Results for Knowledge Graph Creation Challenge 2025: SDM-RDFizer

Enrique Iglesias[1,2,*], Maria-Esther Vidal[1,2,3]

[1]L3S Research Center, Hannover, Germany
[2]Leibniz University of Hannover, Hannover, Germany
[3]TIB Leibniz Information Centre for Science and Technology, Hannover, Germany

Abstract

In recent years, knowledge graphs (KGs) have grown in popularity. Major companies have incorporated them into their products to enhance the user experience. Consequently, numerous methods have emerged to address KG construction. The RDF Mapping Language (RML) is an extension of the W3C mapping language recommendation, R2RML. RML allows for the declarative definition of KG structures based on unified ontologies and data sources in various formats, including CSV, JSON, and XML files. However, RML incorporates CSV, JSON, and XML files. Furthermore, RML has additional functionalities, such as executing functions, using quoted triples, working with collections, and creating logical views. Thus, RML has become a standalone mapping language separate from R2RML. The KGCW 2025 Challenge defines a dataset comprising a series of test cases that cover all RML functionalities. These test cases evaluate the compliance of state-of-the-art knowledge graph (KG) creation engines. This paper reports on the conformance evaluation of SDM-RDFizer executing this dataset, highlighting its strengths and areas for improvement to enhance performance.

Keywords

Knowledge Graph Creation, Data Integration System, RDF Mapping Languages

1. Introduction

Knowledge graphs (KGs) have become increasingly commonplace in recent years, driven by the significant growth in daily data generation. Major companies such as Google, Netflix, Amazon, and Microsoft use KGs to create relationships between products, concepts, and themes to enhance the user experience [1]. As a result, multiple methods and mapping languages have been developed to support KG creation. One such mapping language is the RDF Mapping Language (RML) [2]. RML was introduced initially as an extension of R2RML, which focused solely on relational databases. RML expanded support to additional data source formats, including CSV, JSON, and XML. Both RML and R2RML adhere to the rules established by the Resource Description Framework (RDF)[1]. Over time, RML has incorporated additional functionalities such as the definition and execution of value transformation functions (RML+FnO [3]), support

Portorož'25: Sixth International Workshop on Knowledge Graph Construction, June 1, 2025, Portorož, SI
*Corresponding author.
†These authors contributed equally.
iglesias@l3s.de (E. Iglesias); maria.vidal@tib.eu (M. Vidal)
0000-0002-8734-3123 (E. Iglesias); 0000-0003-1160-8727 (M. Vidal)

[1]https://www.w3.org/TR/2004/REC-rdf-primer-20040210/

for RDF-Star (RML-Star [4]), transformation of collections and containers (RML-CC[2]), and projection and transformation of data sources (RML-LV[3]). The integration of these extensions has allowed RML to evolve into a stand-alone mapping language with its own specification[4].
The dataset used in the KGCW 2025 Challenge defines a informative set of test cases that cover core functionalities of RML, including execution joins, duplicate handling, blank nodes, and empty values. Additionally, the dataset includes test cases for all current RML extensions (e.g., RML-Star, RML-CC, RML-LV). The challenge aims to evaluate the level of compliance of existing KG creation engines with the latest RML specification. This report builds upon the results achieved in the KGCW 2024 Challenge by incorporating evaluations of newly introduced test cases (e.g., RML-LV) and addressing cases that were previously incomplete. It presents the modifications applied to SDM-RDFizer to ensure full compliance with the updated specification and provides the outcomes of executing the 2025 challenge dataset.
This paper is organized into three additional sections. Section 2 provides an overview of SDM-RDFizer, including its techniques, data structures, and physical operators used to optimize KG creation. Section 3 presents the results of the challenge, including a detailed description of the test cases and the required updates for execution. Finally, Section 4 offers concluding remarks and outlines future directions for SDM-RDFizer.

2. SDM-RDFizer

SDM-RDFizer [5, 6] is a KG creation engine capable of transforming structured (i.e., CSV and relational databases) and semi-structured data (i.e., JSON and XML) into RDF triples, following defined mapping rules. It supports the latest RML specification.
SDM-RDFizer adopts a two-fold approach to KG creation, consisting of two main modules: **Triples Maps Planning** (TMP) and **Triples Maps Execution** (TME). Each module plays a distinct role in the KG creation process and employs a set of data structures and operators to handle various aspects such as join execution and duplicate removal. TMP determines the execution order of RML Triples Maps (TMs) with the goal of minimizing memory usage. TME then generates the KG according to the execution plan established by TMP.
To support the transformation of different types of TMs, SDM-RDFizer implements several specialized operators. The **Simple Object Map** (SOM) operator executes *rml:template* and *rml:reference*; the **Object Reference Map** (ORM) operator handles parent triples maps; and the **Object Join Map** (OJM) operator executes joins. For duplicate detection, SDM-RDFizer uses hash tables known as **Predicate Tuple Tables** (PTTs). Each generated triple is compared against the corresponding PTT; if it already exists, it is discarded as a duplicate. Otherwise, the triple is added to both the PTT and the KG. A **Dictionary Table** (DT) is used to compress the resources stored in PTTs. For join operations, SDM-RDFizer caches the results in a structure called the **Predicate Join Tuple Table** (PJTT) to avoid redundant join computations.
For the KGCW 2024 Challenge [7], SDM-RDFizer was extended to support the latest RML modules, including RML-FNML, RML-Star, and RML-IO. In the case of RML-IO, SDM-RDFizer

[2]https://kg-construct.github.io/rml-cc/spec/docs/
[3]https://kg-construct.github.io/rml-lv/ontology/documentation/index-en.html
[4]https://kg-construct.github.io/rml-core/spec/docs/

was enhanced to accept new input types such as compressed files (e.g., ZIP, TAR), SPARQL endpoints, and remote data sources. It also gained the ability to compress the generated KG into various formats, split triples into multiple files, and export in different RDF serializations (e.g., JSON-LD, RDF/XML, Turtle).
To support RML-FNML, a new operator was introduced to execute functions during data transformation, incorporating strategies from FunMap [8]. FunMap is a TM translator that replaces TMs containing functions with equivalent TMs by executing the functions and transforming the input data accordingly. SDM-RDFizer applies the same principle, enabling real-time data transformation through function execution.
For RML-Star, a new operator was developed to handle the *rml:quotedTriplesMap* construct introduced in this module. Additionally, PJTT was extended to store joins that may occur in either the *rml:subjectMap* or the *rml:objectMap*.
Finally, SDM-RDFizer was further enhanced to support the RML-LV and RML-CC modules, which are discussed later in this report. SDM-RDFizer is publicly available on GitHub [5].

3. KGCW 2025 Challenge Test Cases

The KGCW 2025 Challenge [6] aims to assess the compliance of existing KG creation engines with the latest RML specification. The dataset used in this challenge is a refined version of the KGCW 2024 Challenge dataset [7], with a greater focus on cases that test the core functionality of the updated RML specification, along with new test cases (e.g., RML-LV). The dataset is composed of seven sets of test cases:

- **RML-Core:** This set includes basic test cases originally defined in the RML test suite [8] to evaluate the compliance of KG creation engines. While the 2024 dataset used CSV, JSON, XML files, and relational databases (MySQL and PostgreSQL) as data sources, the 2025 dataset focuses solely on JSON files.
- **RML-FNML:** This set includes test cases that apply functions to transform data using a set of predefined operations [9].
- **RML-Star:** This set contains test cases for RDF-Star [10], adapted from the RML-Star test suite [11] to conform to the updated specification.
- **RML-IO:** Formerly part of a unified module (which is now divided into RML-IO and RML-IO-Registry) in the 2024 dataset, this set now includes a wide range of remote data sources such as endpoints, compressed files, and JSON and XML files [12]. It also defines

[5] https://github.com/SDM-TIB/SDM-RDFizer
[6] https://zenodo.org/records/14970817
[7] https://zenodo.org/records/10973433
[8] https://kg-construct.github.io/rml-core/test-cases/docs/
[9] https://kg-construct.github.io/rml-fnml/test-cases/docs/
[10] https://kg-construct.github.io/rml-star/test-cases/docs/
[11] https://zenodo.org/records/6518802
[12] https://kg-construct.github.io/rml-io/test-cases/docs/

Set	# of Test Cases	# of Passed Cases	# of Fail Cases
RML-Core	62	62	0
RML-FNML	19	19	0
RML-Star	18	18	0
RML-IO	32	32	0
RML-IO-Registry	102	102	0
RML-CC	35	35	0
RML-LV	32	32	0
Total	300	300	0

Table 1
Test Cases of the KGCW 2025 Challenge dataset.

output specifications for various formats, including Turtle, RDF/JSON, JSON-LD, and compressed formats like ZIP and TAR.

- **RML-IO-Registry:** This new module extends RML-IO by emphasizing specific data source configurations and datatype mappings.
- **RML-CC:** This set includes test cases that cover collections and containers [13].
- **RML-LV:** This set features test cases where data sources are generated through projection and joins across multiple sources, including mixtures of different data formats [14].

This work presents the results of executing all the modules with SDM-RDFizer. Table 1 shows the total number of test cases in the dataset and which cases were passed and failed by SDM-RDFizer. The full results are available on GitHub [15].

3.1. Results of RML-Core

RML-Core consists of test cases designed to validate fundamental RML functionalities, including the definition of classes, the use of *rml:template* and *rml:reference*, the execution of parent triples maps and joins, and the handling of data types, language tags, and named graphs. In the 2024 challenge, this module utilized a range of data sources such as CSV, JSON, XML, and relational databases. For the 2025 challenge, however, the dataset has been streamlined to focus exclusively on JSON files.
Given this focus, one of the primary challenges lies in correctly navigating nested JSON structures. To address this, SDM-RDFizer implements a recursive traversal mechanism that locates the relevant data by descending through the JSON hierarchy according to the specified iterator. SDM-RDFizer employs a parser query to process the input mappings. To comply with the latest RML specification, the parser has been updated to adopt the new *rml* namespace, remove references to the deprecated R2RML namespace, and revise the *rml:logicalSource* definition. This includes support for *rml:path* and *rml:root*, and replaces *rml:query* with *rml:iterator*.
SDM-RDFizer successfully executed all 62 test cases in the RML-Core module.

[13] https://kg-construct.github.io/rml-cc/test-cases/docs/
[14] https://kg-construct.github.io/rml-lv/test-cases/docs/
[15] https://github.com/SDM-TIB/SDM-RDFizer/tree/master/kgcw_2025_challenge

3.2. Results of RML-FNML

RML-FNML includes test cases that apply functions for value transformations, such as replacing, concatenating, or changing the case of strings, based on the RML+FnO specification. SDM-RDFizer supports these transformations by leveraging FunMap [8], a tool that rewrites triples maps (TMs) containing functions into equivalent ones that reflect the function results. SDM-RDFizer also implements a dedicated operator that executes these functions on the fly, ensuring that the values incorporated into the KG are those produced by the function maps.
The parser query is extended to recognize function maps. In the case of nested functions, an additional parser query is used to isolate functional maps. This enables proper handling of nested structures, as each function map is extracted and executed individually. SDM-RDFizer dynamically resolves function dependencies when one function receives the output of another. SDM-RDFizer successfully performs all 19 test cases from this set.

3.3. Results of RML-Star

RML-Star includes test cases based on RDF-Star, an RDF extension that introduces quoted triples, which allow a triple to be used as the subject or object of another triple. This capability supports the expression of metadata about statements—such as provenance, certainty, or attribution. To accommodate this feature, RML-Star introduces the *rml:quotedTriplesMap* construct for defining quoted triples within a KG. SDM-RDFizer supports this by implementing a dedicated operator capable of generating quoted triples, including recursively nested ones.
A particular challenge addressed in this module is the execution of joins in the *rml:subjectMap*, in addition to the more common use in the *rml:objectMap*. SDM-RDFizer handles both scenarios consistently using its OJM operator for join processing and PJTT for managing intermediate results. The parser was also extended to recognize and process *rml:quotedTriplesMap*.
RML-Star introduces a new type of TM, *rml:NonAssertedTriplesMap*. This type of TM is designed solely for generating quoted triples and does not contribute asserted triples to the KG. SDM-RDFizer treats these mappings as auxiliary, invoking them only when needed to produce quoted triples, and ignoring them otherwise.
SDM-RDFizer successfully executes all 18 test cases in this module.

3.4. Results of RML-IO and RML-IO-Registry

RML-IO and RML-IO-Registry modules comprise test cases that cover a wide range of input data source formats, including compressed files; JSON and XML documents; and data extracted from SPARQL endpoints. These modules also test the ability to generate the KG in various RDF serialization formats, such as Turtle, RDF/XML, and JSON-LD, and to compress outputs into formats like ZIP and TAR. Some cases introduce the concept of directing specific triples to designated output files. These modules aim to assess the capacity of KG creation engines to manage diverse input types and output requirements. Initially, the 2024 dataset included such test cases under RML-Core, but in the 2025 dataset, they are handled by these two modules.
RML-IO-Registry focuses on specialized source specifications and reference formulations, including datatype mappings, different encodings, and handling inputs containing comments.
To extract data from diverse sources, SDM-RDFizer utilizes several Python libraries: *csv* for

CSV files, *json* for JSON, *xml* for XML, *mysql-connector* for MySQL, *psycopg2* for PostgreSQL, and *pyodbc* for SQL Server.
SDM-RDFizer uses the *requests* library to retrieve remote data. For SPARQL endpoints, it employs the *SPARQLWrapper* library to execute queries and format results similarly to CSV. Compressed input files are downloaded locally and decompressed using appropriate libraries (e.g., *zip* for ZIP files). Output files are serialized into RDF formats using the *rdflib* library.
Some test cases explore the use of alternative output destinations for specific triples. These outputs can be defined within various mapping components, such as *rml:subjectMap*, *rml:predicateMap*, or *rml:objectMap*. Depending on the component, SDM-RDFizer directs the relevant triples to a specified output file. For example, if an alternate output is declared in the *subjectMap*, all triples from that mapping are sent to the alternative file. When defined elsewhere, only the relevant triples are redirected. Triples without an alternate destination go to the default output file set at execution. SDM-RDFizer prioritizes the creation of these auxiliary files and can compress them when needed.
SDM-RDFizer successfully executed all 134 test cases from both modules.

3.5. Results of RML-CC

The RML-CC module contains test cases designed to generate RDF collections and containers. It introduces the terms *rml:gather* and *rml:gatherAs* into RML. The *rml:gather* term specifies which data elements should be grouped into a collection, while *rml:gatherAs* defines the type of RDF container or collection to be used (e.g., *rdf:Alt*, *rdf:Bag*, *rdf:List*, or *rdf:Seq*). The gathered data may consist of literals, IRIs, or results from join operations. In some cases, nested collections are formed—for example, an *rdf:Bag* might contain multiple *rdf:List* instances. The *rml:gather* term can be used within both the *rml:subjectMap* and *rml:objectMap*. Properly handling blank nodes is essential in this module, as they are intermediate nodes linking elements within a collection. SDM-RDFizer extends its parser query to support *rml:gather* and *rml:gatherAs*, and introduces additional logic to retrieve and manage nested collections. A new operator is implemented to collect data and generate the corresponding triples based on the specified collection type. This operator supports recursive application, enabling the construction of nested structures. When the collected data originates from join operations, SDM-RDFizer utilizes the PJTT data structure to store the intermediate results, which are accessed as needed during collection generation.
SDM-RDFizer successfully executed all 35 test cases from this module.

3.6. Results of RML-LV

RML-LV is a collection of test cases that apply RML logical views to input data. This module is distinctive in that it focuses on generating new input data by combining and projecting existing sources and using various types of joins (e.g., inner and left joins). The output of a logical view is expected to be flat, meaning that the projected data must not retain any nested structures, even when the original input is a JSON file. RML-LV extends the definition of a TM's *logicalSource* by introducing the terms *rml:viewOn* and *rml:field*. The *rml:viewOn* term specifies the source of data for the logical view, while *rml:field* defines which values are extracted and their corresponding names. Logical views may also be nested. RML-LV supports data format

intermixing; for example, a JSON structure may contain values resembling CSV content.
SDM-RDFizer extends its parser to support logical views and implements a separate mechanism to extract nested views. A new operator is introduced and executed at the TMP module level, unlike the other operators executed at the TME module level; it produces raw values instead of RDF triples. Following the same design philosophy as with other data source formats, SDM-RDFizer ensures that each logical view is generated only once per execution. This operator can also be executed recursively to process nested logical views. When executing joins, SDM-RDFizer uses a modified version of the PJTT data structure, which stores raw values instead of RDF entities. In the case of data format intermixing, these values are extracted and loaded into memory as independent files of the corresponding type (e.g., JSON structures within a CSV file). Finally, all projected data is flattened to eliminate any nested structures.
SDM-RDFizer successfully executed all 32 test cases of this module.

4. Conclusions

The KGCW 2025 Challenge dataset evaluates the compliance of state-of-the-art engines with the new RML formulation. It comprises 300 test cases across seven modules: RML-Core, RML-IO, RML-IO-Registry, RML-FNML, RML-Star, RML-CC, and RML-LV. SDM-RDFizer successfully executed all test cases, covering all proposed modules. The SDM-RDFizer is fully RML compliant. To achieve this, SDM-RDFizer introduced a new parsing query and separate queries for the extraction of nested functions, collections, and views, the extension of existing data structures for the proper handling of the new operators, an operator for the execution of functions on the fly, an operator for generating quoted triples, an operator for the generation of RDF collection, and an operator for the creation of logical views. Moving forward, the authors will apply more extensive testing on the operators implemented for the compliance of RML-CC and RML-LV (especially the flattening logical views) to determine that all border cases are appropriately covered. Additionally, the operator for the execution of RML-Star is currently being redesigned to improve the handling of intermediate results and remove redundant code. Finally, an improved mapping parser is planned to remove the reliance on ever more complicated parsing queries.

Acknowledgments

This work was supported by the “Leibniz Best Minds: Programme for Women Professors”, through funding of the “TrustKG-Transforming Data in Trustable Insights” project (Grant P99/2020), and by the Lower Saxony Ministry of Science and Culture (MWK) with funds from the Volkswagen Foundation’s zukunft.niedersachsen program (CAIMed - Lower Saxony Center for AI and Causal Methods in Medicine; GA No. ZN4257).

References

[1] N. F. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, J. Taylor, Industry-scale knowledge graphs: lessons and challenges, Commun. ACM 62 (2019) 36–43. URL: https://doi.org/10.1145/3331166. doi:10.1145/3331166.

[2] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, R. Van de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: Workshop on Linked Data on the Web, 2014.

[3] B. De Meester, A. Dimou, R. Verborgh, E. Mannens, An ontology to semantically declare and describe functions, in: European Semantic Web Conference, Springer, 2016, pp. 46–49.

[4] T. Delva, J. Arenas-Guerrero, A. Iglesias-Molina, Ó. Corcho, D. Chaves-Fraga, A. Dimou, Rml-star: A declarative mapping language for rdf-star generation, in: O. Seneviratne, C. Pesquita, J. Sequeda, L. Etcheverry (Eds.), 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, volume 2980 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021. URL: https://ceur-ws.org/Vol-2980/paper374.pdf.

[5] E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana, M.-E. Vidal, SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs, in: CIKM, 2020. doi:10.1145/3340531.3412881.

[6] E. Iglesias, M.-E. Vidal, D. Collarana, D. Chaves-Fraga, Empowering the sdm-rdfizer tool for scaling up to complex knowledge graph creation pipelines1, Semantic Web 16 (2025) SW–243580. URL: https://journals.sagepub.com/doi/abs/10.3233/SW-243580. doi:10.3233/SW-243580. arXiv:https://journals.sagepub.com/doi/pdf/10.3233/SW-243580.

[7] E. A. Iglesias, M. Vidal, Results for knowledge graph creation challenge 2024: Sdm-rdfizer, in: D. Chaves-Fraga, A. Dimou, A. Iglesias-Molina, U. Serles, D. V. Assche (Eds.), Proceedings of the 5th International Workshop on Knowledge Graph Construction co-located with 21th Extended Semantic Web Conference (ESWC 2024), Hersonissos, Greece, May 27, 2024, volume 3718 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3718/paper12.pdf.

[8] S. Jozashoori, D. Chaves-Fraga, E. Iglesias, M. Vidal, Ó. Corcho, Funmap: Efficient execution of functional mappings for knowledge graph creation, in: J. Z. Pan, V. A. M. Tamma, C. d'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne, L. Kagal (Eds.), The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part I, volume 12506 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 276–293. URL: https://doi.org/10.1007/978-3-030-62419-4_16. doi:10.1007/978-3-030-62419-4_16.