An RML-FNML module for Python user-defined functions in Morph-KGC

Julián Arenas-Guerrero\textsuperscript{a,*}, Paola Espinoza-Arias\textsuperscript{b}, José Antonio Bernabé-Diaz\textsuperscript{b}, Prashant Deshmukh\textsuperscript{c}, José Luis Sánchez-Fernández\textsuperscript{b}, Oscar Corcho\textsuperscript{a}

\textsuperscript{a} Universidad Politécnica de Madrid, Madrid, Spain
\textsuperscript{b} BASF Digital Solutions, Madrid, Spain
\textsuperscript{c} BASF, Ludwigshafen am Rhein, Germany

A R T I C L E I N F O

Keywords:
Knowledge graphs
Python user-defined functions
RML-FNML
Chemical industry

A B S T R A C T

The RML mapping language declares schema transformations to map heterogeneous data into knowledge graphs. Although the schema transformations provided by RML are sufficient for simple use cases, large real-world ones typically involve diverse data and require complex computations. User-defined functions provide the flexibility to enable the creation and application of knowledge graphs for these use cases and also allow reusing existing software packages for data processing in multiple domains. In this work, we present an implementation for data transformations in Morph-KGC. This implementation has the following benefits: (i) it conforms to RML-FNML, the standard module for data transformations in RML, (ii) it can handle user-defined functions written in Python, a widely used programming language for data processing, (iii) it includes support for YARRRML, a user-friendly syntax of RML to maximize usability. The implementation is currently being used by BASF, a large multinational chemical company, to semantically integrate its industrial data in a maintainable and reproducible manner.

1. Motivation and significance

Several private and public organizations are currently building knowledge graphs [1] (KGs) to integrate data in a standardized and structured way, and exploiting them to get new insights that may be harder to gain with traditional data representation mechanisms. To build KGs, many organizations [2–7] have adopted a declarative construction approach that consists in defining machine-readable mapping rules [8], represented in the RDF mapping language [9] (RML), to describe the correspondences (i.e., schema transformations) between the ontology elements that define the structure of the KGs and the values from heterogeneous data sources (e.g., relational and NoSQL databases, spreadsheets or CSVs). This approach allows building KGs in a standardized, easily maintainable, and reproducible manner; instead

* Corresponding author.

\textit{E-mail addresses:} julian.arenas.guerrero@upm.es (Julián Arenas-Guerrero), paola.espinoza@basf.com (Paola Espinoza-Arias), jose-antonio.bernabe-diaz@basf.com (José Antonio Bernabé-Díaz), prashant.deshmukh@basf.com (Prashant Deshmukh), jose-luis.c.sanchez@basf.com (José Luis Sánchez-Fernández), oscar.corcho@upm.es (Oscar Corcho).

https://doi.org/10.1016/j.softx.2024.101709

Received 20 September 2023; Received in revised form 30 January 2024; Accepted 22 March 2024

Available online 1 April 2024

2352-7110/© 2024 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).
of using ad hoc scripting code or graphical tools that are not easily extensible. Besides the schema transformations that RML provides support for, data transformations are achieved via the RML-FNML module that describes functions using the Function Ontology [10] (FnO), with systems typically offering built-in functions sets such as the General Refine Expression Language\(^1\) (GREL).

Disparate data sources, complex transformations or computations, and large volumes of data are prevalent in current data integration scenarios. Built-in functions in RML systems are not enough to handle the heterogeneity in input data and address data processing tasks where more flexibility is needed. This additional expressive power is usually achieved with user-defined functions (UDFs) which can be written in languages such as Java, C++, or Python. Many researchers and data scientists typically rely on interpreted languages like Python [11], as it is high-level, easy to learn and use, with features such as dynamic typing or automatic memory management, and comes with an extensive set of data processing packages. For these reasons, many modern data processing systems such as databases [12] offer Python UDFs. Therefore, our hypothesis is that making Python UDFs available in RML systems would empower KG creation. Fig. 1 motivates data transformations to create a KG of materials from a data source in German. It first shows a KG built without data transformations which contains errors, due to the lack of data cleaning. In contrast, in the KG built with data transformations, data is not only clean, but it is also more complete, since a natural language translation function was applied to also include the names of the materials in English.

Furthermore, the RDF-based Turtle syntax\(^2\) of RML is verbose, error-prone, hindering usability. When considering data transformations with RML-FNML the complexity increases even more, since additional constructs in the mappings are required. This is relieved with YARRRML [13], a user-friendly syntax of RML using YAML. However, it comes at the cost of an additional mapping translation step during mapping development, which typically involves the following iterative tasks: (i) write or modify the YARRRML rules, (ii) convert YARRRML to RML using translation tools [14], (iii) create the KG with an RML system, and (iv) verify that the output is correct. Therefore, our second hypothesis is that direct consumption of YARRRML from KG creation systems would simplify and accelerate mapping development, a task that usually requires large human efforts [7].

In this paper we describe the implementation of an RML-FNML module for the Morph-KGC [15] system that supports Python UDFs. This functionality is relevant for industrial data integration at BASF\(^3\), a large multinational chemical company, due to the wide variety of its use cases requiring data transformations to create KGs. The implementation copes with the requirements of BASF and many other organizations, where Python is the language of choice in data processing pipelines. Furthermore, we also implement a YARRRML module that transparently translates YARRRML to RML on-the-fly during the creation (also referred to as materialization) of the KG. As a result, mapping development is simplified, reducing human efforts and accelerating KG creation.

Fig. 1. Example of KGs built from a data source of chemical materials (upper left). The first KG (upper right) was built without applying data transformations. The second KG (bottom) was constructed by applying a set of RML-FNML functions resulting in a refined KG with clean material names (without blank spaces and special characters) and with material names also in English.

---

\(^1\) https://www.w3.org/TR/r2rml/#syntax

\(^2\) https://www.w3.org/TR/r2rml/#syntax

\(^3\) https://www.basf.com
2. Software description

This section first introduces RML-FNML, the RML module responsible for data transformations for which we provide an implementation. Next, we describe the architecture of the software, including the native YARRRML support. Finally, we summarize the features of the software.

2.1. RML-FNML

The RDB to RDF mapping language [17] (R2RML) is a W3C Recommendation to map relational databases to RDF. R2RML declares schema transformation rules that associate the input data sources with the output KG. RML\(^4\) is a popular extension of R2RML that generalizes it beyond relational databases, allowing for any input data source (e.g., CSV, XML, JSON). However, data transformations are not included in RML but are declared using a separate module, namely RML-FNML, which defines them using the Function Ontology [10].

The structure of RML-Core and RML-FNML is shown in Fig. 2. The mapping rules in RML are known as triples maps. They contain a logical source to access the input data. Triples maps also contain operators known as term maps that generate the RDF terms of the triples; they can be subject, predicate, object or graphs maps depending on the position taken by the terms. Joins of logical sources can be defined with referencing object maps, in which case the object terms of the mapping rule will be generated by the subject map of the parent mapping rule. RML-FNML declares the evaluation of FnO functions over data from a logical source. The evaluation of a transformation function is defined via a function execution, with the function given by a function map. An input links an FnO parameter (parameter map) to some input value (input value map).

2.2. Software architecture

An overview of the software is depicted in Fig. 3. It takes as input (i) the set of Python UDFs, (ii) the functional YARRRML mapping rules, and (iii) the set of heterogeneous data sources, and produces as output an RDF KG. In the following, we describe the tasks performed by the software.

YARRRML translation. Initially, the YARRRML module checks whether the input mappings use this syntax, in which case they are translated to RML(-FNML). The translation process is accomplished in two stages: (i) normalization and (ii) translation. The former simplifies the YARRRML rules to facilitate translation; this includes the harmonization of YARRRML keys (e.g., function, f and fn are all represented as function), expansion of prefixes in term maps, conversion of inverse predicates\(^5\) to new predicate–object maps, etc. Normalization significantly simplifies the translation phase, which consists in iterating over the YARRRML tree creating for each clause the associated RML rule. Performing mapping translation at the beginning abstracts other system’s procedures from YARRRML.

RML-FNML parser. The RML-Core parser of Morph-KGC transforms the RDF-based mappings to a table representation, namely an rml\_df Pandas DataFrame. For each term map, rml\_df stores its type (viz., rml:constant, rml:template, rml:reference, rml:parentTriplesMap) and associated value. This was extended to consider rml:functionExecution as a new type, with the associated value pointing to the function execution. To store the actual functions, and similar to the case of rml\_df, the parser has been extended to transform them to a new fnml\_df DataFrame, which stores the function executions along with function, parameter and value maps. As mentioned above, the values of function executions in rml\_df point to function executions in fnml\_df.

```python
import re

@udf(
    fun_id= "http://ontology.basf.com/BASF_Functions/format_name",
    material= "http://users.ugent.be/bjdmeest/function/grel.ttl#valueParam"
)
```

---

\(^4\) RML is also referred to as RML-Core, which is the main module of the language declaring schema transformations [9].

\(^5\) https://rml.io/yarrrml/spec/#example-po-inverse
def format_name(material):
    material = material.strip()
    clean_material = " ".join(re.split(" \s+ ", material))
    return re.sub(r"[\w\s]+", " ", clean_material)

Listing 1: Example of a Python UDF that removes blank spaces and special characters from a string representing a material in BASF.

Built-in and user-defined functions. The RML-FNML interpreter supports built-in and user-defined functions. Both of them use Python decorators to specify the function identifier and to link the parameters of theFnO function to the procedural Python parameters. Built-in functions use the bif decorator (this is transparent to the user since the function does not have to be provided), and UDFs use the udf decorator with fun_id defining the identifier to refer to the function from the RML-FNML mappings. An example of the latter is shown in Listing 1, which presents a UDF that removes blank spaces and special characters; the Python function receives material as input parameter, which is linked in the decorator to theFnO grel:valueParam parameter. To filter values in a UDF, None can be returned, in which case the RML-FNML module will not generate triples for those values.

Materialization procedure. The software processes one mapping rule at a time. It first loads the data from the logical source, converting it to a Pandas DataFrame. Data manipulation, such as percent-encoding, NULL removal, and string escaping and concatenation are performed over the created DataFrame. The procedure handles the different kinds of term maps and produces the RDF terms that are aggregated to create the final output triples. In the case of functions, the procedure identifies function-valued term maps and delegates its execution to the associated operator, which is described next.

Function executor. The RML-FNML interpreter first uses the identifier of the function to retrieve it along with its decorator. When the function is user-defined, this entails loading it from a script provided by the user. The columns of the internal DataFrame are the input values for the function, and a new column is produced to store the results of its evaluation (the RDF term specifying the function execution in the mapping is used as the name of the new column). Chaining (or nesting) of functions is achieved following operator-at-a-time processing: the individual operators (i.e., the functions) of a mapping rule are executed over entire columns before moving to the next operator. Because the results of the function are stored in the DataFrame, the output of one function can be used as the input of subsequent functions, hence enabling chaining. It must be noted that, in accordance with the R2RML standard, NULL values resulting from the evaluation of a function are filtered after its execution.

2.3. Software functionalities

Data transformation. The implementation comes with built-in functions, namely a subset of OpenRefine’s GREL functions. UDFs can be used when additional expressive power is required, in which case the path to a script with the functions must be provided in the configuration of Morph-KGC (an example is presented in Section 3). The built-in set can be extended on demand to adapt to users’ needs.

RML-FNML compliant. To our knowledge, this is the first KG creation system that is compliant with the new RML-FNML module [9]. Compliance and technical quality are ensured with official test cases from the W3C Community Group on KG Construction. These test cases have been included in the continuous integration pipeline of Morph-KGC with GitHub Actions.

Native YARRRML support. The usability of the software is enhanced by supporting YARRRML mappings out-of-the-box. This simplifies mapping development and avoids the need for additional steps to translate the mapping to RML-FNML.

3. Illustrative example

BASF offers the world’s largest portfolio and variety of chemical materials to meet the specific needs of several industries such as pharmaceuticals, automotive, agricultural, among others. Most of these materials are the product of in-house inventions that have been refined over more than a century of work. In this refinement process, there are several legacy systems that handle data related to these materials in different laboratories around the world. Unfortunately, some of the names of these materials include incorrect characters because legacy systems allowed names to be added as a free text entry, limiting searchability. In addition, the names of most of the materials are provided only in German, which hinders its exploitation by non-German experts in laboratories across the world, who typically use English.

To format and translate materials from an internal BASF’s instance of the Databricks lakehouse [19], UDFs are applied to create the refined KG (see Fig. 1). Listing 2 shows the RML-FNML rules that address these transformations. Here, the basf_function:format_name, presented in Listing 1, takes the $(name) value as input parameter and removes blank spaces, dots, colons, among other characters that were erroneously added to the name. Additionally, to generate material names in English, a nested function is applied to (i) first clean the material name using the basf_function:format_name
function, and (ii) then translate the cleaned name into English using the `basf_function:translate_name` function, which calls an external translation service. Language tags are specified in the mapping rules according to the language of the generated material names.

```
mappings:
  materials:
    sources:
      table: materials
    s: basf_data:{id}
    po:
      - [rdf:type, basf_onto:Material]
      - p: basf_onto:alternative_name
        o:
          - function: basf_function:translate_name
            parameters:
              - parameter: grel:valueParam
                value: $(name)
            language: de
      - p: basf_onto:name
        o:
          - function: basf_function:translate_name
            parameters:
              - parameter: grel:valueParam
                value: $(name)
            language: en
```

Listing 2: Mapping to format and translate material names into English.

Listing 3 shows the code snippet to execute Morph-KGC. The configuration of the system is done via an INI configuration file, which can also be provided as a string, as shown in the listing. In the `CONFIGURATION` section of the INI string, the `udfs` parameter is used to specify the location of the script that contains the UDFs (in this case the script in Listing 1). The `DATABRICKS` section contains the path to the mapping file and the details of the Databricks connection to the BASF lakehouse containing the material data. It must be noted that the parameters of the Databricks' connection provided in the listing only contain placeholders (inside `"<"`) instead of the real values that are used in the BASF's connection.

```
import morph_kgc

# create the config as a string
config = ""
  [CONFIGURATION]
    udfs: path/to/udfs.py
  [DATABRICKS]
    mappings: path/to/mapping_file.yml
    db_url: databricks:connector://token:<databricks_token>@<databricks_host>:443/<database_or_schema_name>
    connect_args: {"http_path": "<cluster_http_path>"}
  ""

# generate the triples and load them to an RDFLib graph
g_rdflib = morph_kgc.materialize(config)
```

Listing 3: Execution of Morph-KGC to create the BASF's materials KG and load it to RDFLib.

4. Impact

The presented software implementation is specially intended for the construction of KGs from messy data sources that must be cleaned or where custom computations have to be performed. As previously mentioned, the flexibility of UDFs is a fundamental requirement when creating KGs in such complex scenarios and the RML-FNML software module presented here will enable applications where the additional expressive power that they provide is needed. We expect that the fact that Python is the preferred language for many researchers and data engineers will contribute to the potential adoption of the software. In addition, the RML-FNML module is also appropriate for reusing Python packages implementing specialized computations or machine learning tasks (e.g., natural language translation packages to create multilingual KGs) which can be now applied to KG construction. The wide amount of Python packages that are available significantly increases the number of use cases in which a KG can be created.

The presented RML-FNML and YARRRML modules are built on top of a robust and well-known system, Morph-KGC [15], already used in domains such as biodiversity [3], biomedicine [5,20] or climate change [21]. The YARRRML module is now extensively applied in projects where the Ontology Engineering Group is involved. The most significant impact of this functionality is the reduction of the time spent by researchers on mapping development and testing, which can now be invested in other tasks. An application in which the YARRRML functionality was employed is the creation of a research software citation KG [22] from a dataset of millions of papers from PubMed Central [23].

The RML-FNML module and the support of Python UDFs are used in semantic data integration pipelines at BASF. In fact, this functionality was implemented due to the need of the organization for data transformations given its complex data use cases, and the lack of tools implementing RML-FNML and Python UDFs. The fact that UDFs are written in Python, which is considered as one of the easiest programming languages for beginners, also makes the RML-FNML module appropriate for educational purposes. In addition, the native support of YARRRML prevents users from having to learn the verbose RML syntax, making it even more suitable for learning. This extension is currently used in courses at the graduate and undergraduate levels at Universidad Politécnica de Madrid and can be adopted by other universities in their KG-related courses.

New research questions in semantic data integration can be addressed with the RML-FNML module, especially concerning the performance of KG construction. Some of the research questions that can be pursued as a result of this software are:

- As an alternative to RML-FNML, data transformations can also be achieved with RML Views [24] for relational databases and tabular files. An RML View is an SQL query in the logical source of a mapping rule which may define data transformations that are pushed down to the underlying data sources. However, it is unclear how the efficiency of data transformations in RML-FNML and RML Views compares. Since Morph-KGC now supports both, the system could be used to analyze this in future works. **RQ1:** Is the execution of data transformation functions in RML-FNML more efficient than in RML Views?

- Beyond RML, other mapping languages have extended the syntax of the SPARQL query language for mapping heterogeneous data to RDF [25,26]. The efficiency of RML systems and SPARQL-based systems have been previously compared, but the impact of data transformation functions in each approach is not clear. **RQ2:** How does the performance of data transformation in RML-based and SPARQL-based mapping systems compare?

7. https://oeg.fi.upm.es/
• Recently, mapping planning strategies have been applied to optimize the execution of RML mappings [27]. The impact of planning strategies in RML-FNML materialization remains unstudied. **RQ3:** Can mapping planning strategies be applied to optimize the execution of RML-FNML?

Regarding usability, the native integration of YARRRML with a materialization system removes the need of an additional mapping translation step and the use of additional systems to perform this task. Given the large human efforts required by the mapping development process, which can amount for several person-months [7], it is important to analyze and quantify its impact with user studies.

• Native YARRRML support reduces human efforts in mapping development, but a quantitative investigation of the time saved by this functionality is missing. **RQ4:** What is the impact of direct consumption of YARRRML in the mapping development process?

5. Conclusions

The RML-FNML module in Morph-KGC was adopted by BASF to build KGs at scale in a standardized, maintainable, and reproducible manner to semantically integrate its industrial data. Given the diversity and heterogeneity of this data, support for UDFs is a fundamental requirement in the system, for which we provide an implementation in this work. UDFs are written in Python, the language used at BASF and many other organizations for data processing. This implementation will enable semantic integration of data in new domains where complex computations are required.

The presented software also facilitates mapping development by natively supporting the YARRRML mapping syntax. This reduces the complexity of mapping development and accelerates the creation of KGs, which are essential factors that contribute to the potential adoption of the software by other organizations.

**References**


[6] Kalyace EG, Grangel González I, Lösch F, Xiao G, ul Mehdli A, Kharlamov E, et al. Native YARRRML support reduces human efforts in mapping development, but a quantitative investigation of the time saved by this functionality is missing. **RQ4:** What is the impact of direct consumption of YARRRML in the mapping development process?

The RML-FNML module in Morph-KGC was adopted by BASF to build KGs at scale in a standardized, maintainable, and reproducible manner to semantically integrate its industrial data. Given the diversity and heterogeneity of this data, support for UDFs is a fundamental requirement in the system, for which we provide an implementation in this work. UDFs are written in Python, the language used at BASF and many other organizations for data processing. This implementation will enable semantic integration of data in new domains where complex computations are required.

The presented software also facilitates mapping development by natively supporting the YARRRML mapping syntax. This reduces the complexity of mapping development and accelerates the creation of KGs, which are essential factors that contribute to the potential adoption of the software by other organizations.

**Declaration of competing interest**

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Julián Arenas-Guerrero reports financial support was provided by Euratom Research and Training Programme (grant agreement No 900018). Julián Arenas-Guerrero and Oscar Corcho have patent #Morph-KGC: Scalable Knowledge Graph Construction with [R2]RML Mappings issued to Julián Arenas-Guerrero and Oscar Corcho.

**Data availability**

No data was used for the research described in the article.

**Acknowledgments**

The authors would like to thank Jürgen Müller as the BASF sponsor of this software. This work also received partial financial support in the frame of the Euratom Research and Training Programme 2019–2020 under grant agreement No 900018 (ENTENTE project).


