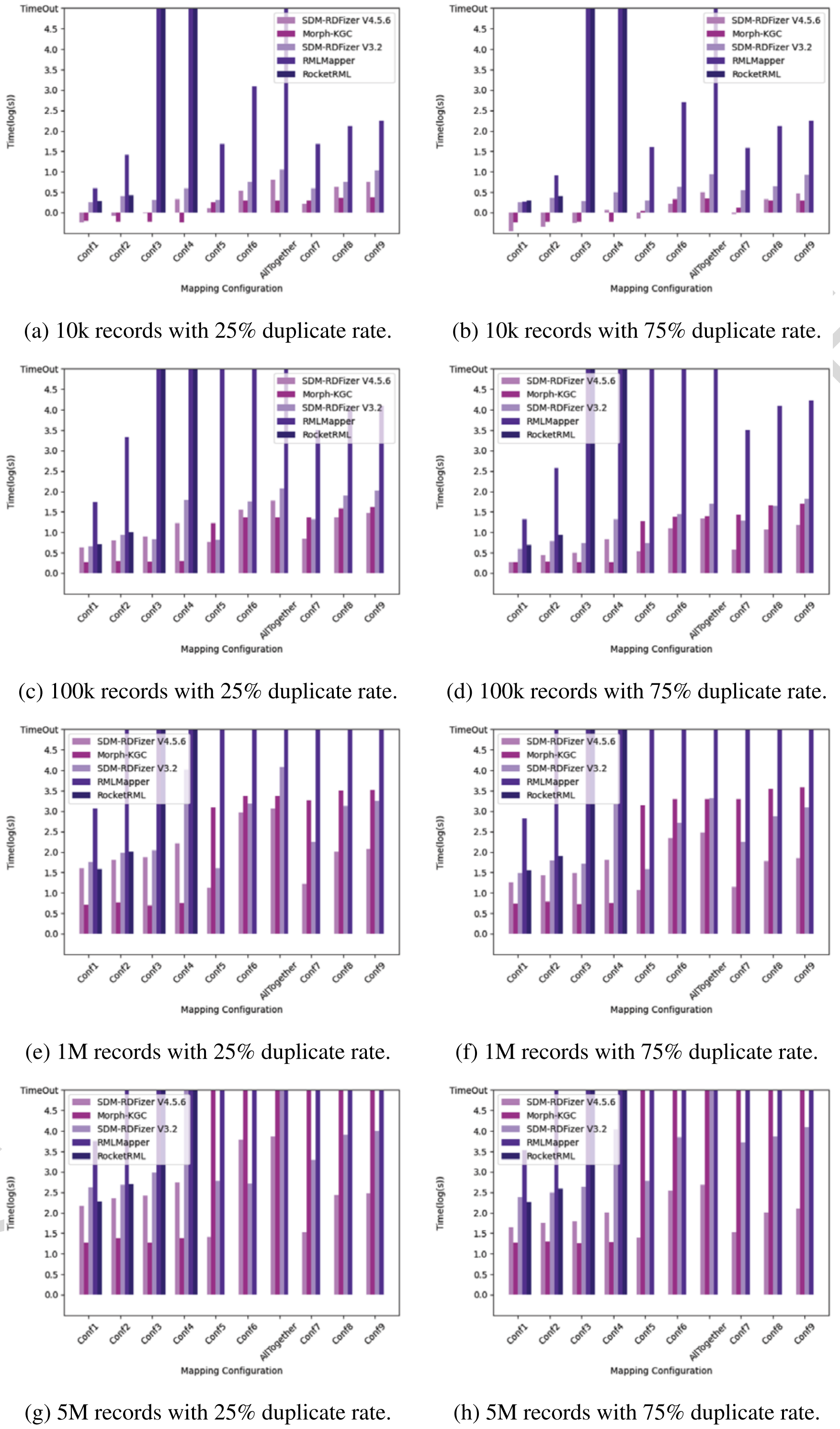
Figure 7a illustrates the maximum memory consumption of the different versions of SDM-RDFizer used. It can be seen in Fig. 7a that the SDM-RDFizer v3.2 is the one that consumes the most memory. By applying data compression, there is a significant reduction in memory consumption caused by the data stored in PTT for duplicate removal, which is much smaller than the data stored in the SDM-RDFizer v3.2. Flushing unneeded data reduces the maximum memory used even further, but not as much as with data compression. Finally, SDM-RDFizer+HDT+Flush and SDM-RDFizer+HDT+Flush+Order have the same maximum memory consumption since the only difference between them is the order in which the triples maps are executed. The benefit of executing the triples maps in a predetermined order is that the maximum amount of data is flushed after finishing the execution of a triples map, thus minimizing the amount of memory being used.

5.3. Performance of the RML engines in the SDM-genomic-datasets

This experiment seeks to prove the impact of using real-world data for KG creation. Even though the triples maps defined for SDM-Genomic-Datasets are simpler than those defined for GTFS-Madrid-Bench, they cover all the triples map types defined in Fig. 1. We evaluate the performance of each engine, i.e., SDM-RDFizer v3.2 and v4.5.6 (i.e., SDM-RDFizer+HDT+Flush+Order), Morph-KGC, RMLMapper, and RocketRML, by measuring the overall execution time it took the engine to complete the KG creation process. As it can be seen in Fig. 8, in case of having ORMs (i.e., **Conf3**, **Conf4**, and **AllTogether**), the engines RMLMapper and RocketRML are not capable of completing these configurations before timing out (i.e., five hours). In addition, RocketRML is not capable of executing N-M joins, thus the corresponding test cases were not executed (i.e., **Conf5**, **Conf6**, **AllTogether**, **Conf7**, **Conf8**, and **Conf9**). Regarding the simpler cases (i.e., **Conf1** and **Conf2**), Morph-KGC presents the lowest execution time since they partition each data source into chunks and then apply the triples map to multiple rows simultaneously. In the cases with ORMs (i.e., **Conf3**, **Conf4**, and **AllTogether**), Morph-KGC also presents the lowest execution times; this is because they apply a transformation that turns all ORMs into equivalent SOMs. For all the engines, there is a reduction in execution time when transforming cases with high duplicate rates compared to the execution time of cases with lower duplicate rates. Mapping configurations transformed with larger data sources have longer execution times regardless of the engine. We compared the performance of the SDM-RDFizer v3.2 and SDM-RDFizer v4.5.6, and Fig. 8 illustrates that SDM-RDFizer v4.5.6 has a reduction in execution time, especially in the configurations with OJMs. We can attribute this reduction in execution time to the new data structures (i.e., DT, OTML, and PL) introduced in this work, which help to reduce memory consumption and, by extension, execution time. Additionally, Fig. 8 shows that SDM-RDFizer v4.5.6 is the only engine capable of executing all the test cases. For most of the cases containing OJMs except the cases with data sources with 10k rows (i.e., Fig. 8a and Fig. 8b), the SDM-RDFizer v4.5.6 presents the lowest execution time among the tested engines. Note that SDM-RDFizer v3.2 can also execute all the triples maps; however, SDM-RDFizer v4.5.6 is more efficient. The PJTT data structure allowed the triples maps to be executed as efficiently as possible. PJTT performs the join between the parent and child data source and stores the results in main memory, thus avoiding uploading the parent data source multiple times. The SDM-RDFizer v3.2 is also capable of executing these triples maps without timing out but in a much less efficient manner; the reason for this is that the amount of memory consumed is much greater because it lacks techniques that the newer version has. In the case of Morph-KGC, this engine depends on the Python library `Pandas` for the execution of joins, which has a decent execution time for small data sources (i.e., Fig. 8a, Fig. 8b, Fig. 8c, and Fig. 8d). Still, when dealing with larger data sources (i.e., Fig. 8e, Fig. 8f, Fig. 8g, and Fig. 8h), there is an increase in the execution time. Finally, RMLMapper lacks operators capable of executing OJMs efficiently; thus, it executes a Cartesian product between the two data sources.

5.4. Discussion

After observing the performance of SDM-RDFizer during the experimental study and considering the established research questions, the following conclusions have been reached. SDM-RDFizer presented a lower execution time when transforming cases with high duplicate rates, as seen in Fig. 8b, Fig. 8d, Fig. 8f, and Fig. 8h. Therefore, the data duplicate rate is inversely proportional to the execution time. These figures illustrate that SDM-RDFizer has the lowest execution time in complex cases (Conf5, Conf6, AllTogether, Conf7, Conf8, and Conf9) with a high duplicate rate, therefore proving that the duplicate rate of the data impacts the performance of KG creation engines and answering **RQ1**. These results also demonstrate that SDM-RDFizer can handle data with high duplicates, thus fulfilling **RE4-Duplicated data**. The execution time of the proposed solution increases as the size of the data sources, as can be observed in Fig. 7 and Fig. 8. Thus, the size of the data source is directly proportional to the execution time. Figure 8e, Fig. 8f, Fig. 8g, and Fig. 8h illustrate that SDM-RDFizer has lower execution than the other RML engines when performing complex cases with large data sources, hence proving that the size of the input data influences the performance of KG creation engines and answering **RQ2**. Furthermore, these results confirm that SDM-RDFizer can transform large data sources, thus, fulfilling **RE2-Large data** and **RE3-Fragmented data**. Finally, SDM-RDFizer presented higher execution time when transforming complex cases than simple cases (Conf1, Conf2, Conf3, and Conf4), as seen in Fig. 8. Therefore, the complexity of the mapping rules is directly proportional



(a) 10k records with 25% duplicate rate.

(b) 10k records with 75% duplicate rate.

(c) 100k records with 25% duplicate rate.

(d) 100k records with 75% duplicate rate.

(e) 1M records with 25% duplicate rate.

(f) 1M records with 75% duplicate rate.

(g) 5M records with 25% duplicate rate.

(h) 5M records with 75% duplicate rate.

Fig. 8. **Results of the execution of the SDM-genomic-datasets benchmark.** Execution time of Conf1, Conf2, Conf3, Conf4, Conf5, Conf6, AllTogether, Conf7, Conf8, and Conf9 for SDM-RDFizer, morph-KGC, RMLMapper, and RocketRML.

to the execution time. In the Fig. 8 , SDM-RDFizer has a lower execution time than the other state-of-the-art engines, especially in complex cases with high duplicate rates and large data sources, then, proving that the complexity of the mapping rules impacts the performance of KG creation engines and answering **RQ3**. These results prove that the proposed approach can transform complex mapping rules, fulfilling **RE5-Data diversity** and **RE8-Mapping complexity**.

6. The SDM-RDFizer characteristics and applications

This section provides an overview of the key features of SDM-RDFizer and outlines its involvement in various projects, highlighting the role it has played.

6.1. Main characteristics of SDM-RDFizer

The SDM-RDFizer engine presents a distinctive set of characteristics that make it a valuable contribution for users and practitioners who create KGs for their projects and use cases.

Novelty: SDM-RDFizer is an RML engine that implements a hybrid approach to ensure high performance and scalability in complex data integration scenarios. On the one hand, the heuristic-based mapping planning based on the input sources and the list of predicates ensures efficient use of the main memory (requirements **RE1-Heterogeneous data**, **RE2-Large data**, **RE3-Fragmented data**, and **RE6-Semi-structured data**). On the other hand, the encoding approach and the physical data structures with their corresponding operators guarantee the fulfillment of the requirements **RE4-Duplicated data**, **RE5-Data diversity**, **RE7-Standardized data integration**, and **RE8-Mapping complexity**. To our knowledge, this is the first KG creation engine based on RML that implements mapping planning and physical data structures, demonstrating its efficiency over several testbeds on well-known benchmarks.

Availability: SDM-RDFizer is available for (re)use in multiple ways. The source code is accessible through the GitHub repository[27] under an Apache 2.0 license so that any developer can extend and modify it. The GitHub repository is also linked to the Zenodo platform, which provides a Digital Object Identifier (DOI) for the general repository[29] and also a DOI for each specific software release.[30] The engine is also available on the Python Package Index (PyPI), so it can be easily installed and integrated in other developments.[31] Finally, we also provide a docker image that deploys the SDM-RDFizer as a web service.

Utility: As demonstrated in our experimental evaluation, SDM-RDFizer is an efficient RML engine in terms of performance and scalability. The implementation of heuristic-based planning and physical data structures permitted SDM-RDFizer to scale up the construction of KGs, overcoming other state-of-the-art solutions. With the different configurations and optimizations implemented in our engine, it can be applied to different use cases, efficiently handling the parameters that affect the creation of KGs and satisfying data integration requirements. In addition, SDM-RDFizer passed all proposed RML test-cases [30],[32] which means that our engine is fully compliant with the RML specification. Lastly, SDM-RDFizer v4.5.6 was awarded with the Task-specific prize at the Knowledge Graph Construction Workshop 2023 Challenge at ESWC 2023 [63].

Impact: The number of commits in the GitHub repository[27] and the number of releases reflect the continuous improvements and support we give to our tool. At the time of writing, 25 users have forked the source code to reuse or extend it with additional features, and the repository has 98 stars. With the implementation of the novel techniques presented in this paper, we expect that SDM-RDFizer will become a reference implementation for RML and also convince industry partners to adopt declarative data integration solutions that ensure the maintenance of their KG creation pipelines. The new developments have provided the basis to scale up to real-world settings where large and heterogeneous data sources must be integrated. These use cases have demanded the fulfillment of specific requirements, and SDM-RDFizer has effectively contributed:

[29] The DOI for the SDM-RDFizer repository is: https://doi.org/10.5281/zenodo.3872103.

[30] For example, the DOI for the v4.5.6 used in the experiments of this paper is: https://doi.org/10.5281/zenodo.7027549.

[31] https://pypi.org/project/rdfizer/.

[32] https://rml.io/implementation-report

- *Scalability of large energy-related data*: In the context of the PLATOON project[45], data integration poses challenges related to interoperability and the need for a unified view of data and metadata. To address this, SDM-RDFizer was integrated into a pipeline to develop a semantic connector for creating a KG focused on electricity balance and predictive maintenance [37]. The KG created using SDM-RDFizer consists of 80,762,377 entities and 220,204,301 RDF triples. The European Commission has recognized this connector as a Key Innovation Tool, highlighting its significance in integrating renewable energy sources (RES) data.[33]
- *Efficient execution to complex KG pipelines*: In the domain of lung cancer research, diverse data sources need to be integrated to facilitate the discovery of patterns relevant to therapy effectiveness, long-term toxicities, and disease progression. SDM-RDFizer has been incorporated into a knowledge-driven framework to overcome challenges related to interoperability and data quality in lung cancer data. This framework provides a foundation for addressing clinical research questions in lung cancer (Fotis et al. [2] and Torrente et al. [62]).

6.2. *Applications of SDM-RDFizer*

The SDM-RDFizer has been utilized in various industrial and research projects to create KGs from heterogeneous data sources. The following list highlights a selection of these projects.

- iASiS,[34] EU H2020 funded project to exploit patient data insights towards precision medicine. SDM-RDFizer played a pivotal role in the iASiS project, facilitating the creation of ten versions of the iASiS KG within a span of three years. These KGs, encompassing more than 1.2 billion RDF triples, seamlessly integrate data from over 40 heterogeneous sources using 1,300 RML triples maps [64]. The data sources includes clinical records and genomic data from UK Biobank[35] for dementia patients, as well as data from lung cancer patients at the Hospital Puerta del Hierro in Madrid.[36] Additionally, the iASiS KGs incorporate structured representations of scientific publications from PubMed,[37] drug-drug interactions from DrugBank,[38] drug side effects from SIDER,[39] and UMLS.[40] The integration of these heterogeneous and large datasets underscores the role of SDM-RDFizer in achieving the data management objectives in the iASiS project and specifically fulfilling the requirements **RE2-Large data**, **RE3-Fragmented data**, **RE5-Data diversity**, **RE7-Standardized data integration**, and **RE8-Mapping complexity**.
- Lung Cancer Pilot of BigMedilytics,[41] where the KG is defined in terms of 800 RML triples maps from around 25 data sources; it comprises 149,484,936 RDF triples. SDM-RDFizer allowed for the integration of structured clinical records of 1,200 lung cancer patients from Hospital Puerta del Hierro in Madrid and the clinical services visited by these patients, with data about the interactions between the drugs that compose their oncological therapies and their treatments for the comorbidities they may suffer. The use of RML was crucial for defining and maintaining the correspondences among the unified schema and 88 different data sources; SDM-RDFizer enabled the evaluation of these mappings to create nine versions of the KG of the lung cancer pilot of BigMedilytics. Similarly in iASiS, the requirements **RE2-Large data**, **RE3-Fragmented data**, **RE5-Data diversity**, **RE7-Standardized data integration**, and **RE8-Mapping complexity** were satisfied.
- In CLARIFY,[42] nine versions of the project KG were created; they integrate data from lung and breast cancer patients collected in various formats (e.g., CSV and relational databases) with structured representations of publications from PubMed, drug-drug interactions collected from DrugBank, side effects from SIDER, and

[33] A dedicated Knowledge Graph for integration of RES data sources https://innovation-radar.ec.europa.eu/innovation/43139.
[34] http://project-iasis.eu/
[35] https://www.ukbiobank.ac.uk/
[36] https://www.comunidad.madrid/hospital/puertadehierro/
[37] https://pubmed.ncbi.nlm.nih.gov/
[38] https://go.drugbank.com/
[39] http://sideeffects.embl.de/
[40] https://www.nlm.nih.gov/research/umls/index.html
[41] https://www.bigmedilytics.eu/
[42] https://www.clarify2020.eu

medical terms from UMLS. The KG definition comprises 1,749 RML triples maps establishing the correspondences with 258 different logical sources. The CLARIFY KG comprises 78M RDF triples and 16M RDF resources. In addition to the requirements satisfied in iASiS and BigMedilytics, here SDM-RDFizer allowed for meeting the requirements of **RE1-Heterogeneous data**.

- P4-LUCAT[43] has 676 RML triples maps that define the P4-LUCAT KG in terms of a unified schema of 318 attributes and 177 classes; it comprises 178M of RDF triples. Data is collected from various sources in different formats, such as CSV and relational databases. As a result, SDM-RDFizer enabled the fulfillment of all the requirements outlined in Section 2.3.
- The ImProVIT KG[44] integrates immune system data into a unified schema consisting of 102 classes, 88 predicates, and 175 attributes. SDM-RDFizer was employed to construct the project KG, generating 6,005,844 RDF triples and 220,414 entities through the evaluation of 577 triples maps. Additionally, for various studies [26], SDM-RDFizer executed multiple versions of these triples maps, resulting in over 40 versions of the ImProVIT KG. The requirements **RE2-Large data**, **RE3-Fragmented data**, **RE5-Data diversity**, **RE7-Standardized data integration**, and **RE8-Mapping complexity** were satisfied.
- The PLATOON project[45] is dedicated to creating the KG for a pilot [37] defined in terms of 2,093 RML triples maps. These mappings define 158 classes and 107 predicates of SEDMOON, the Semantic Data Models of Energy.[46] Data were collected from a relational database comprising 600 GB of energy-related observational data collected over six years. The resulting KG comprises 220M RDF triples and 80 million RDF resources. SDM-RDFizer played a crucial role in meeting the requirements of **RE3-Fragmented data**, **RE5-Data diversity**, **RE7-Standardized data integration**, and **RE8-Mapping complexity**. However, given the substantial volume of the data sources, the data management techniques implemented in SDM-RDFizer v4.5.6 were pivotal in fulfilling requirement **RE2-Large data**.
- The Knowledge4COVID-19 KG [52] comprises 80M RDF triples integrating COVID-19 scientific publications and COVID-19 related concepts (e.g., drugs, drug-drug interactions, and molecular dysfunctions). It is defined in terms of 57 RML triples maps. All the data was collected from tabular CSV files; SDM-RDFizer allowed for the satisfaction of **RE2-Large data**, **RE3-Fragmented data**, **RE5-Data diversity**, **RE7-Standardized data integration**, and **RE8-Mapping complexity** [52].
- H2020 – SPRINT[47] studies performance and scalability of different semantic architecture for the Interoperability Framework on Transport across Europe. The SDM-RDFizer was used to evaluate the impact of different parameters on the transport domain during the creation of KGs to identify bottlenecks and allow optimizations. Additionally, under this project, the GTFS-Madrid-Bench [16] was also defined, where the SDM-RDFizer was used to materialize the KG used for comparing the performance between native triplestores and virtual KG creation engines.
- EIT-SNAP[48] innovation project on the application of semantic technologies for transport national access points, and SDM-RDFizer allowed the integration of transportation data in Spain. In this specific project, the SDM-RDFizer was integrated in a sustainable workflow to construct KGs based on the Transmodel ontology [51] in a systematic manner [14].
- Open Cities[49] is a Spanish national project on creating common and shared vocabularies for Spanish cities; SDM-RDFizer executes the RML mapping rule for integrating geographical data for Spanish cities, hence ensuring the interoperability between open data in Spain throughout the created KGs.
- Virtual Platform for the H2020 European Joint Programme on Rare Disease,[50] and SDM-RDFizer merges data collected from the consortium partners, thus, satisfying the requirements in Section 2.3.

[43] https://p4-lucat.eu/

[44] https://www.tib.eu/en/research-development/project-overview/project-summary/improvit

[45] https://platoon-project.eu/.

[46] https://github.com/PLATOONProject/SEDMOON/tree/main

[47] http://sprint-transport.eu/

[48] https://www.snap-project.eu/

[49] https://ciudades-abiertas.es/

[50] https://www.ejprarediseases.org

- CoyPU[51] is a German-funded project where SDM-RDFizer generates KGs for various events collected from economic value networks in the industrial environment and social context. Specifically, SDM-RDFizer is utilized to create a federation of KGs that integrates data from World Bank, Wikidata, DBpedia, and the CoyPU KG.[52] The World Bank dataset comprises 21 topics, ranging from Agriculture and Rural Development to Trade. Each topic has three CSV files associated with it. The CSV files contain annual statistics per country from 1960 to 2022. In total, 63 CSV files are transformed, and a KG comprised of 111 Million RDF triples is generated. **RE2-Large data**, **RE3-Fragmented data**, **RE5-Data diversity**, **RE7-Standardized data integration**, and **RE8-Mapping complexity** were satisfied during the generation of this KG.

7. Conclusions and future work

This paper introduces novel data management techniques that leverage innovative data structures and physical operators for the efficient execution of RML triples maps. These techniques have been implemented in SDM-RDFizer v4.5.6, and their effectiveness has been empirically evaluated through 416 testbeds encompassing state-of-the-art RML engines and benchmarks. The results highlight the significant computational power of well-designed data structures and algorithm operators, particularly in complex scenarios involving star joins across multiple triples maps. We anticipate that the reported findings and the availability of the new version of SDM-RDFizer will inspire the community to adopt declarative approaches in defining KG creation pipelines using RML and to explore data management techniques that can further enhance the performance of their engines. For future work, we aim to develop a flushing policy for the Predicate Join Tuple Table (PJTT) to reduce memory consumption by eliminating values of redundant joins. Additionally, we pursue optimizing the Simple Object Map (SOM) and Object Reference Map (ORM) operators to enhance their respective transformation capabilities. Furthermore, we plan to devise efficient data management techniques to empower SDM-RDFizer for executing RML-Star mapping rules; an initial version of this implementation is already available on GitHub[53] and extending the current data structures and physical operators is part of our future tasks. We aim to enable the evaluation of observational data, such as sensor data, within the SDM-RDFizer framework. Lastly, we plan to develop parallel operators to empower SDM-RDFizer with fine-grained parallelization – as proposed by Wang et al. [66] – and speed up the process of KG creation, specifically when data sources presented as relational databases are integrated. Extensive empirical assessments with different parallel approaches are also part of our future agenda.

Acknowledgements

Enrique Iglesias is supported by Federal Ministry for Economic Affairs and Energy of Germany (BMWK) in the project CoyPu (project number 01MK21007[A-L]). Maria-Esther Vidal is partially funded by Leibniz Association, program “Leibniz Best Minds: Programme for Women Professors”, project TrustKG-Transforming Data in Trustable Insights; Grant P99/2020. This research was funded by the Galician Ministry of Education, University and Professional Training and the European Regional Development Fund (ERDF/FEDER program) through grants ED431C2018/29 and ED431G2019/04.

References

[1] S. Abiteboul, P. Buneman and D. Suciu, *Data on the Web: From Relations to Semistructured Data and XML*, Morgan Kaufmann, 1999. ISBN 1-55860-622-X.

[51] https://coypu.org/
[52] https://www.youtube.com/watch?v=CciNaaMyi8A
[53] https://github.com/SDM-TIB/SDM-RDFizer-Star

[2] F. Aisopos, S. Jozashoori, E. Niazmand, D. Purohit, A. Rivas, A. Sakor, E. Iglesias, D. Vogiatzis, E. Menasalvas, A.R. Gonzá, G. Vigueras, D. Gó, M. Torrente, R.H. Ló, M.P. Pulla, A. Dalianis, A. Triantafillou, G. Paliouras and M. Vidal, Knowledge graphs for enhancing transparency in health data ecosystems, *Semantic Web* **14**(5) (2023), 943–976. doi:10.3233/SW-223294.

[3] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M.S. Pé and O. Corcho, Morph-KGC: Scalable knowledge graph materialization with mapping partitions, *Semantic Web* **15**(1) (2024), 1–20. doi:10.3233/SW-223135.

[4] J. Arenas-Guerrero, M. Scrocca, A. Iglesias-Molina, J. Toledo, L. Pozo-Gilo, D. Doñ, O. Corcho and D. Chaves-Fraga, Knowledge graph construction with R2RML and RML: An ETL system-based overview, in: *Proceedings of the 2nd International Workshop on Knowledge Graph Construction Co-Located with 18th Extended Semantic Web Conference (ESWC 2021)*, Online, June 6, 2021, D. Chaves-Fraga, A. Dimou, P. Heyvaert, F. Priyatna and J.F. Sequeda, eds, CEUR Workshop Proceedings, Vols 2873, CEUR-WS.org, 2021, https://ceur-ws.org/Vol-2873/paper11.pdf.

[5] L. Asprino, E. Daga, A. Gangemi and P. Mulholland, Knowledge graph construction with a façade: A unified method to access heterogeneous data sources on the web, *ACM Trans. Internet Techn.* **23**(1) (2023), 6–1631. doi:10.1145/3555312.

[6] D.V. Assche, T. Delva, G. Haesendonck, P. Heyvaert, B.D. Meester and A. Dimou, Declarative RDF graph generation from heterogeneous (semi-)structured data: A systematic literature review, *J. Web Semant.* **75** (2023), 100753. doi:10.1016/J.WEBSEM.2022.100753.

[7] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann and D. Aumueller, Triplify: Light-weight linked data publication from relational databases, in: *Proceedings of the 18th International Conference on World Wide Web, WWW 2009*, Madrid, Spain, April 20–24, 2009, J. Quemada, G. Leó, Y.S. Maarek and W. Nejdl, eds, ACM, 2009, pp. 621–630. doi:10.1145/1526709.1526793.

[8] S. Battle, Gloze: XML to RDF and back again, in: *Proceedings of the First Jena User Conference HP Labs*, Bristol, 2006.

[9] S. Bischof, N. Lopes and A. Polleres, Improve efficiency of mapping data between XML and RDF with XSPARQL, in: *Web Reasoning and Rule Systems – 5th International Conference, RR 2011, Proceedings*, Galway, Ireland, August 29–30, 2011, S. Rudolph and C. Gutierrez, eds, Lecture Notes in Computer Science, Vol. 6902, Springer, 2011, pp. 232–237. doi:10.1007/978-3-642-23580-1_17.

[10] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro and G. Xiao, Ontop: Answering SPARQL queries over relational databases, *Semantic Web* **8**(3) (2017), 471–487. doi:10.3233/SW-160217.

[11] D. Calvanese, G.D. Giacomo, M. Lenzerini, D. Nardi and R. Rosati, Data integration in data warehousing, *Int. J. Cooperative Inf. Syst.* **10**(3) (2001), 237–271. doi:10.1142/S0218843001000345.

[12] P. Chandak, K. Huang and M. Zitnik, Building a knowledge graph to enable precision medicine, *Sci Data* **10** (2023), 2052–4463. doi:10.1038/s41597-023-01960-3.

[13] D. Chaves-Fraga, Knowledge graph construction from heterogeneous data sources exploiting declarative mapping rules, PhD thesis, Technical University of Madrid, Spain, 2021. doi:10.20868/UPM.thesis.67890.

[14] D. Chaves-Fraga, A. Antó, J. Toledo and O. Corcho, ONETT: Systematic knowledge graph generation for national access points, in: *Joint Proceedings of the 1st International Workshop on Semantics for Transport and the 1st International Workshop on Approaches for Making Data Interoperable Co-Located with 15th Semantics Conference (SEMANTiCS 2019)*, Karlsruhe, Germany, September 9, 2019, L. Kaffee, K.M. Endris, M. Vidal, M. Comerio, M. Sadeghi, D. Chaves-Fraga and P. Colpaert, eds, CEUR Workshop Proceedings, Vols 2447, CEUR-WS.org, 2019, https://ceur-ws.org/Vol-2447/paper8.pdf.

[15] D. Chaves-Fraga, K.M. Endris, E. Iglesias, O. Corcho and M. Vidal, What are the parameters that affect the construction of a knowledge graph? in: *On the Move to Meaningful Internet Systems: OTM 2019 Conferences – Confederated International Conferences: CoopIS, ODBASE, C&TC 2019*, Rhodes, Greece, October 21–25, 2019, H. Panetto, C. Debruyne, M. Hepp, D. Lewis, C.A. Ardagna and R. Meersman, eds, Proceedings, Vol. 11877, Springer, 2019, pp. 695–713. doi:10.1007/978-3-030-33246-4_43.

[16] D. Chaves-Fraga, F. Priyatna, A. Cimmino, J. Toledo, E. Ruckhaus and O. Corcho, GTFS-Madrid-bench: A benchmark for virtual knowledge graph access in the transport domain, *J. Web Semant.* **65** (2020), 100596. doi:10.1016/J.WEBSEM.2020.100596.

[17] D. Chaves-Fraga, E. Ruckhaus, F. Priyatna, M. Vidal and O. Corcho, Enhancing virtual ontology based access over tabular data with morph-CSV, *Semantic Web* **12**(6) (2021), 869–902. doi:10.3233/SW-210432.

[18] A. Chebotko, S. Lu and F. Fotouhi, Semantics preserving SPARQL-to-SQL translation, *Data Knowl. Eng.* **68**(10) (2009), 973–1000. doi:10.1016/J.DATAK.2009.04.001.

[19] O. Corcho, D. Chaves-Fraga, J. Toledo, J. Arenas-Guerrero, C. Badenes-Olmedo, M. Wang, H. Peng, N. Burrett, J. Mora and P. Zhang, A high-level ontology network for ICT infrastructures, in: *The Semantic Web – ISWC 2021–20th International Semantic Web Conference, ISWC 2021, Proceedings*, Virtual Event, October 24-28, 2021, A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue, Y. Ding, P.M. Barnaghi, A. Haller, M. Dragoni and H. Alani, eds, Lecture Notes in Computer Science, Vol. 12922, Springer, 2021, pp. 446–462. doi:10.1007/978-3-030-88361-4_26.

[20] E. Costetchi, A. Vassiliades and C.I. Nyulas, Towards a mapping framework for the tenders electronic daily standard forms, in: *Proceedings of the 4th International Workshop on Knowledge Graph Construction Co-Located with 20th Extended Semantic Web Conference ESWC 2023*, Hersonissos, Greece, May 28, 2023, D. Chaves-Fraga, A. Dimou, A. Iglesias-Molina, U. Serles and D.V. Assche, eds, CEUR Workshop Proceedings, Vols 3471, CEUR-WS.org, 2023, https://ceur-ws.org/Vol-3471/paper5.pdf.

[21] S. Das, S. Sundara and R. Cyganiak, R2RML: RDB to RDF Mapping Language, W3C Recommendation 27 September 2012, *W3C*, 2012, http://www.w3.org/TR/r2rml/.

[22] T. Delva, J. Arenas-Guerrero, A. Iglesias-Molina, O. Corcho, D. Chaves-Fraga and A. Dimou, RML-star: A declarative mapping language for RDF-star generation, in: *Proceedings of the ISWC 2021 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice Co-Located with 20th International Semantic Web Conference (ISWC 2021)*, Virtual Conference, October 24–28, 2021, O. Seneviratne, C. Pesquita, J. Sequeda and L. Etcheverry, eds, CEUR Workshop Proceedings, Vols 2980, CEUR-WS.org, 2021, https://ceur-ws.org/Vol-2980/paper374.pdf.

[23] A. Dimou, Creation of knowledge graphs, in: *Knowledge Graphs and Big Data Processing*, V. Janev, D. Graux, H. Jabeen and E. Sallinger, eds, Lecture Notes in Computer Science, Vol. 12072, Springer, 2020, pp. 59–72. doi:10.1007/978-3-030-53199-7_4.

[24] A. Dimou, T.D. Nies, R. Verborgh, E. Mannens and R.V. de Walle, Automated metadata generation for linked data generation and publishing workflows, in: *Proceedings of the Workshop on Linked Data on the Web, LDOW 2016, Co-Located with 25th International World Wide Web Conference (WWW 2016)*, S. Auer, T. Berners-Lee, C. Bizer and T. Heath, eds, CEUR Workshop Proceedings, Vols 1593, CEUR-WS.org, 2016, https://ceur-ws.org/Vol-1593/article-04.pdf.

[25] A. Dimou, M.V. Sande, P. Colpaert, R. Verborgh, E. Mannens and R.V. de Walle, RML: A generic language for integrated RDF mappings of heterogeneous data, in: *Proceedings of the Workshop on Linked Data on the Web Co-Located with the 23rd International World Wide Web Conference (WWW 2014)*, Seoul, Korea, April 8, 2014, C. Bizer, T. Heath, S. Auer and T. Berners-Lee, eds, CEUR Workshop Proceedings, Vols 1184, CEUR-WS.org, 2014, https://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

[26] S. Dost, A. Rivas, H. Begali, A. Ziegler, E. Aliabadi, M. Cornberg, A.R. Kraft and M. Vidal, Unraveling the hepatitis b cure: A hybrid AI approach for capturing knowledge about the immune system's impact, in: *Proceedings of the 12th Knowledge Capture Conference 2023, K-CAP 2023*, Pensacola, FL, USA, December 5–7, 2023, K.B. Venable, D. Garijo and B. Jalaian, eds, ACM, 2023, pp. 241–249. doi:10.1145/3587259.3627558.

[27] K.M. Endris, P.D. Rohde, M. Vidal and S. Auer, Ontario: Federated query processing against a semantic data lake, in: *Database and Expert Systems Applications – 30th International Conference, DEXA 2019, Proceedings, Part I*, Linz, Austria, August 26–29, 2019, S. Hartmann, J. Kü, S. Chakravarthy, G. Anderst-Kotsis, A.M. Tjoa and I. Khalil, eds, Lecture Notes in Computer Science, Vol. 11706, Springer, 2019, pp. 379–395. doi:10.1007/978-3-030-27615-7_29.

[28] B. Golshan, A.Y. Halevy, G.A. Mihaila and W. Tan, Data integration: After the teenage years, in: *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017*, Chicago, IL, USA, May 14–19, 2017, E. Sallinger, J.V. den Bussche and F. Geerts, eds, ACM, 2017, pp. 101–106. doi:10.1145/3034786.3056124.

[29] C. Gutierrez and J.F. Sequeda, Knowledge graphs, *Commun. ACM* **64**(3) (2021), 96–104. doi:10.1145/3418294.

[30] P. Heyvaert, D. Chaves-Fraga, F. Priyatna, O. Corcho, E. Mannens, R. Verborgh and A. Dimou, Conformance test cases for the RDF mapping language (RML), in: *Knowledge Graphs and Semantic Web – First Iberoamerican Conference, KGSWC 2019, Proceedings*, Villa Clara, Cuba, June 23–30, 2019, B. Villazó and Y. Hidalgo-Delgado, eds, Communications in Computer and Information Science, Vol. 1029, Springer, 2019, pp. 162–173. doi:10.1007/978-3-030-21395-4_12.

[31] M. Hofer, D. Obraczka, A. Saeedi, H. Kö and E. Rahm, Construction of Knowledge Graphs: State and Challenges, CoRR arXiv:2302.11509, 2023. doi:10.48550/ARXIV.2302.11509.

[32] 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 and A. Zimmermann, *Knowledge Graphs, ACM Comput. Surv.* **54**(4) (2021), 0360. doi:10.1145/3447772.

[33] E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana and M. Vidal, SDM-RDFizer: An RML interpreter for the efficient creation of RDF knowledge graphs, in: *CIKM'20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event*, Ireland, October 19–23, 2020, M. d'Aquin, S. Dietze, C. Hauff, E. Curry and P. Cudré, eds, ACM, 2020, pp. 3039–3046. doi:10.1145/3340531.3412881.

[34] E. Iglesias, S. Jozashoori and M. Vidal, Scaling up knowledge graph creation to large and heterogeneous data sources, *J. Web Semant.* **75** (2023), 100755. doi:10.1016/J.WEBSEM.2022.100755.

[35] E. Iglesias and M. Vidal, Knowledge graph creation challenge: Results for SDM-RDFizer, in: *Proceedings of the 4th International Workshop on Knowledge Graph Construction Co-Located with 20th Extended Semantic Web Conference ESWC 2023*, Hersonissos, Greece, May 28, 2023, D. Chaves-Fraga, A. Dimou, A. Iglesias-Molina, U. Serles and D.V. Assche, eds, CEUR Workshop Proceedings, Vols 3471, CEUR-WS.org, 2023, https://ceur-ws.org/Vol-3471/paper13.pdf.

[36] A. Iglesias-Molina, D.V. Assche, J. Arenas-Guerrero, B.D. Meester, C. Debruyne, S. Jozashoori, P. Maria, F. Michel, D. Chaves-Fraga and A. Dimou, The RML ontology: A community-driven modular redesign after a decade of experience in mapping heterogeneous data to RDF, in: *Proceedings, Part II*, The Semantic Web – ISWC 2023–22nd International Semantic Web Conference, Athens, Greece, November 6–10, 2023, T.R. Payne, V. Presutti, G. Qi, M. Poveda-Villaló, G. Stoilos, L. Hollink, Z. Kaoudi, G. Cheng and J. Li, eds, Lecture Notes in Computer Science, Vol. 14266, Springer, 2023, pp. 152–175. doi:10.1007/978-3-031-47243-5_9.

[37] V. Janev, M.-E. Vidal, D. Pujić, D. Popadić, E. Iglesias, A. Sakor and A. Č, Responsible knowledge management in energy data ecosystems, *Energies* **15**(11) (2022). doi:10.3390/en15113973.

[38] M. Kejriwal, J.F. Sequeda and V. Ló, Knowledge graphs: Construction, management and querying, *Semantic Web* **10**(6) (2019), 961–962. doi:10.3233/SW-190370.

[39] B. Kinast, H. Ulrich, B. Bergh and B. Schreiweis, Functional requirements for medical data integration into knowledge management environments: Requirements elicitation approach based on systematic literature analysis, *Journal of Medical Internet Research* **25** (2023), 41344. doi:10.2196/41344.

[40] M. Lefranç, A. Zimmermann and N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, in: *The Semantic Web – 14th International Conference, ESWC 2017, Proceedings, Part I*, Portorož, Slovenia, May 28–June 1, 201, E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler and O. Hartig, eds, Lecture Notes in Computer Science, Vol. 10249, 2017, pp. 35–50. doi:10.1007/978-3-319-58068-5_3.

[41] H. Lei and K.A. Ross, Faster joins, self-joins and multi-way joins using join indices, data knowl, *Eng.* **29**(2) (1999), 179–200. doi:10.1016/S0169-023X(98)00031-7.

[42] M. Lenzerini, Data integration: A theoretical perspective, in: *Proceedings of the Twenty-First ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Madison, Wisconsin, USA, June 3–5, L. Popa, S. Abiteboul and P.G. Kolaitis, eds, ACM, 2002, pp. 233–246. doi:10.1145/543613.543644.

[43] M.N. Mami, D. Graux, S. Scerri, H. Jabeen, S. Auer and J. Lehmann, Squerall: Virtual ontology-based access to heterogeneous and large data sources, in: *The Semantic Web – ISWC 2019 – 18th International Semantic Web Conference, Proceedings, Part II*, Auckland, New Zealand, October 26–30, 2019, C. Ghidini, O. Hartig, M. Maleshkova, V. Svá, I.F. Cruz, A. Hogan, J. Song, M. Lefranç and F. Gandon, eds, Lecture Notes in Computer Science, Vol. 11779, Springer, Auckland, New Zealand, 2019, pp. 229–245. doi:10.1007/978-3-030-30796-7_15.

[44] M. Namici and G.D. Giacomo, Comparing query answering in OBDA tools over W3C-compliant specifications, in: *Proceedings of the 31st International Workshop on Description Logics Co-Located with 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018)*, Tempe, Arizona, US, October 27th – to – 29th, 2018, M. Ortiz and T. Schneider, eds, CEUR Workshop Proceedings, Vols 2211, CEUR-WS.org, 2018, https://ceur-ws.org/Vol-2211/paper-25.pdf.

[45] S.M. Oo, G. Haesendonck, B.D. Meester and A. Dimou, RMLStreamer-SISO: An RDF stream generator from streaming heterogeneous data, in: *The Semantic Web – ISWC 2022–21st International Semantic Web Conference, Proceedings*, Virtual Event, October 23–27, 2022, U. Sattler, A. Hogan, C.M. Keet, V. Presutti, J.P.A. Almeida, H. Takeda, P. Monnin, G. Pirrò and C. d'Amato, eds, Lecture Notes in Computer Science, Vol. 13489, Springer, 2022, pp. 697–713. doi:10.1007/978-3-031-19433-7_40.

[46] A. Poggi, D. Lembo, D. Calvanese, G.D. Giacomo, M. Lenzerini and R. Rosati, Linking data to ontologies, *J. Data Semant.* **10** (2008), 133–173. doi:10.1007/978-3-540-77688-8_5.

[47] S. Polfliet and R. Ichise, Automated mapping generation for converting databases into linked data, in: *Proceedings of the ISWC 2010 Posters & Demonstrations Track: Collected Abstracts*, Shanghai, China, November 9, 2010, A. Polleres and H. Chen, eds, CEUR Workshop Proceedings, Vols 658, CEUR-WS.org, 2010, https://ceur-ws.org/Vol-658/paper525.pdf.

[48] F. Priyatna, O. Corcho and J.F. Sequeda, Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph, in: *23rd International World Wide Web Conference, WWW'14*, Seoul, Republic of Korea, April 7–11, 2014, C. Chung, A.Z. Broder, K. Shim and T. Suel, eds, ACM, 2014, pp. 479–490. doi:10.1145/2566486.2567981.

[49] P.D. Rohde, E. Iglesias and M. Vidal, SHACL-ACL: Access control with SHACL, in: *The Semantic Web: ESWC 2023 Satellite Events – Hersonissos, Proceedings*, Crete, Greece, May 28–June 1, 2023, C. Pesquita, H. Skaf-Molli, V. Efthymiou, S. Kirrane, A. Ngonga, D. Collarana, R. Cerqueira, M. Alam, C. Trojahn and S. Hertling, eds, Lecture Notes in Computer Science, Vol. 13998, Springer, 2023, pp. 22–26. doi:10.1007/978-3-031-43458-7_4.

[50] J.A. Rojas, M. Aguado, P. Vasilopoulou, I. Velitchkov, D.V. Assche, P. Colpaert and R. Verborgh, Leveraging semantic technologies for digital interoperability in the European railway domain, in: *The Semantic Web – ISWC 2021–20th International Semantic Web Conference, ISWC 2021, Proceedings*, Virtual Event, October 24-28, 2021, A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue, Y. Ding, P.M. Barnaghi, A. Haller, M. Dragoni and H. Alani, eds, Lecture Notes in Computer Science, Vol. 12922, Springer, 2021, pp. 648–664. doi:10.1007/978-3-030-88361-4_38.

[51] E. Ruckhaus, A. Anton-Bravo, M. Scrocca and O. Corcho, Applying the LOT methodology to a public bus transport ontology aligned with transmodel: Challenges and results, *Semantic Web* **14**(4) (2023), 639–657. doi:10.3233/SW-210451.

[52] A. Sakor, S. Jozashoori, E. Niazmand, A. Rivas, K. Bougiatiotis, F. Aisopos, E. Iglesias, P.D. Rohde, T. Padiya, A. Krithara, G. Paliouras and M. Vidal, Knowledge4Covid-19: A semantic-based approach for constructing a Covid-19 related knowledge graph from various sources and analyzing treatments' toxicities, *J. Web Semant.* **75** (2023), 100760. doi:10.1016/J.WEBSEM.2022.100760.

[53] M. Scrocca, M. Comerio, A. Carenini and I. Celino, Turning transport data to comply with EU standards while enabling a multimodal transport knowledge graph, in: *The Semantic Web – ISWC 2020 – 19th International Semantic Web Conference, Proceedings, Part II*, Athens, Greece, November 2–6, 2020, J.Z. Pan, V.A.M. Tamma, C. d'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne and L. Kagal, eds, Lecture Notes in Computer Science, Vol. 12507, Springer, Athens, Greece, 2020, pp. 411–429. doi:10.1007/978-3-030-62466-8_26.

[54] J.F. Sequeda, W.J. Briggs, D.P. Miranker and W.P. Heideman, A pay-as-you-go methodology to design and build enterprise knowledge graphs from relational databases, in: *The Semantic Web – ISWC 2019 – 18th International Semantic Web Conference*, Auckland, New Zealand, October 26–30, 2019, Proceedings, Part II, Lecture Notes in Computer Science, Vol. 11779, Springer, 2019, pp. 526–545. doi:10.1007/978-3-030-30796-7_32.

[55] J.F. Sequeda and D.P. Miranker, Ultrawrap: SPARQL execution on relational data, *J. Web Semant.* **22** (2013), 19–39. doi:10.1016/J.WEBSEM.2013.08.002.

[56] J.F. Sequeda and D.P. Miranker, Mapping relational databases to linked data, in: *Linked Data Management*, A. Harth, K. Hose and R. Schenkel, eds, Chapman and Hall/CRC, 2014, pp. 95–115, http://www.crcnetbase.com/doi/abs/10.1201/b16859-7. doi:10.1201/B16859-7.

[57] U. Simsek, E. Kä and D. Fensel, RocketRML – a NodeJS implementation of a use case specific RML mapper, in: *Joint Proceedings of the 1st International Workshop on Knowledge Graph Building and 1st International Workshop on Large Scale RDF Analytics Co-Located with 16th Extended Semantic Web Conference (ESWC 2019)*, Portorož, Slovenia, June 3, 2019, CEUR Workshop Proceedings, Vol. 2489, CEUR-WS.org, 2019, pp. 46–53, https://ceur-ws.org/Vol-2489/paper5.pdf.

[58] C. Stadler, L. Bü, L. Meyer and M. Martin, Scaling RML and SPARQL-based knowledge graph construction with apache spark, in: *Proceedings of the 4th International Workshop on Knowledge Graph Construction Co-Located with 20th Extended Semantic Web Conference ESWC 2023*, Hersonissos, Greece, May 28, 2023, D. Chaves-Fraga, A. Dimou, A. Iglesias-Molina, U. Serles and D.V. Assche, eds, CEUR Workshop Proceedings, Vols 3471, CEUR-WS.org, 2023, https://ceur-ws.org/Vol-3471/paper8.pdf.

[59] B. Steenwinckel, G. Vandewiele, I. Rausch, P. Heyvaert, R. Taelman, P. Colpaert, P. Simoens, A. Dimou, F.D. Turck and F. Ongenae, Facilitating the analysis of Covid-19 literature through a knowledge graph, in: *The Semantic Web – ISWC 2020 – 19th International Semantic Web Conference, Athens, Proceedings, Part II,*, The Semantic Web – ISWC 2020–19th International Semantic Web Conference, Proceedings, Part II, Athens, Greece, November 2–6, 2020, J.Z. Pan, V.A.M. Tamma, C. d'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne and L. Kagal, eds, Lecture Notes in Computer Science, Vol. 12507, Springer, 2020, pp. 344–357. doi:10.1007/978-3-030-62466-8_22.

[60] L. Tailhardat, Y. Chabot and R. Troncy, Designing NORIA: A knowledge graph-based platform for anomaly detection and incident management in ICT systems, in: *Proceedings of the 4th International Workshop on Knowledge Graph Construction Co-Located with 20th Extended Semantic Web Conference ESWC 2023*, Hersonissos, Greece, May 28, 2023, D. Chaves-Fraga, A. Dimou, A. Iglesias-Molina, U. Serles and D.V. Assche, eds, CEUR Workshop Proceedings, Vols 3471, CEUR-WS.org, 2023, https://ceur-ws.org/Vol-3471/paper3.pdf.

[61] H.V. Thakker, On Supporting Interoperability between RDF and Property Graph Databases, PhD thesis, Rheinische Friedrich-Wilhelms-Universität, Bonn, 2021.

[62] M. Torrente, P.A. Sousa, R. Herná, M. Blanco, V. Calvo, A. Collazo, G.R. Guerreiro, B. Nú, J. Pimentao, J.C. Sá, M. Campos, L. Costabello, V. Novacek, E. Menasalvas, M.E. Vidal and M. Provencio, An artificial intelligence-based tool for data analysis and prognosis in cancer patients: Results from the clarify study, *Cancers* **14**(16) (2022). https://www.mdpi.com/2072-6694/14/16/4041. doi:10.3390/cancers14164041.

[63] D. Van Assche, D. Chaves-Fraga, A. Dimou, U. Ş and A. Iglesias, in: *KGCW 2023 Challenge @ ESWC 2023*, Zenodo, 2023. doi:10.5281/zenodo.7837289.

[64] M. Vidal, K.M. Endris, S. Jazashoori, A. Sakor and A. Rivas, Transforming heterogeneous data into knowledge for personalized treatments – a use case, *Datenbank-Spektrum* **19**(2) (2019), 95–106. doi:10.1007/S13222-019-00312-Z.

[65] M. Vidal, E. Niazmand, P.D. Rohde, E. Iglesias and A. Sakor, Challenges for healthcare data analytics over knowledge graphs, *Trans. Large Scale Data Knowl. Centered Syst.* **54** (2023), 89–118. doi:10.1007/978-3-662-68014-8_4.

[66] S. Wang, J. Yan, Y. Liu, P. Hu, H. Cai and L. Jiang, Parallel construction of knowledge graphs from relational databases, in: *PRICAI 2023: Trends in Artificial Intelligence – 20th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2023*, Jakarta, Indonesia, November 15–19, 2023, F. Liu, A.A. Sadanandan, D.N. Pham, P. Mursanto and D. Lukose, eds, Proceedings, Part I, Vol. 14325, Springer, 2022, pp. 467–479, Lecture Notes in Computer Science. doi:10.1007/978-981-99-7019-3_42.

[67] G. Xiao, L. Ding, B. Cogrel and D. Calvanese, Virtual knowledge graphs: An overview of systems and use cases, *Data Intell.* **1**(3) (2019), 201–223. doi:10.1162/DINT_A_00011.

[68] G. Xiao, D. Hovland, D. Bilidas, M. Rezk, M. Giese and D. Calvanese, Efficient ontology-based data integration with canonical IRIs, in: *The Semantic Web – 15th International Conference, ESWC 2018, Proceedings*, Heraklion, Crete, Greece, June 3–7, 2018, A. Gangemi, R. Navigli, M. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai and M. Alam, eds, Lecture Notes in Computer Science, Vol. 10843, Springer, 2018, pp. 697–713. doi:10.1007/978-3-319-93417-4_45.

[69] G. Xiao, R. Kontchakov, B. Cogrel, D. Calvanese and E. Botoeva, Efficient handling of SPARQL OPTIONAL for OBDA, in: *The Semantic Web – ISWC 2018 – 17th International Semantic Web Conference, Proceedings, Part I*, Monterey, CA, USA, October 8–12, 2018, D. Vrandecic, K. Bontcheva, M.C. Suá, V. Presutti, I. Celino, M. Sabou, L. Kaffee and E. Simperl, eds, Lecture Notes in Computer Science, Vol. 11136, Springer, 2018, pp. 354–373. doi:10.1007/978-3-030-00671-6_21.