# Mapeathor: Simplifying the Specification of Declarative Rules for Knowledge Graph Construction

Ana Iglesias-Molina, Luis Pozo-Gilo, Daniel Doña, Edna Ruckhaus, David Chaves-Fraga, and Oscar Corcho

Ontology Engineering Group, Universidad Politécnica de Madrid, Spain
{ana.iglesiasm,luis.pozo}@upm.es, ddona@delicias.dia.fi.upm.es,
{eruckhaus,dchaves,ocorcho}@fi.upm.es

**Abstract.** In recent years we have observed an increasing interest by the scientific community, from social sciences to biomedicine, in the generation and publication of RDF-based knowledge graphs. One possibility for creating knowledge graphs consists in using declarative mappings together with their associated parsers. These mappings describe the relationship between the source data and a reference ontology. However, the learning curve to create these mapping files is steep, hindering its use by a wider community. In this paper we present a user-friendly mapping-language-independent tool, Mapeathor, to declare transformation rules based on spreadsheets and translate them into two different mapping languages with the purpose of easing the mappings creation process.

**Keywords:** Knowledge Graph · Declarative mapping · Spreadsheet

## 1 Introduction

In the last few decades, we have seen a significant increase in the publication of data in a machine understandable manner following Linked Data principles[1] (e.g., DBpedia[2], Wikidata[3]). Knowledge Graph construction requires integrating different data sources in a structured way, usually following the schema of an ontology or group of ontologies. This facilitates the posterior task of mining the knowledge graph with several applications, such as searching recommendations and learning implicit data patterns.

Knowledge graphs can be built in diverse ways. One option is creating ad-hoc scripts to transform data, which requires the user to repeat the process of script



[1] https://5stardata.info/en/
[2] https://wiki.dbpedia.org/
[3] https://www.wikidata.org/

writing in every specific use case. Another option is using tools like OpenRefine[4] to perform data transformation through the creation of an RDF skeleton, which includes proprietary transformation rules and a functionality for knowledge graph construction. Lastly, there is an option to keep the transformation rules in specific files that can be later processed by engines that either transform the data to RDF or create a virtual knowledge graph that can be queried without transforming the source data. These rules can be written in a wide variety of languages (e.g., R2RML [2], RML [3]) that cover different user's needs (e.g., the source data format or the engine that will be used). Although the use of these mapping files is more flexible and independent, since they can be processed by a wide variety of engines, their creation is still not easy for new users. Experts are usually needed to carry out these tasks, hindering the use of semantic web technologies across the scientific community. That is why it is necessary to lower the learning curve and improve mapping reuse and reproducibility.

Since mapping languages started to be used by the community, there have been multiple approaches for the development of editors to ease their specification. Most of them enable editing through graphical visualization [4, 6], others provide a writing environment (e.g. the Protégé extension OntopPro). These editors are language-oriented, they help to create one kind of mapping, not taking into account the wide variety of mapping languages that currently exist. Moreover, when managing a considerable amount of mapping rules, a graphical approach may not be easily handled.

Our work focuses on providing a straightforward way to create these mappings, specifying the transformation rules in spreadsheets, so they are later translated into one of the implemented mapping languages. The purpose of this proposal is to increase the interoperability between these languages [1] as well as to ease the creation process. To perform the mapping rules translation we developed Mapeathor[5], a tool able to parse the spreadsheets and generate the corresponding mappings in two different languages. This work is an extension and improvement on the work previously presented in [5], where the first version of the spreadsheet design and the tool were presented. The spreadsheet includes now more options to maintain the language's expressiveness, and the implementation has become simpler to use and more accurate in the translation.

This paper is structured as follows: Section 2 describes the design of the spreadsheet. Section 3 explains the functionalities of the tool and a real-world use case. Finally, section 4 presents the main conclusions and future work.

## 2 Spreadsheet design

The rules required to generate a knowledge graph can be specified in multiple languages. The language is chosen by the user depending on the specific use case. However, the rules themselves are equivalent across languages, so they can be written in a language-independent way, in this case, we chose a spreadsheet for

[4] http://openrefine.org/
[5] https://morph.oeg.fi.upm.es/demo/mapeathor

**(a) Prefix sheet**

| Prefix | URI |
|---|---|
| noise | http://v.ciudadesabiertas.es/cont-acustica# |
| noise-res | http://v.ciudadesabiertas.es/res/cont-acustica# |
| sosa | http://www.w3.org/ns/sosa/ |

**(b) Subject sheet**

| ID | Class | URI |
|---|---|---|
| Station | noise:EstacionMedida | noise-res:estacion-medida/{id} |
| Observation | noise:Observacion | noise-res:observacion/{idx} |

**(c) Source sheet**

| ID | Feature | Value |
|---|---|---|
| Station | query | SELECT id, name FROM Station |
| Observation | source | data/station.json |
| Observation | format | JSON |
| Observation | iterator | $ |

**(e) Function sheet**

| FunctionID | Feature | Value |
|---|---|---|
| <Fun1> | fno:executes | grel:replace |
| <Fun1> | ex:param1 | {obsProperty} |
| <Fun1> | ex:param2 | " " |
| <Fun1> | ex:param3 | "-" |

**(d) Predicate_Object sheet**

| ID | Predicate | Object | DataType | ReferenceID | InnerRef | OuterRef |
|---|---|---|---|---|---|---|
| Station | dcterms:identifier | {id} | string | | | |
| Station | schema:name | {name} | string | | | |
| Station | geosparql:hasGeometry | noise-res:punto/{id} | iri | | | |
| Observation | sosa:resultTime | {resTime} | Time | | | |
| Observation | sosa:madeBySensor | | | Station | {madeBySensor} | {id} |
| Observation | sosa:observedProperty | <Fun1> | | | | |

Fig. 1: Example of a spreadsheet representing the **(a)** Prefix sheet, **(b)** Subject sheet, **(c)** Source sheet, **(d)** Predicate_Object sheet and **(e)** Function sheet.

rule specification. The spreadsheet template is devised to contain the rules in a compact and understandable way, in a format widely used by the scientific community. The design is aimed to be language-independent and to ease the writing process so the user does not have to learn a mapping language. In addition, the functionalities of a spreadsheet editor can be used to speed up the writing process. Reusing mappings for similar use cases is also easier in this specification format. The spreadsheet contains the mapping essential elements structured in five different sheets: Prefix, Source, Subject, Predicate_Object and Function.

**Prefix sheet:** This sheet contains the namespaces and corresponding prefixes used when declaring the transformation rules (Figure 1a). It is composed of two columns: `Prefix` for the prefix and `URI` for the corresponding namespace.

**Subject sheet:** This sheet defines the subjects to be generated and the key ID that links the information in the sheets (Figure 1b). It is organized in three columns: `ID`, `Class` and `URI`. `URI` defines the template URI for the subject, its class is specified in `Class`. `ID` contains a unique identifier for each subject's set of rules in order to relate to information on these rules in the remaining sheets.

**Source sheet:** Here we specify where the data is retrieved from (Figure 1c). The information is organized in three columns: `ID`, `Feature` and `Value`. `Feature` declares the type of information provided in `Value`. In `Value` it can be specified the path to the source data (with the feature *source*), the format (*format*), the iterator (*iterator*, loop used to map the data from JSON and XML files), database table (*table*), SQL query (*query*) and SQL version (*SQLVersion*). Any language option may be included. Finally, `ID` indicates the rule it refers to.

**Predicate_Object sheet:** This sheet defines the triples through the predicates and its correspondent objects (Figure 1d). The columns `Predicate` and `Object` specify the predicate and object in a rule. The XSD datatype of `Object`

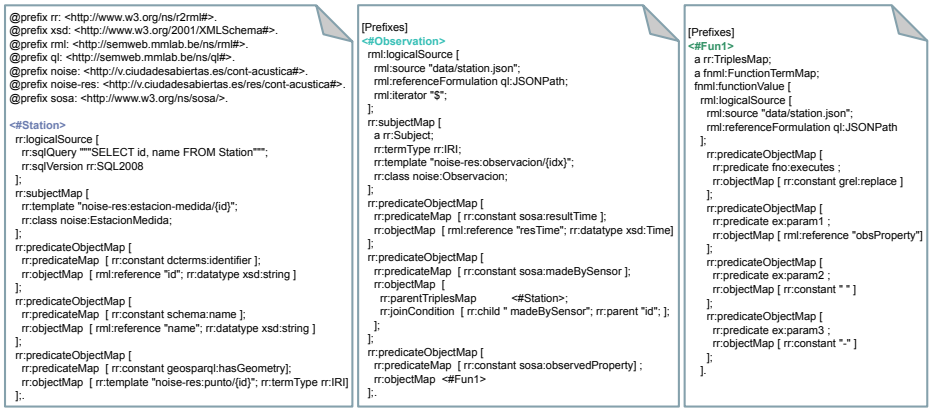


Fig. 2: Output RML mapping file resulting from the translation of the rules shown in Figure 1. The following sets of rules are shown: (a) Station, (b) Observation and the function (c) Fun1.

is defined in `DataType`. When the object refers to a subject defined in another rule, the rule is written differently. There are three fields that allow the specification of the linking condition between the object of the triple and the referenced subject. They specify which is the ID of the target subject (`ReferenceID`), and the "join" fields in the source data (`InnerRef` for the field of the object of the current triple, and `OuterRef` for the field of the referred subject). Lastly, the column `ID` indicates the rule it belongs to.

**Function sheet:** Some languages are able to process transformation functions over the data (e.g. FnO+RML), which can be detailed in this sheet (Figure 1e). Some well known options are the SQL and GREL functions, but any option can be used. The functions are referred in the Predicate_Object sheet or in other function rows with the identifier specified in `FunctionID`. The column `Feature` is used to specify the type of information provided in `Value`, where the name of the function and the value of the parameters are written.

## 3 Demonstration

The spreadsheet containing the transformation rules is processed by the tool Mapeathor to create a mapping file. For example, Figure 2 depicts the mapping file written in the RML language that results when translating the rules in Figure 1. Currently, this tool translates Google spreadsheets and XLSX files to the following languages: the W3C recommendation R2RML [2], RML [3], and its serialization, YARRRML[6]. It can be used as a web service[7] and as a CLI[8].

[6] https://rml.io/yarrrml/
[7] https://morph.oeg.fi.upm.es/tool/mapeathor/swagger/
[8] https://github.com/oeg-upm/Mapeathor

Currently, Mapeathor is being used to generate mappings for city open data publication related to traffic, public bus transport, budget and noise pollution in the context of the Ciudades Abiertas project. Six spreadsheets have been completed, containing 31 subjects and 104 predicate-objects rules. The process of spreadsheet completion and mapping creation for the languages implemented will be shown in the demo with data from this real-world use case.

## 4 Conclusions and future work

This paper presents Mapeathor, a tool able to translate transformation rules specified in spreadsheets to three different mapping languages. The key part of the work are the spreadsheets containing the mapping rules, since they are designed to facilitate the specification process for the user. Currently, the tool is being tested in several use cases from the Ciudades Abiertas project.

The purpose of this work is to create a framework to declare in a user-friendly manner the transformation rules in a language-independent way and to be able to generate these rules in any mapping language. Future work includes a user study to test the usefulness of this tool and find guidelines for improvement, extend the tool to cover more languages, and implement changes that make rule specification more user-friendly.

**Acknowledgements.** The work presented in this paper is supported by the Spanish Ministerio de Economía, Industria y Competitividad and EU FEDER funds under the DATOS 4.0: RETOS Y SOLUCIONES - UPM Spanish national project (TIN2016-78011-C4-4-R).

## References

1. Corcho, O., Priyatna, F., Chaves-Fraga, D.: Towards a New Generation of Ontology Based Data Access. Semantic Web **11**, 153–160 (2020)
2. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language, W3C Recommendation 27 September 2012, https://www.w3.org/TR/r2rml/
3. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: a generic language for integrated RDF mappings of heterogeneous data. In: Ldow (2014)
4. Heyvaert, P., Dimou, A., Herregodts, A.L., Verborgh, R., Schuurman, D., Mannens, E., Van de Walle, R.: Rmleditor: a graph-based mapping editor for linked data mappings. In: European Semantic Web Conference. pp. 709–723. Springer (2016)
5. Iglesias-Molina, A., Chaves-Fraga, D., Priyatna, F., Corcho, O.: Towards the definition of a language-independent mapping template for knowledge graph creation. In: Proceedings of the Third International Workshop on Capturing Scientific Knowledge. pp. 33–36 (2019)
6. Sicilia, Á., Nemirovski, G., Nolle, A.: Map-On: A web-based editor for visual ontology mapping. Semantic Web **8**(6), 969–980 (2017)