

Empowering the SDM-RDFizer Tool for Scaling Up to Complex Knowledge Graph Creation Pipelines ¹

Enrique Iglesias ^{a,*}, Maria-Esther Vidal ^{a,c,b}, Samaneh Jozashoori ^{b,c}, Diego Collarana ^d and David Chaves-Fraga ^{e,f,g}

^a *L3S Research Center Germany, Hannover Germany*
E-mail: iglesias@l3s.de

^b *TIB Leibniz Information Centre for Science and Technology, Hannover Germany*
E-mails: maria.vidal@tib.eu, samaneh.jozashoori@tib.eu

^c *Leibniz University, Hannover, Germany*

^d *University of Bonn, Fraunhofer IAIS, Germany*
E-mail: diego.collarana.vargas@iais.fraunhofer.de

^e *Ontology Engineering Group, Universidad Politécnica de Madrid, Spain*
E-mail: david.chaves@upm.es

^f *Declarative Languages and Artificial Intelligence Group, KU Leuven, Belgium*

^g *Flanders Make, DTAI-FET, Belgium*

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 arising from factors such as large data volumes, data heterogeneity, and high duplicate rates. This work addresses these challenges by focusing on scaling up declarative knowledge graph creation specified using the RDF Mapping Language (RML). We propose SDM-RDFizer, a two-fold solution designed to address these complexities. Firstly, we introduce a reordering approach for RML triples maps, prioritizing the evaluation of the most selective maps first to reduce the memory consumption. 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 evaluate the effectiveness of SDM-RDFizer using established benchmarks, which demonstrate its superiority over existing state-of-the-art RML engines, highlighting the tangible benefits of our proposed 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 SDM-RDFizer.

Keywords: Data Integration Systems , Knowledge Graphs, RDF Mapping Languages

1. Introduction

Advancements in data collection devices and methods (e.g., sensors, wearables, and genomic tests) have led to the generation of vast amounts of heterogeneous data, including omics and patient health data. Data are available across

¹E. Iglesias and M-E. Vidal equally contributed and are the first authors of this work

*Corresponding author. E-mail: iglesias@l3s.de.

different organizations and companies like hospitals, universities, and pharmaceuticals. However, data silos often hinder the combination, analysis, and reuse of this valuable data, preventing the discovery of insights crucial for decision-making. Knowledge graphs (KGs) have gained significant traction in both industrial and academic sectors to address this challenge [1]. KGs provide a unified representation of heterogeneous data, enabling the convergence of data and their meaning. KGs can be defined as data integration systems (DISs) comprising 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; they also enable users to trace the process of KG creation, enhancing transparency and maintenance. Thus, KGs represent expressive data structures to model integrated data and metadata, and their declarative specifications can be explored, validated, and traced.

The Semantic Web community has actively contributed to the problem of integrating heterogeneous datasets into KGs, as well as to provide methodologies, formalisms, and engines to facilitate their creation and maintenance [2–4]. Declarative mapping languages (e.g., R2RML [5], RML [6], and SPARQL-Generate [7]) allow knowledge engineers to define DISs whose evaluations generate KGs expressed in RDF². DISs enable the definition of concepts in the unified schema (e.g., classes, properties, and attributes) based on data collected from diverse sources in various formats (e.g., tabular, CSV, JSON, or RDF) and improve the flexibility and efficiency of KG creation processes, enabling organizations to unlock valuable insights and drive informed decision-making. Furthermore, the community has defined various engines to execute mapping rules [8–11]. Moreover, many engines are available such as Morph-KGC [12], RMLMapper³, RocketRML [13], and SDM-RDFizer v3.2 [14]. 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 objective is to address the challenge of creating Knowledge Graphs (KGs) specified using RML. We present novel data management techniques implemented in the latest version of SDM-RDFizer (named SDM-RDFizer v4.5.6). These enhancements in SDM-RDFizer allow us to effectively scale up to complex KG creation pipelines found in real-world scenarios where the integration of heterogeneous data sources into KGs is essential. Examples of these complex use cases are part of the Knowledge Graph Construction Workshop 2023 Challenge at ESWC 2023⁴ where SDM-RDFizer v4.5.6 was awarded with the Task-specific award as a result of “performing amazingly for empty value and duplicate removal in execution time and resource usage”. Additionally, pipelines for KG creation in various domains (biomedicine [16–18] or energy [19]) are exemplary complex use cases where the declarative definition of KGs facilitate the integration of large volumes of heterogeneous data. Thus, this paper extends the work reported by Iglesias et al. [14]. The contributions of this paper are as follows:

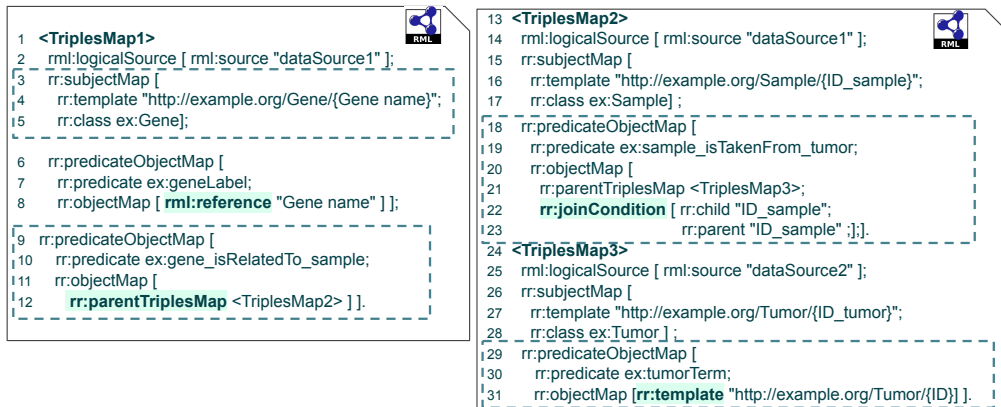
- Data structures for RDF data compression and planning the mapping rule execution.
- Physical operators that leverage these data structures to efficiently handle complex RML mappings, including multiple joins and large data sources with high duplicate rates.
- A new version of the SDM-RDFizer tool (named 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 [20] and SDM-Genomic-Datasets⁵. 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.

²<https://www.w3.org/RDF/>

³<https://github.com/RMLio/rmlmapper-java>

⁴<https://zenodo.org/record/7689310>

⁵<https://figshare.com/articles/dataset/SDM-Genomic-Datasets/14838342>

Fig. 1. **Example.** Main concepts of the RDF Mapping Language (RML).

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 to understand the problem of KG creation. First, some basic concepts related to knowledge graphs, data integration systems, KG creation pipelines, and RML (RDF Mapping Language) are defined. Next, we will 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. Finally, we will illustrate the problem of KG creation within the context of a use case derived from the data integration challenges faced by the EU H2020 funded iASiS project⁶.

2.1. Preliminaries

Knowledge graphs (KG) are directed edge-labeled graphs that represent statements as entities and their relationships as labeled edges (Gutierrez et al. [21]). The creation of a KG, denoted as \mathcal{G} , can be defined as a data integration system $DIS_{\mathcal{G}} = \langle O, S, M \rangle$. Here, O represents a unified ontology consisting of classes and properties, S denotes a set of data sources, and M corresponds to mapping rules or assertions that are formulated as conjunctive queries over the sources in S . The execution of these mapping rules M over the data sources in S generates the instances in \mathcal{G} . The rules in M (Namici et al. [22]) can be represented as Horn clauses, such as $body(\bar{X}) : \neg head(\bar{Y})$, following the Global As View (GAV) approach [23].

A KG creation pipeline encompasses the steps or plan required to evaluate all the mapping rules in M and generate a KG. According to Iglesias et al. [24], there is a vast space of potential plans consisting of mapping rules, and their costs, such as execution time and memory consumption, can vary significantly. Therefore, the KG creation engine must be equipped with data management methods to identify efficient pipelines that can be executed effectively.

The RDF Mapping Language (RML) [25] allows specifying declarative mapping rules to transform into an RDF graph collected from heterogeneous formats, e.g., JSON, CSV, and XML. As the W3C-standard R2RML, TriplesMap corresponds to mapping rules where the resources ($rr:subjectMap$) of an RDF class and their properties ($rr:predicateObjectMap$) are assigned to values ($rr:objectMap$) based on logical data

⁶<https://cordis.europa.eu/project/id/727658>

sources (`rml:logicalSource`). A `rr:objectMap` can be also defined as a reference or a join with the `rr:subjectMap` in another TriplesMap 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. Figure 1 depicts the specification of three triples maps (lines 1, 13, and 24). A *Triples Map* defines a group of mapping rules that define 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. Declaratively Specifying 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 [26]:

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 $A_{k,1}, \dots, A_{k,p}$ from S_k that compose the Internationalized Resource Identifier (IRI) of the entities of C_j . Figure 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 Figure 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 $B_{k,1}, \dots, B_{k,q}$ 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 $B_{k,1}, \dots, B_{k,q}$ from S_k that compose the Internationalized Resource Identifier (IRI) of the object values of op . Lines 29-31 in Figure 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 Figure 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

The declarative specification of a KG \mathcal{G} as a data integration system $DIS_{\mathcal{G}} = \langle O, S, M \rangle$ promotes modularity by allowing individual definition, maintenance, testing, and debugging of concepts in O . Moreover, mapping rules can be reused and collaboratively defined. Additionally, the RDF knowledge graph, composed of the ontology O and the RML triples maps in M , serves as a human- and machine-readable documentation, providing provenance information for each class and property in a KG \mathcal{G} .

To create a KG, the execution of mapping rules in M involves exploring numerous possible pipelines for KG creation ([24]). Furthermore, Chaves-Fraga et al. [15] have established that the efficiency of declaratively specified

KG creation as a data integration system is influenced by various parameters. 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. Additionally, These parameters encompass the complexity of the mapping rules (including shape, number of joins between triples maps, and selectivity of the joins).

Based on these findings, we can identify the requirements that an RML engine must meet to execute KG creation pipelines efficiently. These requirements (REs) are as follows:

- **RE1-Execution of Complex Triples Maps:** An RML engine should be capable of producing the RDF KG, regardless of the type or complexity of the triples maps involved.
- **RE2-Scalable Integration of Large Data Sources:** An RML engine should be able to handle the execution of KG creation pipelines, even when dealing with large data sources.
- **RE3-Efficient Management of Duplicates:** An RML engine should be capable of detecting duplicated data within the data sources of a KG creation pipeline and prevent the generation of duplicate RDF triples.
- **RE4-Performance Agnostic of the Data Source Format:** The performance of an RML engine should not be affected by the format of the data sources used in the KG creation pipeline.

2.4. A Use Case for Knowledge Graph Creation

This section illustrates the problem of defining a knowledge graph (KG) using a declarative approach similar to a data integration system (DIS), where mapping rules are specified in the RDF Mapping Language (RML). In this example, we focus on a portion of the DIS that defines the KG of the EU H2020 project iASiS⁶. The data sources, referred to as SDM-Genomic-Datasets [14] vary in size, with 100k, 1M, and 5M rows, and each dataset has a different percentage of data duplicate rate (25% or 75%). The iASiS ontology⁷ consisting of classes like `iasis:Mutation`, `iasis:Gene`, `iasis:Sample`, `iasis:GenomePosition`, `iasis:Tumor`, and `iasis:SomanticStatus`, is used to define the KG. The ontology and its properties are defined in six RML triples maps, as presented in Figure 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.

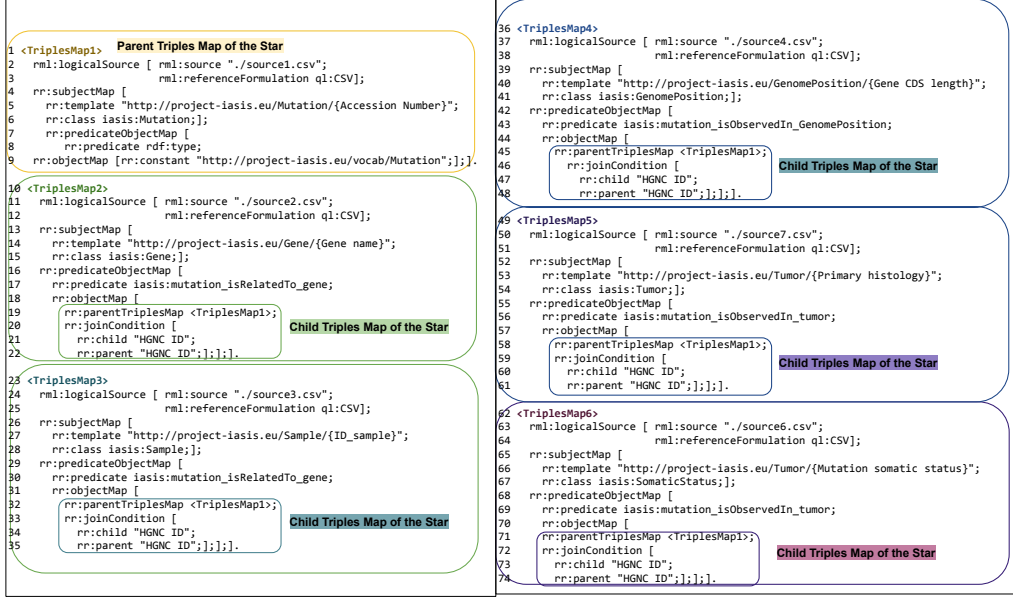
Three state-of-the-art RML engines, i.e., RMLMapper v6.0⁸, Morph-KGC v2.1.1⁹, and SDM-RDFizer v3.2¹⁹, are utilized to create this portion of the KG; the engines timed out in five hours. The results of executing these engines in six testbeds are shown in Figure 2. The execution time was significantly 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 good performance compared to RMLMapper and Morph-KGC, it still required a considerable amount of time to create the KG. In this paper, we present a new version of SDM-RDFizer (v4.5.6¹⁰) 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.

⁷<https://github.com/SDM-TIB/iASiSOntology>

⁸<https://github.com/RMLio/rmlmapper-java/releases/tag/v6.0.0>

⁹<https://doi.org/10.5281/zenodo.6524684>

¹⁰<https://doi.org/10.5281/zenodo.7027549>



(a) A Star Triples 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.

3. Related Work

This section summarizes the key contributions from the existing literature regarding creating RDF KGs. First, 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 [2].

3.1. Knowledge Graph Creation and Existing Technologies

The creation of a knowledge graph (KG) \mathcal{G} can be defined as a data integration system [23], 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 [27]. The connection between these input sources and the ontology is often established through mapping rules, such as the W3C recommendation R2RML [5] for relational databases or its main extension for heterogeneous data formats (e.g., CSV, XML, and JSON) called RML [6]. However, there are other solutions that adapt different languages (e.g., SPARQL) for integration purposes, such as SPARQL-Anything [28] and SPARQL-Generate [7]. The works by Hofer et al.[29] and Van Assche et al.[2] 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.[30] 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.

3.2. Virtual Data Integration in Knowledge Graphs

The creation of a virtual knowledge graph (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 [31]. 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 [32, 33]. Several solutions have been proposed, with a focus on translating SPARQL queries into SQL queries. These solutions include Ontop [9], Ultrawrap [34], Morph [11], Squerall [35], Ontario [36], and Morph-CSV [10]. 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. [37] demonstrates that it can be easily extended to efficiently create a KG by rewriting an input query based on RML mapping rules.

3.3. Materialized Knowledge Graph Creation

A Knowledge Graph (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 RMLMapper³ is an in-memory RML compliant engine designed to address source heterogeneity during KG creation. RMLMapper is capable of executing RML mapping rules 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 [38]. Morph-KGC [12] is a tool that processes R2RML, RML, and RML-star [39] 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 [14] 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. [40] present data management methods that resort to the translation of RML mapping rules into SPARQL queries over the Apache Spark Big Data framework to scale up the process of KG creation. RMLStreamer [41] also prioritizes scalability, and efficiently generates RDF triples continuously

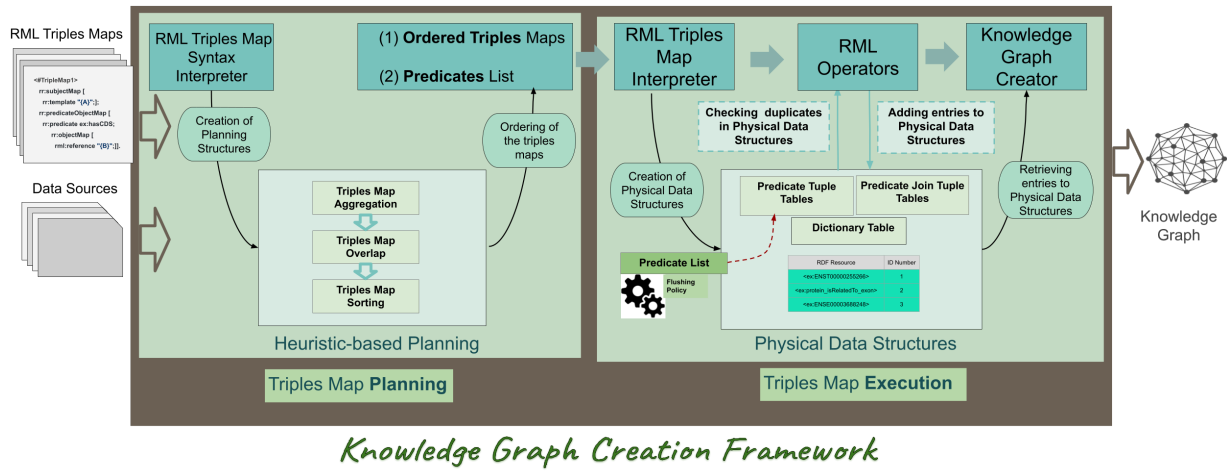


Fig. 3. **The SDM-RDFizer Architecture.** SDM-RDFizer receives as an input RML triples maps with its corresponding data sources. The Triples Map Planning component plans and orders the execution of the triples maps. The Triples Map Execution component resorts to physical operators, data structures, and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

while eliminating duplicates at the end. Other RML-compliant engines, such as RocketRML [13], focus on heterogeneity rather than scalability. Meanwhile, tools like Chimera [42] and CARML¹¹ 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 2023⁴, participants such as CARML, SANSAs (Stadler et al. [40]), RMLStreamer, and SDM-RDFizer showcased their capabilities. The results demonstrated that there is no single engine capable of outperforming 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, SANSAs 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 subsection 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.

¹¹<https://github.com/carml/carml>

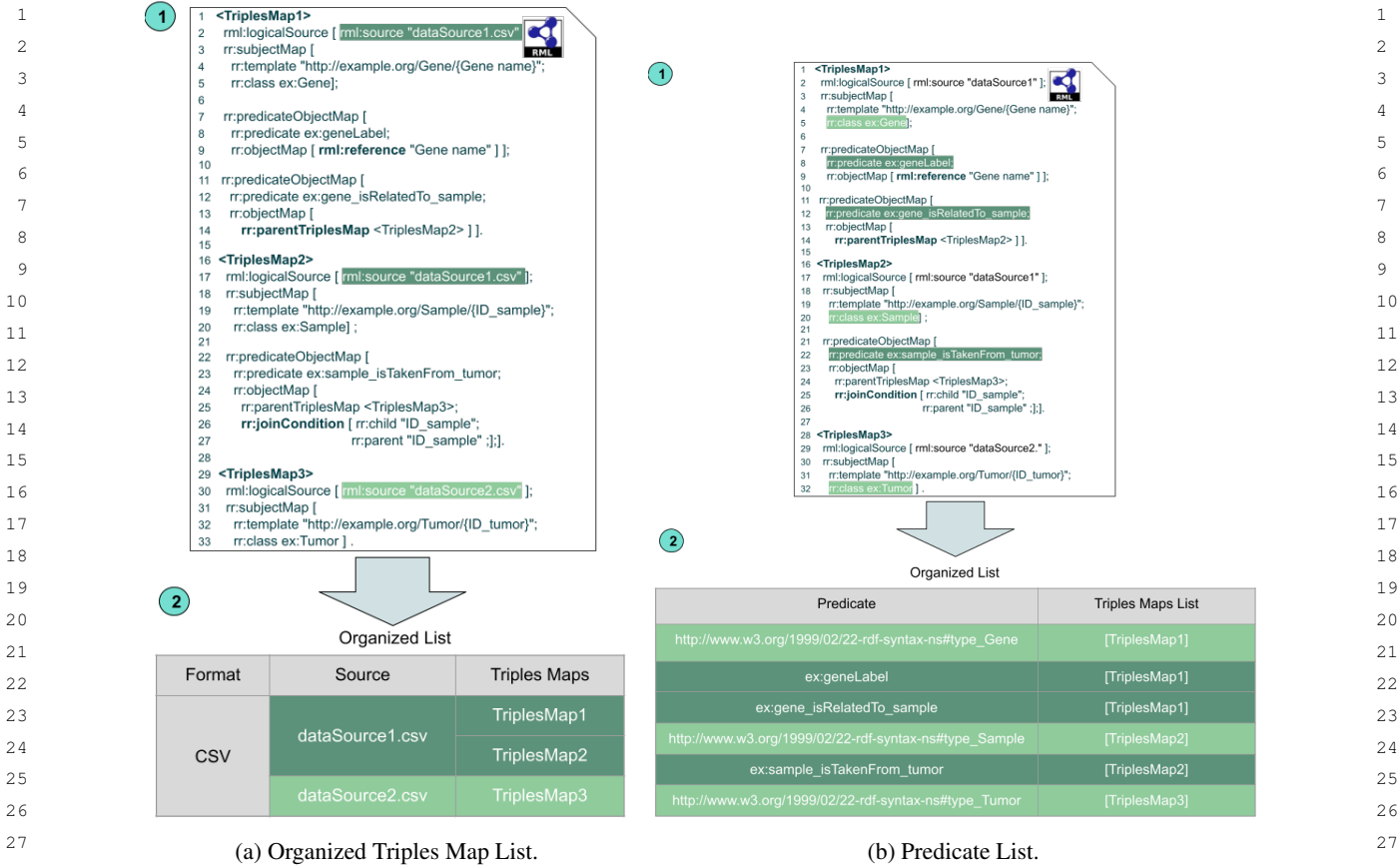


Fig. 4. **The Data Structures for the Planning Phase.** The Organized Triples Map List groups Triples Maps (TMs) first by data source and then by data source format, thus reducing the need to open any particular data source more than once. Predicate List groups the TMs by predicate. Each time a TM is 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.

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.

Figure 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 **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 TMs should be executed. OTML groups the TMs by data source, while PL groups the TMs by predicate and orders them. Triples maps are sorted to determine which of them define the same predicates.

Organized Triples Maps List (OTML) groups TMs by their data source; see Figure 4a. OTML is only used with TMs 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 TM using SQL queries.

During the Triples Map Planning phase, TMs 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 TMs. Implementing this data structure into the SDM-RDFizer causes the processor to adopt a hybrid approach. A data-driven approach is used for TMs with file data sources, while a mapping-driven approach is used for TMs with relational databases. Figure 4a depicts the OTML for the TMs in Figure 1. Since all these TMs are over CSV files, only one group is created.

Predicate List (PL) groups TMs by their predicates by creating a list of TMs associated with a particular predicate; see Figure 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 TMs require the corresponding predicate, and the associated PTT is safe to be flushed. We add the range of the predicate to the predicate in the case of generic predicates, such as `rdf:type`. For example, as it can be seen in Figure 4b in 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 is also utilized to organize OTML, determined by which TMs have overlapping predicates. By default, the order of execution considers the TMs with the least overlap first and the one with the most overlap last. Following each TM is the TM that overlaps with it so that when a TM is finished executing, the most memory is released, and if any PTT remains, they will be flushed when the following TM 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 maps. 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 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 (requirement **RE4**).

The 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

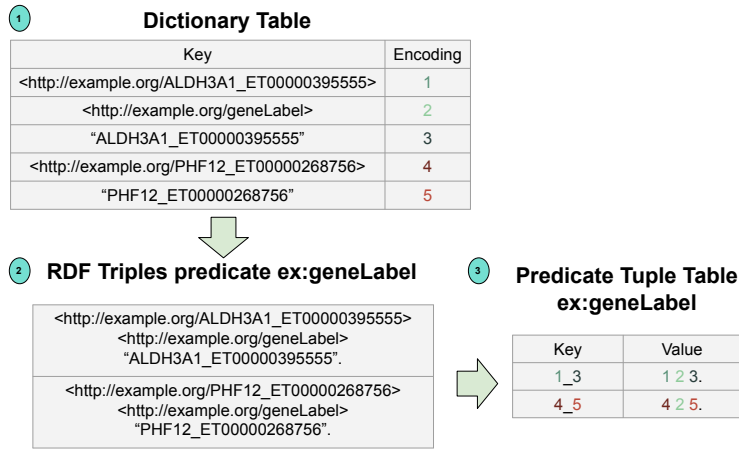


Fig. 5. **Dictionary Table and Predicate Triple Table.** Each generated RDF resource or literal is encoded in terms of an identification number in base 36. The Dictionary Table stores this encoding. RDF triples generated for a predicate, e.g., `ex:geneLabel`, are stored into its corresponding Predicate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

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. Figure 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 . Figure 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 Figure 5, the identification number stored in the Dictionary Table 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 knowledge graph. Whenever an RDF triple is generated during the execution of a TM, 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. Thus, the Dictionary Table and Predicate Triple Tables empower SDM-RDFizer to satisfy the requirements **RE2** and **RE3**.

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. The Dictionary Table is used to retrieve the encodings of the RDF resources or literals. Thus, a PJTT minimizes the space to store intermediate results. Figure 6 illustrates the PJTT for the join condition between triples maps `TriplesMap2` and `TriplesMap3` in Figure 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 which corresponds to the most efficient implementation of a join [43]. As a result, the Predicate Join Triple Tables provide the basis for the implementation of the physical operators which allow SDM-RDFizer to satisfy the requirements **RE1** and **RE2**.

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 TM and its respective data source, SOM generates the

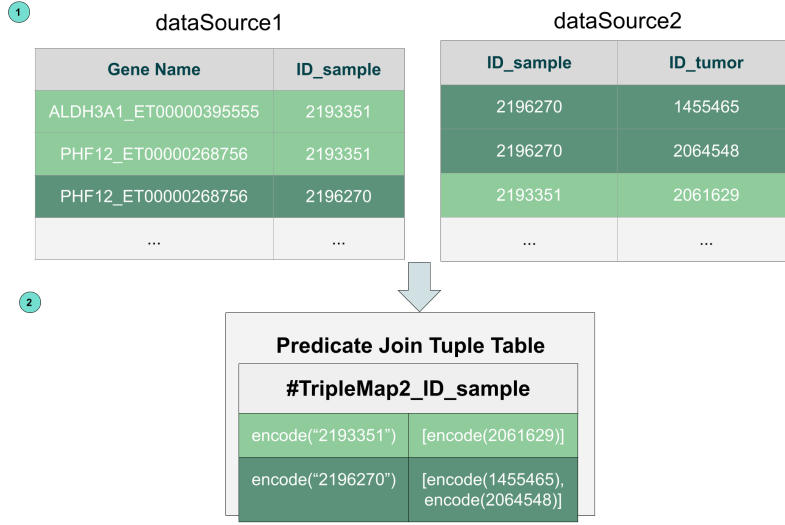


Fig. 6. **Predicate Join Tuple Table.** The PJTT for the join between triples maps `TriplesMap2` and `TriplesMap3` in Figure 1 (lines 21-23). The encoding stored in the Dictionary Table is used to minimize the space to store intermediate results.

RDF triples. The encoding of each generated RDF triple is checked against the corresponding predicate tuple table (PTT). If the RDF triple already exists in PTT, it is discarded. If not, it is added both to PTT and to 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: $row \in S$

- 1: Create RDF triple $t = (f1(row.att1), p, f2(row.att1))$
- 2: Add $f1(row.att1)$ and $f2(row.att1)$ to the Dictionary Table

- 3: **if** $encode(f1(row.att1), f2(row.att1))$ does not belong to the PTT for p **then**

- 4: Add $encode(f1(row.att1), f2(row.att1))$ to PTT for p

- 5: Add $(f1(row.att1), p, f2(row.att1))$ to the KG

- 6: **end if**

- 7: **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 TM. Thus, the same process as in SOM is applied to the generated RDF triples, i.e., the encoding of the triples are checked against PTT to determine if they will be added to the KG. Algorithm 2 illustrates the ORM implementation.

Object Join Map (OJM) implements an index join in executing a join condition between two TMs 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 2 Object Reference Map (ORM)

```

1 Input: Triples Map  $tm1$  refers to Triples Map  $tm2$  to define predicate  $p$  on logical source  $S$ 
2 and  $tm1$  subjectMap is  $f1(att1)$  and  $tm2$ 
3 subjectMap is  $f2(att2)$ 
4 Output: RDF Triples for  $p$  generated from  $tm1$ 
5 for each:  $row \in S$ 
6   1: Create RDF triple  $t = (f1(row.att1), p, f2(row.att1))$ 
7   2: Add  $f1(row.att1)$  and  $f2(row.att1)$  to the Dic-
8       tionary Table
9   3: if  $encode(f1(row.att1), f2(row.att1))$  does not
10      belong to the PTT for  $p$  then
11     4: Add  $encode(f1(row.att1), f2(row.att1))$  to
12        PTT for  $p$ 
13     5: Add  $(f1(row.att1), p, f2(row.att1))$  to the KG
14     6: end if
15     7: return  $KG$ 

```

Algorithm 3 Object Join Map (OJM)

```

16 Input: Triples Map  $tm1$  refers to Triples Map  $tm2$ 
17 to define predicate  $p$  on logical sources  $S1$  and
18  $S2$  and join condition  $B$  on attributes  $S1Att$  and
19  $S2Att$ , respectively, and  $tm1$  subjectMap is
20  $f1(att1)$  and  $tm2$  subjectMap is  $f2(att2)$ 
21 Output: RDF Triples for  $p$  generated from  $tm1$ 
22   1: for  $row1 \in S1$  do
23     2: if  $encode(row1.S1Att)$  belongs to the PJTT for
24         $tm2$  then
25       3: for  $v \in valueSet(encode(row1.S1Att))$  in
26          PJTT for  $tm2$  do
27         4: Create  $t = (f1(row1.att1), p, f2(decode(v)))$ 
28         5: Add  $f1(row1.att1)$  and  $f2(decode(v))$  to
29            the Dictionary Table
30         6: if  $encode(f1(row1.att1), f2(decode(v)))$ 
31            does not belong to the PTT for  $p$  then
32           7: Add  $encode(f1(row1.att1), f2(decode(v)))$ 
33              to PTT for  $p$ 
34           8: Add  $(f1(row1.att1), p, f2(decode(v)))$ 
35              to the KG
36           9: end if
37          10: end for
38         11: end if
39         12: end for
40         13: return  $KG$ 

```

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, TM type and complexity, selectivity of the results, and types of join between the TMs. In total, we consider three different mapping types: Simple Object Map (SOM), Object Reference Map (ORM), and Object Join Map (OJM). All re-

sources and experimental settings used in this evaluation are publicly available¹² and Table 1 summarizes the used configurations.

GTFS-Madrid-Bench [20]: 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 mapping rules comprised of 13 TMs with 73 SOMs, and 12 OJMs involving ten data sources.

SDM-Genomic-Datasets [14]: This benchmark is created by randomly sampling data records from somatic mutation data collected in COSMIC¹³. 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 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. We group the aforementioned TM 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 frameworks¹⁴. **Conf7** aims at evaluating the impact of defining the same predicates using different TMs. It has a set of four TMs with two OJMs. For each pair of TMs, there is an OJM. The data sources of one pair of the TMs are a subset of the other pair. Both pairs of TMs share the same predicate. **Conf8** provides a TM that is connected to five other TMs with different logical sources through join, i.e., this TM is connected via a five-star join with the other TMs. It comprises a set of six TMs with five OJMs; five child TMs refer to the same parent TM. **Conf9** combines the first two configurations in one testbed; it is composed of a set of ten TMs with seven OJMs.

State-of-the-art RML Engines and Metrics.

The following RML tools are included in the empirical study. RMLMapper v6.0¹⁵, RocketRML v1.11.3 [13]¹⁶, Morph-KGC v2.1.1 [12]¹⁷, and SDM-RDFizer v4.5.6¹⁸; it implements all the techniques described in this paper, i.e., planning techniques, physical operators, and data compression techniques to reduce the size of the main memory structures required to store intermediate results that were generated. The SDM-RDFizer v3.2¹⁹ is also used to determine if there is an increase in performance between the older and newer version.

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 an Intel(R) Xeon(R) equipped with a CPU E5-2603 v3 @ 1.60GHz 20 cores, 64GB 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 that 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 the SDM-RDFizer (i.e.,

¹²<https://github.com/SDM-TIB/SDM-RDFizer-Experiments/tree/master/swj2022>

¹³<https://cancer.sanger.ac.uk/cosmic/GRCh37>, version90, released August 2019

¹⁴<https://doi.org/10.6084/m9.figshare.17142371>

¹⁵<https://github.com/RMLio/rmlmapper-java>

¹⁶<https://github.com/semantifyit/RocketRML/>

¹⁷<https://github.com/oeg-upm/Morph-KGC>

¹⁸<https://github.com/SDM-TIB/SDM-RDFizer>

¹⁹<https://doi.org/10.5281/zenodo.3872104>

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 MAs refer to the same parent MA.
Conf9	Set of ten TMs with seven OJMs. Combines Conf7+Conf8	

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).

SDM-RDFizer v3.2) only contains the operators for TM 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 Figure 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 (Figure 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 mapping rules contain 13 TMs, 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 the SDM-RDFizer used. It can be seen in Figure 7a that the SDM-RDFizer v3.2 is the one that consumes the most memory. By applying data

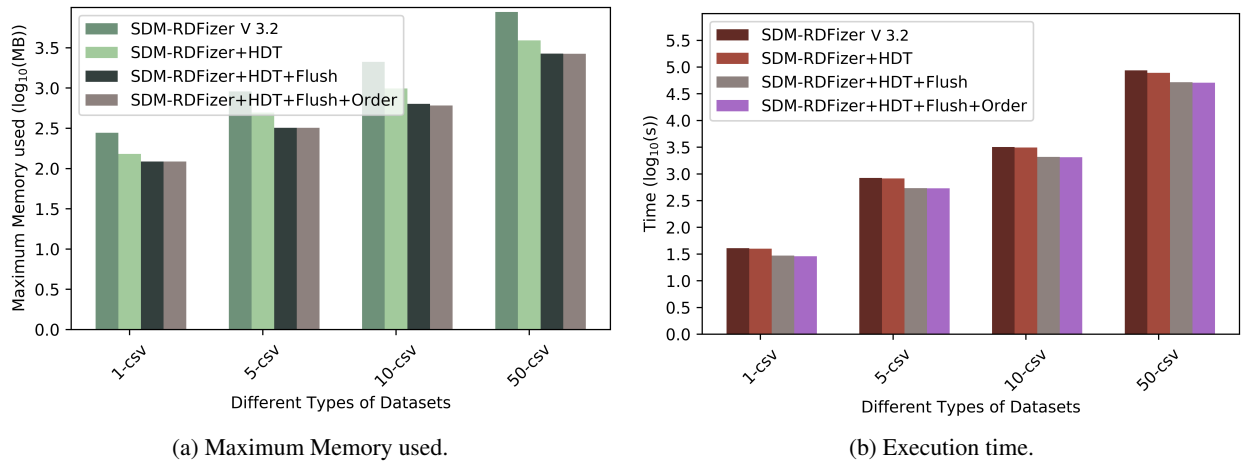
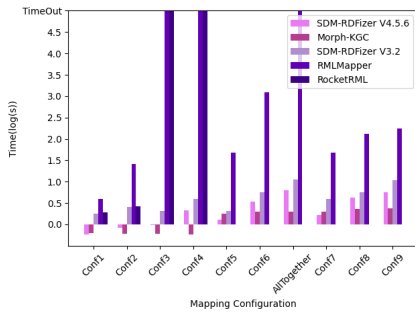


Fig. 7. Results of the execution of the GTFS-Madrid-Bench benchmark. Execution time and memory consumption of various version of the SDM-RDFizer when transforming the GTFS-Madrid-Bench benchmark.

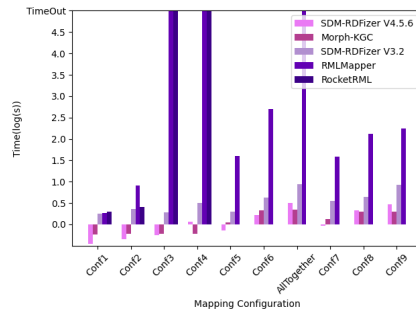
compression, there is a significant reduction in memory consumption caused by the data stored in PTT for duplicate removal is much smaller than the data stored in the SDM-RDFizer v3.2. Flushing unneeded data reduces the maximum memory used even further, but not as much as with data compression. Finally, 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 TMs are executed. The benefit of executing the TMs in a predetermined order is that the maximum amount of data is flushed after finishing the execution of a TM, thus minimizing the amount of memory being used.

5.3. Performance of the RML Engines in the SDM-Genomic-Datasets

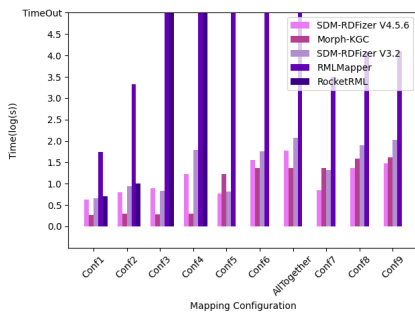
This experiment seeks to prove the impact of using real-world data for KG creation. Even though the TMs defined for SDM-Genomic-Datasets are simpler than those defined for GTFS-Madrid-Bench, they cover all the TM types defined in Figure 1. We evaluate the performance of each engine, i.e., SDM-RDFizer 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 Figure 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 best performance, since they partition each data source into chunks and then apply the TM to multiple rows simultaneously. In the cases with ORMs (i.e., **Conf3**, **Conf4**, and **AllTogether**), Morph-KGC also presents the best performance; 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 duplicates 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 Figure 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, Figure 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., Figure 8a and Figure 8b), the SDM-RDFizer v4.5.6 presents the best performance among the tested engines. Note that SDM-RDFizer v3.2 can also execute all the TMs; however, SDM-RDFizer v4.5.6 is more efficient. The PJTT data structure allowed the TMs to be executed as efficiently as possible. PJTT performs the join between the parent and child data source and stores the results



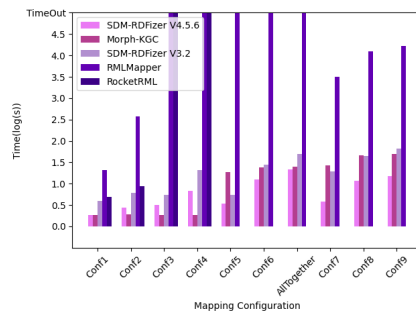
(a) 10k records with 25% duplicate rate.



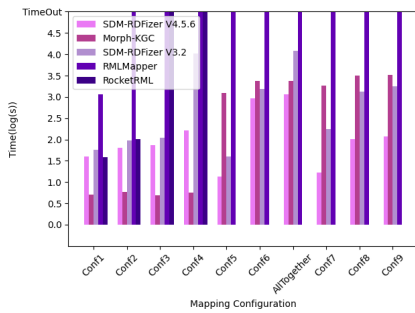
(b) 10k records with 75% duplicate rate.



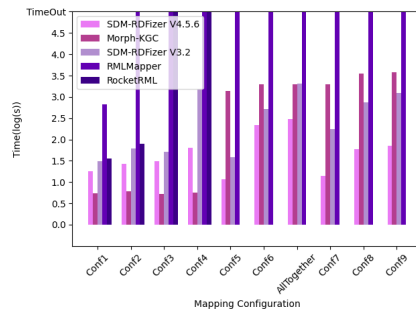
(c) 100k records with 25% duplicate rate.



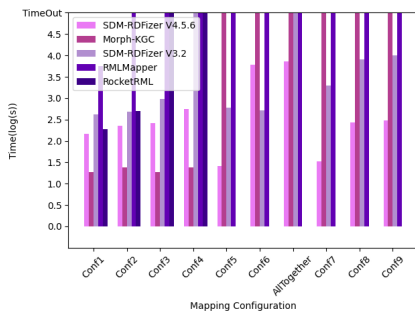
(d) 100k records with 75% duplicate rate.



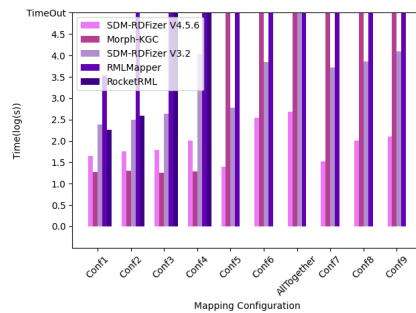
(e) 1M records with 25% duplicate rate.



(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.

in main memory, thus avoiding uploading the parent data source multiple times. The SDM-RDFizer v3.2 is also capable of executing these TMs 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., Figure 8a, Figure 8b, Figure 8c, and Figure 8d). Still, when dealing with larger data sources (i.e., Figure 8e, Figure 8f, Figure 8g, and Figure 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.

6. The SDM-RDFizer Characteristics

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 own projects and use cases.

Novelty: SDM-RDFizer is the first 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 **RE2** and **RE4**). On the other hand, the encoding approach and the physical data structures with their corresponding operators guarantee the fulfillment of the requirements **RE1** and **RE3**. To the best of 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 repository¹⁸ under an Apache 2.0 license so that it can be extended and modified by any developer. The GitHub repository is also linked to the Zenodo platform, which provides a Digital Object Identifier (DOI) for the general repository²⁰ and also a DOI for each specific software release²¹. The engine is also available on the Python Package Index (PyPI), so it can be easily installed and integrated in other developments²². Finally, we also provide a docker image that deploys the SDM-RDFizer as a web service.

Utility: As demonstrated in our experimental evaluation, the SDM-RDFizer is one of the best RML engines 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 providing a useful tool for different and heterogeneous use cases. In addition, SDM-RDFizer passed all proposed RML test-cases [44]²³, which means that our engine is fully compliant with the RML specification.

Adoption: The SDM-RDFizer has been successfully utilized in various industrial and research projects to create knowledge graphs from heterogeneous data sources. The following list highlights a selection of these projects, demonstrating the versatility and practical application of the SDM-RDFizer in diverse domains: **a)** iASiS²⁴, EU H2020 funded project to exploit patient data insights towards precision medicine. The iASiS RDF knowledge graph comprises more than 1.2B RDF triples collected from more than 40 heterogeneous sources using over 1,300 RML TMs [45]. **b)** Lung Cancer Pilot of BigMedilytics²⁵, where the KG is defined in terms of 800 RML TMs from around 25 data sources; it comprises around 500M RDF triples. **c)** In CLARIFY²⁶, the KG integrates data from lung and breast cancer patients. It comprises 76M RDF triples and 16M RDF resources. 626 RML TMs define the CLARIFY KG. **d)** P4-LUCAT²⁷ has 676 RML TMs that define the P4-LUCAT KG in terms of a unified schema of

²⁰The DOI for the SDM-RDFizer repository is: <https://doi.org/10.5281/zenodo.3872103>

²¹For example, the DOI for the v4.5.6 used in the experiments of this paper is: <https://doi.org/10.5281/zenodo.7027549>

²²<https://pypi.org/project/rdfizer/>

²³<https://rml.io/implementation-report>

²⁴<http://project-iasis.eu/>

²⁵<https://www.bigmedilytics.eu/>

²⁶<https://www.clarify2020.eu>

²⁷<https://p4-lucacat.eu/>

318 attributes and 177 classes; it comprises 178M of RDF triples. **e)** The ImProVIT KG²⁸ integrates immune system data into a unified schema of 176 classes, 66 predicates, and 151 attributes. **f)** PLATOON²⁹ project creates the KG for a pilot [19] that is defined in terms of 2,093 RML mapping rules; it comprises 220M RDF triples and 80M of RDF resources. **g)** The Knowledge4COVID-19 KG [17] 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. **h)** H2020 - SPRINT³⁰ studies performance and scalability of different semantic architecture for the Interoperability Framework on Transport across Europe. **i)** EIT-SNAP³¹ innovation project on the application of semantic technologies for transport national access points, and SDM-RDFizer allowed the integration of transportation data in Spain. **j)** Open Cities³² 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. **k)** Virtual Platform for the H2020 European Joint Programme on Rare Disease³³, and SDM-RDFizer merges data collected from the consortium partners. **l)** CoyPU³⁴ is a German-funded project where SDM-RDFizer generates KGs for various types of 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 CoyPU KG³⁵.

Impact: Since its first release, the SDM-RDFizer has caught the attention of practitioners and knowledge engineers due to its good results w.r.t. other RML engines. The number of commits in the GitHub repository¹⁸ 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 95 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 needed to be integrated. These use cases have demanded the fulfillment of specific requirements, and SDM-RDFizer has effectively contributed:

- *Scalability of large energy-related data:* In the context of the PLATOON project²⁹, 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 [19]. 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) data³⁶.
- *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. [16] and Torrente et al. [46]).

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

²⁸<https://www.tib.eu/en/research-development/project-overview/project-summary/improvit>

²⁹<https://platoon-project.eu/>

³⁰<http://sprint-transport.eu/>

³¹<https://www.snap-project.eu/>

³²<https://ciudades-abiertas.es/>

³³<https://www.ejprarediseases.org>

³⁴<https://coypu.org/>

³⁵<https://www.youtube.com/watch?v=CciNaaMyi8A>

³⁶A dedicated Knowledge Graph for integration of RES data sources <https://innovation-radar.ec.europa.eu/innovation/43139>

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 GitHub³⁷ and extending the current data structures and physical operators is part of our future tasks. Lastly, our ambition is to enable the evaluation of observational data, such as sensor data, within the SDM-RDFizer framework.

References

- [1] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, S. Kirrane, J.E.L. Gayo, R. Navigli, S. Neumaier, A.N. Ngomo, A. Polleres, S.M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab and A. Zimmermann, Knowledge Graphs (2021). doi:10.2200/S01125ED1V01Y202109DSK022.
- [2] 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.
- [3] M. Kejriwal, J.F. Sequeda and V. Lopez, Knowledge graphs: Construction, management and querying, *Semantic Web* **10**(6) (2019), 961–962. doi:10.3233/SW-190370.
- [4] 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.
- [5] 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/>.
- [6] 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)*, CEUR Workshop Proceedings, Vol. 1593, CEUR-WS.org, 2016. <https://ceur-ws.org/Vol-1593/article-04.pdf>.
- [7] M. Lefrançois, A. Zimmermann and N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, in: *European Semantic Web Conference*, Springer, 2017, pp. 35–50.
- [8] 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.
- [9] 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.
- [10] D. Chaves-Fraga, E. Ruckhaus, F. Priyatna, M. Vidal and Ó. Corcho, Enhancing virtual ontology based access over tabular data with Morph-CSV, *Semantic Web* **12**(6) (2021), 869–902. doi:10.3233/SW-210432.
- [11] F. Priyatna, Ó. 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*, ACM, 2014, pp. 479–490. doi:10.1145/2566486.2567981.
- [12] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M.S. Pérez and O. Corcho, Morph-KGC: Scalable knowledge graph materialization with mapping partitions, *Semantic Web* (2022). doi:10.3233/SW-223135.
- [13] U. Şimşek, E. Kärle and D. Fensel, RocketRML-A NodeJS implementation of a use-case specific RML mapper, in: *Proceeding of the First International Workshop on Knowledge Graph Building*, 2019.
- [14] 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, 2020. <https://arxiv.org/abs/2008.07176>.
- [15] D. Chaves-Fraga, K.M. Endris, E. Iglesias, Ó. 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, Proceedings*, Lecture Notes in Computer Science, Vol. 11877, Springer, 2019, pp. 695–713. doi:10.1007/978-3-030-33246-4_43.

³⁷<https://github.com/SDM-TIB/SDM-RDFizer-Star>

- [16] F. Aisopos, S. Jozashoori, E. Niazmand, D. Purohit, A. Rivas, A. Sakor, E. Iglesias, D. Vogiatzis, E. Menasalvas, A.R. González, G. Viguera, D. Gómez-Bravo, M. Torrente, R.H. López, M.P. Pulla, A. Dalianis, A. Triantafyllou, 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.
- [17] 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.
- [18] B. Steenwinckel, G. Vandewiele, I. Rausch, P. Heyvaert, R. Taelman, P. Colpaert, P. Simoens, A. Dimou, F.D. Turck and F. Ongenaes, Facilitating the Analysis of COVID-19 Literature Through a Knowledge Graph, in: *The Semantic Web - ISWC 2020*, 2020, pp. 344–357. doi:10.1007/978-3-030-62466-8_22.
- [19] V. Janev, M.-E. Vidal, D. Pujić, D. Popadić, E. Iglesias, A. Sakor and A. Čampa, Responsible Knowledge Management in Energy Data Ecosystems, *Energies* **15**(11) (2022). doi:10.3390/en15113973.
- [20] D. Chaves-Fraga, F. Priyatna, A. Cimmino, J. Toledo, E. Ruckhaus and Ó. 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.
- [21] C. Gutiérrez and J.F. Sequeda, Knowledge graphs, *Communications of the ACM* **64**(3) (2021), 96–104.
- [22] 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, CEUR Workshop Proceedings, Vol. 2211, CEUR-WS.org, 2018. <https://ceur-ws.org/Vol-2211/paper-25.pdf>.
- [23] M. Lenzerini, Data Integration: A Theoretical Perspective, in: *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, ACM, 2002, pp. 233–246. doi:10.1145/543613.543644.
- [24] 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.
- [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, Vol. 1184, CEUR-WS.org, 2014. https://ceur-ws.org/Vol-1184/dow2014_paper_01.pdf.
- [26] 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.
- [27] A. Poggi, D. Lembo, D. Calvanese, G.D. Giacomo, M. Lenzerini and R. Rosati, Linking Data to Ontologies, 2008, pp. 133–173. doi:10.1007/978-3-540-77688-8_5.
- [28] 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, *Transactions on Internet Technology* (2022), accepted for publication.
- [29] M. Hofer, D. Obraczka, A. Saeedi, H. Köpcke and E. Rahm, Construction of Knowledge Graphs: State and Challenges, *CoRR abs/2302.11509* (2023). doi:10.48550/arXiv.2302.11509.
- [30] 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.
- [31] 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.
- [32] 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, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part I*, Lecture Notes in Computer Science, Vol. 11136, Springer, 2018, pp. 354–373. doi:10.1007/978-3-030-00671-6_21.
- [33] 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, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, Lecture Notes in Computer Science, Vol. 10843, Springer, 2018, pp. 697–713. doi:10.1007/978-3-319-93417-4_45.
- [34] J.F. Sequeda and D.P. Miranker, Ultrawrap: SPARQL execution on relational data, *J. Web Semant.* **22** (2013), 19–39.
- [35] 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: *International Semantic Web Conference*, Springer, 2019, pp. 229–245.
- [36] K.M. Endris, P.D. Rohde, M.-E. Vidal and S. Auer, Ontario: Federated query processing against a semantic data lake, in: *International Conference on Database and Expert Systems Applications*, Springer, 2019, pp. 379–395.
- [37] P.D. Rohde, E. Iglesias and M.-E. Vidal, SHACL-ACL: Access Control with SHACL, in: *ESWC 2023 Poster and Demos*, 2023. https://2023.eswc-conferences.org/wp-content/uploads/2023/05/paper_Rohde_2023_SHACL-ACL.pdf.
- [38] J. Arenas-Guerrero, M. Scrocca, A. Iglesias-Molina, J. Toledo, L. Pozo-Gilo, D. Doña, Ó. 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, CEUR Workshop Proceedings, Vol. 2873, CEUR-WS.org, 2021. <https://ceur-ws.org/Vol-2873/paper11.pdf>.
- [39] T. Delva, J. Arenas-Guerrero, A. Iglesias-Molina, Ó. 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, CEUR Workshop Proceedings, Vol. 2980, CEUR-WS.org, 2021. <https://ceur-ws.org/Vol-2980/paper374.pdf>.

- [40] C. Stadler, L. Bühmann, L.-P. Meyer and M. Martin, Scaling RML and SPARQL-based Knowledge Graph Construction with Apache Spark, in: *Fourth International Workshop On Knowledge Graph Construction Co-located with ESWC 2023*, 2023. <https://kg-construct.github.io/workshop/2023/resources/paper9.pdf>.
- [41] 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, Virtual Event, October 23-27, 2022, Proceedings*, Lecture Notes in Computer Science, Vol. 13489, Springer, 2022, pp. 697–713. doi:10.1007/978-3-031-19433-7_40.
- [42] 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, Athens, Greece, November 2-6, 2020, Proceedings, Part II*, Lecture Notes in Computer Science, Vol. 12507, Springer, 2020, pp. 411–429. doi:10.1007/978-3-030-62466-8_26.
- [43] H. Lei and K.A. Ross, Faster Joins, Self-Joins and Multi-Way Joins Using Join Indices, *Data Knowl. Eng.* **28**(3) (1998), 277–298.
- [44] P. Heyvaert, D. Chaves-Fraga, F. Priyatna, Ó. 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, Villa Clara, Cuba, June 23-30, 2019, Proceedings*, Communications in Computer and Information Science, Vol. 1029, Springer, 2019, pp. 162–173. doi:10.1007/978-3-030-21395-4_12.
- [45] 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.
- [46] M. Torrente, P.A. Sousa, R. Hernández, M. Blanco, V. Calvo, A. Collazo, G.R. Guerreiro, B. Núñez, J. Pimentao, J.C. Sánchez, 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). doi:10.3390/cancers14164041. <https://www.mdpi.com/2072-6694/14/16/4041>.