UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Informáticos

Knowledge Graph Construction and Evolution using Declarative Mapping Languages

D O C T O R A L T H E S I S

Submitted for the degree of Doctor by:

Ana Iglesias Molina
MSc in Computational Biology

Madrid, 2024


UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Informáticos

Doctoral Degree in Artificial Intelligence

Knowledge Graph Construction and Evolution using Declarative Mapping Languages

D O C T O R A L T H E S I S

Submitted for the degree of Doctor by:

Ana Iglesias Molina
MSc in Computational Biology

Under the supervision of:
Dr. Óscar Corcho García

Madrid, 2024

Title: Knowledge Graph Construction and Evolution using Declarative Mapping Languages
Author: Ana Iglesias Molina
Doctoral Programme: Artificial Intelligence

Thesis Supervision:

Dr. Óscar Corcho García, full professor, Universidad Politécnica de Madrid (Supervisor)

External Reviewers:

Thesis Defense Committee:

Thesis Defense Date:

A mis padres, y a mi hermano

Acknowledgement

Soy porque somos. Como hay una parte de mí en esta tesis, hay pedacitos de mucha más gente también, de todas las personas maravillosas con las que he tenido la suerte de cruzarme estos años.

Quiero empezar agradeciendo a Oscar y Asun, que desde las clases de Tecnologías Semánticas del máster despertaron mi interés, y me dieron la oportunidad de entrar en el Ontology Engineering Group. Especialmente a Oscar, hacer la tesis bajo tu supervisión ha sido una experiencia que me ha permitido aprender y crecer mucho durante estos años, tanto con todas las reuniones, consejos y conversaciones, como con la libertad que siempre me has dado para que encontrase mi camino. Siempre has sido un referente y una motivación en cada paso y en cada crisis, gracias por la confianza y poder trabajar contigo estos años.

El camino hubiera sido infinitamente más duro si no hubiera aterrizado en el OEG, y sin la increíble red de apoyo que tejen todas las personas que me han acompañado estos años desde que llegué. A la "antigua" generación, tanto los que están como los que ya se fueron, nuevos y antiguos profesores, María, Carlos, Patri, Pablo, Paola, Alba, Andrea, Esteban, Elvira, Elena, Raúl, Víctor, siempre referentes y siguiendo vuestros pasos. A la "nueva" generación, que tanto me ha amenizado los últimos años, a Clark (about time to learn spanish), Camilla, Ibai, Diego, Carlos, Sergio, Isam y Lucía Palacios. A Salva, por el compañerismo, los ánimos y el apoyo constante para seguir adelante. A Dani, por siempre estar tan dispuesto a echar una mano. A Alejandra, por saber siempre cuáles eran las palabras adecuadas para dar en la fibra, centrarme y seguir escribiendo. A Lucía Sánchez y Paula, mis compañeras de mesa y faltadas, de animos y risas, de meriendas y dramas, gracias por hacerlo tan fácil. Y por suspuesto, a lo que fue del *data (des)integration group*: A Edna, por el cariño, la paciencia y la compañía en los momentos más difíciles; a Jhon, por las reuniones disfrutonas, y las sesiones de post-pádel; y especialmente a David, por todo. Desde revisarme los emails que no me atrevía a mandar, a los cafés para ponernos al día. Me has acompañado desde el principio, mostrándome lo motivamente que puede ser este camino, descubriéndome nuevos retos, abriéndome puertas y siempre dispuesto a ayudar y avanzar. Ha sido (y sigue siendo!) un placer trabajar a vuestro lado. Gracias también a JARG, Raúl Alcázar y Ana Ibarrola, que integran una parte esencial del esqueleto del OEG y hacen posible todo nuestro trabajo.

I also want to thank Pedro, Filip and Jay for granting me the opportunity to work at ISI and discover new horizons in research, showing me so many different perspectives; and Kian, I'll never forget the long conversations at your office, that I took as mine. To my good colleague in Belgium, Dylan, I promise I will be a better tourist guide if you ever come back to Madrid. And to Anastasia, working with you showed me so much about collaboration, team work and research, I will never be grateful enough for all I have learned from you. It is an honor and a pleasure working by your side.

Miles de gracias también a todas las personas de mi día a día, que se preguntan qué hago desde que empecé la carrera y seguirán preguntando hasta después de terminar el doctorado. A Eva, Arturo y Xana por aterrizarme, y recordarme la vida detrás de la tesis. A Laura y Patri, ¿quién nos iba a decir que iba a terminar haciendo la tesis con todo lo que me

escuchasteis renegar de ella? Gracias por las risas, los viajes, la ayuda *(please!)* y siempre estar ahí. Y a Evelyn, por todos los años de amistad a pesar lo opuestas que somos, por las noches de gyozas, las aventuras, risas y dramas, y tu compañía y cariño siempre presentes.

Y por último, a mi familia, sin la que nada de esto hubiera sido posible. A mis padres, Sara y Antonio, por vuestro cariño y apoyo incondicional, gracias por haberme dado siempre alas para poder llegar a donde quisiera. A mi hermano, Miguel, por tu energía y optimismo, por hacerme reflexionar y rebatir, hacerme ver la vida de otra forma. A pesar de la distancia, y lo que echo de menos sentarnos por las noches a hablar, y que vengas a molestarme a cualquier hora, nunca estás lejos.

Gracias a todos los que estáis y los que habéis estado por motivarme a seguir cada día. No me olvido de ninguna las personas que han pasado por mi vida y que de una forma u otra, han sido una luz en este camino.

Abstract

Knowledge graphs have gained momentum in the past few decades, becoming evermore essential for data interoperability, management, analysis and exploitation. An aspect that plays an essential role in their uptake and use is the ease of their construction. There are several ways in which knowledge graphs can be constructed from heterogeneous data, ranging from ad hoc scripting to declaratively defining the mapping transformation rules.

The use of declarative approaches enables a reusable, maintainable and understandable manner for a seamless knowledge graph construction process. They rely on mapping languages to express how to transform the source data into the desired graph structure. Although these approaches are progressively used and adopted, they still lack some expressiveness for the increasing complexity of available data. Therefore, this thesis analyses the expressiveness of these languages, and extracts and defines the requirements for constructing knowledge graphs with the current needs. In addition, it extends the RDF Mapping Language (RML) with features to generate knowledge graphs enriched with annotations, following the latest developments in the area.

In order to facilitate the creation of the transformation rules for users with different backgrounds and expertise, this thesis proposes a spreadsheet-based approach to write them, providing a familiar environment and suppressing the need of learning the syntax of the language. It also updates a user-friendly serialization with the latest additions from its target language, suitable for users with more technical profiles. Both approaches are supported by implementations that can interoperate with different languages.

Finally, this thesis evaluates the role that these declarative approaches can play in different tasks involved in the knowledge graph life cycle. More specifically, it assesses how they can be beneficial when refactoring the schema of knowledge graphs.

Overall, this thesis contributes to the understanding of the capabilities that declarative languages have for knowledge graph construction and refactoring, while providing extended support for their creation and interoperability.

Resumen

Los grafos de conocimiento han ganado impulso en las últimas décadas, posicionándose como un recurso clave para potenciar la interoperabilidad entre datos, su gestión, análisis y explotación. Un aspecto esencial para incrementar su acogida y uso es asegurar que se pueden construir fácilmente. Hay muchas maneras en las que los grafos de conocimiento se pueden construir, desde usando *scripts* ad hoc hasta definiendo reglas de transformación declarativas.

El uso de los métodos declarativos posibilitan la construcción de grafos de conocimiento de manera reusable, mantenible y comprensible. Estos métodos se basan en lenguajes que permiten expresar las reglas de transformación. Aunque se usan de manera cada vez más extendida, hay casos para los que su expresividad no es suficiente para lidiar con la complejidad de los datos. Por ello, esta tesis analiza la expresividad de estos lenguajes, extrae y define los requisitos para construir grafos de conocimiento acorde a las necesidades actuales. Además, extiende un conocido lenguaje para posibilitar la creación de grafos enriquecidos con anotaciones, siguiendo los últimos avances en el área.

Para facilitar la creación de las reglas de transformación con estos lenguajes para usuarios con distintos perfiles y experiencia, esta tesis propone un método basado en hojas de cálculo para escribir las reglas. Este método provee de un entorno familiar para su escritura, al tiempo que evita que los usuarios tengan que aprender la sintaxis de los lenguajes. Además, esta tesis propone una actualización de una serialización amigable acorde a los últimos avances en los lenguages. Ambas propuestas se proponen junto con un servicio que permite respectivamente la generación de reglas en varios lenguajes, facilitando asímismo la interoperabilidad entre ellos.

Finalmente, esta tesis evalúa el rol que los métodos declarativos pueden jugar en fases distintas del ciclo de vida de los grafos de conocimiento. Más específicamente, se valora cómo pueden beneficiar al proceso de cambio de estructura de los grafos.

En general, esta tesis contribuye al mejor entendimiento de las capacidades de los lenguajes declarativos para la construcción y evolución de grafos de conocimiento, al tiempo que propone un soporte extendido para facilitar su creación e interoperabilidad.

Table of Contents

Acknowledgement v
Abstract vii
Resumen ix
List of Figures xiii
List of Tables xvii
List of Algorithms xix
Abbreviations and acronyms xxi

1 Introduction **1**
1.1 Main Contributions 4
1.2 Thesis Structure 5
1.3 Derived Outcomes 6
1.4 Related Research Activities 8

2 State of the Art **11**
2.1 Knowledge Representation in the Semantic Web 11
2.1.1 Resource Description Framework (RDF) 11
2.1.2 Statements About Statements in RDF 12
2.2 Knowledge Graph Construction with Declarative Mapping Rules 14
2.2.1 R2RML: RDB to RDF Mapping Language 15
2.2.2 RML: RDF Mapping Language 18
2.2.3 Additional Mapping Languages 20
RDF-based Mapping Languages. 20
SPARQL-based Mapping Languages. 22
Mapping Languages Based on Alternative Schemas. 23
2.2.4 Comparison of Mapping Languages Expressiveness 24
2.2.5 Interoperability for Mapping Languages 25
2.3 User-Friendly Knowledge Graph Construction Approaches 26
2.3.1 Visual Editors 26
2.3.2 Serializations 28
2.4 Knowledge Graph Life Cycle 29
2.5 Conclusions and Limitations of the State of the Art 32

3 Objectives and Contributions **35**
3.1 Objectives 35

3.2 Contributions . . . 36
3.3 Assumptions . . . 37
3.4 Hypotheses . . . 38
3.5 Restrictions . . . 38

4 Understanding, Representing and Extending the Expressiveness of Declarative Mapping Languages 41
4.1 Comparison framework . . . 41
4.1.1 Methodology . . . 41
4.1.2 Data Sources Description . . . 42
4.1.3 Triples Generation . . . 44
4.1.4 General Features for Graph Construction . . . 46
4.2 A Unified Model to Represent Mapping Rules: Conceptual Mapping . . . 48
4.2.1 Methodology . . . 49
4.2.2 Requirements . . . 50
Purpose and Scope . . . 50
Mapping Challenges . . . 51
Conceptual Mapping Requirements . . . 52
4.2.3 Implementation . . . 52
Ontology Conceptualization . . . 53
Ontology Design Patterns . . . 56
Ontology Evaluation . . . 56
4.2.4 Publication and Maintenance . . . 57
4.2.5 Extensions . . . 57
CSV Description: CM-CSV . . . 58
RDF-star Generation: CM-star . . . 59
4.2.6 Ontology Usage Example . . . 60
4.3 Featuring New Mapping Needs in RML: Support for RDF-star Construction 64
4.3.1 Reification with RML-star . . . 65
4.3.2 Validation . . . 67
RML-star Test Cases . . . 67
Use Cases . . . 68
4.4 Summary . . . 70

5 Enhancing the Creation of Mapping Rules 73
5.1 Easing Mapping Rule Creation with Spreadsheets . . . 73
5.1.1 Spreadsheet-based Mapping Rules . . . 74
Prefix Sheet . . . 74
Subject Sheet . . . 75
Source Sheet . . . 75
Predicate_Object Sheet . . . 76
Function Sheet . . . 77
5.1.2 Implementation: Mapeathor . . . 77
5.1.3 Evaluation . . . 79
Methodology . . . 79

Results . . . 83
5.1.4 Discussion . . . 83
5.2 User-friendly Creation of Mapping Rules for RDF-star with YARRRML . . . 84
5.2.1 User-friendly Syntax for Mapping Rules . . . 85
YARRRML-star . . . 85
Additional Updates . . . 86
5.2.2 Implementation: Yatter . . . 87
5.2.3 Validation . . . 88
YARRRML Test Cases . . . 89
Serializations Comparison . . . 89
Systems Comparison . . . 92
5.3 Summary . . . 93

6 Representation Impact on Knowledge Graph Refactoring from a Reification Perspective 95
6.1 Motivation . . . 96
6.1.1 Re-construction Example . . . 96
6.1.2 Representation Impact on Knowledge Graph Consumption . . . 97
6.2 Methodology . . . 98
6.2.1 Dataset . . . 98
6.2.2 Mappings and Queries . . . 99
6.2.3 Engines . . . 102
6.2.4 Experimental Setup and Metrics . . . 102
6.3 Results . . . 103
6.4 Discussion . . . 105
6.5 Summary . . . 106

7 Conclusions and Future Work 109
7.1 Contributions Summary and Conclusions . . . 109
7.2 Future Work . . . 113

References 115

Annexes 131
A Conceptual Mapping Requirements . . . 131
B Mapeathor Mappings . . . 132
C Supplementary Tables for KG Refactoring Evaluation . . . 139

List of Figures

2.1 RDF graph example . . . 12
2.2 Approaches for statement reification in RDF . . . 13
2.3 Declarative knowledge graph construction workflow . . . 15
2.4 Structure of Triples Map from R2RML . . . 16
2.5 Structure of Term Map from R2RML . . . 17
2.6 Input "Athletes" table . . . 17
2.7 Existing mapping languages and the relationships among them . . . 21
2.8 Graphical representations approaches in visual mapping editors. . . . 27
2.9 Linked Data life cycle proposed by Ngomo et al., 2014. . . . 29
2.10 Linked Data generation and publishing tasks proposed by Radulovic et al., 2015 30
2.11 Knowledge graph life cycle proposed by Simsek et al., 2021. . . . 31
2.12 Knowledge graph development proposed by Tamašauskaitė and Groth, 2023. 31

3.1 Relations between objectives, contributions, hypotheses, assumptions and restrictions of this thesis . . . 39

4.1 LOT Methodology . . . 50
4.2 Conceptual Mapping ontology overview . . . 54
4.3 CM-CSV module . . . 58
4.4 CM-star module . . . 59
4.5 Example data and ontology about cities for the example mapping . . . 60
4.6 RML-star module . . . 65

5.1 Ontology diagram for the user study exercise of Mapeathor . . . 80
5.2 Expertise of participants of the user study . . . 81
5.3 Accuracy results of user study . . . 82

6.1 KG re-construction alternatives . . . 96
6.2 Overall execution times of KG re-construction evaluation . . . 103
6.3 Execution times of KG re-construction in triplestores . . . 104

List of Tables

2.1 Mapping languages overview 20
2.2 Approaches for easy mapping creation 26

4.1 Comparison framework: Data source description 43
4.2 Comparison framework: Triple generation 45
4.3 Comparison Framework: General features 47

5.1 Prefix sheet. 75
5.2 Subject sheet. 75
5.3 Source sheet. 76
5.4 Predicate_Object sheet. 76
5.5 Function sheet. 77
5.6 Accuracy results and significance of user study 83
5.7 Features of user-friendly serializations 90
5.8 Features of user-friendly serialization systems 93

6.1 Summary results of the representation impact on KG consumption scenarios 98
6.2 SemMedDB scale sizes 99
6.3 Characteristics of evaluation mapping 100
6.4 Characteristics of evaluation SPARQL queries 101
6.5 Execution time results for re-construction evaluation with triplestores 105

A.1 Requirements identified for declarative knowledge graph construction. 131
C.1 Execution times of KG re-construction in KG construction engines 139
C.2 Execution times of KG re-construction in triplestores 140

List of Algorithms

1 Mapeathor translation algorithm 78
2 YARRRML-star translation algorithm 88

Abbreviations and Acronyms

API	Application Programming Interface
CSV	Comma Separated Values
CSVW	CSV on the Web
FTP	File Transfer Protocol
GREL	Google Refine Expression Language
GTFS	General Transit Feed Specification
HTTP	Hypertext Transfer Protocol
HTML	HyperText Markup Language
IRI	Internationalised Resource Identifier
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
JSON-LD	JSON for Linking Data
KG	Knowledge Graph
LD	Linked Data
LOT	Linked Open Terms
MIME	Multipurpose Internet Mail Extensions
NT	N-Triples
ODBC	Open Database Connectivity
OWL	Web Ontology Language
RDB	Relational Database
RDF	Resource Description Format
RDFs	RDF Schema
REST	Representational State Transfer
R2RML	RDB2RDF Mapping Language

RML RDF Mapping Language

ShEx Shape Expressions

SHACL Shapes Constraint Language

SKOS Simple Knowledge Organization System

SPARQL SPARQL Protocol and RDF Query Language

SQL Structured Query Language

TTL Terse RDF Triple Language (Turtle)

UPM Universidad Politécnica de Madrid

URI Uniform Resource Identifier

URL Uniform Resource Locator

XML Extensible Markup Language

XSD XML Schema Definition

W3C World Wide Web Consortium

WoT Web of Things

Chapter 1

Introduction

The nature of the World Wide Web has made it possible for every user and organization across the world to publish and access data on a global scale. The size of the data available on the Web has increased hand-in-hand with its heterogeneity. These data are often provided in different formats (e.g., plain text, CSV or JSON) and with diverse entry points, (e.g., data portals, APIs, or databases). This diversity makes it increasingly difficult to enable large-scale data access and processing for both humans and machines. The Semantic Web (Berners-Lee et al., 2001) was envisioned to represent information as *Linked Data* or *Knowledge Graphs* that can be exploited by both humans and machines, becoming an active research field in the last decades. Multiple Recommendations have been released from the World Wide Web Consortium (W3C) for modelling data in a uniform model (i.e., RDF (Cyganiak et al., 2014)), querying (i.e., SPARQL (Harris et al., 2013)), constructing from other data sources (i.e., R2RML (Das et al., 2012)) and validating against a set of restrictions (i.e., SHACL (Knublauch & Kontokostas, 2017)) among others.

Knowledge graphs (KGs) integrate and convey real-world knowledge, representing entities and how they relate to each other. KGs are usually compliant with the W3C Recommendations, and adequate for downstream consumption applications (Hogan et al., 2021). There are multiple success stories of knowledge graphs being currently used, from publicly available and representing common knowledge or domain-specific knowledge, to privately owned graphs. Largely known and widely used Open Knowledge Graphs include examples containing common knowledge, such as Wikidata (Vrandečić & Krötzsch, 2014), DBpedia (Auer et al., 2007), YAGO (Pellissier Tanon et al., 2020) and Freebase (Bollacker et al., 2007); and about a specific domain, such as libraries (Vila-Suero et al., 2013), scientific articles (Färber et al., 2023; Stocker et al., 2023), tourism (Kärle et al., 2018), cultural heritage (Carriero et al., 2019), life sciences (Dumontier et al., 2014; Piñero et al., 2020), railway infrastructure (Rojas et al., 2021), among many others. Not only within the scientific community, but also companies realized their potential and are taking advantage of them. Therefore, companies such as Google[1], Microsoft (Färber, 2019), LinkedIn[2], Pinterest (Gonçalves et al., 2019), produce their own knowledge graphs.

[1] https://blog.google/products/search/introducing-knowledge-graph-things-not/
[2] https://engineering.linkedin.com/blog/2016/10/building-the-linkedin-knowledge-graph

An essential step for producing knowledge graph able to fulfill their stakeholder's needs is the *construction*. Over the years, a high variety of ways in which knowledge graphs are constructed have been developed (Tamašauskaitė & Groth, 2023): requiring a different level of manual effort, suitable for diverse data sizes and formats, addressing (semi-)structured to unstructured data sources. Even within the systematic construction from semi-structured data (Poggi et al., 2008), there are likewise multiple manners to carry out the task, using (i) programming languages, (ii) visual interactive interfaces, and (iii) declarative mapping rules.

The use of *programming languages* for KG construction is a common practice. For instance, libraries such as RDFLib[3] and Pyoxigraph[4] for Python or Jena[5] for Java enable the source data processing and extraction of relationships to construct KGs. This method brings benefits regarding flexibility to address data heterogeneity, processing and generation. However, it usually supposes an ad-hoc approach restricted for users with specific technical background, and can hinder the maintainability, understandability and reusability of the construction resources (Iglesias-Molina et al., 2019).

However, constructing KGs should not only be restricted to practitioners with technical skills (Karger, 2014). As a result, different *visual interactive interfaces* and methods were developed to facilitate the KG construction process for a wider range of users. For instance, OpenRefine[6] comprises a framework to upload tabular data and manually edit them, including the capability to generate RDF datasets. Semantify.it (Kärle et al., 2017) provides a user-friendly editor to manually annotate data to create structured content with Schema.org (Guha et al., 2016) for web pages. One drawback of this kind of editors is that they cannot usually manage large data sizes[7] (Petrova-Antonova & Tancheva, 2020). In addition, similarly to the *ad-hoc programming* approach, providing manual environments for performing the transformations runs against the reproducibility, maintainability and automation of the KG construction process.

A compromise between these two approaches emerged with *declarative mapping rules*. These mappings define the rules that hold of the correspondences between source data and a target ontology to create a knowledge graph. The main milestone in the progress of these technologies came with the standardization of the RDB2RDF Mapping Language (R2RML) in 2012 (Das et al., 2012). This language, based on previous efforts such as XLWrap (Langegger, 2009), D2RQ (Bizer & Seaborne, 2004) or R2O (Barrasa et al., 2004), focused on describing the transformation of data in relational databases (RDBs) into RDF. Since R2RML started to be adopted and used in real-world scenarios, its limitations became evident. Starting from the limited input data format it is able to describe (RDBs), the first extensions appeared to extend the scope for more heterogeneous data formats (e.g., RML (Dimou et al., 2014), xR2RML (Michel, Djimenou, Zucker, & Montagnat, 2015)). R2RML relies on SQL queries and views for performing data transformation functions over the original data, a feature that

[3] https://rdflib.readthedocs.io/en/stable/
[4] https://pyoxigraph.readthedocs.io/en/latest/
[5] https://jena.apache.org/
[6] https://openrefine.org/
[7] https://github.com/OpenRefine/openrefine.org/issues/136

cannot be applied to other data formats. For this reason, other extensions appeared to allow describing data transformations in the language (e.g., FunUL (Crotti Junior et al., 2016), KR2RML (Slepicka et al., 2015)). Likewise, with new needs and requirements, additional extensions and languages were developed subsequently (e.g., RML-Target (Van Assche et al., 2021), SPARQL-Generate (Lefrançois et al., 2017), ShExML (García-González et al., 2020), SPARQL-Anything (Asprino et al., 2023)). The use of mapping languages over this decade has proven to be a suitable approach for semi-automated KG construction, maintainable in the long term and scalable for large data sizes (Iglesias et al., 2022, 2023; Vidal et al., 2023; Xiao et al., 2020).

Hence, mapping languages provide a means (usually a vocabulary or syntax) to describe these transformation rules declaratively in a file. All languages share a common core of characteristics: (i) the description of the input data, and (ii) the specification on how to generate the output triples in the graph according to the schema of an ontology. The extent, level of detail and variability on how to perform this task is related to the heterogeneity of the proposed languages. They were developed taking into account different needs and use cases, resulting in diverse features, therefore providing a wide set of possibilities for users. Knowing the capabilities of each language can help users decide which one to use depending on their needs. Moreover, despite the efforts over the years to improve and refine the languages, there are still use cases that cannot be solved using them. For instance, in RML, datatypes or language tags can only be generated as constant values, not dynamically from the source data; and there is not much support for generating RDF-star graphs (Hartig, 2017), an RDF extension for allowing triples to be placed as subjects and/or objects of other triples. Understanding their limitations and open issues to address these uncovered use cases can help develop further features for an enhanced KG construction process. However, **there is a lack of comprehensive and extensive studies analyzing the expressiveness of mapping languages in fine-grained detail**, which can greatly help in (i) the choice of a language depending on the use case requirements, and (ii) identifying the challenges and open issues to further improve the languages.

Mapping files are processed by an engine along with the input data to construct KGs. Thus, users are not required to have coding skills, but in turn, they need to learn the mapping language. Most mapping languages are defined as an ontology (e.g., R2RML and its extensions). Therefore, they can use RDF serializations, usually Turtle (Beckett et al., 2014), as the syntax to write the file. Other popular approaches use SPARQL (Harris et al., 2013) as the basis, extending the language to tackle input data description (e.g., SPARQL-Generate (Lefrançois et al., 2017), SPARQL-Anything (Asprino et al., 2023), Tarql[8]). For practitioners with a background in semantic technologies, the barrier to learn the mapping languages is lower, as they probably have previous know-how in these syntaxes. Without this, the learning curve for any non-expert user is high. For this reason, to help in the adoption of these technologies and make them more accessible for non-expert users, several different approaches were developed, ranging from visual approaches (e.g., Karma (Gupta et al., 2012), RMLEditor (Heyvaert et al., 2016)) to user-friendly syntaxes (e.g., YARRRML (Heyvaert et al., 2018), SMS2 (Stardog, 2021)). While these syntaxes present a wide adoption by more experienced practitioners, they

[8] https://tarql.github.io/

still pose a barrier for learning the language's grammar, which may hinder its use by novel users. In addition, visual approaches are limited for complex or large use cases. Hence, further research is needed to **identify approaches that facilitate the writing of mappings, reducing the need to learn a language or syntax, while remaining scalable for large use cases**, in order to facilitate their adoption. In addition, each language is processed by different engines with different capabilities. When dealing with complex use cases or evolving KGs, it is sometimes necessary to use different engines, which may not implement the same language. These kind of situations force the user to learn more than one single language, as there is little support for translating mapping rules among different mapping languages (Corcho et al., 2020). Hence, **it is required to improve the interoperability between existing mapping languages** and bridge the gap between user-friendly mapping creation and existing KG construction implementations.

The technologies revolving mappings are suitable for composing declarative, modular, automated approaches to be integrated into larger pipelines that manage knowledge graphs (Cimmino & García-Castro, 2024; Grassi et al., 2023; Simsek et al., 2021). Additionally, these declarative approaches can play an important role in other different tasks of the life cycle apart from the construction. For instance, they enable metadata annotation about the data sources, which can help enhance the process in terms of transparency, completeness and performance (Chaves-Fraga et al., 2021; Vidal et al., 2023); as well as data processing and cleaning, thanks to the inclusion of data transformation functions in the language (Crotti Junior et al., 2016; De Meester et al., 2017; Debruyne & O'Sullivan, 2016; Jozashoori et al., 2020). However, **there is a lack of research to assess how these technologies can play a beneficial role in the evolution of knowledge graphs**, beyond their construction.

1.1 Main Contributions

The overall objective of this thesis is to improve the understanding and operational management of declarative KG construction languages. The contributions of this thesis are listed below, organized according to the main thesis sub-objectives:

1. The first objective consists on gathering, understanding and implementing the current needs for knowledge graph construction from heterogeneous data sources with mapping languages.
    - **A comparison framework with a fine-grained analysis of the characteristics of current mapping languages.** We design a framework with the features that a mapping language may provide, from data source description to triple transformation and additional rules that apply to triple creation, and check whether current mapping languages provide such features in the language itself or supported by a compliant engine.
    - **A set of requirements for declarative mapping languages**, extracted from the comparison framework and the needs of the community of practitioners. These requirements are implemented in a formal language as an ontology.
    - **Update the RML mapping language with new requirements.** We present

how some of these requirements have served to update RML, providing a novel solution for constructing RDF-star graphs.

2. The second objective consists on improving the user experience for knowledge engineers and domain experts to write mapping rules.
    - **A spreadsheet-based approach to write mapping rules**. This approach enables users to write mapping rules in well-known spreadsheet editors (e.g., MS Excel, Google Spreadsheets) without the need to learn a language's constructs or syntax, to be automatically translated into a correctly formatted mapping file with the support of a compliant tool.
    - **The update of the user-friendly serialization YARRRML for RML**, incorporating the latest features of the language, along with a compliant tool to translate them into human-readable [R2]RML mapping files.
3. The third objective consists on assessing the role of declarative KG construction technologies to support the evolution of knowledge graphs.
    - **An evaluation of how declarative KG construction technologies can be beneficial when the schema of a KG changes.** We test these approaches for re-constructing a knowledge graph, comparing them with triplestore-based re-construction. We assess the performance of each approach and the features that intervene in different scenarios. This study sheds light on how KG construction technologies can play a role in supporting the refactoring of knowledge graphs.

1.2 Thesis Structure

The remainder of this thesis is structured as follows:

- Chapter 2 describes the main concepts used in this thesis, reviews the state of the art on the topics of interest and identifies the current limitations. It first introduces the basic notions of knowledge representation on the Web, to then proceed to describe the current landscape of mapping languages and methods to write them in a user-friendly manner. Then it presents related work on assessing the role of these declarative mappings in the different phases that take place within the knowledge graph life cycle. The chapter concludes identifying the gaps in the state of the art, leading to the work towards obtaining the main contributions of the thesis.
- Chapter 3 presents the main objectives and contributions of this thesis, along with the assumptions, hypotheses and restrictions of the work and research methodology.
- In Chapter 4, we define a comparison framework to analyze the features of current mapping languages. This framework helps to extract a set of requirements for knowledge graph construction, that are gathered and represented in an ontology. Additionally, we present a particular case of the update of the RML mapping language to address some of the extracted requirements, specifically the RML-star module to construct RDF-star graphs.

- In Chapter 5, we present two approaches to facilitate the creation of mapping documents for users. The first relies on spreadsheets to write mapping rules, and it is tested with users with different backgrounds to evaluate its usability. The second proposes an update on the YARRRML user-friendly syntax with respect to recent modifications in the RML language. Both approaches are supported by tools to create mapping documents in a set of target languages.
- In Chapter 6, we assess the role that mapping-compliant technologies can play in knowledge graph evolution. We conduct an empirical evaluation performing schema changes with KG construction engines that use mappings, and SPARQL `CONSTRUCT` queries, and evaluate which approach is more suitable in different situations.
- Finally, Chapter 7 draws the main conclusions of the presented work, and outlines the future research directions.

1.3 Derived Outcomes

This section lists the publications derived from the work in this thesis.

- **Journal Publications**
  - Arenas-Guerrero, J.*, **Iglesias-Molina, A.***, Chaves-Fraga, D., Garijo, D., Corcho, O. and Dimou, A. (2024) Declarative generation of RDF-star graphs from heterogeneous data. *Semantic Web*, in press. This work presents and validates the latest version of RML-star, described in Chapter 4.
  - **Iglesias-Molina, A.**, Cimmino, A., Ruckhaus, E., Chaves-Fraga, D., García-Castro, R., Corcho, O. (2024) An Ontological Approach for Representing Declarative Mapping Languages. *Semantic Web*, *5* (1), 191–221. This work analyzes the mapping languages capabilities and limitations, gathers the requirements for KG construction and presents their formalization as an ontology, described in Chapter 4.
- **Conference Publications**
  - **Iglesias-Molina A.**, Toledo J., Corcho O. and Chaves-Fraga D. (2023) Re-Construction Impact on Metadata Representation Models. In *Proceedings of The Twelfth International Conference on Knowledge Capture (K-CAP23)*, December 5–7, Pensacola. This work assesses the value of declarative KG construction in the evolution of the KG's schemas, described in Chapter 6.
  - **Iglesias-Molina, A.**, Van Assche, D., Arenas-Guerrero, J., De Meester, B., Debruyne, C., Jozashoori, S., Maria, P., Michel, F., Chaves-Fraga, D. and Dimou, A. (2023) The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF. In *Proceedings of the 22nd International Semantic Web Conference (ISWC2023)*, November 6–10, Athens. This work presents the updates in RML with the new needs and requirements identified for declarative KG construction, described in Chapter 4.

  - **Iglesias-Molina, A.**, Ahrabian, K., Ilievski, F., Pujara, J. and Corcho, O. (2023) Comparison of Knowledge Graph Representations for Consumer Scenarios. In *Proceedings of the 22nd International Semantic Web Conference (ISWC2023)*, November 6–10, Athens. This work motivates the contribution, presented in Chapter 6.
  - **Iglesias-Molina, A.**, Chaves-Fraga, D., Dasoulas, I. and Dimou, A. (2023) Human-Friendly RDF Graph Construction: Which One Do You Chose?. In *Proceedings of the 23rd International Conference on Web Engineering 2023 (ICWE2023)*, June 6–9, Alicante. This work updates the YARRRML syntax with the new features, described in Chapter 5.

- **Workshop Publications**
  - **Iglesias-Molina, A.**, Cimmino, A., Corcho, O. (2022) Devising Mapping Interoperability with Mapping Translation. In *Proceedings of the Third International Workshop on Knowledge Graph Construction, co-located with the 19th Extended Semantic Web Conference*, May 29 – June 2, Hersonissos. This works looks into the interoperability status of current mapping languages, presented in Chapter 2.
  - **Iglesias-Molina, A.**, Chaves-Fraga, D., Priyatna, F. and Corcho, O. (2019) Towards the Definition of a Language-independent Mapping Template for Knowledge Graph Creation. In *Proceedings of the Third International Workshop on Capturing Scientific Knowledge co-located with the Eleventh International Conference on Knowledge Capture*, November 19–21, Marina del Rey. This work presents the first version of the spreadsheet-based approach for writing declarative mapping rules, described in Chapter 5.

- **Posters and demos**
  - **Iglesias-Molina, A.** and Garijo D. (2023) Towards Assessing FAIR Research Software Best Practices in an Organization Using RDF-star. In *Proceedings of the Semantics 2023 Posters and Demos Track*, September 19–22, Leipzig. This work presents a use case that uses RML-star, described in Chapter 4.
  - Delva, T., Arenas-Guerrero, J., **Iglesias-Molina, A.**, Corcho, O., Chaves-Fraga, D., and Dimou, A. (2021) RML-star: A declarative mapping language for RDF-star generation. In *Proceedings of the ISWC 2021 Posters, Demos and Industry Tracks*, October 24–28, online. This work presents the first version of RML-star, described in Chapter 4.
  - **Iglesias-Molina, A.**, Pozo-Gilo, L., Dona, D., Ruckhaus, E., Chaves-Fraga, D. and Corcho, O. (2020) Mapeathor: Simplifying the Specification of Declarative Rules for Knowledge Graph Construction. In *Proceedings of the ISWC 2020 Demos and Industry Tracks*, November 2–6, online. This work refines the spreadsheet-based approach for writing declarative mapping rules and presents its implementation in a tool described in Chapter 5.

1.4 Related Research Activities

Research Stay

During the development of this thesis, one research stay took place in the following institution:

- 06/07/2022 – 06/10/2022. Research stay at the **Information Sciences Institute of the University of Southern California**, supervised by Prof. Dr. Filip Ilievski. During this stay, we analyzed the differential impact of diverse knowledge graph representation over different consumption scenarios: knowledge exploration performed by users, systematic query performance and graph completion tasks. This stay was funded by a scholarship from Programa Propio I+D+i of UPM, oriented to research personnel in predoctoral formation for doing an international research stay. This stay lead to the publication "Comparison of Knowledge Graph Representations for Consumer Scenarios", presented in the International Semantic Web Conference (ISWC) 2023 (Iglesias-Molina, Ahrabian, et al., 2023).

Organization of Workshops and Tutorials

As part of the research activity and interest in contributing further to this community, the author has participated in the organization of the following events:

- Organization of the **Fourth and Fifth Editions of the Knowledge Graph Construction Workshop (KGCW)**[9,10], co-located with the Extended Semantic Web Conference (ESWC 2023 and 2024), held both times in Hersonissos, Greece, organized together with Anastasia Dimou (KU Leuven), David Chaves-Fraga (Universidade de Santiago de Compostela), Umutcan Serles (STI Innsbruck) and Dylan Van Assche (Universiteit Gent). This workshop gathers in every edition the community of KG Construction to present relevant research, breakthroughs and resources in the field, complementing the activities of the Knowledge Graph Construction W3C Community Group. The number of attendees was around 40-60 people.
- Organization of the **Tutorial on Knowledge Graph Construction using Declarative Mapping Rules**[11], co-located with the 19th International Semantic Web Conference (ISWC2020), held on November 2-6 2020 online, organized together with Oscar Corcho, David Chaves-Fraga and Andrea Cimmino (UPM); and the **Tutorial on Declarative Construction and Validation of Knowledge Graphs**[12], co-located with the 12th International Conference on Knowledge Capture (K-CAP2023), held on December 5-7 2023 in Pensacola, USA, organized together with Xuemin Duan (KU Leuven). Presenter in the **Knowledge Graph Construction Tutorial**[13], co-located with the 19th Extended Semantic Web Conference (ESWC2022), held on May 29 – June 2 2022. All tutorials focused on explaining in detail, from a practical perspective, the

[9] https://w3id.org/kg-construct/workshop/2023
[10] https://w3id.org/kg-construct/workshop/2024
[11] https://oeg-dataintegration.github.io/kgc-tutorial-2020/
[12] https://w3id.org/kg-construct/tutorials/kcap2023
[13] https://w3id.org/kg-construct/costdkg-eswc-tutorial

process of constructing knowledge graphs, from writing mappings to their execution, publication and validation with suitable tools. The number of attendees in all tutorials was around 20-30 people.

Chapter 2

State of the Art

In this chapter, we discuss the state of the Art on the topics of interest for this thesis, so as to provide an overview of recent developments in the field as well as to identify the open research problems and challenges in the area, some of which will be addressed in this thesis. The chapter starts describing some of the essential concepts as background information for the reader, about knowledge representation and annotation in the Web in Section 2.1. Then, in Section 2.2, we focus on describing current techniques to construct knowledge graphs, particularly with declarative approaches. The challenges in the adoption of declarative KGC approaches lead us to present the user-friendly approaches developed to facilitate the creation of the mapping documents in Section 2.3. Next, Section 2.4 introduces several knowledge graph life cycle proposals, analysing in which stages declarative mappings can play a beneficial role for its management. The chapter concludes with Section 2.5 drawing the conclusions from each area of research and its main gaps, introducing how this thesis contributes to advance the state of the art.

2.1 Knowledge Representation in the Semantic Web

This section describes the concepts that play a relevant role in knowledge representation on the Web, hence also relevant to its creation. First we present the Resource Description Framework (RDF) (Cyganiak et al., 2014), used as backbone of knowledge graphs and also for specifying declarative transformation rules (Section 2.1.1). Then, we discuss different approaches for annotating statements in RDF to add additional information (Section 2.1.2).

2.1.1 Resource Description Framework (RDF)

The Resource Description Framework (RDF) (Cyganiak et al., 2014; Lassila & Swick, 1999) is a standard model for data interchange on the Web. Its basic unit are triples: two resources linked by a relationship. These triples are represented in the form of $< s,p,o >$, where s is the subject, p is the predicate and o the object. This model relies on the use of the linking structure of the Web, using IRIs to identify uniquely every resource and relationship.

Listing 2.1 presents an example of an RDF graph composed of a set of triples about

pole vault records, which is also visually depicted in Fig. 2.1. In total there are three triples. They all share the same subject, which is a resource uniquely identified by the IRI <http://example.com/athlete/1>. Likewise, all predicates in the triples are also defined by an IRI, e.g., <http://example.com/ns#name>. The first triple (Line 5) defines that the subject is an instance of the class ex:Athlete with the predicate rdf:type. The rest of the triples define attributes of this instance with name (ex:name) and birth date (ex:birthdate). The objects of both triples are literals, that may be strings ("Wilma Murto", Line 6) or typed literals ("1998-06-11"^^xsd:date, Line 7). Type literals refer to data values attached with a tag that represents their data type.

```
1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
2 @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
3 @prefix ex: <http://example.com/ns#>.
4
5 <http://example.com/athlete/1> rdf:type ex:Athlete .
6 <http://example.com/athlete/1> ex:name "Wilma Murto" .
7 <http://example.com/athlete/1> ex:birthdate "1998-06-11"^^xsd:date .
```

Listing 2.1: Example of RDF graph in Turtle serialization.

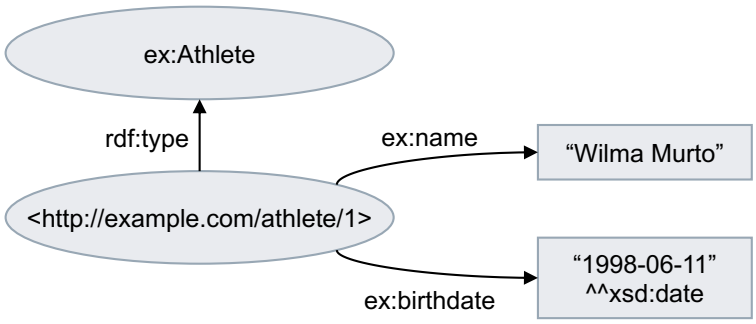


Figure 2.1: Visual representation of the RDF graph shown in Listing 2.1.

There are different syntaxes to serialize RDF graphs. RDF/XML (Gandon & Schreiber, 2014) was the first to be proposed, and relies on XML. Notation3 (N3) (Berners-Lee & Connolly, 2011) was developed as a human readable syntax, but its use is not common. Instead, the N-Triples (Carothers & Seaborne, 2014) and Turtle (Beckett et al., 2014) serializations, subsets of N3, are more widely used. JSON-LD (Kellogg et al., 2020) was developed for programming environments, as it is written in JSON documents. Lastly, RDFa (Herman et al., 2017) extends HTML to markup structured content in webpages, with the objective of improving the results retrieved by search engines.

2.1.2 Statements About Statements in RDF

Plain triples are not always able to represent the complexity of all knowledge. This is the case of statement annotation, when it is required to add information to a triple that does not correspond to a single resource, but the entire statement. This situation triggered the

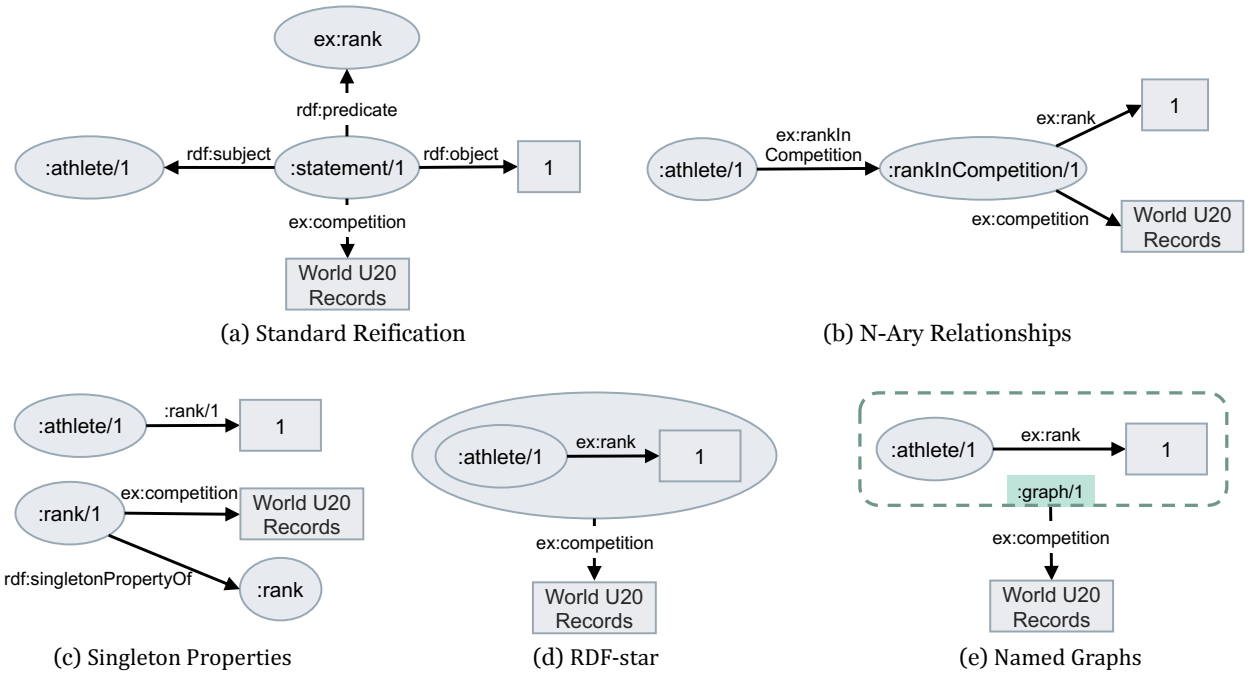


Figure 2.2: Approaches for statement reification in RDF: (a) Standard Reification, (b) N-Ary Relationships, (c) Singleton Properties, (d) RDF-star and (e) Named Graphs.

development of different approaches to enable triple annotation (also known as reification). Fig. 2.2 illustrates instances of these models for the main statement *Wilma Murto obtained the first position in the rank*, annotated with the additional statement *in the World U20 Records competition.*

Standard Reification (Lassila & Swick, 1999) explicitly declares a resource to denote an `rdf:Statement`. This statement has `rdf:subject`, `rdf:predicate`, and `rdf:object` attached to it and can be further annotated with additional statements. The resource is typically a blank node, but an IRI can be used. In Fig. 2.2a, the resource `:statement/1` is an `rdf:Statement` with four associated triples, where the objects of these triples are the resources of the original triple (i.e., `:athlete/1` for the subject, `ex:rank` for the predicate, and `1` for the object). The property `ex:competition` is used with its own value as object.

N-Ary Relationships (Noy & Rector, 2006) converts a relationship into an instance that describes the relation, which can have attached both the main object and additional statements. This representation is widely used in ontology engineering as an ontology design pattern (Gangemi & Presutti, 2013). In Fig. 2.2b, the entity `:athlete/1` points to an intermediate node (`:rankInCompetition/1`) which holds the triples for both the position in rank and competition when it took place.

Singleton Properties (Nguyen et al., 2014) uses unique predicates in the main triple. This unique predicate can then become the subject of the additional triple. The unique predicate is linked to the original predicate using `rdf:singletonPropertyOf`. In Fig. 2.2c, the main

triple uses the predicate `:rank/1`, the unique version of the original `ex:rank`. This predicate is then annotated with the competition of the ranking.

RDF-star (Hartig, 2017; Hartig et al., 2023) extends RDF with a new syntax for compact triple reification. It introduces the notion of triple recursiveness with `Quoted Triples`, which can be used as subjects and/or objects of other triples. This is the only approach that extends the standard RDF features. This representation is being incorporated into the RDF 1.2 specification (Hartig et al., 2023), which is being developed under the RDF-star W3C Working Group[1]. In Figure 2.2d we observe the example as an RDF-star graph, and it is represented in RDF as `«:athlete/1 ex:rank 1» ex:competition "World U20 Records"`.

Named Graphs (Prud'hommeaux & Seaborne, 2008) are a SPARQL feature that allows the assignment of an IRI to one or several triples as a graph identifier. Hence, graph IRIs allow the unique identification of triples. These IRIs can be used as subjects to add additional statements. In Fig. 2.2e, the main triple indicating the rank position of an athlete is assigned the named graph `:graph/1`. The graph IRI is subsequently used as subject in a triple that adds information about the triple within the graph.

2.2 Knowledge Graph Construction with Declarative Mapping Rules

Knowledge graphs can be constructed in diverse manners. Some KGs are constructed natively in RDF by collecting knowledge from contributions of the community, such as Wikidata. Other KGs are constructed by transforming data from heterogeneous formats and sources, unstructured or (semi-)structured, into RDF. This section provides an overview of the latter, with a focus on declarative KG construction from heterogeneous data sources.

Constructing RDF knowledge graphs from heterogeneous data sources involves a schema transformation of the data to the desired graph structure. This transformation may be done in different ways, usually involving mapping languages that allow expressing the transformation rules to create the target graphs. Hence, these mappings hold declaratively the relationships between the source and target data schemas. This comprises a domain-agnostic approach that can (and has been) applied to multiple different use cases, such as agriculture (Bilbao-Arechabala & Martínez-Rodríguez, 2022), public procurement (Soylu et al., 2022), biomedicine (Aisopos et al., 2023; Iglesias-Molina et al., 2019; Michel et al., 2020), cultural heritage (Calvanese et al., 2016), IoT (Cimmino et al., 2020; González-Gerpe et al., 2022), among others.

The usual workflow in which declarative approaches are involved is depicted in Fig. 2.3. They enable both materialization and virtualization of knowledge graphs (Poggi et al., 2008). In materialization scenarios, data is transformed into the target graph, usually following the schema provided by an ontology (Arenas-Guerrero, Chaves-Fraga, et al., 2024; Iglesias et al., 2020; Lefrançois et al., 2017). In virtualization scenarios, data is not transformed; instead, the original data source is accessed also following the schema of the target (virtual) graph.

[1] https://www.w3.org/groups/wg/rdf-star/

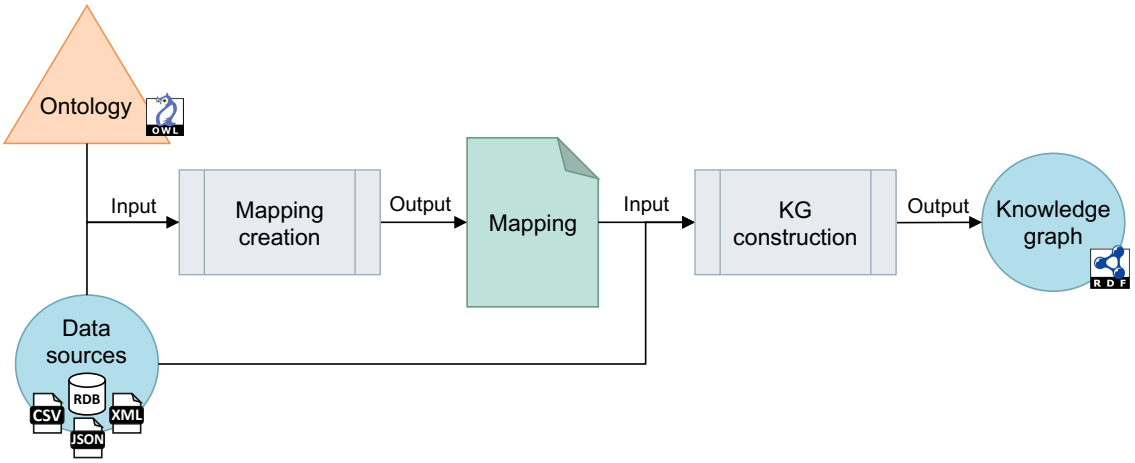


Figure 2.3: Declarative knowledge graph construction workflow.

This process requires translating the original query into the language of the original data source (Calvanese et al., 2017; Priyatna et al., 2014; Rodríguez-Muro & Rezk, 2015).

This section presents an overview of existing mapping languages. We first show two well known RDF-based mapping languages that are widely adopted: the W3C Recommendation R2RML (Section 2.2.1), its extension, RML (Section 2.2.2); and then other relevant languages (Section 2.2.3). Lastly, we discuss the approaches that compare the diversity and expressiveness of these mapping languages (Section 2.2.4), and the current interoperability among them (Section 2.2.5).

2.2.1 R2RML: RDB to RDF Mapping Language

R2RML (Das et al., 2012) addresses the transformation of data in relational databases into RDF. This language specifies how to write the transformation rules in R2RML mapping documents. An R2RML mapping document is composed by at least one *Triples Map* (`rr:TriplesMap`). *Triples Maps* are used to define the transformation rules to generate RDF triples, their structure is depicted in Fig. 2.4. It contains:

- One *Logical Table* (`rr:LogicalTable`), that describes the source relational table or view of a database.
- One *Subject Map* (`rr:SubjectMap`), that defines how to generate the subject of the triples, and optionally, the corresponding class(es) it may be instance of (`rr:class`).
- Zero to multiple *Predicate Object Maps* (`rr:PredicateObjectMap`). A *Predicate Object Map* (POM) is formed by one to multiple *Predicate Maps* (`rr:PredicateMap`), and one to many *Object Maps* (`rr:ObjectMap`).

Graph Maps (`rr:GraphMap`) indicate how to generate named graphs. Zero or more *Graph Maps* can be assigned to both *Subject Maps* and *Predicate Object Maps*. It is also possible to join *Logical Tables* by replacing an *Object Map* for a*Referencing Object Map* (`rr:RefObjectMap`), which uses the subject of another *Triples Map*, indicated with `rr:parentTriplesMap`, as the

object of the triple. This join may have a condition to be performed, which is indicated using `rr:joinCondition`, `rr:child`, and `rr:parent`. Additionally, *Object Maps* may include additional information, such as datatypes (`rr:datatype`) and languages (`rr:language`) of literals.

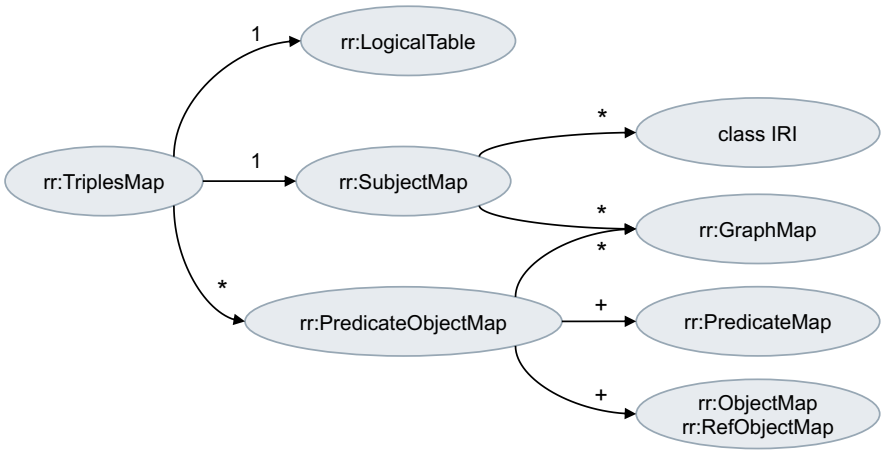


Figure 2.4: Structure of a *Triples Map* and its properties (adapted from Das et al., 2012).

Subject Map, *Predicate Map*, *Object Map*, and *Graph Map* are subclasses of *Term Map* (`rr:TermMap`), which define how to generate RDF terms. *Term Maps* can be (i) *constant-valued* (`rr:constant`), i.e., always generating the same RDF term; (ii) *column-valued* (`rr:column`), i.e., the RDF terms are directly obtained from cells of a column in the RDB; or (iii) *template-valued* (`rr:template`), i.e., the RDF terms are composed from the data in columns and constant strings. These terms can be generated either as IRIs (`rr:IRI`), Blank Nodes (`rr:BlankNode`) or literals (`rr:Literal`) using `rr:termType`. The structure of a *Term Map* is shown in Fig. 2.5.

Listing 2.2 shows an example of an R2RML mapping to construct the RDF graph in Listing 2.3. The R2RML mapping contains one *Triples Map*, "TriplesMapAthletes" (Line 5). It takes as input the "Athletes" table shown in Fig. 2.6 (Line 6), which contains information about pole vault athletes, including their name, position in the ranking and height of a jump. The subject is formed with a template using one data source column (Line 8). The set of *Predicate Object Maps* are used to describe with further attributes each instance (Lines 11-26). Two of them add the datatype to the object, one as an integer (Line 19) and other as a float (Line 25).

```
1  @prefix rr: <http://www.w3.org/ns/r2rml#>.
2  @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
3  @prefix ex: <http://example.com/ns#>.
4
5  <#TriplesMapAthletes>
6      rr:logicalTable [ rr:tableName "Athletes" ];
7      rr:subjectMap [
8          rr:template "http://example.com/athlete/{RANK}";
9          rr:class ex:Athlete;
10     ];
```

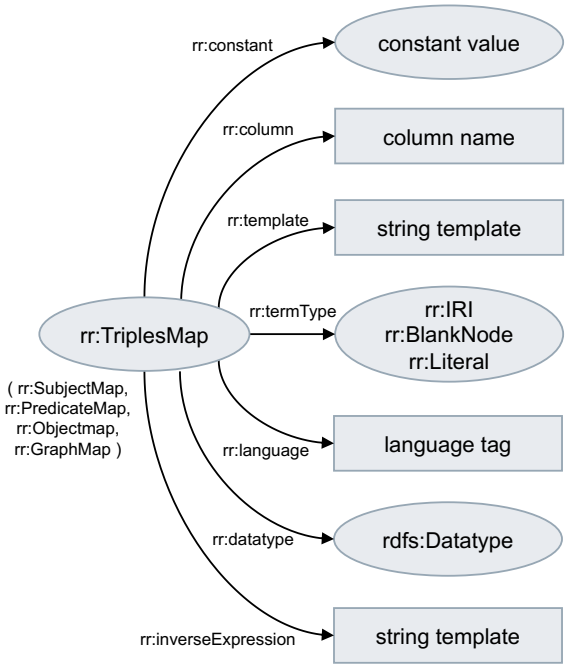


Figure 2.5: Structure of a *Term Map* and its properties (adapted from Das et al., 2012).

NAME	RANK	MARK
Armand Duplantis	1	6.22
Sondre Guttormsen	2	6.00
Menno Vloon	3	5.91

Figure 2.6: Input "Athletes" table with names, ranks and marks for athletes.

```
11      rr:predicateObjectMap [
12          rr:predicate ex:name;
13          rr:objectMap [ rr:column "NAME" ];
14      ];
15      rr:predicateObjectMap [
16          rr:predicate ex:rank;
17          rr:objectMap [ rr:column "RANK" ; rr:datatype xsd:integer ];
18      ];
19      rr:predicateObjectMap [
20          rr:predicate ex:mark;
21          rr:objectMap [ rr:column "MARK" ; rr:datatype xsd:float ];
22      ].
```

Listing 2.2: R2RML mapping to generate the RDF graph in Listing 2.3 with data from the database table shown in Fig. 2.6.

```
<athlete/1> a ex:Athlete ;
    ex:name "Armand Duplantis" ;
    ex:rank "1"^^xsd:integer ;
    ex:mark "6.22"^^xsd:float .
<athlete/2> a ex:Athlete ;
    ex:name "Sondre Guttormsen" ;
    ex:rank "2"^^xsd:integer ;
    ex:mark "6.00"^^xsd:float .
<athlete/3> a ex:Athlete ;
    ex:name "Menno Vloon" ;
    ex:rank "3"^^xsd:integer ;
    ex:mark "5.91"^^xsd:float .
```

Listing 2.3: RDF graph resulting from transforming data from the database table shown in Fig. 2.6 using the R2RML mapping in Listing 2.2.

2.2.2 RML: RDF Mapping Language

RML (Dimou et al., 2014) extends R2RML to address the transformation of heterogeneous data formats, such as CSV, XML or JSON. It broadens the possibilities for KG construction with minimal changes to R2RML's vocabulary. This language rapidly gained importance, and currently gathers an active community of users and developers.

The main differences between RML and R2RML (Dimou, 2020) are listed as follows:

- **Data source reference**. RML introduces the *Logical Source* (`rml:LogicalSource`), that extends R2RML's *Logical Table*. The *Logical Table* focuses on relational databases, whereas the *Logical source* enables the description of more heterogeneous data sources in addition to relational databases.
- **Data source language**. In R2RML, the data source language is implicitly assumed to be SQL. To address heterogeneous data formats, RML uses the *Reference Formulation* (`rml:ReferenceFormulation`) to indicate which formulation is suitable to parse data in a specific format. For instance, the JSONPath formulation can be used for parsing data in a JSON file.
- **Iteration**. For tabular data sources, the iteration of the data source is per row. This strategy is followed by every R2RML processor to construct RDF graphs from RDBs. For non-tabular data sources, the iteration cannot always be implicitly assumed. For this reason, RML introduces the *Iterator* (`rml:iterator`) enabling the explicit determination of the iteration pattern to specify the specific data used in each iteration to build the RDF triples. The use of the *Iterator* is optional, its use relies on the needs and particularities of each data source and use case.
- **Value reference**. R2RML relies on column names when referring to values in a table or view of a RDB (`rr:column`). RML broadens the scope of this concept using *References* (`rml:reference`), that may refer to columns, objects (such as in the case of XML and JSON) or other elements valid with respect to the used *Reference Formulation*.

Listing 2.5 shows an example RML mapping that generates the same output RDF graph (Listing 2.3) as the R2RML mapping previously shown in Listing 2.2. Instead of a database table, this mapping takes as input a JSON file (Listing 2.4), which is described with the *Logical Source)* (Lines 6-19). The fields of the input JSON file are referred to using the `rml:reference` property.

```
1 [ { "NAME": "Duplantis",
2     "RANK": "1",
3     "MARK": "6.22"},
4   { "NAME": "Guttormsen",
5     "RANK": "2",
6     "MARK": "6.00"},
7   { "NAME": "Vloon",
8     "RANK": "3",
9     "MARK": "5.91"} ]
```

Listing 2.4: Input "Athletes.json" file with names, ranks and marks for athletes.

```
1  @prefix rr: <http://www.w3.org/ns/r2rml#>.
2  @prefix rml: <http://semweb.mmlab.be/ns/rml#>.
3  @prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
4  @prefix ex: <http://example.com/ns#>.
5  @prefix ql: <http://semweb.mmlab.be/ns/ql#>.
6
7  <#TriplesMapAthletes>
8    rml:logicalSource [
9      rml:source "Athletes.json";
10     rml:referenceFormulation ql:JSONPath;
11     rml:iterator "$.*"
12   ];
13     rr:subjectMap [
14         rr:template "http://example.com/athlete/{RANK}";
15         rr:class ex:Athlete;
16     ];
17     rr:predicateObjectMap [
18         rr:predicate ex:name;
19         rr:objectMap [ rml:reference "NAME" ];
20     ];
21     rr:predicateObjectMap [
22         rr:predicate ex:rank;
23         rr:objectMap [ rml:reference "RANK" ; rr:datatype xsd:integer ];
24     ];
25     rr:predicateObjectMap [
26         rr:predicate ex:mark;
27         rr:objectMap [ rml:reference "MARK" ; rr:datatype xsd:float ];
28     ].
```

Listing 2.5: RML mapping to generate the RDF graph in Listing 2.3 with data from the JSON file shown in Listing 2.4.

Table 2.1: Analyzed mapping languages and their corresponding references.

Classification	Language	Reference(s)
RDF-based	R2RML	(Das et al., 2012)
	RML	(Dimou et al., 2014, 2020)
	D2RQ	(Bizer & Seaborne, 2004; Cyganiak et al., 2012)
	XLWrap	(Langegger, 2009; Langegger & Wöß, 2009)
	CSVW	(Tennison et al., 2015)
	KR2RML	(Slepicka et al., 2015)
	xR2RML	(Michel, Djimenou, Zucker, & Montagnat, 2015; Michel et al., 2017)
	R2RML-F	(Debruyne & O'Sullivan, 2016)
	FunUL	(Crotti Junior et al., 2016)
	R2RMLcc	(Debruyne et al., 2017)
	D2RML	(Chortaras & Stamou, 2018)
	WoT mappings	(Cimmino et al., 2020)
SPARQL-based	XSPARQL	(Bischof et al., 2012; Polleres et al., 2009)
	TARQL	(TARQL, 2019)
	SPARQL-Generate	(Lefrançois et al., 2017, 2022)
	Facade-X	(Daga et al., 2021; SPARQL-Anything, 2023)
	NORSE	(Stadler et al., 2023)
Alternative schema	R_2O	(Barrasa et al., 2004)
	D-REPR	(Vu et al., 2019)
	T2WML	(Szekely et al., 2019)
	ShExML	(García-González, 2022; García-González et al., 2020)
	Helio mappings	(Cimmino & García-Castro, 2024)

There are several proposals to extend the capabilities of RML. The initial features for addressing heterogeneous data sources were expanded by Dimou et al., 2015, allowing more diverse data accesses and fine-grained description of the data sources. De Meester et al., 2017 proposed **RML+FnO**, which provided RML with a connector to the Function Ontology (De Meester et al., 2016) to include data transformation functions in the mappings. Delva, Van Assche, et al., 2021 introduced **RML Fields**, that addressed several challenges to accurately access nested data. **RML Target** was developed by Van Assche et al., 2021 to specify how to create the output triples regarding, for instance, RDF serialization, compression format or encoding. Lastly, **RML-star** was proposed by Delva, Arenas-Guerrero, et al., 2021, and later refined by Arenas-Guerrero, Iglesias-Molina, et al., 2024, to allow the construction of RDF-star graphs.

2.2.3 Additional Mapping Languages

Following, we present an overview of existing mapping languages, listed in Table 2.1 and depicted in Fig. 2.7. This section presents additional relevant languages classified in three categories, depending on the model they are based on or extend: (i) RDF-based, (ii) SPARQL-based, and (iii) based on alternative schemas.

RDF-based Mapping Languages.

Mapping languages in this group are formalized using RDF. They are usually modelled as ontologies or vocabularies that enable the description of data transformation rules. Mappings

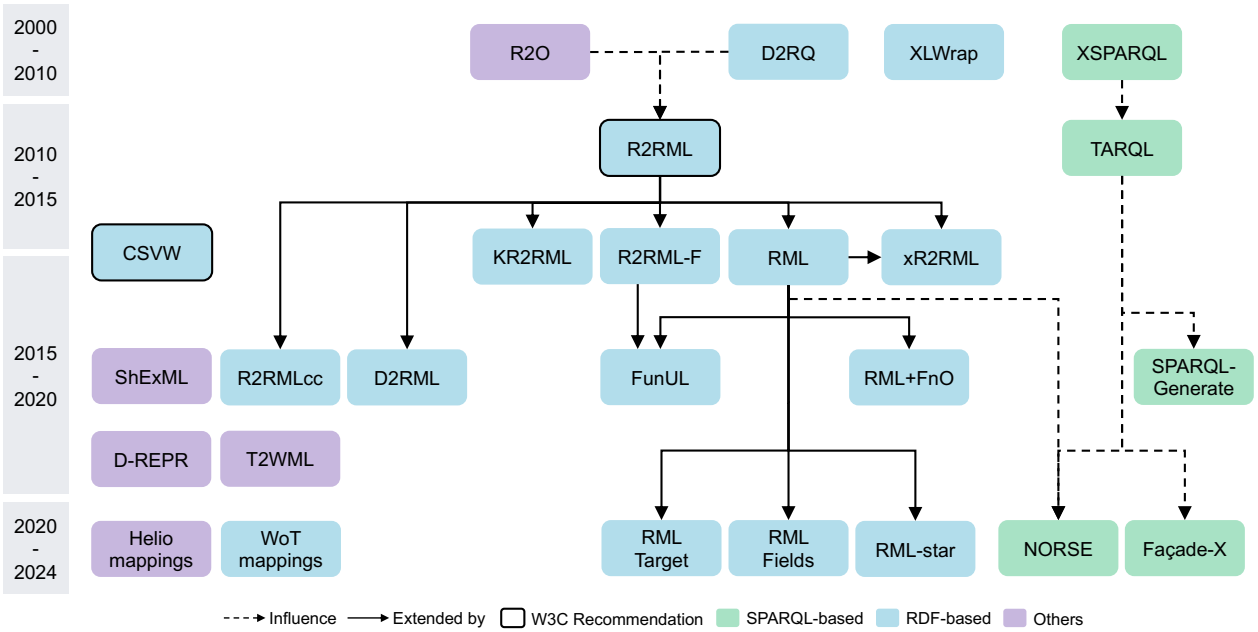


Figure 2.7: Mapping languages release and extensions in the last two decades and relationships among them, classified according to the schema they are based on. Adapted and extended from Chaves Fraga, 2021 and Iglesias-Molina et al., 2024.

in these languages are usually written as files that follow the Turtle syntax (Beckett et al., 2014). R2RML and RML, previously presented in Section 2.2.1 and Section 2.2.2 respectively, also belong to this group of languages.

D2RQ (Bizer & Seaborne, 2004) is one of the first mapping languages, developed to access the contents of relational databases. The main components of this language are the *Class Map* (`d2rq:ClassMap`) and the *Property Bridge* (`d2rq:PropertyBridge`) (Cyganiak et al., 2012). A *Class Map* specifies how the instances of a class are generated, indicating the class that the instances belong to, the database's column(s) and URI pattern to create the URI for the instances. Each *Class Map* also specifies the access to the input database, and refer to *Property Bridges*. These *Property Bridges* are used to generate the pairs of predicate-objects for the *Class Map*, specifying the property for the predicate, and patterns and column references for the object.

XLWrap (Langegger & Wöß, 2009) enables the transformation of spreadsheets to RDF. These mappings are written using the TriG syntax. They contain a graph instance of `xl:Mapping`, that describes the input file (workbook) (`xl:fileName`) and worksheet (`xl:sheetName`), a constant graph with information about the mapping (`xl:constantGraph`), references to *Template Graphs* (`xl:templateGraph`) and *Transformation Operations* (`xl:transformationOperation`). The *Template Graphs* specify how to create the desired RDF graphs with references from the spreadsheet's cells and XLWrap expressions. This language provides further features to accurately access data in spreadsheets, described in the specification (Langegger, 2009).

CSV on the Web (CSVW) (Tennison et al., 2015) is a W3C Recommendation that enables

RDF generation providing a fine-grained annotation of CSV files and other tabular data formats, such as datatypes, valid values, data transformations, and primary and foreign key constraints.

KR2RML (Slepicka et al., 2015) extended R2RML to: (i) address multiple input data formats, such as CSV, JSON and XML; (ii) select the RDF serialization to create the output graph and (iii) integrate data transformation functions for pre-processing messy data without relying on SQL queries or views. To this end, the authors developed the nested Relational Model (NRM), as an intermediate form to represent data. NRM allowed the unification of the way to access data without loosing expressiveness.

xR2RML (Michel, Djimenou, Zucker, & Montagnat, 2015) extends R2RML and RML to include additional heterogeneous data sources and additional features not usually tackled by other mapping languages. It extends RML's *Logical Source* to include NoSQL databases (e.g., MongoDB), and refines the access to different levels of iterations for hierarchical data sources with `xrr:pushDown`. It enables the creation of RDF lists and containers, by including additional *Term Map* types (`xrr:RdfList`, `xrr:RdfSeq`, `xrr:RdfBag` and `xrr:RdfAlt`). Additionally, it allows the creation of dynamic language tags (i.e., with references from the input data) with `xrr:languageReference`.

R2RML-F (Debruyne & O'Sullivan, 2016) is a dedicated extension of R2RML to include data transformation functions, later refined in **FunUL** (Crotti Junior et al., 2016). It introduces the *Function* class (`rrf:Function`), which defines a data transformation function with name and body. An additional type of *Term Map* is defined, the *Function Valued Term Map*, which connects with the *Function* and supplies it with parameter bindings.

R2RML for collections and containers (Debruyne et al., 2017) (henceforth, R2RMLcc) is another R2RML extensions focused on the generation of RDF lists and containers. These terms are created with *Gather Maps*, that create a group of *Object Maps* within an *Object Map* with the `rrf:gather` property.

D2RML (Chortaras & Stamou, 2018) proposes a vocabulary extension to R2RML and RML with a large amount of features to describe in detail different heterogeneous data formats (e.g., JSON, CSV, XML) and accesses (e.g., RESTful APIs, SPARQL endpoints, documents) with a high level of detail. For instance, it adds to R2RML's *Logical Table* and RML's *Logical Source* the *SQL Tables*, *SPARQL Tables*, *CSV Tables* and *Information Source*. This proposal also includes data transformation functions to be used within the data source descriptions, and enables conditional generation of *Term Maps*.

WoT mappings (Cimmino et al., 2020) specify how data from IoT (Internet of Things) devices can be translated to RDF. It is an extension of the WoT (Web of Things) Thing Descriptions (Kaebisch et al., 2023) designed to deal with the heterogeneity of the IoT devices.

SPARQL-based Mapping Languages.

This group is integrated by languages that leverage the SPARQL query language, usually by extending its features to describe non-RDF data sources (Harris et al., 2013).

XSPARQL (Bischof et al., 2012) combines XQuery and SPARQL to enable the transformation

between XML and RDF data sources bidirectionally. This language inherits all inherent features of both languages, excluding the `ASK` and `DESCRIBE` query forms. To perform the translation from XML to RDF, the data is extracted using the XQuery language with variables, that are later used in a `CONSTRUCT` clause to create the RDF graph.

TARQL (TARQL, 2019) allows CSV transformation using the SPARQL syntax. The columns names of the input CSV files are treated as variables, which can be used in a `CONSTRUCT` clause to construct the RDF graph. The input file is referred to using the `FROM` clause.

SPARQL-Generate (Lefrançois et al., 2017) presents a more holistic approach than its predecessors providing access to more than one input data source format. This language is capable of generating RDF and text streams from a wide variety of data formats and access protocols. To this end, it introduces three new clauses to the SPARQL syntax: (i) `SOURCE` to bind variables to the input data source, (ii) `ITERATOR` to for accurately extracting data from the data sources; and (iii) `GENERATE`, that replaces and extends the `CONSTRUCT` clause with embedded SPARQL-Generate queries.

Facade-X (Daga et al., 2021) is proposed to also deal with a variety heterogeneous data sources relying in the SPARQL syntax, but without extending it with additional clauses. This proposal overrides the `SERVICE` operator to access heterogeneous data that can be queried as if it was RDF with a *façade*. Hence, data is retrieved in the `SERVICE` operator and bound as variables, that are later used in the `CONSTRUCT` clause to construct RDF graphs.

NORSE (Stadler et al., 2023) was developed as a vocabulary to place non-RDF data source descriptions within the SERVICE operator. This vocabulary allows including RML *Logical Source* descriptions inside queries, bound to SPARQL variables. In this approach, graphs are also generated using the `CONSTRUCT` clause.

Mapping Languages Based on Alternative Schemas.

Lastly, the languages of this group rely on models different from RDF or SPARQL, equally able and expressive enough to allow users to write transformation rules.

R_2O (Barrasa et al., 2004) is an XML-based language able to describe the transformation rules for constructing graphs from relational databases. An R_2O mapping consists on the following components: description of the database schema, ontology URI(s), mapping definitions between the ontology and database, and optionally a set of instance URIs to be added. The mapping definitions enable the description on how to generate the triples, as well as conditional expression, joins over the data source elements and data transformation functions.

D-REPR (Vu et al., 2019), Dataset REPResentation language, can be written as YAML or JSON files, providing an easier approach to describe the transformation rules from a wide range of data source formats. This language requires four components: dataset resources, attributes of data sources, a semantic model to map the attributes to the ontology classes, and rules for the values of the attributes. This language also enables the use of data transformation functions.

T2WML (Szekely et al., 2019), Table to Wikidata Mapping Language, is a YAML-based language that enables the mapping of CSV and Excel files to graphs following the Wikidata

data model (Vrandečić & Krötzsch, 2014). It allows for a more flexible description of tables. While other languages process tabular data in a row-wise manner, T2WML is a cell-centric model. Thus, this language can create, for instance, statements by referring to cells in distant rows. Cells can be linked to entities and property nodes in Wikidata.

ShExML (García-González, 2022; García-González et al., 2020) proposes a mapping language based on the ShEx validation language for knowledge graphs (Prud'hommeaux et al., 2014). This language enables the access to heterogeneous data sources, such as JSON, XML, CSV, RDBs and RDF with SPARQL queries. A ShExML mapping is divided in a declaration and a generation part. The declaration part provides the specification of prefixes, data sources, iterators, fields and expressions for merging or transforming data. The generation part describes how to generate triples and graphs with the fields and expressions defined in the declaration part.

Helio (Cimmino & García-Castro, 2024) is a JSON-based language that provides a variety of possibilities for heterogeneous data formats and accesses, as well as for extended functions for linking resources further than left joins, and data transformation functions. A Helio mapping is composed of a *data source* description, defining also its provider and handler; a set of *resource rules* that conform the triple generation rules; and optionally a set of *linking rules* to be applied over the resource rules.

2.2.4 Comparison of Mapping Languages Expressiveness

As the number of mapping languages increased and their adoption for KG construction grew wider, comparisons among these languages started emerging. This is the case of, for instance, SPARQL-Generate (Lefrançois et al., 2017), which is compared to RML in terms of query/mapping complexity; and ShExML (García-González et al., 2020), which is compared to SPARQL-Generate and YARRRML from a usability perspective.

Some studies dig deeper, providing qualitative complex comparison frameworks. Hert et al., 2011 provide a comparison framework for mapping languages focused on transforming relational databases to RDF. The framework is composed of 15 features: Logical table to class, M:N relationships, project attributes, select conditions, user-defined instance URIs, literal to URIs, vocabulary reuse, transformation functions, datatypes, named graphs, blank nodes, integrity constraints, static metadata, one table to n classes, and write support. The results led authors to divide the mappings into four categories (direct mapping, read-only general-purpose mapping, read-write general-purpose mapping, and special-purpose mapping), and ponder on the heavy reliance of most languages on SQL to implement the mapping, and the usefulness of read-write mappings (i.e., mappings able to write data in the database). De Meester et al., 2019 show an initial analysis of 5 similar languages (RML+FnO, xR2RML, FunUL, SPARQL-Generate, YARRRML) discussing their characteristics, according to three categories: non-functional, functional and data source support. The study concludes by remarking on the need to build a more complete and precise comparative framework and asking for a more active participation from the community to build it.

More recently, two related systematic reviews were published. Van Assche et al., 2023 focused specifically on declarative KG construction. This survey analyses the characteristics of

mapping languages assessing 15 characteristics for schema transformations and 5 for data transformation; as well as of compliant systems with 14 characteristics. It provides a more in-depth and systematic analysis than previous studies, but it is limited to reviewing peer-reviewed publications. Then, the expressiveness of languages such as R2RML, CSVW or TARQL that lack this kind of publication are not included. The review presented by Ryen et al., 2022 was more limited in extension with respect to Van Assche et al., 2023, but broader in scope considering knowledge graph generation and publication. Hence, the scope of these works leave out relevant languages that are worth analysing to understand their characteristics and capabilities.

2.2.5 Interoperability for Mapping Languages

The large amount of mapping languages presented in this section opens the door to a wide range of possibilities for KG construction. The downside of this situation comes with the general lack of interoperability among them. The concept of mapping translation (Corcho et al., 2020) emerged to bridge this gap and enable transforming transformation rules to different languages, but its actual use in real cases is limited.

We find some examples of unidirectional pairwise translations between languages. Helio and SPARQL-Generate, with the aim of extending their outreach, implement a plugin to translate from RML to their corresponding compliant language[2,3,4]. Inversely, ShExML provides translation to RML in its playground[5], thus presenting the language also as a user-friendly syntax for RML (García-González et al., 2020).

There are also examples of tools that provide a set of optimizations for KG construction exploiting the translation of mapping rules. This is the case of Morph-CSV (Chaves-Fraga et al., 2021) and FunMap (Jozashoori et al., 2020). Morph-CSV performs a transformation over the tabular data with RML+FnO mappings and CSVW annotations, and outputs a SQL database and R2RML mappings. FunMap takes an RML+FnO mapping, performs the transformation functions indicated, outputs the parsed data and generates a function-free RML mapping.

Lastly, the user-friendly serializations presented later in the chapter are always supported with translations to a mapping language, which is explained in detail in Section 2.3.

These cases of translations between mapping languages serve particular objectives, and provide limited support for real interoperability among them. Each mapping processor usually provides support for one mapping language, therefore, forcing users to learn different languages depending on the features that the language and compliant engine can provide to tackle the needs of each use case. Even for similar languages of the same family (e.g., R2RML and extensions with overlapping features), there is no approach that facilitates translating mappings rules to one another.

[2] https://github.com/oeg-upm/helio/wiki/Streamlined-use-cases#materialising-rdf-from-csv--xml-and-json-files-using-rml
[3] https://helio-playground.linkeddata.es/tour
[4] https://github.com/sparql-generate/rml-to-sparql-generate
[5] https://shexml.herminiogarcia.com/editor/

Table 2.2: Approaches for facilitating the mapping creation process for users. The proposals are divided into (i) visual editors and (ii) text-based serializations. For each proposal, the reference, compliant mapping language and subtype is provided.

Classification	Approach	Mapping Language	Type
Visual Editor	ODEMapster (Barrasa & Gómez-Pérez, 2006)	R2O	Tree
	Karma (Gupta et al., 2012)	R2RML	Tree
	RBA (Neto et al., 2013)	R2RML	Tree
	Sengupta et al., 2013	R2RML	Building blocks
	Lembo et al., 2014	R2RML	Graph
	RMLEditor (Heyvaert et al., 2016)	R2RML/RML	Graph
	SQuaRE (Blinkiewicz & Bak, 2016)	R2RML	Graph
	OntopPro (Calvanese et al., 2017)	Proprietary	Building blocks
	Juma (Crotti Junior et al., 2017)	R2RML	Building blocks
	RMLx (Aryan et al., 2017)	RML	Building blocks
	Map-On (Sicilia et al., 2017)	R2RML	Graph
	gra.fo[6]	R2RML	Graph
	Stardog designer[7]	SMS2	Graph
	Ontopic Studio[8]	R2RML/Proprietary	Building blocks
	Eccenca Corporate Memory[9]	Proprietary	Building blocks
Serialization	SML (Stadler et al., 2015)	R2RML	SPARQL-based
	Ontop proprietary (Calvanese et al., 2017)	R2RML	TTL- and SQL-based
	YARRRML (Heyvaert et al., 2018)	RML	YAML-based
	SMS2[10]	R2RML	SPARQL-based
	XRM[11]	R2RML/RML/CSVW	Proprietary syntax

2.3 User-Friendly Knowledge Graph Construction Approaches

This section presents the different approaches developed for easing the writing of mapping rules for practitioners. We divide these approaches into two categories, (i) visual editors (Section 2.3.1) that rely on an interactive application to graphically depict and edit mappings, and (ii) serializations (Section 2.3.2) that rely on a simplified syntax for writing the mapping rules. Table 2.2 lists the approaches described in the remaining of the section.

2.3.1 Visual Editors

Visual editors were first developed for easing the mapping writing process. We can subdivide the proposals released over the years based on how the mapping is graphically represented in the tool: either with a tree or graph layout, or using building blocks (Fig. 2.8).

The tree layout (Fig. 2.8a) succeeded in the first approaches developed. The first one was developed for the R_2O mapping language in **ODEMapster** (Barrasa & Gómez-Pérez, 2006), providing a graphical interface to visualize and edit the mappings by linking the database elements with the ontology resources. After the release of R2RML, **Karma** (Gupta et al., 2012) and **RBA** (R2RML by assertion) (Neto et al., 2013) were developed for this language. Both work similarly to ODEMapster, and Karma additionally provides automatic mapping suggestions.

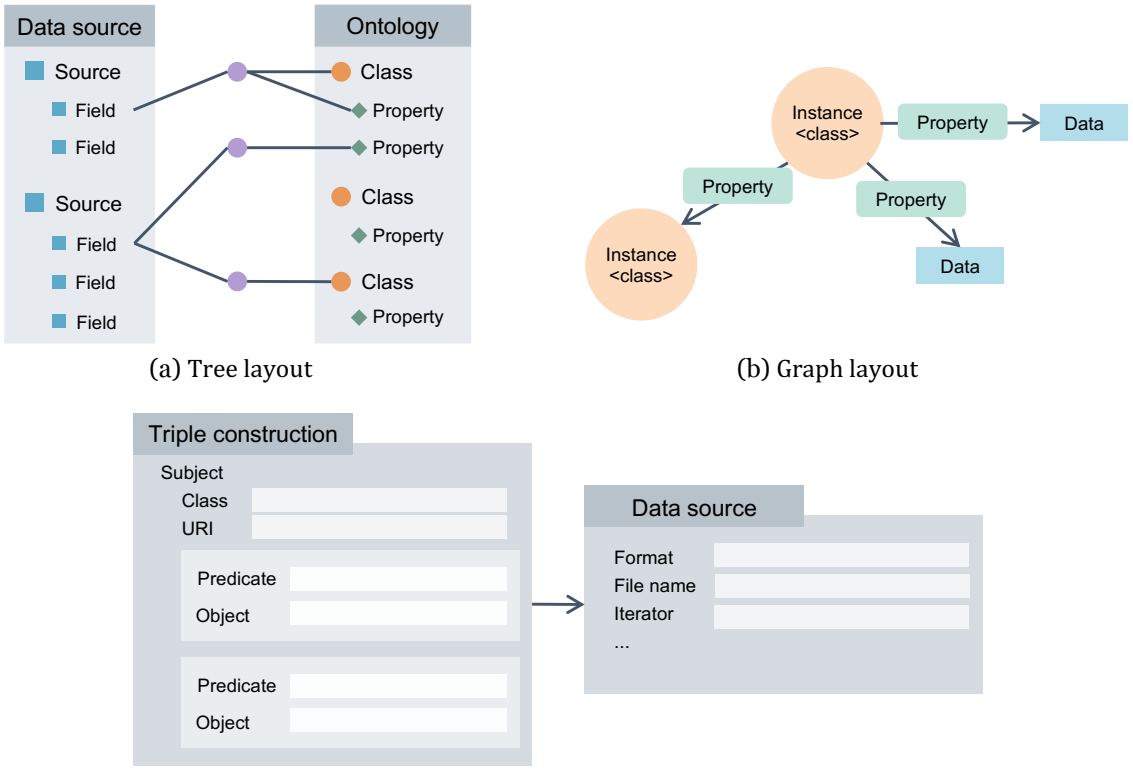


Figure 2.8: Graphical representations approaches in visual mapping editors: (a) tree layout, (b) graph layout, and (c) building blocks.

The graph display (Fig. 2.8b) was later adopted by multiple editors, such as in Lembo et al., 2014, **SQuaRE** (Blinkiewicz & Bak, 2016), **RMLEditor** (Heyvaert et al., 2016) and **Map-On** (Sicilia et al., 2017). These editors provide a graph overview of the mapping while constructing it, while also showing the data sources and ontology. All of them are able to create R2RML mappings, and in addition, the RMLEditor can also produce RML mappings. There are also non open-source editors developed by companies that use a graph layout, such as **gra.fo**[6] and the **Stardog designer**[7]. The former works with R2RML, and the latter with the Stardog proprietary syntax, SMS2, described in the next section.

Some tools were also developed that follow an alternative approach to the tree and graph layouts, broadly used in the semantic web supporting applications. This approach comprises the use of building blocks or templates (Fig. 2.8c), that are the components to build a mapping between data source and ontology. The first editor following this approach was proposed

[6] https://gra.fo/
[7] https://www.stardog.com/designer/
[8] https://ontopic.ai/en/ontopic-studio/
[9] https://documentation.eccenca.com/latest/build/lift-data-from-tabular-data-such-as-csv-xslx-or-database-tables
[10] https://docs.stardog.com/archive/7.5.0/virtual-graphs/mapping-data-sources.html#sms2-stardog-mapping-syntax-2
[11] https://zazuko.com/products/expressive-rdf-mapper/

in Sengupta et al., 2013, later refined in Pinkel et al., 2014. **OntopPro** (Calvanese et al., 2017) released a Protégé plugin to create and edit mappings (in their proprietary language) with templates, allowing also the creation of RDF triples and running SPARQL queries. **Juma** (Crotti Junior et al., 2017) and **RMLx** (Aryan et al., 2017) allow building R2RML and RML mappings respectively with building blocks, correspondent to the different parts of the mappings. There are also a couple of examples that enable this kind of visualization for mapping construction using a proprietary languages, such as **Ontopic Studio**[8] and **Eccenca Corporate Memory**[9]. There are studies that suggest that these visual studies reduce the cognitive load to write mappings (Crotti Junior et al., 2018). However, the uptake of these approaches is limited.

2.3.2 Serializations

As an alternative to visual approaches, text-based user-friendly serializations were developed. This approach suited most practitioners with technical profiles or with preferences for a text oriented environment. **SML** (Stadler et al., 2015) was developed as user-friendly syntax for R2RML. It provides an SPARQL-based syntax, enhancing the simplicity for writing mappings but maintaining the same expressiveness. The virtual KG processor **Ontop** (Calvanese et al., 2017) also provides a simplified serialization for R2RML, which combines the Turtle syntax for triple generation and SQL for data access. The Stardog triplestore developed another SPARQL-based proprietary serialization, the Stardog Mapping Syntax 2 (**SMS2**)[10].

Later on, as more mapping languages emerged, new serializations with a broader language compliance range emerged. This is the case of **XRM**[11] (Expressive RDF Mapper), that provides a unique syntax for writing R2RML, RML and CSVW mappings. It can be used with a service plugin integrated into common text editors, Visual Studio Code and Eclipse. This service warns the users about errors in the mapping while writing, and translates the mapping rules into one of the aforementioned languages.

YARRRML (Heyvaert et al., 2018) was developed as a YAML-based[12] user-friendly syntax for RML. Listing 2.6 shows an example of a YARRRML mapping equivalent to the RML example shown in Listing 2.5. YARRRML mappings need as well the declaration of the prefixes at the beginning of the document, which is done using the `prefixes` key (Lines 1-2). The mapping rules are defined within the `mappings` key (Lines 4-13), which are aggregated in a rule set with keys named by the user (e.g., `athletes` key, Line 5). Each mapping rule set describes the input data sources (`sources` key, Lines 6-7), subject (`s` key, Line 8) and predicate-object pairs (`po` key, Lines 9-11). This serialization can be used with the Matey[13], a web service able to translate YARRRML into RML mapping files, or directly generate RDF triples.

```
1 prefixes:
2  ex: "http://example.com/ns#"
3
4 mappings:
5   athletes:
```

[12] https://yaml.org/spec/1.2.2/
[13] https://rml.io/yarrrml/matey/

```
6     sources:
7       - ["data.json~jsonpath", "$.*"]
8     s: http://example.com/athlete/$(RANK)
9     po:
10      - [a, ex:Athlete]
11      - [ex:name, $(NAME)]
12      - [ex:rank, $(RANK)]
13      - [ex:mark, $(MARK)]
```

Listing 2.6: YARRRML mapping to generate the RDF graph in Listing 2.3 with data from the JSON file shown in Listing 2.4. This mapping translates into the RML mapping shown in Listing 2.5.

These serializations are in general widely adopted as they have proven useful for facilitating the writing of mapping rules. Hence, as mapping languages change and incorporate new features, these serializations need to be updated as well.

2.4 Knowledge Graph Life Cycle

A high number of knowledge graphs has been released over the years. Along with this adoption, the question arose about which are the processes involved in their life cycle. As a result, different proposals for life cycles have been published for Linked Data and knowledge graphs.

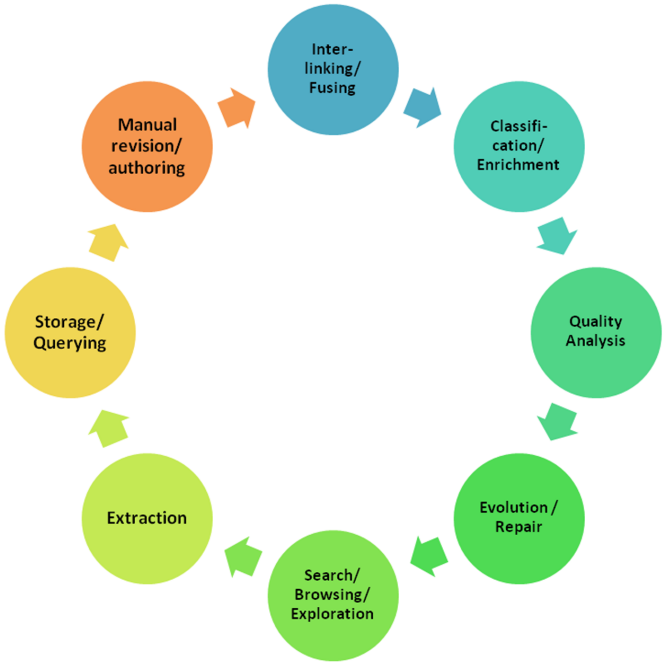


Figure 2.9: Linked Data life cycle proposed by Ngomo et al., 2014.

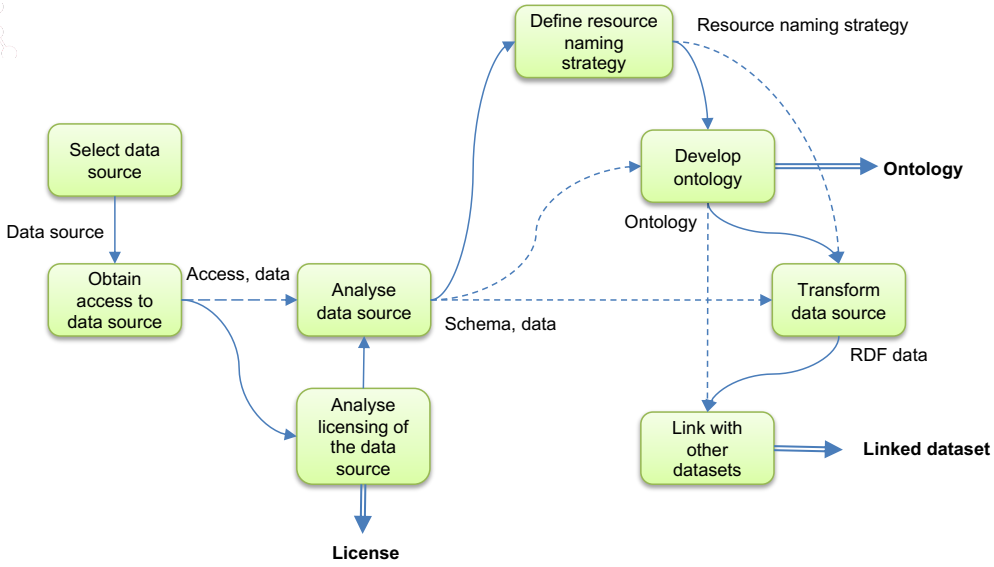


Figure 2.10: Linked Data generation and publishing tasks proposed by Radulovic et al., 2015

Ngomo et al., 2014 proposed a series of steps that define the life cycle of Linked Data (Fig. 2.9). These steps include (i) *Extraction* of information from (semi-)structured and unstructured data sources mapping them to an RDF schema; (ii) *Storage and Querying* the graph in a graph store; (iii) *Authoring* for users to create, modify or extend the information in the graph; (iv) *Linking* to external related resources; (v) *Enrichment* with higher-level structures to aggregate and query data more efficiently; (vi) *Quality Analysis* to assess the quality of the published data; (vii) *Evolution and Repair* to address issues and errors detected; (viii) *Search, Browse and Exploration* for users to navigate the graph in a user-friendly manner.

Radulovic et al., 2015 proposed a series of interconnected tasks for generating and publishing Linked Data (Fig. 2.10). Is is oriented to the domain of energy consumption in buildings, but the steps proposed can be applied to other domains. These steps are: (i) *Selecting data sources*, (ii) *Obtaining access* to data sources, (iii) *Analyzing licensing* of data sources, (iv) *Analyzing data sources*, (v) *Defining resource naming strategy*, (vi) *Developing an ontology*, (vii) *Transforming the data sources*, and (viii) *Linking* with other datasets.

Simsek et al., 2021 proposed a life cycle for knowledge graphs based on the authors' experience, considering a series of steps and iterations (Fig. 2.11). The life cycle starts with the (i) *Knowledge Creation* to construct the knowledge graph from heterogeneous data sources, followed by the (ii) *Knowledge Hosting* in an appropriate graph store. Then, the (iii) *Knowledge Curation* cycle takes place, which is comprised of three steps. First, the *Knowledge Assessment* of the quality (considering completeness and correctness) of the knowledge graph, which triggers the other two steps: *Knowledge Cleaning* for error detection and correction, and *Knowledge Enrichment* with related resources. Once the *Knowledge Assessment* is satisfactory and the knowledge graph is curated, the (iv) *Knowledge Deployment* stage takes place to publish the knowledge graph and make it available for consumption.

Tamašauskaitė and Groth, 2023 carried out a systematic literature review to find the common

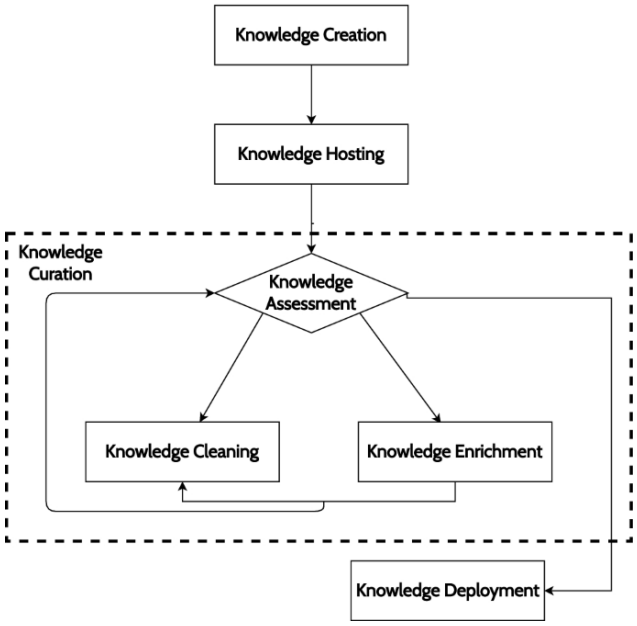


Figure 2.11: Knowledge graph life cycle proposed by Simsek et al., 2021.

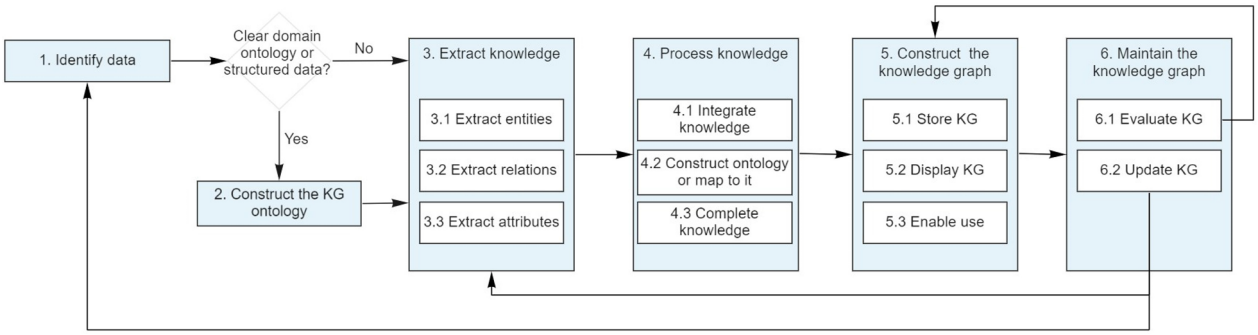


Figure 2.12: Knowledge graph development proposed by Tamašauskaitė and Groth, 2023.

stages of knowledge graph management process analysing several papers that have described the process followed for their knowledge graphs (Fig. 2.12). This process include (i) *Identification of Data* and domain of interest of the knowledge graph; (ii) *Construction of the Ontology* whenever there is no suitable ontology available; (iii) *Extraction of Knowledge* from the identified data, and more specifically, extracting entities, relations between them and their attributes; (iv) *Processing of Knowledge* to ensure its high quality, which involves integration from different sources, cleaning, mapping to the ontology, completion, enrichment and validation; (v) *Knowledge Graph Construction* to store it and make it available and accessible for consumption; and (vi) *Knowledge Graph Maintenance* for evaluating the knowledge graph quality and updates for evolution.

All these proposed life cycles share common stages and consider similar aspects. We can group the particular tasks and stages of each proposed life cycle in tree main categories, (i) *Creation*, (ii) *Deployment* and (iii) *Maintenance* of knowledge graphs.

- **Creation.** This phase comprises the tasks dedicated to ultimately produce a high-quality knowledge graph. Examples of tasks in this phase are knowledge extraction, data cleaning and processing, data mapping and transformation, validation, quality assessment, enrichment, linking and completion.
- **Deployment.** We aggregate in this phase tasks related to making the knowledge graph available for consumption, such as hosting in a graph store, displaying for users and enabling its use for downstream tasks.
- **Maintenance.** The tasks that takes place in this phase tackle the management over time of the knowledge graph, and may involve iterations over any of the tasks previously carried out for their creation and deployment. The objective of this phase is to curate the graph, detect and correct errors and implement new features according to new needs or requirements for evolving the knowledge graph.

The use of declarative mapping approaches have proven to be valuable for different tasks within the knowledge graph life cycle. The main task for which they were developed is *construction*, relating the input data sources with the target schema to produce the knowledge graph. For this task, a high amount of effort has been made to improve the process (as has been explained in detail over this chapter), for increasing mapping expressiveness (see Section 2.2), facilitating their creation (see Section 2.3) and for optimizing the use of resources by transformation engines (Arenas-Guerrero, Chaves-Fraga, et al., 2024; Iglesias et al., 2022, 2023; Jozashoori & Vidal, 2019; Xiao et al., 2020).

Apart from the *construction* task, these declarative approaches can play an important role in other different tasks of the life cycle. They enable *metadata annotation* about the data sources (e.g., DCAT in RML, CSVW for tabular files), which can help enhance the process in terms of transparency, completeness and performance (Chaves-Fraga et al., 2021; Vidal et al., 2023). They also allow for *data processing and cleaning*, thanks to the increasing integration of data transformation functions into the languages' features (Crotti Junior et al., 2016; De Meester et al., 2017; Debruyne & O'Sullivan, 2016; Jozashoori et al., 2020). The possibility of expressing how the output graph is generated (e.g., serialization, compression) and managing streams of data can help the *deployment* phase as well (e.g., RML Target and SPARQL-Generate). However, there is no analysis so far looking into the role of these approaches in the *evolution and maintenance* of knowledge graphs.

2.5 Conclusions and Limitations of the State of the Art

This chapter presents the relevant work in the state of the art about declarative mapping languages for knowledge graph construction and evolution. We conclude this section with the identification of the challenges in the field that motivate the contributions of this thesis, which are presented in the following chapters.

- **Different mapping language expressiveness and processing characteristics of their associated engines.** Many different mapping languages for knowledge graph construction from heterogeneous data sources have been published over the years. These languages offer a variety of features, providing a wide variety of possibilities for

addressing real-world use cases. Deciding which language is the most suitable for the particularities of each use case and requirement becomes hard without a fine-grained comparison of the features that each language provides.

- **Updating mapping languages according to new needs.** As KG technologies advance, new needs and requirements emerge in KGC. Despite the options that mapping languages currently provide, they have limitations and have not adopted yet some of the latest developments (e.g., the currently work in progress RDF 1.2 specification (Hartig et al., 2023)). These limitations negatively impact on the adoption of declarative approaches for KGC, as they must be competitive with ad-hoc approaches (e.g., programming scripts) in terms of expressiveness and keep up with the current progress in the field.
- **Adoption barrier for writing declarative mappings.** Mappings usually follow a syntax that is not easy or intuitive to write, especially for non-experts. Several developments have been published, providing user interfaces and user-friendly serializations to improve the writing task for users. However, visual approaches have not been adopted in general, and serializations still pose a barrier for users with non-technical profiles.
- **The role of declarative mappings in the knowledge graph life cycle.** Mapping languages were developed for constructing KGs from diverse data sources, relating them to the schema provided by an ontology. However, the analysis shows that they can also be part of other tasks of the KG life cycle, such as data pre-processing and metadata annotation. Yet, the role that they can play in other parts of the life cycle has not been assessed.

Chapter 3

Objectives and Contributions

This chapter presents the main objectives of this thesis and identifies the contributions to the state of the art. We enumerate the assumptions considered in this work, describe the main hypotheses and set the scope of the thesis presenting the restrictions. Fig. 3.1 provides an overview of the objectives and contributions presented in this thesis, along with their relations with identified hypotheses, assumptions and restrictions.

3.1 Objectives

The general objective of this thesis is to *improve the understanding (features and limitations) and operational management of declarative KG construction languages*. In order to achieve this goal, the following sub-objectives are defined:

O1 To analyze the needs for declarative knowledge graph construction from heterogeneous data sources, in order to facilitate the support of advances in existing mapping languages to address the most relevant ones.

O2 To help knowledge engineers and domain experts with building declarative mappings in a user-friendly manner.

O3 To assess declarative knowledge graph construction technologies in terms of their benefits for supporting the creation and evolution of knowledge graphs.

In order to achieve the first objective, the following open research problem must be solved:

- Ever since the first mapping languages began to be developed, many more have been proposed and released over the years. These languages address different needs and propose diverse features, but they share some characteristics, as in the end they all serve the same purpose of constructing KGs. In order to keep improving the KG construction process from heterogeneous sources, we need to identify which are the needs and the current challenges in the mapping languages. Thus we can facilitate the selection of languages according to their capabilities for addressing different needs. However, there is a lack of a fine-grained analysis of the competences and expressiveness of these mapping languages.

From a technological perspective, the following problem must be solved:

- The needs for KG construction are not static, as they evolve with the use cases and evolution of surrounding technology. For instance, this is the case of the currently under development new specification of RDF 1.2 (Hartig et al., 2023), that includes RDF-star triples. Mapping languages so far cannot produce RDF-star graphs without a dedicated extension, thus new technological support is needed to assist the creation of KGs with the evolving needs, and more specifically, to build RDF-star graphs.

In order to achieve the second objective, the following research problem must be solved:

- Most mapping languages are formalized as ontologies. In these cases, mappings are required to be written using an RDF serialization. Other common languages extend SPARQL to write transformation rules. This implies a double obstacle for new users, to learn the language and the language's syntax (i.e., Turtle or SPARQL) in case they are not familiar with them. There is a lack of methods that facilitate the writing process for both expert practitioners and new users to speed-up the process and become less error-prone.

From a technological perspective, the following problem must be solved:

- User-friendly approaches usually involve developing new syntaxes that cannot always be directly process by with KGC systems. In addition, different KGC systems are usually compliant with only one language. There is currently little interoperability among the existing languages, which limits the possibilities for users to use a wider variety of systems with different capabilities. Hence, new technological support is needed to allow translations among current mapping languages and reduce the barrier adoption enhancing the communication between user-friendly syntaxes and existing KGC implementations.

Finally, for achieving the third objective, the following open research problem must be solved:

- Knowledge graphs undergo a complex process from creation to consumption that comprise their life cycle. Declarative mapping rules and their supporting technologies are a key in the construction step, but can also play a beneficial role in other steps, such as data pre-processing and metadata annotation for transparency. However, there is a lack of research on their support in other tasks involved in the KG life cycle.

3.2 Contributions

In this section, we describe the solutions corresponding to the objectives and open research problems described in Section 3.1. We present as follows the contributions that support the advance of the current state of the art regarding the first objective (understand and gather the needs for KG construction):

C1 Design of a **comparison framework analysing the state-of-the-art mapping languages** proposed to generate knowledge graphs from heterogeneous data sources. We extract the characteristics of these languages and compare them over a set of detailed

features that show their expressiveness.

C2 Identification, definition and implementation of requirements for knowledge graph construction from heterogeneous data sources. Based on the analysis performed on the comparison framework, we are able to extract what are the needs for building knowledge graphs, which are possible to achieve with the current languages and which need to be addressed yet. We implement these requirements in a formal language.

C3 Development of new features for the RML mapping language to address the limitations of the language according to the new needs in knowledge graph construction. We focus on the development of the extension to create RDF-star graphs.

Regarding the second objective (to help knowledge engineers and domain experts to build mappings), we present new advances with the following contributions:

C4 Design of a user-friendly approach for writing mapping rules based on spreadsheets. We propose this approach as a way of writing mappings by only specifying the essential information, reducing learning curve of the language constructs and the syntax's peculiarities. We also develop an implementation of this approach to translate the mapping rules in spreadsheets into different mapping languages.

C5 Update of the YARRRML syntax for RML with new features. We include in a new version of this syntax the features incorporated in the evolution of the RML mapping language, to keep this language accessible for a broader community of users that already use YARRRML. We also provide an implementation to translate between the updated YARRRML version into other mapping languages to facilitate its adoption.

Finally, with regard to the third objective (to assess the role of mapping technologies in KG evolution), this work presents the following contribution:

C6 Analysis of the scenarios in which declarative KG construction technologies play a valuable role in the refactoring of knowledge graphs. We conduct an evaluation to study how mapping-based processes be beneficial when the schema used in a knowledge graph changes with respect to other established methods.

3.3 Assumptions

Our work is based on the set of assumptions listed below. These assumptions provide a background to facilitate the comprehension of the decisions taken during the development of this work.

A1 Mapping languages are declarative.

A2 Mapping languages provide human readable documentation available online.

A3 The schema of knowledge graphs (ontology) used for creating mapping documents is available and implemented in OWL or RDF(S).

A4 Mapping rules can be translated into different languages with information preservation.

A5 The refactoring of the schema used in a KG does not involve changes in the data.

3.4 Hypotheses

After the identification of the assumptions, we can describe the research hypotheses of this thesis. They cover the general characteristics of the contributions:

H1 Current mapping languages lack some expressiveness to construct knowledge graphs from heterogeneous data sources for all use cases.

H2 It is possible to update current mapping languages with new features that address the current needs in the construction of knowledge graphs.

H3 Writing mapping rules in spreadsheet environments improves the user experience for practitioners of different backgrounds for writing mappings, whilst reducing errors.

H4 Declarative KG construction technologies brings benefits in the refactoring of knowledge graphs within their life cycle.

3.5 Restrictions

Finally, there is a set of restrictions that describe the limitations and define the future work objectives:

R1 Requirements for KG construction are considered up to January 2023, since it is an issue that is evolving during the time of writing this thesis.

R2 The reference for the RML specification in the mapping languages analysis is the release on 2014 (Dimou et al., 2014).

R3 Features specific of non RDF-based mapping languages are not ensured to be modelled in an ontology (e.g., SPARQL recursive clauses and FILTER).

R4 The implementations proposed in contributions C4 and C5 are not fully compliant with all modules of the RML release on 2023 (Iglesias-Molina, Van Assche, et al., 2023b).

R5 The evaluation of the value of mappings in KG refactoring considers only changes in the schema used in the KG, not in the data.

R6 The changes considered for KG refactoring are schema changes for switching among metadata representation approaches for RDF graphs.

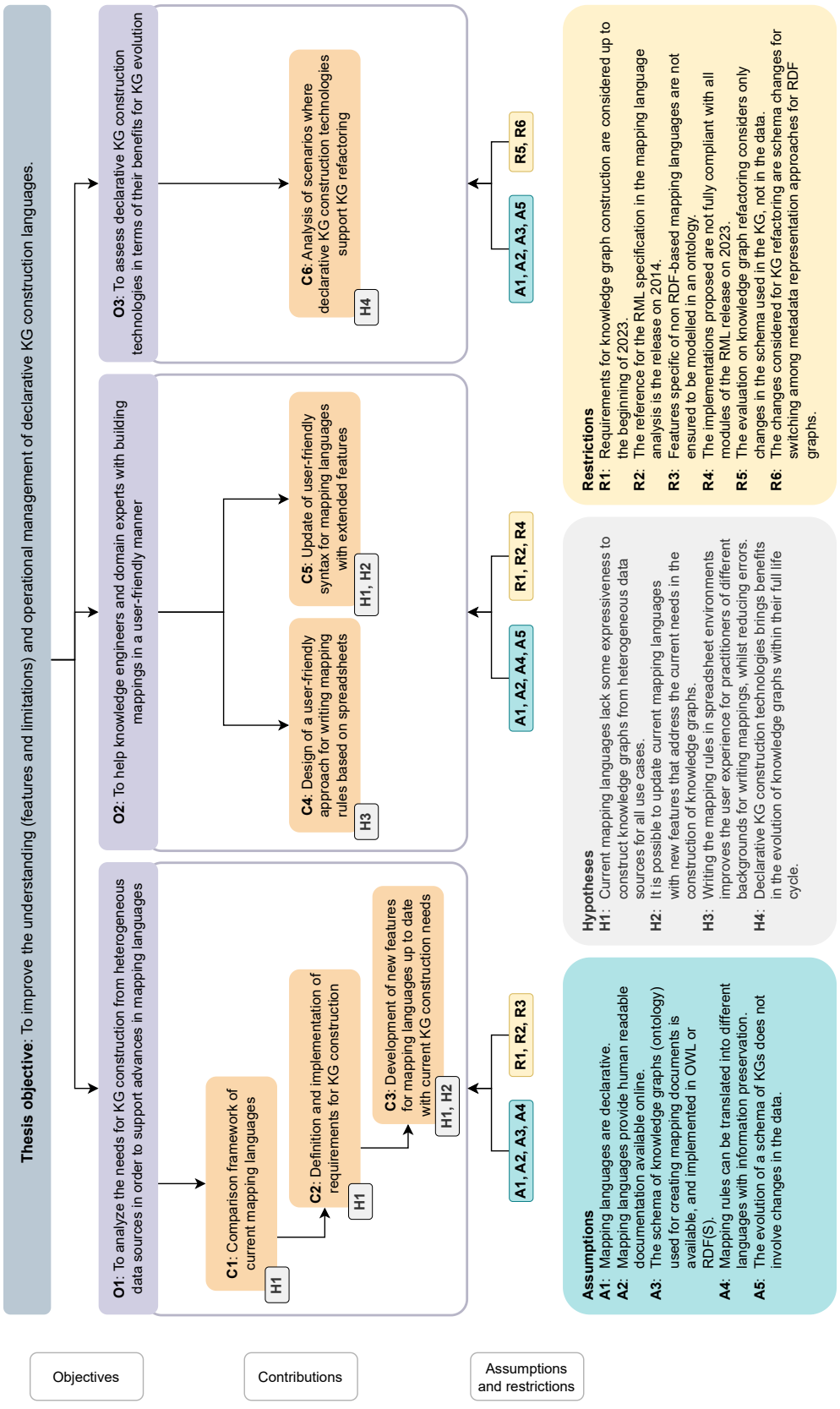


Figure 3.1: Overview of objectives, contributions, hypotheses, assumptions and restrictions, and their relations.

Chapter 4

Understanding, Representing and Extending the Expressiveness of Declarative Mapping Languages

Declarative mappings are a key element for the knowledge graph construction process to enhance maintainability, understandability and reproducibility. This chapter first presents an extensive analysis of current mapping languages in the form of a comparison framework (Section 4.1). Based on this comparison, a set of requirements are extracted and used for building an ontology that aims at representing the expressiveness of current mapping languages (Section 4.2). Finally, we present the extension of a widely used mapping language to construct RDF-star graphs (Section 4.3).

4.1 Comparison framework

This section presents a comparison framework that collects and analyzes the main features included in mapping languages. The diversity of the languages that have been analyzed is crucial for understanding the needs for constructing knowledge graphs from heterogeneous data sources. Thus, we can extract relevant shared features and requirements along with the peculiarities of each language.

4.1.1 Methodology

The framework presented in this section analyzes languages from the three categories identified in Section 2.2. The selected languages fulfill the following requirements: (i) are widely used, relevant and/or include novel or unique features; (ii) are currently maintained, and not deprecated; (iii) are not a serialization or a user-friendly representation of another language; and (iv) are not tailored for a specific use case. For instance, D2RQ (Bizer & Seaborne, 2004) and R_2O (Barrasa et al., 2004) are not included since they were superseded by R2RML, which is analysed in the comparison. T2WML is specific for transforming data following only the schema of Wikidata, and WoT mappings for the IoT domain, which are the reasons why they

are also excluded. And lastly, NORSE is not included either since it is conceived for being generated by translating from RML mappings.

The following RDF-based languages are included: R2RML (Das et al., 2012), RML (Dimou et al., 2014), KR2RML (Slepicka et al., 2015), xR2RML (Michel, Djimenou, Zucker, & Montagnat, 2015), R2RML-F (Debruyne & O'Sullivan, 2016), FunUL (Crotti Junior et al., 2016), XLWrap (Langegger & Wöß, 2009), CSVW (Tennison et al., 2015), D2RML (Chortaras & Stamou, 2018) and R2RMLcc (Debruyne et al., 2017). The analyzed SPARQL-based languages are: XSPARQL (Bischof et al., 2012), TARQL (TARQL, 2019), SPARQL-Generate (Lefrançois et al., 2017) and Facade-X (Daga et al., 2021). Finally, we selected the following languages based on other formats: ShExML (García-González et al., 2020), Helio Mappings (Cimmino & García-Castro, 2024) and D-REPR (Vu et al., 2019).

The framework has been built as a result of analyzing the common features of these mapping languages, and also the specific features that make them unique and suitable for some scenarios. It includes information on data sources, general features for the construction of RDF graphs, and features related to the creation of subjects, predicates, and objects. In the following subsections, the features of each part of the framework are explained in detail. The language comparison for data sources is provided in Table 4.1, for triples creation in Table 4.2, and for general features in Table 4.3.

These languages have been analyzed based on their official specification, documentation, or reference paper (listed in Table 2.1). Specific implementations and extensions that are not included in the official documentation are not considered in our framework. The cells (i.e., language feature) marked with "*" in the framework tables indicate that there are non-official implementations or extensions that include the feature.

4.1.2 Data Sources Description

Table 4.1 shows the ability of each mapping language to describe a data source in terms of retrieval, description, security, data format and protocol.

Data Retrieval. Data from data sources may be retrieved in a continuous manner (e.g., *Streams*), periodically (e.g., *Asynchronous sources*), or just once, when the mapping is executed (e.g., *Synchronous sources*). As shown in Table 4.1, all mapping languages are able to represent synchronous data sources. Additionally, SPARQL-Generate and Helio are able to represent periodical data sources, and SPARQL-Generate also represents continuous data sources (e.g., `it:WebSocket()` in SPARQL-Generate). Other languages do not explicitly express that feature in the language, but a compliant engine may implement it.

Representing Data Sources. Extracting and retrieving heterogeneous data involves several elements that mapping languages need to consider: *Security terms* to describe access (e.g., relational databases (RDB), API Key, OAuth2, etc); *Retrieval protocol* such as local files, HTTP(S), JDBC, etc; *Features that describe the data* to define particular characteristics of the source data (e.g., queries, regex, iterator, delimiter, etc); *Data formats* such as CSV,

Table 4.1: Data retrieval and data source expression for the analysed mapping languages from the references stated in Table 2.1. (*) indicates features not explicitly declared in the language, but that are implemented by compliant tools.

Feature		ShExML	XSPARQL	TARQL	CSVW	R2RML	RML	KR2RML	xR2RML	SPARQL-Generate	R2RML-F	FnUL	Helio	D-REPR	XLWrap	D2RML	Facade-X	R2RMLce
Retrieval of data	**Streams**	×	×	×	×	×	✓*[a]	×	×	✓	×	×	×	×	×	×	✓	×
	Synchronous sources	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Asynchronous sources	–	–	–	–	–	–	–	–	Events, Periodic	–	–	Periodic	–	–	–	–	–
	Security terms	–	–	–	–	Basic (DB)	Basic (DB)	Basic (DB)	Basic (DB)	–	Basic (DB)	–	API Key, OAuth2, Bearer, Basic	–	–	Basic (DB)	–	Basic (DB)
	Encoding	×	×	✓*[b]	✓	×	✓	×	×	×	×	×	✓	×	×	×	✓	×
	MIME Type	×	×	×	×	×	×	×	×	×	×	×	✓	×	×	×	✓	×
Expressing data sources	**Features describing data**	Iterator, Queries	–	Delimiter, Separator	Delimiter, Separator, Regex	Queries	Delimiter, Regex, Iterator, Queries, Separator	Queries	Regex, Iterator, Queries	Delimiter, Regex, Iterator, Queries, Separator	Iterator, Queries	Iterator, Queries	Delimiter, Regex, Iterator, Queries, Separator	Delimiter, Regex, Iterator	Separator	Delimiter, Regex, Iterator, Queries	Delimiter, Regex, Iterator, Queries, Separator	Iterator, Queries
	Retrieval protocol	file, http(s), odbc/jdbc	file	file	file, http(s)	file, http(s), odbc/jdbc	file, http(s), odbc/jdbc	file, odbc/jdbc	file, odbc/jdbc	file, http(s), odbc/jdbc WebSocket, MQTT	file, http(s), odbc/jdbc	file, http(s)	file, any URI-based	file	file	file, http(s), odbc/jdbc	file, http(s)	file, http(s), odbc/jdbc
	Data formats	Tabular, Tree, Graph	Tree (XML)	Tabular (CSV)	Tabular	Tabular	Tabular, Tree, Graph	Tabular, Tree	Tabular, Tree	Tabular, Tree, Plain Text, Graph	Tabular	Tabular, Graph	Tabular, Tree, Plain Text, Graph	Tabular (CSV), Tree	Tabular (CSV, Excel)	Tabular, Tree, Plain Text, Graph	Tabular, Tree, Plain Text, Graph	Tabular

[a]Implemented by RMLSreamer, available at https://github.com/RMLio/RMLStreamer.
[b]Command line input option `--encoding` (TARQL, 2019).

RDB, and JSON; *Encoding* and content negotiation (i.e., *MIME Type*).

Half of the languages do not allow the definition of security terms. Some languages are specific for RDB terms (R2RML and extensions, with `rr:logicalTable`), and only Helio can define security terms. These two languages are also the only ones that allow the specification of MIME Types, and can also specify the encoding along with TARQL and CSVW (e.g., `csvw:encoding` attribute of `csvw:Dialect` in CSVW).

Regarding protocols, all languages consider the use of local files. It is highly usual to consider HTTP(s) and database access (especially with the ODBC and JDBC protocols). Only XSPARQL, TARQL, D-REPR, and XLWrap describe exclusively local files.

The features provided by each language are closely related to the data formats that are covered. Queries are usual for relational databases and NoSQL document stores and iterators for tree-like formats. Some languages also enable the description of delimiters and separators for tabular formats (e.g., CSVW defines the class `Dialect` to describe these features; this class is reused by RML), and finally, less common Regular Expressions can be defined to match specific parts of the data in languages such as CSVW, SPARQL-Generate, Helio, D-REPR, and D2RML (e.g., `RegexHandler` in Helio, `format` in CSVW).

The most used format is tabular (i.e., RDB, CSV, TSV). Some languages can also process RDF graphs such as ShExML, RML, SPARQL-Generate, Helio, and D2RML (e.g., `QUERY` in ShExML, SPARQL service description[1] in RML), and the last three languages can also process plain text.

4.1.3 Triples Generation

Table 4.2 represents how different languages describe the generation of triples. We assess whether they generate the *Subject*, *Predicate*, and *Object*: in (1) a *Constant* manner, i.e., non-dependant on the data field to be created; or in (2) a *Dynamic* manner, i.e., changing its value with each data field iteration. For *Objects*, the possibility of adding *Datatype and Language* tags is also considered; this feature assesses whether they can be added, and if they are added in a dynamic (changes with the data) or static (constant) manner. This table also analyzes the use and cardinality of transformation functions and the possibility of iterating over different nested level arrays (i.e., in tree-like formats).

The categories *Constant* and *RDF Resource* (the latter within *Dynamic*) show which kind of resources can be generated by the language (i.e., IRI, Blank Node, Literal, List and/or Container). The *Dynamic* category also considers: the *Data References* (i.e., fields from the data source) that can appear with single of mixed formats; from how many *Data Sources* (e.g., "'1:1" when only data from one file can be used) the term is generated; if *Hierarchy Iteration* over different nested levels in tree-like formats is allowed; and if *Functions* can be used to perform transformations on the data to create the term (e.g., `lowercase`, `toDate`, etc.).

[1] http://www.w3.org/ns/sparql-service-description#

Table 4.2: Features for subject, predicate, and object generation of the studied mapping languages from the references stated in Table 2.1.

	Feature	ShExML	XSPARQL	TARQL	CSVW	R2RML	RML	KR2RML	xR2RML	SPARQL-Generate	R2RML-F	FunUL	Helio	D-REPR	XLWrap	D2RML	Facade-X	R2RMLce
Subject	**Constant**	IRI	BN, IRI	BN, IRI	IRI	BN, IRI	BN, IRI	–	BN, IRI	BN, IRI	BN, IRI	BN, IRI	IRI	BN, IRI	BN, IRI	BN, IRI	BN, IRI	BN, IRI
	RDF Resource	IRI	BN, IRI	BN, IRI	IRI	BN, IRI	BN, IRI	IRI	BN, IRI	IRI	BN, IRI	BN, IRI	IRI	BN, IRI	BN, IRI	BN, IRI	IRI	BN, IRI
	Data Reference	1..* Ref 1..* Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..* Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..1 Format
	Data Sources	1..*	1..*	1..1	1..1	1..1	1..1	1..*	1..*	1..*	1..1	1..1	1..*	1..1	1..1	1..1	1..*	1..1
	Hierarchy Iteration	✓	✓	×	×	×	✓	✓	×	✓	×	×	×	✓	×	✓	✓	×
	Functions	–	1..*	1..*	–	–	1..*	1..*	–	1..*	1..*	1..*	1..*	1..*	1..*	1..*	1..*	1..*
Predicate	**Constant**	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI
	RDF Resource	–	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	IRI	–	IRI	IRI	IRI
	Data Reference	–	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..1 Format	–	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..1 Format
	Data Sources	–	1..1	1..1	1..1	1..1	1..1	1..*	1..1	1..*	1..1	1..1	1..*	1..1	–	1..1	1..*	1..1
	Hierarchy Iteration	×	×	×	×	×	✓	✓	×	×	×	×	×	✓	×	✓	✓	×
	Functions	–	1..*	1..*	–	–	1..*	1..*	–	1..*	1..*	1..*	1..*	1..*	–	1..*	1..*	1..*
Object	**Constant**	IRI, Literal	BN, IRI, Literal	BN, IRI, Literal	IRI, Literal	IRI, Literal	IRI, Literal	IRI, Literal	BN, IRI, Literal, List, Container	BN, IRI, Literal, List	IRI, Literal	IRI, Literal	IRI, Literal	BN, IRI, Literal	IRI, Literal	BN, IRI, Literal	BN, IRI, Literal, List	BN, IRI, Literal, List, Container
	RDF Resource	IRI, Literal, Lists	BN, IRI, Literal	BN, IRI, Literal	IRI, Literal	BN, IRI, Literal	BN, IRI, Literal	IRI, Literal, List	BN, IRI, Literal, List, Container	BN, IRI, Literal, List	BN, IRI, Literal	BN, IRI, Literal	IRI, Literal	BN, IRI, Literal	IRI, Literal	BN, IRI, Literal	BN, IRI, Literal, List	BN, IRI, Literal, List, Container
	Data Reference	1..* Ref 1..* Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..* Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..1 Format	1..* Ref 1..* Format	1..* Ref 1..1 Format
	Data Sources	1..*	1..*	1..1	1..1	1..1	1..1	1..*	1..*	1..*	1..1	1..1	1..*	1..1	1..1	1..1	1..*	1..1
	Hierarchy Iteration	✓	✓	×	×	×	✓	✓	×	✓	×	×	×	✓	×	✓	✓	×
	Functions	1	1..*	1..*	–	–	1..*	1..*	–	1..*	1..*	1..*	1..*	1..*	1..*	1..*	1..*	1..*
	Datatype and Language	static, dynamic	static, dynamic	static, dynamic	static	static	static	–	static	static	static	static	static, dynamic	static	–	static	static	static

Subject Generation. Subjects can be IRIs or Blank Nodes (BN). This is well reflected in the languages, since, with a few exceptions that do not consider Blank Nodes, all languages are able to generate these two types of RDF resources, both constant and dynamically. All can generate a subject with one or more data references (e.g., in RML `rr:template "http://ex.org/{id}{name}"`), ShExML, xR2RML, SPARQL-Generate, Facade-X, and Helio with different formats. For example, in xR2RML a CSV field that contains an array can be expressed as: `xrr:reference "Column(Movies)/JSONPath($.*)`. Part of the languages even allow generating subjects with more than one data source, this is the case of ShExML, XSPARQL, KR2RML, SPARQL-Generate, Facade-X, Helio and xR2RML. About a third of the languages allow hierarchy iterations (ShExML, XSPARQL, KR2RML, SPARQL-Generate, D-REPR, Facade-X and D2RML), and more than a half use functions with N:1 cardinality. Additionally, some of them even allow functions that can output more than one parameter (i.e., 1:N or N:M), but it is less usual.

Predicate Generation. All languages can generate constant predicates as IRIs. Only four languages do not allow dynamic predicates (ShExML, and XLWrap). For those that do, they also allow more than one data reference. The languages that allow subject generation using multiple formats, data sources, functions, and hierarchy iterations, provide the same features for predicate generation.

Object Generation. Generally, languages can generate a wider range of resources for objects, since they can be IRIs, blank nodes, literals, lists, or containers. All of them can generate constant and dynamic literals and IRIs. Those languages that allow blank nodes in the subject also allow them in the object. Additionally, ShExML, KR2RML, SPARQL-Generate, Facade-X, R2RMLcc and xR2RML consider lists, and the last two languages also consider containers (e.g., `rr:termType xrr:RdfBag` in xR2RML). Data references, sources, hierarchy iterations, and functions remain the same as in subject generation. Lastly, datatype and language tags are not allowed in KR2RML and XLWrap; they are defined as constants in the rest of the languages, and dynamically in ShExML, XSPARQL, TARQL and Helio.

4.1.4 General Features for Graph Construction

Table 4.3 shows the features of mapping languages regarding the construction of RDF graphs such as *linking rules*, *metadata* or *conditions*, assignment to *named graphs*, and declaration of *transformation functions* within the mapping.

Statements. General features that apply to statements are described in this section: the capability of a language to assign statements to *named graphs*, to *retrieve data from only one source* or *more than one source*, and to apply *conditions* that have to be met in order to create the statement (e.g., if the value of a field called "required" is `TRUE`, the triple is generated).

Most RDF-based languages allow static assignment to named graphs. R2RML, RML, R2RML-F, FunUL, and D2RML enable also dynamic definitions (e.g., `rr:graphMap` in R2RML and in

Table 4.3: Statements, linking rules, and function properties of the studied mapping languages from the references stated in Table 2.1.

Feature		ShExML	XSPARQL	TARQL	CSVW	R2RML	RML	KR2RML	xR2RML	SPARQL-Generate	R2RML-F	FunUL	Helio	D-REPR	XLWrap	D2RML	Facade-X	R2RMLcc
Statements	Assign to named graphs	static	–	–	–	static, dynamic	static, dynamic	static	static	–	static, dynamic	static, dynamic	–	–	static	static, dynamic	–	static, dynamic
	Retrieve data from one source	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Retrieve data from one or more sources	✓	✓	×	✓	×	×	×	×	✓	✓	×	✓	✓	×	✓	✓	×
	Allow conditions to form statements	✓	✓	✓	×	×	✓[a]	×	×	✓	✓	×	×	×	✓	✓	✓	×
Linking rules	Use one data reference	✓	✓	×	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	×	✓	✓	✓
	Use one or more data references	✓	×	×	×	×	✓	×	✓	✓	×	×	✓	✓	×	×	✓	×
	No condition to link	✓	✓	×	×	✓	✓	×	✓	✓	✓	✓	✓	×	×	✓	✓	✓
	Link with one condition	✓	✓	×	×	✓	✓	×	✓	✓	✓	✓	✓	✓	×	✓	✓	✓
	Link with one or more conditions	×	✓	×	×	✓	✓	×	✓	✓	✓	✓	✓	✓	×	✓	✓	✓
	Use only equal function in condition	✓	✓	×	×	✓	✓	×	✓	✓	✓	✓	✓	✓	×	✓	✓	✓
	Use any similarity function in condition	×	✓	×	×	×	✓	×	×	×	×	×	✓	×	×	✓	✓	×
Functions	Cardinality	1:1, N:1	1:1, N:1, 1:N, N:M	1:1, N:1	–	–	1:1, N:1*[a]	1:1, N:1, 1:N, N:M	–	1:1, N:1, 1:N, N:M	1:1, N:1	1:1, N:1	1:1, N:1	1:1, N:1, 1:N, N:M	1:1, N:1, 1:N, N:M	1:1, N:1, 1:N, N:M	1:1, N:1, 1:N, N:M	–
	Nested functions	×	✓	✓	×	×	✓[a]	×	×	✓	✓	✓	✓	✓	✓	✓	✓	×
	Functions belong to a specification	✓	×	✓	×	×	✓[a]	×	×	✓	×	×	✓	×	✓	×	✓	×
	Declare own functions	✓	✓	×	×	×	✓[a]	×	×	✓	✓	✓	×	×	✓	✓	✓	×

[a]With the Function Ontology (FnO) (De Meester et al., 2017)

its extensions). Theoretically, the rest of R2RML extensions should also implement this feature; however, to the best of our knowledge, it is not mentioned in their respective specifications.

Allowing conditional statements is not usual; it is only considered in the SPARQL-based languages, XLWrap and D2RML (e.g., `xl:breakCondition` in XLWrap). Regarding data sources, all languages allow data retrieval from at least one source; ShExML, XSPARQL, CSVW, SPARQL-Generate, Facade-X, Helio, D-REPR and D2RML enable more sources. That is, using data in the same statement from, e.g., one CSV file and one JSON file.

Linking Rules. Linking rules refer to linking resources that are being created in the mapping. For instance, having as object of a statement a resource that is the subject of another statement. These links are implemented in most languages by joining one or more data fields. Six languages do not allow these links: TARQL, CSVW, KR2RML and XLWrap. The rest is able to perform linking with at least one data reference and one or no condition. Fewer enable more data references and more conditions (e.g., in R2RML and most extensions allow the application of a `rr:joinCondition` over several fields).

Linking rules using join conditions imply evaluating if the fields selected are equal. Since the join condition is the most common, applying the equal logical operator is the preferred choice. Only a few languages consider other similarity functions to perform link discovery, such as the Levenshtein distance and Jaro-Winkler, e.g., Helio.

Transformation functions. Applying functions in mappings allows practitioners transforming data before it is translated. For instance, to generate a label with an initial capital letter (`ex:ID001 rdfs:label "Emily"`) that was originally in lower case ("emily"), a function may be applied (e.g., GREL function `toTitleCase()`). Only four of the analyzed languages do not allow the use of these functions: CSVW, R2RML and xR2RML. Of those that do, some use functions that belong to a specification (e.g., RML+FnO uses GREL functions[2]). All of them consider functions with cardinalities 1:1 and N:1; and half of them also include 1:N and N:M (i.e., output more than one value), for instance, a regular expression that matches and returns more than one value. Nesting functions (i.e., calling a function inside another function) is not unusual; this is the case of SPARQL-based languages, the R2RML extensions that implement functions (except K2RML), Helio, D-REPR, and XLWrap. Finally, some languages even enable extending functions depending on specific user needs, such as XSPARQL, RML+FnO, SPARQL-Generate, Facade-X, R2RML-F, FunUL, XLWrap and D2RML.

4.2 A Unified Model to Represent Mapping Rules: Conceptual Mapping

Based on the analysis performed in Section 4.1, we abstract the features and limitations present in current mapping languages to represent them in an ontology. This ontology aims at gathering the expressiveness of current mapping languages, and is called the Conceptual

[2] https://docs.openrefine.org/manual/grelfunctions

Mapping[3] (Iglesias-Molina et al., 2024). The methodology followed to build this ontology is first presented, and then each step of its construction is described in detail.

4.2.1 Methodology

The Conceptual Mapping ontology was developed following the guidelines provided by the Linked Open Terms (LOT) methodology. LOT is a well-known and mature lightweight methodology for the development of ontologies and vocabularies that has been widely adopted in academic and industrial projects (Poveda-Villalón et al., 2022). It is based on the previous NeOn methodology (Suárez-Figueroa et al., 2015) and includes four major stages: Requirements Specification, Implementation, Publication, and Maintenance (Fig. 4.1). In this section, we describe these stages and how they have been applied and adapted to the development of the Conceptual Mapping ontology.

Requirements specification This stage refers to the activities carried out for defining the requirements that the ontology must meet. At the beginning of the requirements identification stage, the goal and scope of the ontology are defined. Following, the domain is analyzed in more detail by looking at the documentation, data that has been published, standards, formats, etc. In addition, use cases and user stories are identified. Then, the requirements are specified in the form of competency questions and statements.

In this case, the specification of requirements includes purpose, scope, and requirements. The requirements are specified as facts rather than competency questions and validated with Themis (Fernández-Izquierdo et al., 2021), an ontology evaluation tool that allows validating requirements expressed as tests rather than SPARQL queries. We consider this approach to be adequate in this case since (1) there are no use cases as this ontology is a mechanism of representation of mapping language's features; and (2) there are no SPARQL queries because they result from Competency Questions which are in turn extracted from use cases and user stories. Further details are shown in Section 4.2.2.

Implementation The goal of the Implementation stage is to build the ontology using a formal language, based on the ontological requirements identified in the previous stage. From the set of requirements a first version of the model is conceptualized. The model is subsequently refined by running the corresponding evaluations. Thus, the implementation process follows iterative sprints; once it passes all evaluations and meets the requirements, it is considered ready for publication.

The conceptualization is carried out representing the ontology in a graphical language using the Chowlk notation (Chávez-Feria et al., 2022) (as shown in Fig. 4.2). The ontology is implemented in OWL 2 using Protégé. The evaluation checks different aspects of the ontology: (1) requirements are validated using Themis (Fernández-Izquierdo et al., 2021), (2) inconsistencies are found with the Pellet reasoner, (3) OOPS! (Poveda-Villalón et al., 2014) is used to identify modeling pitfalls, and (4) FOOPS! (Garijo et al., 2021) is run to check the FAIRness of the ontology. Further details are described in Section 4.2.3.

Publication The publication stage addresses the tasks related to making the ontology and its

[3] https://w3id.org/conceptual-mapping/portal

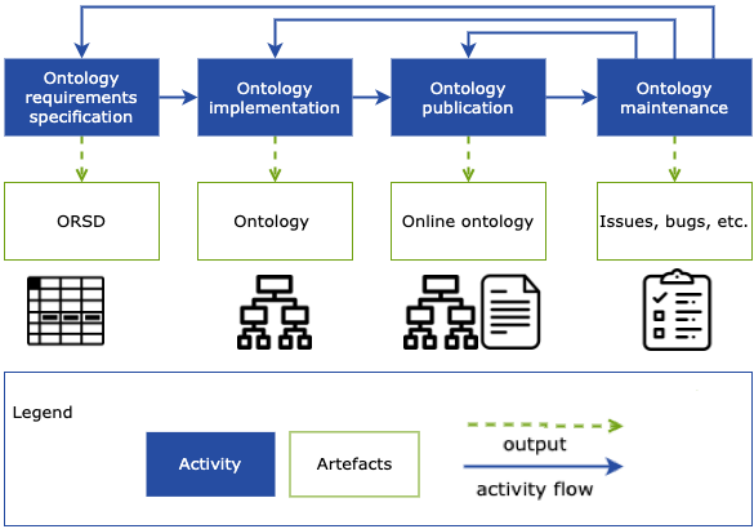


Figure 4.1: Workflow proposed by the LOT Methodology (Poveda-Villalón et al., 2022).

documentation available. The ontology documentation was generated with Widoco (Garijo, 2017), a built-in documentation generator in OnToology (Alobaid et al., 2019), and it is published with a W3ID URL[4]. The ontology and related resources can be accessed in the ontology portal. Further details are presented in Section 4.2.4.

Maintenance Finally, the last stage of the development process, maintenance, refers to ontology updates as new requirements are found and/or errors are fixed. The ontology presented in this work promotes the gathering of issues or new requirements through the use of issues in the ontology GitHub repository. Additionally, it provides control of changes, and the documentation enables access to previous versions. Further details are shown in Section 4.2.4.

4.2.2 Requirements

This section presents the purpose, scope, and requirements of the Conceptual Mapping Ontology. In addition, it also describes from where and how the requirements are extracted: analysing the mapping languages (presented as a comparative framework in Section 4.1) and the Mapping Challenges proposed by the community.

Purpose and Scope

The Conceptual Mapping ontology aims at gathering the expressiveness of declarative mapping languages that describe the transformation of heterogeneous data sources into RDF. This ontology-based language assumes that all mapping languages used for the same basic purpose of describing data sources in terms of an ontology to create RDF, must share some basic patterns and inherent characteristics. Inevitably, not all features are common. As described in previous sections, some languages were developed for specific purposes, others extend

[4] https://w3id.org/conceptual-mapping

existing languages to cover additional use cases, and others are in turn based in languages that already provide them with certain capabilities. The Conceptual Mapping ontology is designed to represent and articulate these core features, which are extracted from two sources: (i) the analysis of current mapping languages, and (ii) the limitations of current languages identified by the community. These limitations, proposed by the W3C Knowledge Graph Construction Community Group[5], are referred to as Mapping Challenges[6] and have been partially implemented by some languages.

This ontology also presents some limitations. As shown in Section 2.2, mapping languages can be classified into three categories according to the schema in which they are based: RDF-based, SPARQL-based and based on other schemes. The Conceptual Mapping is included in the first category and, as such, has the same inherent capabilities and limitations as RDF-based languages regarding the representation of the language as an ontology. This implies that it is feasible to represent their expressiveness, whereas reusing classes and/or properties or creating equivalent constructs. Languages based on other approaches usually follow schemas that make them relatable to ontologies. This can be seen in the correspondence between YARRRML and RML: RML is usually written in Turtle syntax. YARRRML (Heyvaert et al., 2018) is mainly used as a user-friendly syntax to facilitate the writing of RML rules. It is based on YAML, and can easily be translated into RML[7].

Lastly, SPARQL-based languages pose a challenge. SPARQL is a rich and powerful query language (J. Pérez et al., 2009) to which these mapping languages add more capabilities (e.g., SPARQL-Generate, SPARQL-Anything). It has an innate flexibility and capabilities sometimes not comparable to the other languages. For this reason, representing every single capability and feature of SPARQL-based languages is out of the scope of this work. Given the differences of representation paradigm between RDF and SPARQL for creating mappings, it cannot be ensured that the Conceptual Mapping covers all possibilities that a SPARQL-based language can, and as such, is considered a limitation.

Mapping Challenges

Following its inception, the W3C Knowledge Graph Construction Community Group[5] defined a series of challenges for mapping languages based on the experience of members in using declarative mappings[6]. These challenges are a summary of the limitations of current languages. They have been partially addressed independently in some of the analyzed languages, such as RML (Delva, Van Assche, et al., 2021; Iglesias-Molina, Van Assche, et al., 2023b) and ShExML (García-González, 2021). These challenges are summarized as follows:

- **[C1] Language Tags and Datatype.** It refers to dynamically building language tags [C1a] and datatypes [C1b], that is, from data rather than as constant values.
- **[C2] Iterators.** This challenge addresses the need to access data values 'outside' the iteration pattern [C2a], especially in some tree-like data sources such as JSON; and iterating over multi-value references [C2b].

[5] https://www.w3.org/community/kg-construct/
[6] https://w3id.org/kg-construct/workshop/2021/challenges.html
[7] https://rml.io/yarrrml/matey/

- **[C3] Multi-value References.** It discusses how languages handle data fields that contain multiple values [C3a], their datatypes and associated language tags [C3b].
- **[C4] RDF Collections and Containers.** This challenge addresses the need to handle RDF collections and containers.
- **[C5] Joins.** It refers to joining resources with zero join conditions [C5a] and joining literals instead of IRIs [C5b].

Conceptual Mapping Requirements

In order to extract the requirements that serve as the basis for the development of the Conceptual Mapping ontology, we take as input the analysis from the comparison framework (Section 4.1) and the Mapping Challenges (Section 4.2.2) described previously and the expertise of the authors. From a combination of their features, we extract 28 requirements (see Appendix A). These requirements are expressed as facts, and are available in the ontology repository and portal[8]. Each requirement has a unique identifier, its provenance (comparison framework or mapping challenge id) and the corresponding constructs in the ontology. The constructs are written in Turtle, and lack cardinality restrictions for the sake of understandability. These requirements are tested with Themis, and its corresponding tests include these restrictions. More details on the evaluation of the requirements are provided in Section 4.2.3.

The requirements gathered range from general-purpose to fine-grained details. The general-purpose requirements refer to the basic fundamental capabilities of mappings, e.g., to create the rules to generate RDF triples (cm-r8) from reference data sources (cm-r7). The requirements with the next level of detail involve some specific restrictions and functionalities, e.g., to indicate the specific type (whether they are IRIs, Blank nodes, etc.) of subjects (cm-r16), predicates (cm-r17), objects (cm-r18), named graphs (cm-r19), datatypes (cm-r20) and language tags (cm-r21); the possibility of using linking conditions (cm-r23) and functions (cm-r15). Finally, some requirements refer to specific details or features regarding the description of data sources (e.g., cm-r4, cm-r6) and transformation rules (e.g., cm-r14, cm-r22, cm-r25).

Not all the observed features in the comparison framework have been added to the set of requirements. Some features are really specific, and supported by a minority of languages, sometimes only one language. As a result, we selected the detailed features in these requirements to build the core specification of the Conceptual Mapping when they tackled the basic functionalities of the language. The rest of the details are left to be included as extensions. This differentiation and the modeling criteria is explained further in Section 4.2.3.

4.2.3 Implementation

This section describes in detail the activities and tasks carried out to implement the ontology, that consists in the conceptualization of the model, the encoding in a formal language, and the evaluation to fix errors, inconsistencies, and ensure that it meets the requirements. Additionally, an example of the ontology's use is presented at the end of the section.

[8] https://oeg-upm.github.io/Conceptual-Mapping/requirements/requirements-core.html

Ontology Conceptualization

The ontology's conceptualization is built upon the requirements extracted from experts experience, a thorough analysis of the features and capabilities of current mapping languages presented as a comparative framework; and the languages' limitations discussed by the community and denoted as Mapping Challenges. The resulting ontology model is depicted in Fig. 4.2. This model represents the core specification of the Conceptual Mapping ontology that contains the essential features to cover the requirements. Some detailed features are also included when considered important to the language expressiveness, or needed for the language main functionality. Other detailed features are considered as extensions, as explained further in Section 4.2.5. For description purposes, we divide the ontology into two parts, *Statements* and *Data Sources*, that compose the core model. These two parts, when not used in combination, cannot describe a complete mapping. For that reason they are not separated into single modules.

Data sources. A data source (`DataSource`) describes the source data that will be translated. For this section, the Data Catalog (DCAT) vocabulary (Albertoni et al., 2020) has been reused. `DataSource` is a subclass of `dcat:Distribution`, which is a specific representation of a dataset (`dcat:Dataset`), defined as "data encoded in a certain structure such as lists, tables and databases". A source can be a streaming source (`StreamSource`) that continuously generates data, a synchronous source (`SynchronousSource`) or an asynchronous source (`AsynchronousSource`). Asynchronous sources, in turn, can be event sources (`EventSource`) or periodic sources (`Periodic Source`). The details of the data source access are represented with the data access service class (`Data AccessService`), which in turn is a subclass of `dcat:DataService`. This class represents a collection of operations that provides access to one or more datasets or data processing functions, i.e., a description of how the data is accessed and retrieved. The data access service optionally has a security scheme (e.g., OAuth2, API Key) and an access protocol (e.g., HTTP(s), FTP).

Data properties in the `dcat:Dataset`, `dcat:Distribution` and `dcat:DataService` classes may be reused according to the features that may be represented in each mapping language, e.g., `dcat:endpointURL`, `dcat:accessURL` and `dcat:endpointDescription`. A data access service is related to a security scheme. The class `wot:SecurityScheme` (from the Web of Things (WoT) Security ontology[9]) has been reused. This class has different types of security schemes as subclasses and includes properties to specify the information on the scheme (e.g., the encryption algorithm, the format of the authentication information, the location of the authentication information). The security protocol `hasProtocol` has as set of predefined values that have been organized as a SKOS concept scheme. It contains almost 200 security protocols, e.g., HTTP(s), JDBC, FTP, GEO, among others. This SKOS list can be extended according to the users' needs by adding new concepts.

In order to represent the fragments of data that are referenced in a statement map, the class `Frame` has been defined. They are connected with the property `hasFrame`. A frame can be a `SourceFrame` (base case) or a `CombinedFrame`, the latter representing two source

[9] https://www.w3.org/2019/wot/security

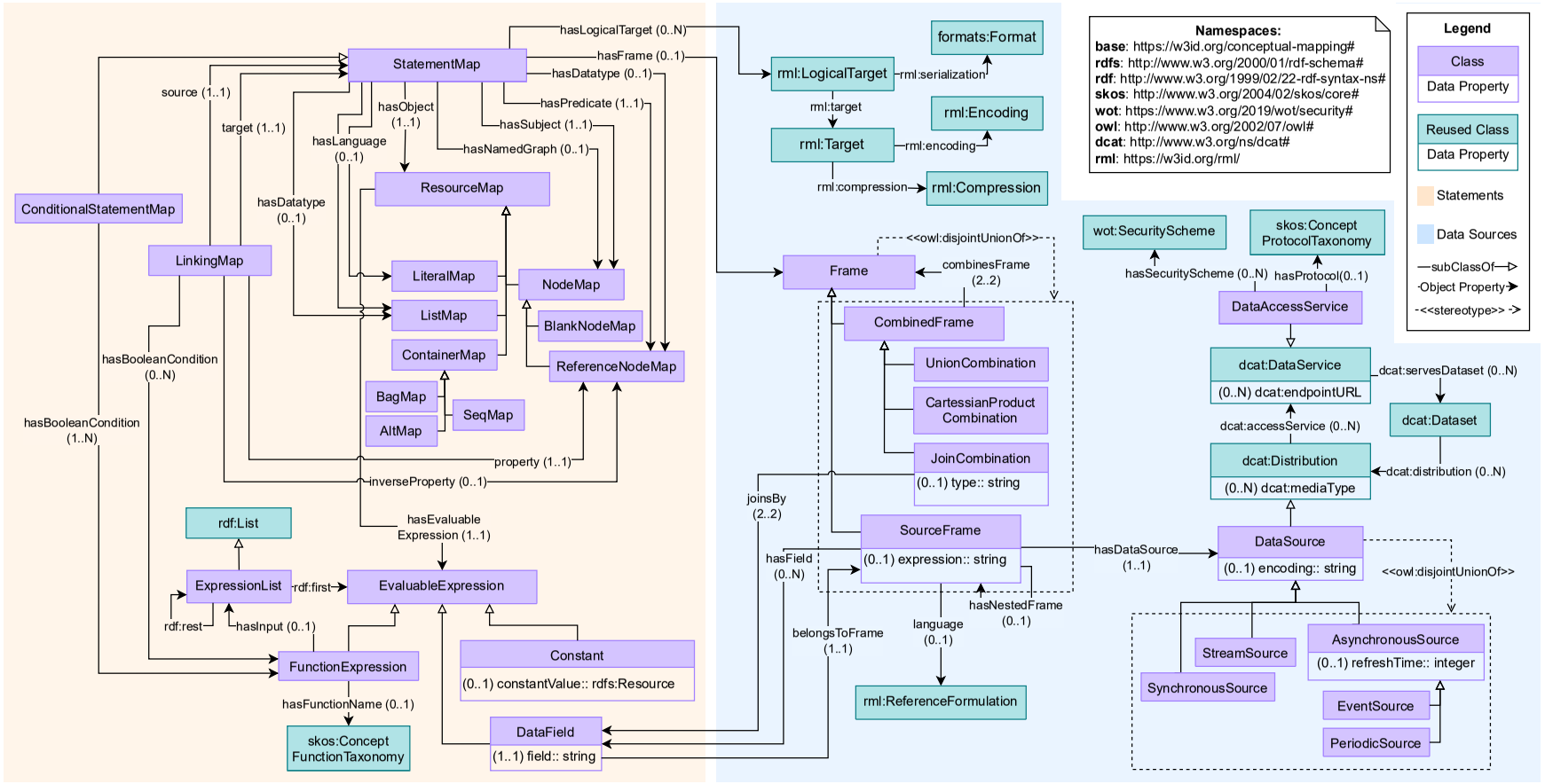


Figure 4.2: Visual representation of the Conceptual Mapping ontology created using the Chowlk visual notation (Chávez-Feria et al., 2022).

frames or combined frames that are combined by means of a join (`JoinCombination`), a union (`UnionCombination`) or a cartessian product (`CartessianProductCombination`).

A source frame corresponds to a data source (with `hasDataSource`) and defines which data is retrieved from the source and how it is fragmented (with `expression`). Among others, JSONPaths, XPaths, queries, or regular expressions can be expressed with this feature. The language of the expression is defined with `language`, of which domain is the reused class from RML `rml:ReferenceFormulation`. A source frame may be related to another source frame with `hasNestedFrame`, e.g., a frame is accessed firstly with a SPARQL query, and their results as a CSV file with this property. A source fragment may refer to many data fields (with `hasField`, which is the inverse property of `belongsToFrame`).

The desired output of the statements that are to be generated can be described with the imported `rml:LogicalTarget`, referenced by the optional property `hasLogicalTarget`. A `rml:LogicalTarget` specifies which RDF serialisation the output should be encoded (`rml:serialization`), and the referred `rml:Target`. The `rml:Target` indicates how the output target can be accessed, and can describe: (i) the compression format for the RDF output (`rml:compression`) and, if so, how e.g., GZip, and (ii) the encoding (`rml:encoding`), e.g., UTF-8.

Statements. The central class of this section is the `StatementMap`, which represents a rule that defines for a triple its subject (`hasSubject`), predicate (`hasPredicate`), and object (`hasObject`). Optionally, it can also specify the object datatype (`hasDatatype`), language (`hasLanguage`) and assigned named graph (`hasNamedGraph`). Therefore, statement maps are similar to RDF statements as both of them are comprised by a subject, predicate and object. In statement maps, objects are resources (`ResourceMap`), and subjects and predicates are more specific, certain subclasses of the resource map: predicates are reference node maps (`ReferenceNodeMap`) that represent resources with an IRI, i.e., ontology properties. Subjects are node maps (`NodeMap`) that may be blank nodes (`Blank Node`) or also reference node maps. An object may be a literal (`LiteralMap`), a blank node, a container (`ContainerMap`) or a collection that defines a list (`ListMap`). The language is expressed as a literal, and the datatype is also a resource with an IRI, i.e., a reference node map.

Resource maps are expressed with an evaluable expression (`EvaluableExpression`) that may be a constant value (`Constant`), a function expression (`FunctionExpression`), or a data field (`DataField`) that belongs to some data source fragment (`belongsToFrame`). For function expressions, the function name (`hasFuntionName`) is taken from a set of predefined names organized in a SKOS concept scheme. This SKOS list can be extended according to the users' needs by adding new concepts for functions that have not been defined. Recursion in this function expression is represented through its input (`hasInput`) as an expression list (`ExpressionList`). Expression lists have been represented as a subclass of RDF lists (`rdf:List`), and the properties (`rdf:first`) and (`rdf:rest`) have been reused. Expression lists may have nested expression lists inside.

A special case of a statement map is a conditional statement map (`ConditionalStatementMap`), a statement map that must satisfy a condition for the triples to be generated.

The condition (`hasBooleanCondition`) is a function expression (e.g., if a value from a field called "present" is set to "False", the statement is not generated). Another relevant class is the linking map (`LinkingMap`), that enables linking subjects from a source (`source`) and a target (`target`) statement maps, i.e., two resources are linked and triples are generated if a linking condition is satisfied. Similarly to the conditional statement map, this condition is represented as a function expression.

Ontology Design Patterns

The following ontology design patterns have been applied in the conceptualization as they are common solutions to the problem of representing taxonomies and linked lists:

- The SKOS vocabulary has been reused to represent some coding schemes such as the protocol taxonomy and the function taxonomy. The design pattern consists on having an instance of `skos:ConceptScheme` for each taxonomy, then each concept or term in the taxonomy, `skos:Concept`, is related to the corresponding concept scheme through the property `skos:inScheme`. The class that uses the taxonomy is then related to `skos:Concept` through an object property, e.g., class `DataAccessService` and object property `hasProtocol`.
- The class `ExpressionList` uses the design pattern for lists developed in RDF where the properties `rdf:first` and `rdf:rest` are used to represent a linked list. The base case (first) is an evaluable expression whereas the rest of the list is (recursively) an `ExpressionList`.

Ontology Evaluation

The ontology, implemented in OWL 2 with Protégé, has been evaluated in different ways to ensure that it is correctly implemented, has no errors, inconsistencies or pitfalls, and meets the requirements.

Reasoner. We used the reasoner Pellet in Protégé to look for inconsistencies in the model, and the results showed no errors.

OOPS!. (Poveda-Villalón et al., 2014) This tool was used to identify modeling pitfalls in the ontology. We executed the tool several times to fix the pitfalls, until there were no important ones. Currently, the results of OOPS! show pitfalls from the reused ontologies, but none important for the newly created terms and axioms. One minor pitfall is returned, P13, regarding the lack of inverse relationships, which we consider are not needed in the ontology. The rest of the pitfalls are as follows: P08 (missing annotations) from DCTERMS; P11 (missing domain or range in properties) for DCTERMS, DCAT and SKOS; and P20 (missing ontology annotations) for DCAT.

Themis. (Fernández-Izquierdo et al., 2021) Themis is a tool to evaluate whether the requirements are implemented in the ontology. To that end, the requirements must be

provided in a specific syntax or described with the Verification Test Case (VTC) ontology[10]. The requirements of the Conceptual Mapping were translated to create the corresponding tests, and were tested in the tool with success. The requirements and associated test along with the complete set of tests annotated with the VTC ontology are available in the GitHub repository[11].

FOOPS!. (Garijo et al., 2021) Additionally, we tried running FOOPS! to check the FAIRness of the ontology, resulting in 73%, which is acceptable. To improve the score, the ontology should be added to a registry, have more metadata describing it, and use a persistent base IRI.

With these validations, we can conclude that the ontology is correctly encoded and implemented, and that it meets the requirements specified in Section 4.2.2.

4.2.4 Publication and Maintenance

To publish the ontology, the first required step is to create the ontology documentation. We used Widoco (Garijo, 2017), integrated inside the OnToology (Alobaid et al., 2019) system, to automatically generate and update the HTML documentation every time there is a commit in the GitHub repository where the ontology is stored. This documentation contains the ontology metadata, links to the previous version, a description of the ontology, the diagram, and detailed examples of the capabilities of the language. It is published using a W3ID URL[4] and under the CC BY-SA 4.0 license.

The HTML documentation is not the only documentation resource provided. An overview of all resources is provided in the ontology portal[3]. This portal shows in a table the ontologies associated with the Conceptual Mapping ontology. For now, the core (Conceptual Mapping) and an extension to describe CSV files in detail (Conceptual Mapping - CSV Description) are available. For each ontology, links to the HTML documentation, the requirements, the GitHub repository, the Issue Tracker, and the releases are provided.

The maintenance is supported by the Issue Tracker[12], where proposals for new requirements, additions, deletions or modifications can be added as GitHub issues. This approach allows authors to review the proposals and discuss their possible implementation.

4.2.5 Extensions

The Conceptual Mapping ontology has been designed as a core ontology. However, as time passes, new requirements may emerge. In order to include these new requirements, new modules of the Conceptual Mapping ontology shall be developed. It is worth mentioning that this is a common practice for ontologies, which is highly suitable for adapting an existing ontology to new scenarios, by ontology modules specialized for a specific set of

[10] https://albaizq.github.io/test-verification-ontology/OnToology/ontology/verification-test-description.ttl/documentation/index-en.html

[11] https://github.com/oeg-upm/Conceptual-Mapping/tree/main/requirements

[12] https://github.com/oeg-upm/Conceptual-Mapping/issues

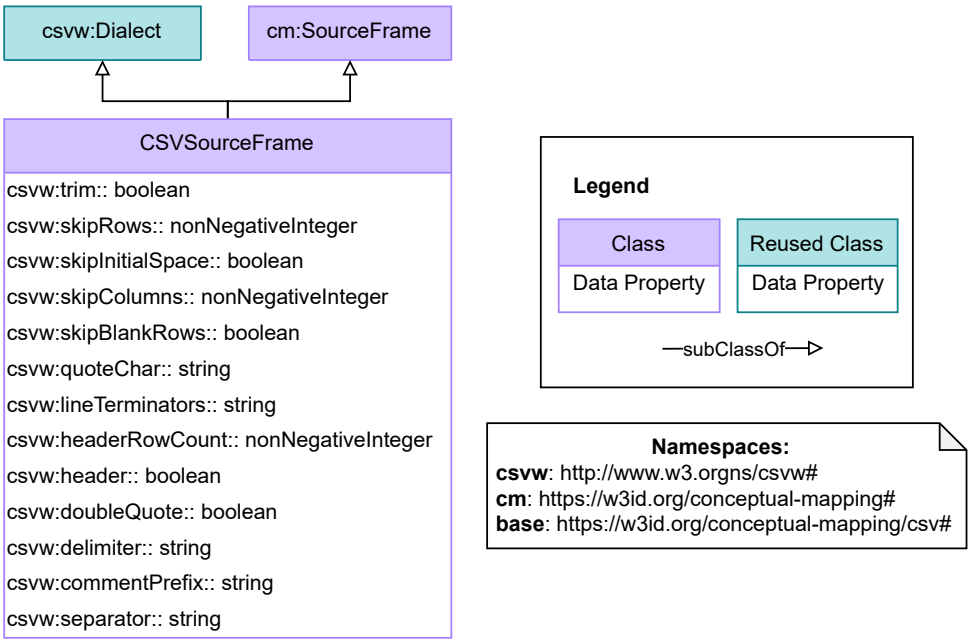


Figure 4.3: Conceptualization of the CM-CSV module following the Chowlk visual notation (Chávez-Feria et al., 2022).

requirements. A clear example of this is the SAREF ontology[13], that has a core module[14] and then specific extensions[15] for certain domains, such as energy (SAREF4ENER) or buildings (SAREF4BLDG) among others. For the Conceptual Mapping we present two extensions: for allowing a more detailed description of CSV files, and for generating RDF-star graphs.

CSV Description: CM-CSV

The core module of the Conceptual Mapping includes limited possibilities for describing CSV files in depth. The extension CM-CSV[16] provides extended features to this end, enabling extended features for describing more complex files. Thus, the CSVW (Tennison et al., 2015) proposal has been blended as an ontology module linked to the core Conceptual Mapping ontology. The class `CSVSourceFrame` is created as a subclass of `SourceFrame` and `csvw:Dialect` to inherit their characteristics. This module is depicted in Fig. 4.3. Thus, this extension allows to describe CSV characteristics such as if the file contains headers (`csvw:header`), its delimiter (`csvw:delimiter`), separator (`csvw:separator`), among many others.

[13] https://saref.etsi.org/
[14] https://saref.etsi.org/core/v3.1.1/
[15] https://saref.etsi.org/extensions.html
[16] https://w3id.org/conceptual-mapping/csv

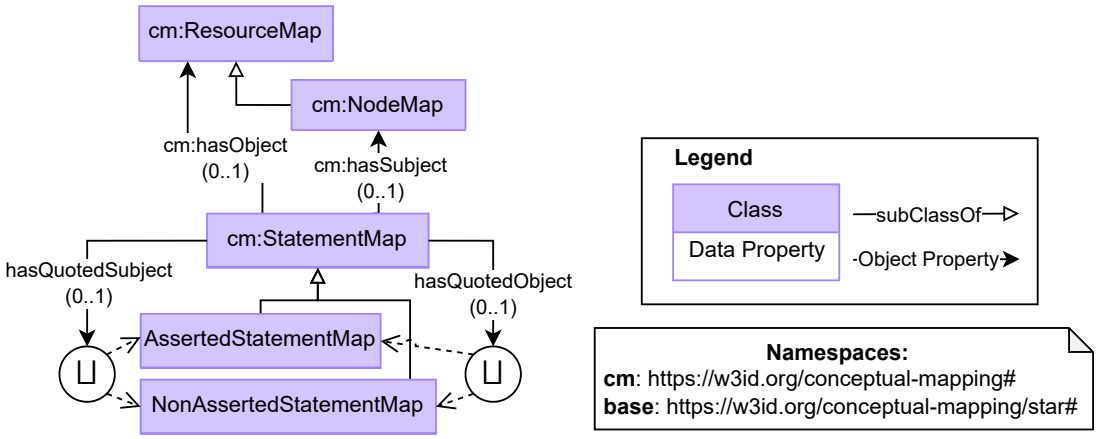


Figure 4.4: Conceptualization of the CM-star module following the Chowlk visual notation (Chávez-Feria et al., 2022).

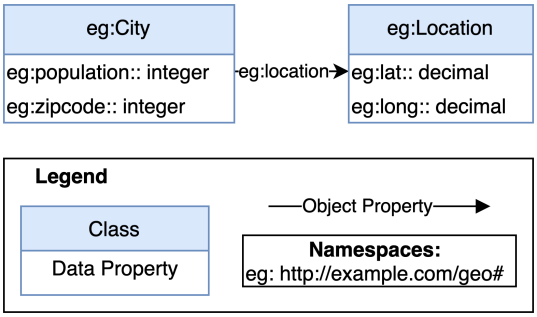
RDF-star Generation: CM-star

RDF-star (Hartig, 2017) was recently proposed as a compact alternative for reification in RDF. It extends the RDF syntax introducing the notion of *Quoted triples*, i.e., triples that can be placed in the position of objects and/or subjects. These triples can be inserted in the graph outside the quoted triple (i.e., they are *asserted*), or appear only inside another triples (i.e., they are *non-asserted*).

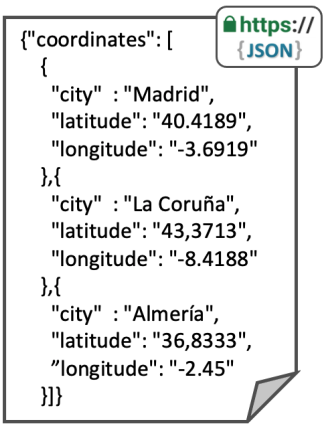
RDF-star has quickly gained popularity, leading to its adoption by a wide range of systems[17] (e.g., Apache Jena [18], Oxigraph (Pellissier Tanon, 2023)) and the formation of the RDF-star Working Group[19]. Regarding mapping languages, SPARQL-Anything is able to generate RDF-graphs as it natively implements the updates from Apache Jena; RML was extended for this purpose with the RML-star module (Delva, Arenas-Guerrero, et al., 2021; Iglesias-Molina, Arenas-Guerrero, et al., 2022; Iglesias-Molina, Van Assche, et al., 2023b) (explained in detail in Section 4.3), as well as R2RML-star (Sundqvist, 2022).

As a result, we developed a module to cover this new feature. The Conceptual Mapping implements the RDF-star graph generation adopting the recursive nature of quoted triples in the CM-star module[20]. Fig. 4.4 represents the overview of this module. Two subclasses of `StatementMap` are created to denote asserted (`AssertedStatementMap`) and non-asserted (`NonAssertedStatementMap`) statement maps. These statements can be quoted as subjects and objects with the properties `hasQuotedSubject` and `hasQuotedObject` respectively. The range of these properties is the union of the introduced statement subclasses. This extension enables potentially an infinite number of quoted statements, as the RDF-star specification indicates (Hartig et al., 2023).

[17] https://w3c.github.io/rdf-star/implementations.html
[18] https://jena.apache.org
[19] https://www.w3.org/groups/wg/rdf-star
[20] https://w3id.org/conceptual-mapping/star



(a) Example reference ontology that represents the classes `City` and `Location`, linked by the property `eg:location`.



(b) Example input JSON file "coordinates.json".

city	population	year_modified	zipcodes
A Coruña	244850	2018	15001, 15002, 15003, 15004
Almeria	201322	2021	04001, 04002
Madrid	3334730	2021	28001, 28002, 28003, 28004, 28005, 28006

(c) Example input MySQL table "cities".

Figure 4.5: Input source data and reference ontology that represents information on cities and their location.

4.2.6 Ontology Usage Example

This section builds a mapping in three steps (data sources in Listing 4.2, triples in Listing 4.3 and special statements in Listing 4.4) to represent how the proposed language can describe data with different features. The mapping uses the data sources "coordinates.json" (Fig. 4.5b) and "cities" (Fig. 4.5c) as input and the ontology depicted in Fig. 4.5a as reference, to create the output RDF shown in Listing 4.1.

```
1  <http://ex.com/loc/40.4189--3.6919> a eg:Location ;
2    eg:lat "40.4189"^^xsd:decimal ;
3    eg:long "-3.6919"^^xsd:decimal .
4  <http://ex.com/loc/43.3713--8.4188> a eg:Location ;
5    eg:lat "43.3713"^^xsd:decimal ;
6    eg:long "-8.4188"^^xsd:decimal .
7  <http://ex.com/loc/36.8333--2.45> a eg:Location ;
8    eg:lat "36.8333"^^xsd:decimal ;
9    eg:long "-2.45"^^xsd:decimal .
10 <http://ex.com/city/ACoruña> a eg:City ;
11   eg:zipcode 15001, 15002, 15003, 15004 ;
12   eg:location <http://ex.com/loc/43.3713--8.4188> .
13 <http://ex.com/city/Almería> a eg:City ;
```

```
  eg:zipcode 04001, 04002 ;
  eg:population 201322 ;
  eg:location <http://ex.com/loc/36.8333--2.45> .
<http://ex.com/city/Madrid> a eg:City ;
  eg:zipcode 28001, 28002, 28003, 28004, 28005, 28006;
  eg:population 3334730 ;
  eg:location <http://ex.com/loc/40.4189--3.6919> .
```

Listing 4.1: Expected RDF output for the data sources and the ontology in Fig. 4.5.

Data sources. Listing 4.2 shows the description of the JSON file "coordinates.json" indicating the protocol from the SKOS concept scheme (`cmp:https`), media type ("application/json"), JSONPath to extract data, access URL "https://ex.com/geodata/coordinates.json", and fields that are going to be used in the transformation. There is no security scheme. The MySQL table "cities" also has no security scheme, the protocol needed is `cmp:jdbc`, the database access is specified in the endpoint URL, and the table as an SQL query. The fields are also specified, with the special case of "zipcodes" that needs a `cm:hasNestedFrame` to extract multiple values inside the field.

```
# Locations
:FrameLoc a cm:SourceFrame;
  cm:expression "$.coordinates[*]";
  cm:language ql:JSONPath ;
  cm:hasField :lat;
  cm:hasField :long;
  cm:hasField :loc_city;
  cm:hasDataSource [ a cm:SynchronousSource;
    dcat:mediaType "text/json";
    dcat:accessService [
      cm:hasProtocol cmp:https;
      dcat:endpointURL "https://ex.com/geodata/coordinates.json"
      cm:hasSecurityScheme [ a wotsec:NoSecurityScheme; ];
    ] ;
  ] .

:lat a cm:DataField ; cm:field "$.latitude" .
:long a cm:DataField ; cm:field "$.longitude" .
:loc_city a cm:DataField; cm:field "$.city" .

# Cities
:FrameCities a cm:SourceFrame ;
  cm:expression "SELECT * FROM cities;";
  cm:hasField :c_city;
  cm:hasField :population;
  cm:hasField :year;
  cm:hasNestedFrame [
    cm:expression "$.zipcodes[*]";
    cm:hasField :zipcode ];
  cm:hasDataSource [ a cm:SynchronousSource;
    dcat:mediaType "text/plain";
```

```
    dcat:accessService [
      cm:hasProtocol cmp:jdbc;
      dcat:endpointURL "jdbc:mysql://localhost:3306/citydb";
      cm:hasSecurityScheme [a wotsec:NoSecurityScheme;] ].

:c_city a cm:DataField; cm:field "city" .
:population a cm:DataField; cm:field "population" .
:year a cm:DataField; cm:field "year_modified" .
:zipcode a cm:DataField cm:field "zipcodes" .
```

Listing 4.2: Description with the Conceptual Mapping of two data sources (a JSON file and a relational database), their access and fields.

Statements. Listing 4.3 contains the rules needed to create instances of the classes `eg:Location` and `eg:City`; and their following attributes: `eg:lat` and `eg:long` for the former; `eg:zipcode` for the latter. To correctly generate the URI for the instances of `eg:City`, a replace function inside a concatenate function is needed to (1) remove the blank spaces in the field "city" and (2) add the field to the base URI "http://ex.com/city/".

```
# Locations
:SubjectLoc a cm:ReferenceNodeMap ;
  cm:hasEvaluableExpression [
    cm:hasFunctionName cmf:concat;
    cm:hasInput ([cm:constantValue "http://ex.com/loc/"]    :lat [cm:constantValue "-" ]
        :long)].

:StatementLoc1 a cm:StatementMap ;
  cm:hasFrame :FrameLoc ;
  cm:subject :SubjectLoc ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue rdf:type ] ];
  cm:object [cm:hasEvaluableExpression [cm:constantValue eg:Location]].

:StatementLoc2 a cm:StatementMap ;
  cm:hasFrame :FrameLoc ;
  cm:subject :SubjectLoc ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue eg:lat]];
  cm:object [ a cm:Literal; cm:hasEvaluableExpression :lat];
  cm:hasDatatype [cm:hasEvaluableExpression xsd:decimal].

:StatementLoc3 a cm:StatementMap ;
  cm:hasFrame :FrameLoc ;
  cm:subject :SubjectLoc ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue eg:long]];
  cm:object [ a cm:Literal; cm:hasEvaluableExpression :long];
  cm:hasDatatype [ cm:hasEvaluableExpression xsd:decimal].

# Cities
```

```
:city_ns a cm:FunctionExpression ;
  cm:functionName cmf:replace ;
  cm:hasInput (c_city " " "").

:SubjectCities a cm:ReferenceNodeMap;
  cm:hasEvaluableExpression [
    cm:hasFunctionName cmf:concat;
    cm:hasInput ([cm:constantValue "http://ex.com/city/"] :city_ns)].

:StatementCit1 a cm:StatementMap ;
  cm:hasFrame :FrameCities ;
  cm:subject :SubjectCities ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue rdf:type]];
  cm:object [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression  [cm:constantValue eg:City]] .

:StatementCit2 a cm:StatementMap ;
  cm:hasFrame :FrameCities ;
  cm:subject :SubjectCities ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue rdfs:label]];
  cm:object [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression  [cm:constantValue :c_city]] .
  cm:hasLanguage [ cm:hasEvaluableExpression [ cm:constantValue "es" ] ].

:StatementCit3 a cm:StatementMap ;
  cm:hasFrame :FrameCities ;
  cm:subject :SubjectCities ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue eg:zipcode] ];
  cm:object [ a cm:Literal;
    cm:hasEvaluableExpression [cm:constantValue :zipcode] ];
  cm:hasDatatype [ cm:hasEvaluableExpression xsd:integer ].
```

Listing 4.3: Description with the Conceptual Mapping of the creation of regular statements from the data sources described in Listing 4.2.

Special statements. Listing 4.4 describes how a conditional statement and a linking rule are generated. This description is represented by means of functions. With the property `cm:hasBooleanCondition`, the conditional statement declares that the field `:year` has to be greater than 2020. The linking rule performs the link between the instances of `eg:City` and `eg:Location` with the predicate `eg:location`, using a distance metric (Levenshtein function) that has to be greater then a threshold of "0.75".

```
:StatementCit4 a cm:ConditionalStatementMap ;
  cm:hasFrame :FrameCities ;
  cm:subject :SubjectCities ;
  cm:predicate [ a cm:ReferenceNodeMap;
    cm:hasEvaluableExpression [cm:constantValue eg:population] ];
```

```
6     cm:object [ a cm:Literal;
7       cm:hasEvaluableExpression  [cm:constantValue :population] ];
8     cm:hasDatatype [ cm:hasEvaluableExpression xsd:integer];
9     cm:hasBooleanCondition [
10      cm:functionName cmf:greater_than ;
11      cm:hasInput ( :year 2020 ) ] .
12
13  :LinkExp1 a cm:LinkingExpression ;
14    cm:source :StatementCit1 ;
15    cm:target :StatementLoc1 ;
16    cm:property eg:location ;
17    cm:hasBooleanCondition [
18      cm:functionName cmf:greater_than ;
19      cm:hasInput ( :levfun 0.75 ) ] .
20
21  :levfun a cm:FunctionExpression ;
22    cm:functionName cmf:levenshtein_distance ;
23    cm:hasInput (:c_city :loc_city) .
```

Listing 4.4: Conditional and linking rules described with the Conceptual Mapping that complement the data source description and regular statements described in Listing 4.2 and Listing 4.3.

4.3 Featuring New Mapping Needs in RML: Support for RDF-star Construction

As the technologies revolving around knowledge graphs advance, so do the needs and requirements for their construction. Taking for instance the W3C Recommendation R2RML (Das et al., 2012), it was only focused on RDF construction from relational databases on its release in 2012. A growing number of extensions were published over the years to widen its possibilities. These proposals only extended its features to describe additional data sources (Michel, Djimenou, Faron-Zucker, & Montagnat, 2015; Van Assche et al., 2021), but also to increase its expressiveness regarding the inclusion of data transformation functions (Crotti Junior et al., 2016; De Meester et al., 2020; Debruyne & O'Sullivan, 2016; Kyzirakos et al., 2018), the generation of more RDF terms (e.g., collections and containers) (Debruyne et al., 2017; Michel, Djimenou, Faron-Zucker, & Montagnat, 2015) and RDF-star graphs (Delva, Arenas-Guerrero, et al., 2021; Sundqvist, 2022).

As a consequence of these growing needs, the W3C Knowledge Graph Construction Community Group was launched[5]. Since its inception, this community has gathered the new requirements for KG construction, published them as Mapping Challenges (see Section 4.2.2), and started implementing them in the widely adopted RML mapping language. The result of these efforts is materialized in a new revision of RML (Iglesias-Molina, Van Assche, et al., 2023b). The language is now published as a modular ontology, integrating: (i) extended possibilities in triple generation (RML-Core) (Van Assche, Iglesias-Molina, Dimou, et al., 2023); (ii) enriched description of data sources and output targets (RML-IO) (Van Assche, Iglesias-Molina, &

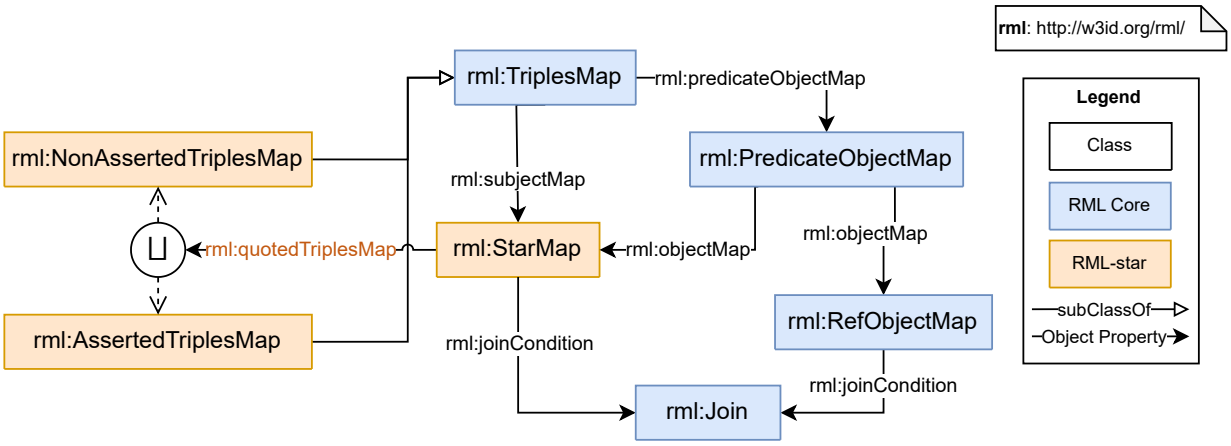


Figure 4.6: The RML-star module (represented using the Chowlk notation (Chávez-Feria et al., 2022)). The RML-star resources are highlighted in orange, while the rest of the represented ontology belongs to the RML-Core module.

Haesendonck, 2023); (iii) generation of RDF collections and containers (RML-CC) (Debruyne et al., 2023); (iv) accurate description of data transformation functions (RML-FNML) (De Meester et al., 2023); and (v) generation of RDF-star graphs (RML-star) (Iglesias-Molina, Van Assche, et al., 2023a).

Thus, this language is comprised of five different modules. For each module, there are more resources associated besides the ontology: its requirements, a specification with the module's details with examples and the restrictions written in SHACL shapes to validate mapping documents. All associated resources of this new version of RML are gathered in the ontology portal[21]. In addition, the semantics of the relationships between the old and this new specification are defined and available online[22] to facilitate backwards compatibility.

This section focuses on presenting one of the abovementioned modules, RML-star, that enables the generation of RDF-star graphs. The essential context about RDF reification is first introduced, followed by the motivation and description of the module and its validation. Finally, the adoption that this module had since its inception is shown, both the implementations and produced graphs.

4.3.1 Reification with RML-star

RML-star (Fig. 4.6) adds a new kind of term map, the `rml:StarMap`, that allows using triples maps to generate quoted triples. Following the RDF-star data model, a quoted triple may appear only in the subject or object of a triple. Thus, star maps can only be used in subject and object maps. Star maps use the property `rml:quotedTriplesMap` to refer to the triples map that generates the quoted triples. The referenced triples map can only

[21] https://w3id.org/rml/portal
[22] https://w3id.org/rml/portal/backwards-compatibility

be one of the following: (i) `rml:AssertedTriplesMap` to be asserted in the output graph, or (ii) `rml:NonAssertedTriplesMap` to not be asserted. A quoted triples map can specify `rml:class` in the subject map or have multiple predicate-object maps. Thus, several triples sharing the same subject map and logical source can be quoted by the same star map.

An example of an RML-star mapping rule is shown in Listing 4.6, which generates the RDF-star triples in Listing 4.7. The example uses the data in Listing 4.5, that contains CSV data related to pole vault: the vaulter (`PERSON`), the height of the jump (`MARK`) and its score (`SCORE`), the date when the jump was performed (`DATE`) and an identifier of the jump (`ID`).

```
1 ID ,  DATE       ,  MARK ,  PERSON   ,  SCORE
2 1  ,  2022-03-21 ,  4.80 ,  Angelica ,  1211
3 2  ,  2022-03-19 ,  4.85 ,  Katerina ,  1224
```

Listing 4.5: Contents of the logical source `:marks` in CSV format.

The mapping rules use an asserted triples map (`<#jumpTM>`) within the subject map of a triples map (`<#dateTM>`). The quoted triples map `<#jumpTM>` contains two predicate-object maps that produce triples annotated with `:date`. The first predicate-object map (*lines 6-9*) produces the triples for the height of the jump, the same as the examples presented previously for the other reification approaches. We extend the RML-star example with a second predicate-object map to also represent the score of the jump within the same quoted triples map (*lines 10-13*). To produce this triple in the other reification approaches an additional triples map for each case would be required.

```
1  <#jumpTM>
2    a rml:AssertedTriplesMap ;
3    rml:logicalSource :marks ;
4    rml:subjectMap [
5      rml:template ":{PERSON}" ] ;
6    rml:predicateObjectMap [
7      rml:predicate :jumps ;
8      rml:objectMap [
9        rml:reference "MARK" ] ] ;
10   rml:predicateObjectMap [
11     rml:predicate :score ;
12     rml:objectMap [
13       rml:reference "SCORE" ] ] .
14 <#dateTM>
15   a rml:TriplesMap ;
16   rml:logicalSource :marks ;
17   rml:subjectMap [
18     rml:quotedTriplesMap <#jumpTM> ] ;
19   rml:predicateObjectMap [
20     rml:predicate :date ;
21     rml:objectMap [
22       rml:reference "DATE" ] ] .
```

Listing 4.6: Example RML-star mapping that transforms data in Listing 4.5.

```
:Angelica :jumps "4.80" .
<< :Angelica :jumps "4.80" >> :date "2022-03-21" .
:Katerina :jumps "4.85" .
<< :Katerina :jumps "4.85" >> :date "2022-03-19" .
:Angelica :score "1211" .
<< :Angelica :score "1211" >> :date "2022-03-21" .
:Katerina :score "1224" .
<< :Katerina :score "1224" >> :date "2022-03-19" .
```

Listing 4.7: RDF-star triples generated by the mapping in Listing 4.6.

Currently, the RML-star specification (Iglesias-Molina, Arenas-Guerrero, et al., 2022) provides a complete description of the language, is published as a W3C Draft Community Group Report, and is maintained by the W3C Knowledge Graph Construction Community Group[5]. This extension belongs to the modules that comprise the new RML specification[23] that is currently under development by the aforementioned Community Group. Both the language and the specification are kept up-to-date reflecting the modifications in RDF-star. For instance, the latest RML-star releases update the term "embedded" to "quoted", according to the modifications in RDF-star. This update renamed the property `rml:embeddedTriplesMap` to `rml:quotedTriplesMap`. In addition, RML-star is implemented by Morph-KGC^{star} (Arenas-Guerrero, Iglesias-Molina, et al., 2024) and the SDM-RDFizer[24] (Iglesias et al., 2024) at the time of writing this document.

4.3.2 Validation

This section validates RML-star with (i) the development of the RML-star test-cases, that enables new implementations to check their conformance to RML-star; and (ii) the creation of RML-star mappings for two real-world use cases: software metadata extraction (Kelley & Garijo, 2021) (SoMEF) and biomedical research literature (Kilicoglu et al., 2012) (SemMedDB). In addition, RML-star is used for the evaluations performed in Chapter 6.

RML-star Test Cases

Test cases are commonly used to evaluate the conformance of an engine with respect to a language specification (e.g., RML test cases (Heyvaert et al., 2019)). A set of RDF-star test cases was proposed covering the syntax of various of its serializations[25]. We adapted these test cases to create the RML-star test cases and evaluate the conformance of Morph-KGC^{star} with respect to RML-star.

To create a representative set of test cases for RML-star, we selected the N-Triples-star syntax tests[26]. For each RDF-star test case, we created two associated RML-star test cases that generate the original RDF-star dataset: one test case with a single input data source (i.e., the

[23] http://w3id.org/rml/portal/
[24] https://github.com/SDM-TIB/SDM-RDFizer/
[25] https://w3c.github.io/rdf-star/tests/
[26] https://w3c.github.io/rdf-star/tests/nt/syntax

mapping does not include joins) and another with two input data sources (i.e., the mapping includes joins among triple maps). For each test case, we manually created the input source(s) in the CSV format and the corresponding RML-star mapping rules to generate the output RDF-star datasets. Following this approach, we obtained 16 RML-star test cases. The test cases are openly available[27], and can be reused by any engine to test its conformance with respect to RML-star.

Use Cases

We create RML-star mappings for in two real-world use cases. The first generates RDF-star graphs from scientific software documentation (Iglesias-Molina & Garijo, 2023), and the second annotates statements extracted from biomedical research publications. For both use cases, we use the tool Morph-KGC^{star}.

Scientific Software Metadata Extraction. Scientific software has become a crucial asset to deliver and reproduce the results described in research publications (Chue Hong et al., 2021). However, scientific software is often time consuming to understand and reuse due to incomplete and heterogeneous documentation, available only in a human-readable manner. The Software Metadata Extraction Framework (SoMEF) (Mao et al., 2019) proposes an approach to automatically extract relevant metadata (description, installation instructions, citation, etc.) from code repositories and their documentation. SoMEF includes different text extraction techniques (e.g., supervised classification, regular expressions, etc.) that yield results with different confidence values. For example, Listing 4.8 shows a JSON snippet with the description that SoMEF obtained from a software repository (Widoco) using the GitHub API. The confidence in this case is high as the extracted description was manually curated by the creators of the code repository. SoMEF extracts more than 30 different metadata fields about software, its source code, its released versions, and their corresponding authors. For transforming the output of SoMEF into RDF-star, we used a total of 35 triples maps to annotate software metadata fields and an additional triples map to annotate source code descriptions. All reified triples follow the same structure (Listings 4.8 & 4.9), i.e., the standard RDF triple contains the excerpt of the extracted feature, and it is annotated with the *technique* used and the *confidence* value. The complete mapping and all input examples and results are available online[28].

Capturing the technique used and the confidence obtained for each extracted metadata field is key for obtaining an accurate representation of the result. Hence, the RDF-star representation corresponding to the JSON in Listing 4.8 includes this information, as depicted in Listing 4.9.

[27] https://github.com/kg-construct/rml-star/tree/main/test-cases
[28] https://github.com/oeg-upm/rdf-star-generation

```
"codeRepository": "https://github.com/oeg-upm/Chowlk",
"description": [
 {
  "confidence": 1.0,
  "excerpt": "Tool to transform an ontology diagram into OWL code.",
  "technique": "GitHub API"
 }
]
```

Listing 4.8: JSON snippet showing the description metadata field extracted by SoMEF on a code repository using the GitHub API as extraction technique.

```
<oeg-upm/Chowlk> :description "Tool to transform an ontology diagram into OWL code
    ." .
<< <oeg-upm/Chowlk> :description "Tool to transform an ontology diagram into OWL
    code." >>
     :technique "GitHub API" .
<< <oeg-upm/Chowlk> :description "Tool to transform an ontology diagram into OWL
    code." >>
     :confidence "1.0"^^xsd:float .
```

Listing 4.9: RDF-star triples snippet showing the results generated for the description field in Listing 4.8. Each asserted triple is annotated with its corresponding confidence and technique.

Biomedical Research Literature. The Semantic MEDLINE Database (SemMedDB) (Kilicoglu et al., 2012) is a repository with biomedical entities and relationships (subject-predicate-object) extracted from biomedical texts, mainly titles and abstracts from PubMed citations. This dataset is available as a relational database and CSV files[29]. It is licensed under the UMLS - Metathesaurus License Agreement[30], which does not allow its distribution, but it may be accessed with an account with the UMLS license.[31] We downloaded the MySQL files for (1) *predication* predictions (PREDICATION and PREDICATION_AUX tables), containing more than 117 million annotations; and (2) *entity* predictions (ENTITY table), which include more than 410 million annotations. Listings 4.10, 4.11 and 4.12 illustrate the columns used from the tables with synthetic data. For *predications*, only data for *subjects* is shown; the missing columns regarding *objects* follow the same structure as *subjects*. *Subjects* and *objects* (from *predications*), and *entities* are assigned a *semantic type* with a confidence score. These *semantic types* categorize the extracted concept in the biomedical domain[32]. In addition, the extraction of *subjects* and *objects* is assigned a timestamp on when it took place. Thus, the

[29] https://lhncbc.nlm.nih.gov/ii/tools/SemRep_SemMedDB_SKR/SemMedDB_down\protect\discretionary{\char\hyphenchar\font}{}{}load.html

[30] https://www.nlm.nih.gov/research/umls/knowledge_sources/metathesaurus/release\protect\discretionary{\char\hyphenchar\font}{}{}/license_agreement.html

[31] An account with the UMLS license can be requested at https://www.nlm.nih.gov/databases/umls.html.

[32] https://www.nlm.nih.gov/research/umls/new_users/online_learning/SEM_003.html

score and timestamp represent metadata about other statements, which are fit to represent using reification. We created an RML-star mapping with 5 triples maps quoting triples: 3 of them are used to annotate the assignation of *semantic types* to *entities*, *subjects*, and *objects* with confidence scores; the remaining 2 provide the timestamps for the extraction of *subjects* and *objects*.

```
1 ENTITY_ID , SEMTYPE , SCORE
2 12345     , orga    , 790
```

Listing 4.10: ENTITY table snippet.

```
1 PREDICATION_ID , SUBJ_SEMTYPE , SUBJ_NAME
2 13579          , Semtype      , SubjName
```

Listing 4.11: PREDICATION table snippet.

```
1 PREDICATION_AUX_ID , PREDICATION_ID , SUBJ_SCORE , TIMESTAMP
2 67890              , 13579          , 800        , 1651740766
```

Listing 4.12: PREDICATION_AUX table snippet.

```
1 << ex:12345 sem:semanticType "orga" >> sem:score 790 .
2 << ex:13579 sem:subject ex:SubjName >> sem:timestamp "1651740766" .
3 << ex:SubjName sem:semanticType "Semtype" >> sem:score 800 .
```

Listing 4.13: RDF-star triples generated from data in Listings 4.10, 4.11 and 4.12.

4.4 Summary

This chapter addresses the fist objective of this thesis *O1: To analyze the needs for declarative knowledge graph construction from heterogeneous data sources, in order to facilitate the support of advances in existing mapping languages to address the most relevant ones.* To reach this objective, we first performed an in-depth analysis of the features of the current mapping languages: from the description of input data sources, to the diverse features for triple generation. This comprises the first contribution of *O1, C1: Design of a comparison framework analysing the state of the art mapping languages.* This analysis and the challenges proposed by the Knowledge Graph Construction W3C Community Group are gathered as requirements for a mapping language to be able to address the complexity of use cases for KG construction.

Then, we presented the Conceptual Mapping, an ontology that, based on the abovementioned requirements, gathers the expressiveness of current mapping languages. This ontology was developed following the LOT methodology, it is evaluated accordingly and made available under a W3ID. This comprises the second contribution ob objective O1, *C2: Identify, define and implement the requirements for knowledge graph construction from heterogeneous data sources.*

Finally, we show that mapping languages evolve as KG technologies advance, in this case with the adoption of RDF-star. To that purpose, we extend one of the most used mapping languages, RML, to create RDF-star graphs, RML-star. Accordingly, we add an extension to the Conceptual Mapping to include this new feature. This comprises the third contribution

of objective O1, *C3: Development of new features for the RML mapping language.*

Our ambition with these contributions is to enhance the understanding of the capabilities of mapping languages and what they need to implement in order to cover the necessities for constructing knowledge graphs, and at the same time be able to evolve as the technologies progress.

Chapter 5

Enhancing the Creation of Mapping Rules

The mappings used for constructing knowledge graphs can be created in different ways. As they are formatted as text documents, they can be directly written in any text editor. However, the verbosity of some languages, the increased cognitive load required (Crotti Junior et al., 2018) and the need of following a specific syntax has fostered the development of user-friendly tools and serializations to ease the mapping writing process. This chapter introduces two alternative solutions for easing the construction of mappings for different potential users, (i) a spreadsheet-based approach to write the mapping rules (Section 5.1), and (ii) a user-friendly mapping syntax updated with the latest needs in knowledge graph construction (Section 5.2).

5.1 Easing Mapping Rule Creation with Spreadsheets

Ever since the adoption of mapping languages increased, multiple editors and user-friendly serializations were developed to ease the mapping writing process and assist the needs of mapping developers. Some of the developed approaches enable editing through graphical visualization (Heyvaert et al., 2016; Sicilia et al., 2017), while others provide a writing environment (e.g., Ontop for Protégé[1]). These proposals present different limitations. Visual approaches do not usually scale well for complex use cases with a high amount of mapping rules. The serializations, despite being easier than the original languages, still require users to learn and follow a syntax. In addition, both alternatives are generally oriented to a single language. This situation, taking also into account the general lack of interoperability (described in Section 2.2.5), forces users to learn more than one language to write the same conceptual mapping rules when one approach does not fully cover their needs.

To overcome this limitations, we devise a spreadsheet-based approach aiming to be (i) syntax independent, (ii) able to generate rules in more than one mapping language and (iii) that leverages a familiar environment for writing mapping rules. These spreadsheets contain the essential elements of the mapping rules without any additional syntax, and are later

[1] https://ontop-vkg.org/tutorial/basic/setup.html#ontop-protege-setup

translated into a mapping document. We choose spreadsheets as they are widely used for data management (Birch et al., 2018; Bradbard et al., 2014; Pemberton & Robson, 2000). The purpose of this approach is to ease the mapping writing process for users with heterogeneous backgrounds, as well as to increase the interoperability among mapping languages (Corcho et al., 2020; Iglesias-Molina, Cimmino, & Corcho, 2022). This proposal is implemented in Mapeathor (Iglesias-Molina, Ruckhaus, et al., 2023), a tool able to parse the spreadsheets and generate the corresponding mappings in R2RML, RML (both the release on 2014 by Dimou et al., 2014 and the release on 2023 by Iglesias-Molina, Van Assche, et al., 2023b) and YARRRML. We evaluate the proposal and how it can improve the mapping writing process for users with heterogeneous backgrounds and expertise performing a user study.

5.1.1 Spreadsheet-based Mapping Rules

This section presents the design of the spreadsheet template to write mapping rules. This template contains only the essential information that needs to be used in order to generate the mapping correctly formatted in a language's syntax. This design is devised to represent the rules in a compact and understandable manner, using a format widely used by the scientific community (i.e., Google Spreadsheets, MS Excel). Thus, this approach prevents users from learning the particular syntax of each language (e.g., keywords, semicolons, brackets, etc.) while using a familiar environment for writing in the spreadsheets.

The spreadsheet template consists of five different sheets, which are described below: *Prefix sheet*, *Source sheet*, *Subject sheet*, *Predicate_Object sheet* and *Function sheet*. We illustrate this section with a running example to describe in detail the expressiveness capabilities of each spreadsheet. This example uses the data represented in the CSV file `people.csv` (Listing 5.1) and the JSON file `sport.json` (Listing 5.2).

```
1 id , name  , birthdate , sport_id
2 1  , Emily , 08/02/90  , 2
3 2  , Jonah , 18/11/75  , 2
```

Listing 5.1: Contents of `people.csv`.

```
1 [ {
2    "id": 1,
3    "sport": "Ice Skating"
4  },{
5    "id": 2,
6    "sport": "Rugby"
7  } ]
```

Listing 5.2: Contents of `sport.json`.

Prefix Sheet

This sheet contains the namespaces and corresponding prefixes used in the creation of the transformation rules. It is composed of two columns: `Prefix` for the prefix and `URI` for the corresponding namespace. The base namespace can be specified writing "@base" in the `Prefix` column. The example shown in Table 5.1 presents how three namespaces and the base namespace are written in this template.

Table 5.1: Prefix sheet.

Prefix	URI
@base	http://example.com/
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
ex	http://ex.com/
grel	http://semweb.datasciencelab.be/ns/grel#

Table 5.2: Subject sheet.

ID	URI	Class	Graph
PERSON	http://ex.com/Person/{name}	ex:Person	
PERSON	http://ex.com/Person/{name}	ex:Athlete	
SPORT	http://ex.com/Sport/{sport}	ex:Sport	ex:SportsGraph

Subject Sheet

This sheet defines how to generate the subjects and a corresponding identifier (`ID`) that groups the mapping rules per subject. It is organized in four columns: `ID`, `URI`, `Class` and `Graph`. `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. `URI` defines the subject URI of the resources that are to be generated by the mapping. `Class` allows assigning the subject to a class with `rdf:type`. A subject may be an instance (`rdf:type`) of one class, more than one or none at all. Finally, `Graph` is an optional column that enables the assignation of a named graph to the triples generated for a subject, that is, generating n-quads.

The example shown in Table 5.2 presents how to write two subjects, each with a different identifier and URI for the instances. Within the `URI` field, what is written between "{" and "}" references a field in the source data. The instances of the subject with the `PERSON` ID are type of two classes, (`ex:Person` and `ex:Athlete`), while all the triples of the instances of the subject identified with the `SPORT` ID are assigned to a named graph (`ex:SportsGraph`).

Source Sheet

This sheet describes the source input data for each set of rules, identified with the identifier previously created in the *Subject sheet* (`ID`). The information is organized in three columns: `ID`, `Feature` and `Value`. `ID` makes reference to the identifier that gathers the mapping rules in the sheets (introduced in the *Subject sheet*). `Feature` declares the type of information provided in `Value`. The allowed keywords in this column are: `source` for the path and name of the file, `format` for the data source format, `iterator` for hierarchical data (e.g., JSON, XML), `table` for name of tables from Relational Data Bases (RDBs), `query` for SQL queries and `SQLVersion` for the version of SQL used. Each ID must have at least the `source` and `format` features specified. The rest of the features are optional. Then, in the `Value` column,

Table 5.3: Source sheet.

ID	Feature	Value
PERSON	source	/home/user/data/people.csv
PERSON	format	CSV
SPORT	source	/home/user/data/sports.json
SPORT	format	JSON
SPORT	iterator	$.*

the corresponding values of each specified `Feature` are written.

Table 5.3 shows the data source description for the `PERSON` and `SPORT` IDs. The former describes as input data the CSV file shown in Listing 5.1, while the latter describes the JSON file in Listing 5.2, for which the iterator is also written (`$.*`).

Predicate_Object Sheet

Table 5.4: Predicate_Object sheet.

ID	Predicate	Object	DataType	Language	ReferenceID	InnerRef	OuterRef
PERSON	ex:name	{name}	string	en			
PERSON	ex:birthdate	<Fun-date>					
PERSON	ex:plays				SPORT	sport_id	id
SPORT	ex:name	{sport}	string	en			
SPORT	ex:code	{id}	integer				

This sheet allows specifying how to generate the triples, that is, predicate-object pairs for the subjects defined in the *Subject sheet*. This sheet contains up to eight columns: `ID`, `Predicate`, `Object`, `DataType`, `Language`, `ReferenceID`, `InnerRef` and `OuterRef`.

The column `ID` indicates the set of rules which the triples belong to, that has been previously defined in the *Subject* and *Source sheets*. The columns `Predicate` and `Object` specify the predicate and object of the triple respectively. The XSD datatype of `Object` is defined in `DataType`, and the language tag in `Language`. Both these columns are optional.

When the object refers to a subject defined in another mapping rule, the rule is written as follows. There are three columns that allow the specification of the linking condition between the object of the current triple and the referenced subject. They specify which is the ID of the referred subject (`ReferenceID`), and the "join" fields in the source data: (i) `InnerRef` for the field of the object of the current triple, and (ii) `OuterRef` for the field of the referred subject. These three fields can be blank when a regular triple is produced.

Table 5.4 shows five predicate-object pairs created for both sets of rules (`PERSON` and `SPORT`). Three of these rules produce literals with datatypes, and two specify a language tag as well. The rule set identified as `PERSON` generates a link to the subject of the `SPORT` rule set using

the predicate `ex:name`, and joining by equal values of the field `sport_id` from `people.csv` and the field `id` from `sport.json`. This rule set also calls a function for generating the object for the predicate `ex:birthdate`, which is identified by being enclosed by "<" and ">".

Function Sheet

Table 5.5: Function sheet.

FunctionID	Feature	Value
<Fun-date>	executes	grel:toDate
<Fun-date>	returns	grel:dateOut
<Fun-date>	grel:valueParam1	{birthdate}
<Fun-date>	grel:valueParam2	dd/MM/yyyy
<Fun-date>	grel:valueParam3	yyyy-MM-dd

Some languages are able to process data transformation functions, which can be detailed in this sheet. Mapeathor implements the lastest release of the RML-FNML specification, that describes how to incorporate functions in RML mappings[2]. 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`. The features admitted in the `Feature` column are `executes` for the name of the function, and `returns` for the expected output type of the function. In addition, this column admits any number of function parameter names. Then, the value of the name of the function, return type and values of the parameters are written correspondingly in the `Value` column.

The example shown in Table 5.5 uses the function `grel:toDate`, to change the formatting of a date. It returns a date as output (`grel:dateOut`) and takes three parameters: the data field to be converted (`grel:valueParam1`), the input data format (`grel:valueParam2`) and the output data format (`grel:valueParam3`).

5.1.2 Implementation: Mapeathor

Mapeathor (Iglesias-Molina, Ruckhaus, et al., 2023) is a a tool that generates mappings from the spreadsheets described in Section 5.1.1. It is able to process MS Excel and Google Spreadsheets, generating human-readable mappings in either R2RML, RML (both the release on 2014 by Dimou et al., 2014 and the release on 2023 by Iglesias-Molina, Van Assche, et al., 2023b) or YARRRML. This tool is being used for building knowledge graphs in soil ecological research (Le Guillarme & Thuiller, 2023), medical evidences (A. Á. Pérez et al., 2022) and scientific literature (Badenes-Olmedo & Corcho, 2023), among others.

Algorithm 1 presents the procedure implemented by Mapeathor to translate an input spreadsheet into a formatted mapping document. First, the rules written in the spreadsheet template

[2] https://w3id.org/rml/fnml/spec

Algorithm 1: Mapeathor translation algorithm

Result: Mapping document in [R2]RML/YARRRML
language ⟵ *output_language*;
spreadsheet ⟵ *input_spreadsheet*;
output_m ⟵ ∅;
rules ⟵ *extract_rules*(*spreadsheet*);
output_m.add(*translate_prefixes*(*rules*, *language*));
for *ruleset* ∈ *rules* **do**
    rules.add(*enrich_ruleset*(*ruleset*));
    output_m.add(*translate_subject*, *language*);
    output_m.add(*translate_source*, *language*);
    output_m.add(*translate_predicate_object*, *language*);

    if *language* == *RML* **then**
        output_m.add(*translate_functions*);
    end
end
return *write*(*output_m*);

are extracted and stored in a JSON file. This step also validates that the spreadsheet follows the template. The prefixes are then added to the output mapping document, along with other prefixes that are used natively in the target language (e.g., `rr`[3]). Then, the rules are translated by grouping them into rulesets with the same `ID` expressed in the spreadsheet (see Section 5.1.1). The extracted rules are enriched with implicit information needed to write the mapping (e.g., to look for `rr:TermTypes` and IRIs). Next, the subject, source and predicate-object pairs in each ruleset are translated. In the case of RML, the system also looks for referencing functions to translate them as well. Finally, the translated rules are written into a final file, which is returned as output.

The mapping documents are generated in a human-readable manner, considering previous experiences (Chaves-Fraga et al., 2020, 2022; Corcho et al., 2021). This helps knowledge engineers in complex data integration contexts to understand more easily whether the mapping document represents the desirable rules originally written and, hence, whether the constructed knowledge graph will be correct or not. For [R2]RML, the output mapping follows a Turtle-based syntax, using predicate object lists within blank node properties[4], as recommended by the [R2]RML specifications. We also ensure the same mapping rules' order as they are written in the spreadsheet. Appendix B shows the output mappings produced by Mapeathor after processing the rules written in the spreadsheet shown in Tables 5.1, 5.2, 5.3, 5.4 and 5.5, also available online[5].

The source code of Mapeathor is openly available under and Apache 2.0 license[6]. It can be

[3] http://www.w3.org/ns/r2rml#
[4] https://www.w3.org/TR/turtle/#unlabeled-bnodes
[5] https://docs.google.com/spreadsheets/d/1TS1r-uR0PJUkUXD-HpMgU2RJfDyRB0FZiGkzd4r3AAI
[6] https://github.com/oeg-upm/mapeathor

run as a CLI using the PyPI package[7] or as an online service[8], where a visual interface is provided. With each release, a new version in PyPI is created and a dedicated DOI archived in Zenodo (Iglesias-Molina, Ruckhaus, et al., 2023).

5.1.3 Evaluation

We evaluate the presented approach by performing a user study to test if the spreadsheet-based approach can improve the mapping writing process for users of different expertise.

Methodology

The evaluation performed consists on a user study that aims at testing whether our approach improves the mapping writing process w.r.t. writing directly the mapping in a target language (i.e., RML) and w.r.t. another mapping editor with a visual interface (i.e., RMLEditor). This section describes the methodology followed.

Procedure The user study took place in May 2023. The spreadsheet-based approach proposed in this work was compared with the RML mapping language (Dimou et al., 2014), and with a visual editor, RMLEditor (Heyvaert et al., 2016). There were 30 participants in total, divided into three groups of 10. Each group had to carry out the same task, to create a mapping from a given dataset and ontology with the assigned tool. All groups were introduced to the needed concepts about mapping languages and specific guidelines with examples of the assigned tool. The groups performed the given task in separated sessions, i.e., one session per tool was carried out, to ensure that all participants were given the same level of assistance independently of the tool. Participants were asked to carry out the proposed task in 30 minutes. Having reached the time limit, they were asked to submit the mapping written so far. In addition, participants were asked to rate in a 5-point Likert scale their expertise in (i) linked data and (ii) mapping languages and tools; and to write their personal opinion on the assigned tool.

Resources We use a subset of the *Vocabulary for data representation of the local business census and activities licenses*[9] that represents local businesses, their postal address and geolocalization (Fig. 5.1). We chose this dataset, that belongs to the common-knowledge domain, to avoid heterogeneous prior domain knowledge influence on the results of the study.

We provided participants with the subset ontology description and associated data, that is comprised of three CSV files: `bar.csv`, `restaurant.csv` and `address.csv`. Information about bars, restaurants and their geolocalization is represented in the files `bar.csv` and `restaurant.csv` respectively. These files contain the same columns, but different data. The file `address.csv` represent the postal address of the businesses present in the other two files. The original data is available in GitHub[10]. To facilitate the exercise for participants, some data cleaning was performed over the original files, as well as translation to all columns from spanish to english. The used resources are published in Zenodo (Iglesias-Molina, 2022).

[7] https://pypi.org/project/mapeathor/
[8] https://morph.oeg.fi.upm.es/demo/mapeathor
[9] http://vocab.ciudadesabiertas.es/def/comercio/tejido-comercial/index-en.html
[10] https://github.com/CiudadesAbiertas/vocab-comercio-censo-locales/

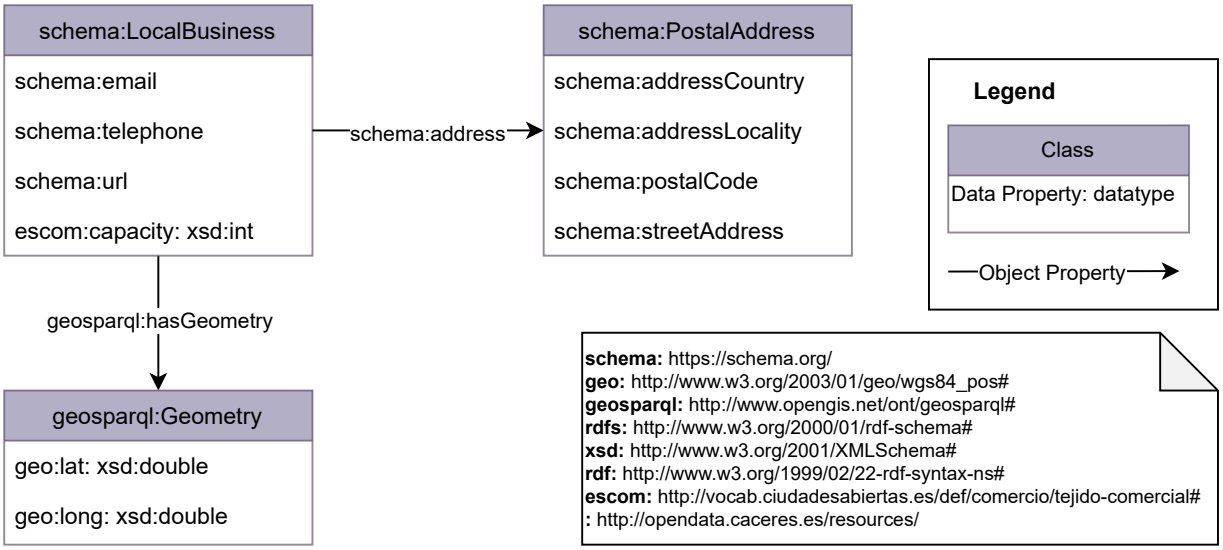
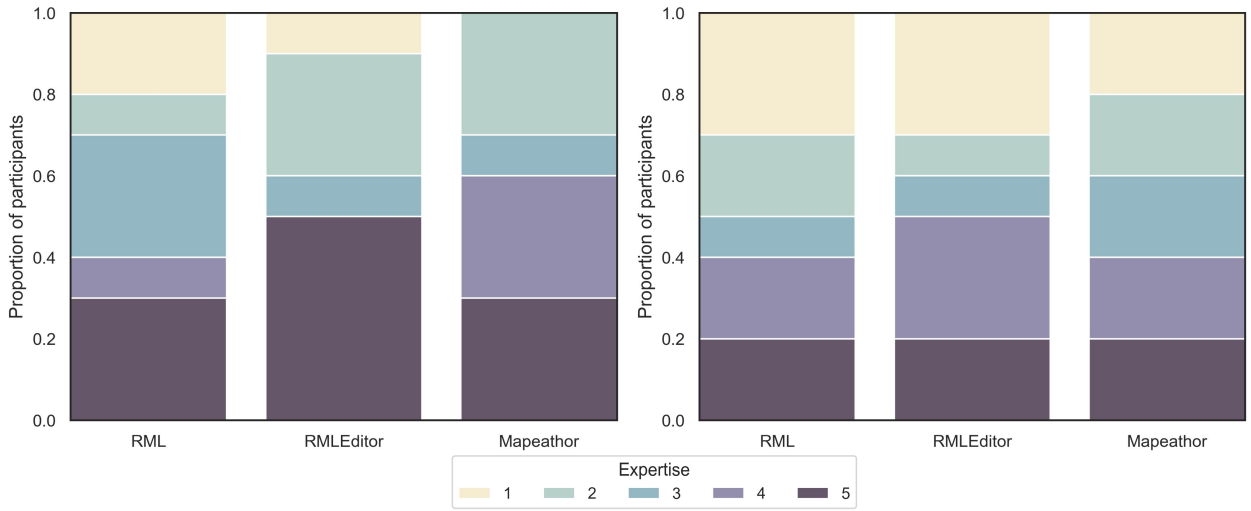


Figure 5.1: Ontology diagram representing a subset of the Vocabulary for data representation of the local business census and activities licenses.

Participants There were in total 30 participants, all sampled from the Ontology Engineering Group, ranging from MSc students to full professors. Their background ranged from having basic knowledge about linked data and mappings, to being knowledgeable practitioners, including participants that were experts in linked data but had no knowledge about mappings. Prior to the assignment of participants to each group, participants were asked to provide information about whether they had prior knowledge of any of the tools, so that no participant would use a tool which they were already familiar with. Taking into account this restriction, they were randomly divided into three groups, one per tool (Fig. 5.2). We run a Chi-Squared Test for homogeneity, obtaining no significant results (p-value > 0.05), thus we ensure that each group was balanced w.r.t. the heterogeneity in background.

Metrics We calculate two kinds of measurements for accuracy: (i) *Total Accuracy* (TA) for the number of correct rules written w.r.t. the reference correct mapping, and (ii) *Local Accuracy* (LA) for the number of correct rules w.r.t. all the written rules. We calculate both accuracies because of the fixed time for completing the exercise. TA can be interpreted as a measure of completeness, while LA focuses on assessing the correctness of the written rules independently on the total completeness. We divide the mapping in four components: *Subject*, *Source*, *Predicate-Object (PO)* and *Join*, and calculate the accuracy for each component and the totality of the mapping. Then a T-Test is performed to look for significant differences of accuracy among the results obtained for the different tools. This test is used under the assumptions of (i) normality, (ii) sameness of variance, (iii) data independence, and (iv) variable continuity.

Threats to validity (Creswell & Creswell, 2017) We identify the following internal and external threats to the validity of our experiment.



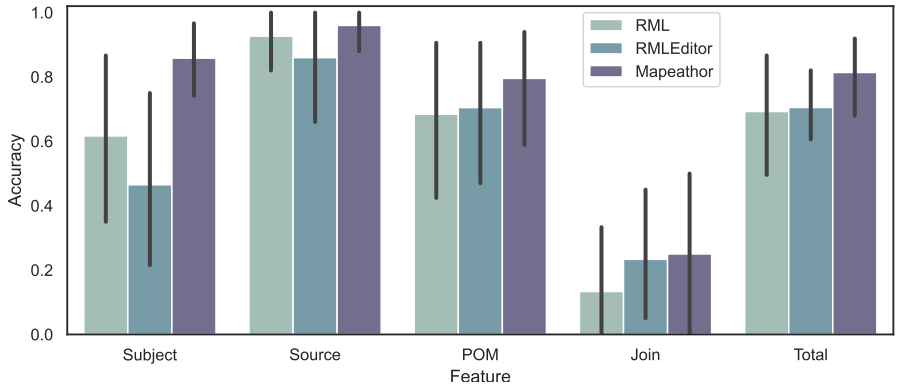
(a) Expertise in Linked Data (b) Expertise in mappings

Figure 5.2: Distribution of expertise of participants according to the 5-point Likert scale in (a) Linked Data and (b) mappings.

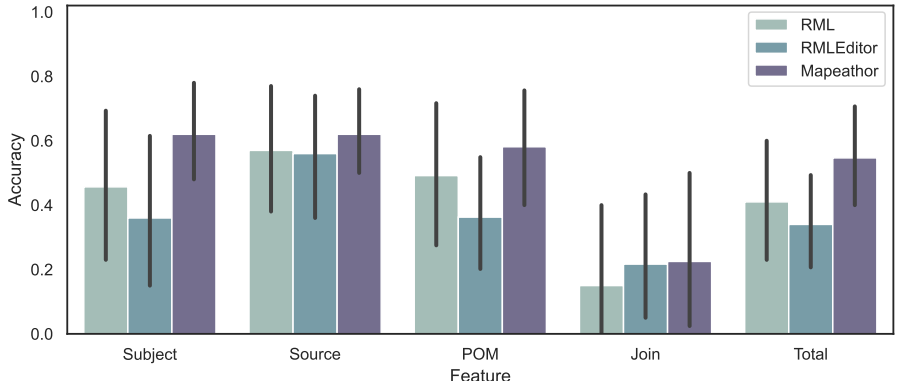
Internal validity threats concern the experimental setup or experience of participants which threaten the ability to draw correct conclusions about the population in the experiment. We identify three internal threats: *tool familiarity bias*, *selection bias* and *instruction bias*.

- **Tool familiarity bias.** Practitioners tend to be more expert in some specific technologies and languages that they use more frequently. Thus, in the sample it is possible to have participants that have used one of the tested approaches, which in turn can influence the results. For creating the groups, participants' experience was inquired to avoid assigning them a tool that they had previously used. There was no participant that had prior knowledge of all tested approaches.
- **Selection bias.** The sample includes participants with different range of skills and previous knowledge about linked data and mappings. We mitigate this threat by defining groups that were homogeneous w.r.t. range of expertise in mappings and linked data, considering also the mitigation measures for the *tool familiarity* threat. That is to say, all groups included from beginner users to knowledgeable practitioners.
- **Instruction bias.** One of the evaluated approaches is designed by the conductors of the user study. This situation could incur in bias on information delivery. To mitigate this threat, we provided participants with instruction guidelines equally detailed for each tool. Three sessions were conducted subsequently to provide the same explanations and attention to each tool. During each sessions, all questions were answered with equal level of detail and guidance for all tools and participants.

External validity threats occur when wrong inferences from sample data are made beyond the studied sample or experimental setup. We identify two external threats: *background of participants sample* and *environment*.



(a) Local accuracy.



(b) Total accuracy.

Figure 5.3: Local accuracy (a) and Total accuracy (b) of the mappings written by participants of the user study.

- **Background of participants sample.** This threat concerns the generalization to individuals outside the study criteria. The sample of participants include a wide range in the variety of backgrounds, from not very familiar with linked data and mappings, to expert practitioners. We deliberately choose to have this variety for each group so that the results can be generalized to a broader sample, while mitigating the risks of internal threats described above that this choice poses.
- **Environment.** This threat concerns the generalization to individuals outside the experiment's setting. Participants were free to use a computer, browser and tools of their choice. Thus, they could perform the tasks required for the study in a well-known environment. No specific experimental setup prevents generalizations to individuals outside our study.

Table 5.6: Results of the user study, showing the mean of the local accuracy (LA), total accuracy (TA) and p-value of the submitted responses per mapping feature and tool.

	Subject		**Source**		**PO**		**Join**		**Total**	
	LA	TA	LA	TA	LA	TA	LA	TA	LA	TA
RML	0.617	0.457	0.927	0.570	0.684	0.458	0.133	0.150	0.693	0.410
RMLEditor	0.465	0.360	0.860	0.560	0.705	0.363	0.233	0.217	0.705	0.340
Mapeathor	**0.858**	**0.620**	**0.960**	**0.620**	**0.795**	**0.581**	**0.250**	**0.225**	**0.831**	**0.547**
p-value	0.090	0.295	0.596	0.890	0.777	0.337	0.746	0.881	0.476	0.264

Results

Fig. 5.3 shows the results obtained from the user study. In general, it can be observed that Mapeathor obtains the highest accuracy rate in all cases. Surprisingly, RMLEditor struggles with the base approach, RML, obtaining in general slightly worse results. Between these two approaches, participants using RML were able to complete a higher number of mapping rules, while for RMLEditor the written rules were more accurate.

Looking at the details of each of the mapping parts, in general the writing of *Sources*, *Subjects* and *Predicate-Object* pairs (*PO*) is mostly successful. It is especially remarkable for *Sources*, that achieves a local accuracy near to 1. However, the results reported for its total accuracy are not that high, which can be due to the time limit and the increasing complexity of the exercise. The accuracy of the creation of *Subjects* with the RMLEditor is rather low w.r.t. the other approaches. The most common mistake done by participants in this case was writing wrongly the class to which the subject is a type of, indicating the path of the source file instead of the class IRI. The mapping element where all approaches fail to assist users is unmistakably the *Join*. Most participants did not even try to write them, and those who did, mostly failed. The most plausible reason to explain this behaviour is the lack of understanding on how and when to use a *Join* in a mapping. RML presents the worst results in this case, as the specification of this element in this language is the one that requires most mapping constructs.

However, none of the differences in the results for any type of mapping rule is significant (p-value > 0.05, see Table 5.6), but we can observe some tendencies. The results for which the p-value is closer to being significant is for the total accuracy of the complete mapping, and for the subjects (specially its local accuracy). In these cases, it can be stated with some confidence that the spreadsheet-based approach proposed in this contribution can help the writing process, improving the baseline and other graphical approaches.

5.1.4 Discussion

In general, the results obtained in the user study show that there is still plenty of room for improvement for facilitating the adoption and use of mappings by a broader community of users. In each group, there were participants able to completely finish the exercise on time. In addition, plenty other participants claimed (in the feedback space left in the questionnaire)

that, without the time limit, it would have been possible for them to complete the mappings. However, the reported local accuracy indicates that, even complete, the mapping would not have been entirely valid. This success rate holds true both for new users as well as for people with expertise on other mapping technologies.

The results obtained for the RMLEditor are specially notable. Overall, this approach does not improve the results of RML. This seems to be a product of the technological support rather than the visual notation. Multiple participants using the RMLEditor stress the lack of certain functionalities, such as the impossibility of copying and pasting sections of the created mapping, and the difficulty of finding created mapping constructs in the working space. The contrary was reported for RML, where the writing got easier for participants once they understood the parts of the mapping the had to copy and modify to produce correct rules. Regarding Mapeathor, multiple participants highlighted that using spreadsheets was convenient for them to carry out the exercise. The most mentioned flaw, however, was the need to separate the rules in different sheets. Especially for the creation and tracking of the rule set's `ID`, participants missed an automated way of filling this ID with the ones created in other sheets. As the process is currently manual, it can induce to errors.

Hence, we can observe that developing tools that aid the mapping writing is generally desired, and can improve the process by speeding up the writing and ensuring that the rules written produce correct mapping documents. Additionally, the higher accuracy reported by the participants using Mapeathor suggests that the spreadsheet-based approach is useful for this aim. Yet, there is still room for improvement regarding (i) the cohesion of the sheets and the shared `IDs` across them, (ii) the potential for hosting more expressiveness from the ever-evolving mapping languages (Iglesias-Molina, Van Assche, et al., 2023b) (see Section 4.3 and Section 5.2) and (iii) the possibility of leveraging more spreadsheet editors features to improve the user experience. Moreover, there are some aspects involving the cognitive effort of the different approaches that has not been analyzed in the evaluation, but are worth looking into in future work.

5.2 User-friendly Creation of Mapping Rules for RDF-star with YARRRML

Along with mapping editors described before, human-friendly serializations emerged to ease the definition of mapping rules. Focusing on R2RML and RML, we highlight the following serializations that have been more widely adopted recently. YARRRML (Heyvaert et al., 2018) leverages YAML to offer a user-friendly representation to define mapping rules, while ShExML (García-González et al., 2020) extends the syntax of the ShEx constraint language (Prud'hommeaux et al., 2014). XRM (Zazuco, 2022) provides an abstract syntax that simulates programming languages and SMS2 (Stardog, 2021), proposed by Stardog[11], is loosely based on the SPARQL query language and extends the features of R2RML to create virtual KGs. Each serialization is accompanied by a system that translates their rules into mapping languages, such as RML or R2RML (henceforth abbreviated as [R2]RML).

[11] https://www.stardog.com/

This section describes YARRRML-star, the extension of YARRRML to support RML-star (Delva, Arenas-Guerrero, et al., 2021) to construct RDF-star graphs. We also improve the serialization to adhere with the latest RML updates (e.g., datatypes, joins, etc.), and develop a translator system –Yatter– that implements the new features, and validate our proposal with test cases and compared it to other user-friendly serializations.

5.2.1 User-friendly Syntax for Mapping Rules

We extend the YARRRML serialization to support the RDF-star construction, two updates (dynamic datatypes and language tags), and a shortcut for join conditions. These are recent features in RML that so far were not considered in YARRRML. To illustrate the extensions, we use a CSV file as input (Listing 5.3) with information about pole vaulters: an identifier, name, language of the country, height of the jump, date when the jump takes place, and type of date format specified.

```
1 ID , PERSON          , COUNTRY , MARK , DATE                    , DATE-TYPE
2 1  , Lisa Ryzih      , de      , 4.40 , 2022-03-21              , date
3 2  , Xu Huiqin       , zh      , 4.55 , 2022-03-19T17:23:37     , dateTime
```

Listing 5.3: Contents of the `jump-source` logical source.

YARRRML-star

RDF-star introduces the notion of RDF-star triples, i.e., triples that are subjects or objects of another triple. These triples are enclosed using "«" and "»", and can be (1) quoted, if they only appear in a graph embedded by another triple (List 5.4 lines 2,4); or (2) asserted, if the quoted triple is also generated outside the triple where it is quoted (lines 1-4). We extend YARRRML to specify how we can construct RDF-star graphs, aligned with the RML-star specification (Iglesias-Molina, Arenas-Guerrero, et al., 2022).

```
1 :1 :jumps 4.40 .
2 << :1 :jumps 4.80 >> :date "2022-03-21" .
3 :2 :jumps 4.55 .
4 << :2 :jumps 4.85 >> :date "2022-03-19T17:23:37" .
```

Listing 5.4: RDF-star triples.

RDF-star triples can be created in YARRRML-star (Listing 5.5) by referencing existing Triples Maps with the tags (1) `quoted` for quoted asserted triples (line 10) and (2) `quotedNonAsserted` for quoted non-asserted triples. The triple that the rule set `jumpTM` creates is used as subject in the rule set `dateTM`, creating RDF-star triples (Listing 5.4). The equivalent mapping rules in RML are shown in Listing 5.6.

```
mappings:
 jumpTM:
  sources: jump-source
  subjects: :$(ID)
  predicateobjects:
   - [:jumps, $(MARK)]
 dateTM:
  sources: jump-source
  subjects:
   quoted: jumpTM
  predicateobjects:
   - [:date, $(DATE)]
```

Listing 5.5: YARRRML-star mapping rules.

```
<#jumpTM>
  a rml:AssertedTriplesMap ;
  rml:logicalSource :jump-source;
  rml:subjectMap [
    rr:template ":{ID}" ] ;
  rr:predicateObjectMap [
    rr:predicate :jumps ;
    rml:objectMap [
      rml:reference "MARK" ] ] .
<#dateTM>
  a rr:TriplesMap ;
  rml:logicalSource :jump-source ;
  rml:subjectMap [
    rml:quotedTriplesMap <#jumpTM> ];
  rr:predicateObjectMap [
    rr:predicate :date ;
    rml:objectMap [
      rml:reference "DATE" ] ] .
```

Listing 5.6: RML-star mapping rules translated from Listing 5.5.

Additional Updates

We enable YARRRML-star with other new features that have been incorporated into RML in recent years. We extend YARRRML to assign datatypes and language tags dynamically to objects. They are generated with the data values, in the following examples the datatype is generated dynamically with the data field `DATA-TYPE` (Listing 5.7), and the language tag with `COUNTRY` (Listing 5.8). They translate into RML as Listings 5.9 and 5.10 show.

```
- [:date, $(DATE), xsd:$(DATE-TYPE)]
```

Listing 5.7: Dynamic datatype in YARRRML-star.

```
- [:name, $(PERSON), $(COUNTRY)~lang]
```

Listing 5.8: Dynamic language tag in YARRRML-star.

```
rr:predicateObjectMap [
 rr:predicate :jumpsOnDate ;
 rml:objectMap [
  rml:reference "DATE";
  rml:datatypeMap [
   rr:template "xsd:{DATE-TYPE}"]]];
```

Listing 5.9: Dynamic datatype in RML translated from Listing 5.7.

```
rr:predicateObjectMap [
 rr:predicate :name ;
 rml:objectMap [
  rml:reference "PERSON";
  rml:languageMap [
   rml:reference "COUNTRY"]]];
```

Listing 5.10: Dynamic language tag in RML translated from Listing 5.8.

We also incorporate a shortcut for specifying join conditions (Listing 5.11). This shortcut

follows the functions' syntax[12]. It is specified as the function `join` that takes as parameters the mapping identifier (with the `mapping=` parameter key) and the similarity function to perform the join. This function can, in turn, take data values as parameters. Quoted and non-asserted triples can also be generated within join conditions by using the parameter keys `quoted=` and `quotedNonAsserted=` respectively. In the example, the join condition is performed using the `equal` function to create as objects the subjects of the mapping set `jumpTM` if the values of the fields `ID` from source and `ID` from target mapping set are the same.

```
1 - predicates: :jumps
2   objects:
3    - function: join(mapping=jumpTM, equal(str1=$(ID), str2=$(ID)))
```

Listing 5.11: Abbreviated syntax for join conditions in YARRRML-star.

5.2.2 Implementation: Yatter

Yatter[13] is a bi-directional YARRRML translator that supports the features described in previous sections. Yatter receives as input a mapping document in the YARRRML serialization and the desirable output format (R2RML or RML), or the other way around.

Algorithm 2 presents the procedure implemented by the system to translate an input YARRRML mapping document into [R2]RML. First, the namespaces defined in YARRRML are added together with a set of predefined ones (e.g., `foaf`[14], `rml`[15], `rdf`[16]) that are used in by [R2]RML.

Second, functions, targets, and databases are identified in the entire YARRRML document and translated into RML. Each of them generates a global identifier mapped into a hash table that can be used by any `rr:TermMap`. For external source declaration, their identifiers are also mapped into a hash table for the next steps. Regarding the RDF-star support, the mapping rules are parsed to identify if it contains `quoted` or `quotedNonAsserted` keys. This determines if the translation requires producing RML-star mapping rules. If true, a hash table is also created for the mapping instances of `rml:NonAssertedTriplesMap`.

In YARRRML, lists of sources and subjects maps can be defined within the same triples map, but [R2]RML triples maps may contain only one source and one subject. Hence, for each mapping document, a list of sources and subjects is first collected and then translated depending on the desirable output format ([R2]RML[-star]). Nevertheless, multiple predicate maps and object maps are allowed within the same triples map, and they are directly translated to RML. Finally, the cartesian product of sources and subject maps together with the predicate object maps is combined to generate the desirable triples map. Before returning the mapping rules, Yatter validates that the generated output is a valid RDF graph.

Although YARRRML leaves the RDF-based syntax of the mapping rules to be processed only

[12] https://rml.io/yarrrml/spec/#functions
[13] https://github.com/oeg-upm/yatter/
[14] http://xmlns.com/foaf/0.1/
[15] http://semweb.mmlab.be/ns/rml#
[16] http://www.w3.org/1999/02/22-rdf-syntax-ns#

Algorithm 2: YARRRML-star translation algorithm

```
Result: [R2]RML mapping document
input_m ⟵ yarrrml_rules;
format ⟵ output_format;
output_m ⟵ ∅;
output_m.add(translate_prefixes(input_m));
if format == RML then
    output_m.add(translate_functions(input_m));
    output_m.add(translate_targets(input_m));
    is_star, non_asserted_maps ← analyze_rml_star(input_m);
end
output_m.add(translate_databases_access(input_m));
ext_sources ← get_external_sources(input_m);
for tm ∈ M.get_triples_map() do
    source_list ← translate_source(format, get_source_list(ext_sources, tm));
    subject_list ← translate_subject(is_star, format, get_subject_list(tm));
    predicates_objects ← translate_predicates_objects(is_star, format, tm);
    for s ∈ source_list do
        for subj ∈ subject_list do
            m ← combine(s, subj, predicates_objects, non_asserted_maps));
            output_m.add(m);
        end
    end
end
return validate(output_m);
```

by the knowledge graph construction systems, we also provide a human-readable output in the same way as Mapeathor does (see Section 5.1.2). Moreover, we ensure that functions, targets, and databases appear first, while for each `rr:TriplesMap`, the sequence is: source, subject map, and the set of predicate object maps.

The source code of Yatter is openly available under an Apache 2.0 license[13]. Following open science best practices, each release automatically generates a dedicated DOI to ensure reproducibility in any experimental evaluation[17]. The development is under continuous integration using GitHub Actions and the YARRRML test-cases (Section 5.2.3) have more than 80% code coverage. Yatter is available through PyPi as a module[18] (Chaves et al., 2024) to be easily integrated in any Python development.

5.2.3 Validation

We validate the extensions to YARRRML and the developed implementation by proposing and testing a set of test cases, and comparing to other proposed user-friendly serializations

[17] https://doi.org/10.5281/zenodo.7024500
[18] https://pypi.org/project/yatter/

and corresponding systems.

YARRRML Test Cases

Test cases are a common method to evaluate the conformance of a system (Arenas-Guerrero, Iglesias-Molina, et al., 2024; Heyvaert et al., 2019). To the best of our knowledge, previous R2RML (Villazón-Terrazas & Hausenblas, 2012) and RML (Heyvaert et al., 2019) test cases were not translated to any human-friendly serialization (e.g., YARRRML). Relying on [R2]RML test cases, we propose a set of representative test cases (including also the new features presented in this work) to assess the conformance of any YARRRML translator system. The proposed test cases serve two objectives, (i) to cover the complete vocabulary of the serialization, and (ii) to have the flexibility to declare the rules (e.g., shortcuts or location of the keys).

We follow a systematic methodology for creating the YARRRML test cases. We analyzed the [R2]RML test cases and observed that several assess correct data generation. Since YARRRML serves as user-friendly serialization for another mapping language, the focus of its test cases is not on assessing data correctness, but on covering the language expressiveness. Hence, we select 15 R2RML test cases that cover the R2RML features and manually translate them into YARRRML. Since RML is a superset of R2RML, it introduces modifications with respect to R2RML to include the definition of heterogenous datasets (e.g., `rr:LogicalTable` is superseded by `rml:LogicalSource`). We propose 8 new test cases to cover these features.

For features not covered by the RML test cases, we follow a similar procedure. We inspected the RML-star test cases (Chaves et al., 2022), and translated to YARRRML the ones that provide a complete coverage of this extension. From the 16 test cases proposed to assess the conformance of RML-star, we adapt 6. Finally, we propose 21 test cases to cover RML-Target, RML-FNML, RML dynamic language tags and datatypes.

In total, we defined 50 YARRRML test cases and their corresponding translation to RML or R2RML. They are openly available[19] to be used by any YARRRML-compliant system. Yatter passes all test cases successfully.

Serializations Comparison

We compare a set of user-friendly serializations and languages, namely SMS2 (Stardog, 2021), XRM (Zazuco, 2022), ShExML (García-González et al., 2020) and YARRRML (Heyvaert et al., 2018) incorporating the updates described in Section 5.2.1, regarding their expressiveness. To that end, we study 15 features that tackle usual characteristics and functionalities in mapping languages. We describe each and discuss how each serialization addresses it (Table 5.7).

LF1. Subject Term Type. This feature indicates what kind of RDF[-star] term the language can generate as subject. In RDF, subjects can be IRIs or blank nodes, while in RDF-star they can also be RDF-star triples. All serializations enable the creation of subjects at least as IRIs, SMS2 and YARRRML additionally implement RDF-star triples and, along with ShExML, blank nodes.

[19] https://github.com/oeg-upm/yarrrml-validation

Table 5.7: Features of user-friendly serializations. BN stands for blank node, L for literal, ST for RDF-star triple, C for constant and D for dynamic. Underlined features indicate the updates of YARRRML-star, while "*" indicates that a feature is possible with the implementation but not explicit in the serialization.

	ShExML	**SMS2**	**XRM**	**YARRRML-star**
LF1	BN, IRI	BN, IRI, ST	IRI	BN, IRI, ST
LF2	C, D (1..1)	C, D, (1..1)	C, D, (1..1)	C, D (0..1)
LF3	IRI	IRI	IRI	IRI
LF4	C (1..1)	C, D (1..1)	C (1..1)	C, D (1..N)
LF5	BN, IRI, L	BN, IRI, L, ST	IRI, L	BN, IRI, L, ST
LF6	C, D (1..1)	C, D (1..N)	C, D (1..1)	C, D (1..N)
LF7	C, D (0..1)	C (0..1)	C (0..1)	C, D (0..1)
LF8	C, D (0..1)	C, D (0..1)	C (0..1)	C, D (0..1)
LF9	C (0..1)	C (0..1)	C, D (0..N)	C, D (0..N)
LF10	(1..N)	(1..N)	(1..N)	(1..N)
LF11	Input	Input	Input	Input, output
LF12	✓	×*	×*	✓
LF13	✓	×	×	×
LF14	✓	✓	×	✓
LF15	✓	×	×	✓

LF2. Subject Generation. This feature indicates if subjects can be generated as constant or dynamic values; and how many subject declarations are allowed at a time. In dynamically generated values, the subject value changes with a field in the data source. In our example, the subject uses the field "`ID`" to generate different subject for each row of input data. All serializations can generate constant and dynamic subjects. For each set of rules, exactly one subject declaration is expected, i.e., one subject for predicate-object pairs. YARRRML can also accept no subject declaration, producing a blank node.

LF3. Predicate Term Type. This feature indicates if the serialization is able to generate an IRI for a predicate and all serializations do so.

LF4. Predicate Generation. This feature indicates if predicates can be generated as constant or dynamic values; and how many predicate declarations are allowed at a time. In dynamically generated values, the subject value changes with a field in the data source. SMS2 and YARRRML enable dynamic predicates, and YARRRML is also able to handle more than one predicate, which avoids repeating the same object for different predicates.

LF5. Object Term Type. This feature indicates what kind of RDF[-star] term the serialization is able to generate as object. The serializations can generate the same kinds of

terms as in subjects (LF1), with the addition of literals.

LF6. Object Generation. This feature indicates if objects can be generated as constant or dynamic values; and how many predicate declarations are allowed at a time. As for subjects, all serializations can generate constant and dynamic objects. In addition, SMS2 and YARRRML allow more than one at a time, which avoids repeating the same predicate for different objects.

LF7. Datatype. This feature indicates if datatypes can be specified constant or dynamically. All serializations enable the optional declaration of constant datatypes, but ShExML and YARRRML also enable dynamic datatypes.

LF8. Language Tag. This feature indicates if language tags can be constant or dynamic. Just as for datatypes, all serializations enable the optional declaration of constant language tags, XRM is the only not allowing dynamic.

LF9. Named Graph. This feature indicates if named graphs can be assigned to the generated statements and how (constant or dynamically). All serializations enable their optional declaration as constant IRI. XRM and YARRRML also enable more than one graph assignation, and allow dynamic values.

LF10. Data references. This features indicates how many data references a term can contain when generated dynamically (i.e., when its value changes with the input data). It applies to subjects, predicates, objects, datatypes, language tags and named graphs when the serialization allows dynamic generation. All serializations allow more than one data reference for dynamic generation.

LF11. Data Description. This feature indicates if the input or output data (e.g., format, iteration, name, path, etc.) can be described. All serializations can describe input data source, and YARRRML also provides the output data source.

LF12. Data Linking. This feature indicates if explicit data linking (e.g., join, fuzzy linking, etc) can be performed with mapping rules. ShExML and YARRRML provide specific features for this end; in XRM and SMS2, however, it is not explicit, but it is possible by using SQL queries.

LF13. Nested Hierarchies. This feature indicates if different levels of a hierarchy source can be accessed in the same data iteration. ShExML is the only language that implements this feature. It is not implemented in YARRRML since is not supported in RML either as a language feature in the time of writing.

LF14. Functions. This indicates if data transformations are applicable to input data (e.g., lowercase). XRM is the only serialization not supporting it.

LF15. Conditions. This feature indicates if a statement is generated or not depending on a condition. Only ShExML and YARRRML implement this.

Discussion. All serializations offer a rich variety of mapping features, but ShExML and YARRRML offer a richer selection with respct to the selected test cases. SMS2 leverages the SPARQL syntax, lowering the learning curve for SPARQL users. At the same time, data

processing is limited to basic SPARQL functions and the user is unable to integrate custom ones. XRM is designed to mimic natural language and adds minimal overhead with its own syntax keywords, which also makes it easy-to-learn, but provides a more limited variety of features.

Systems Comparison

We also compare the systems that support the aforementioned serializations: ShExML translator[20], Stardog[11] (with focus on how Stardog maps data sources to RDF graphs, using R2RML or SMS2), XRM translator (Zazuco, 2022), and YARRRML-parser[21] and our system, Yatter[13]. We study 8 system features to draw conclusions about them including:

SF1. Availability. Stardog and XRM are commercial systems and their implementation is not available. ShExML Java library[20], YARRRML-parser[21] and Yatter[13] are all available as GitHub repositories.

SF2. Programming Language. ShExML, XRM and Stardog are built in Java, YARRRML-parser in Javascript and Yatter in Python.

SF3. Input Data Sources. This feature indicates the data source formats that the system can translate, given the corresponding mapping rules. All systems support relational databases and CSV files as input data sources. Only Stardog and Yatter support NoSQL data sources.

SF4. Input serialization. This feature indicates the input mapping serialization. All systems support their corresponding mapping serialization. Additionally, Stardog can transform R2RML mapping rules to RDF graphs, whereas both YARRRML systems can translate R2RML or RML files into YARRRML.

SF5. Output serialization. This indicates the output mapping serialization. XRM and YARRRML systems translate their mapping rules to [R2]RML, while XRM also supports CARML and CSVW. Stardog directly constructs the RDF graph. ShExML generates both RML mapping rules and RDF graphs.

SF6. RDF-star Support. Only Stardog and Yatter support this feature. Stardog added RDF-star statement support in one of their latest releases using the “Edge Properties" configuration. Yatter improves upon YARRRML-parser by also enabling the construction of RDF-star graphs.

SF7. Dynamic Language Tag Support. ShExML, Yatter and Stardog provide support for this feature.

SF8. Dynamic Datatype Tag Support. Yatter and ShExML are the only systems that enable the reference of datatypes dynamically.

Additionally, based on the YARRRML test cases, we develop the corresponding test cases for the other analyzed serializations. The results of the conformance test of the analysed systems are presented online[19].

[20] https://github.com/herminiogg/ShExML
[21] https://github.com/RMLio/yarrrml-parser

Table 5.8: Features of the systems that support the user-friendly serializations.

	ShExML translator	**Stardog**	**XRM translator**	**YARRRML parser**	**Yatter**
SF1	Open Source	Closed source	Closed source	Open Source	Open Source
SF2	Java	Java	Java	Javascript	Python
SF3	RDB, CSV, JSON, XML, RDF	RDB, NoSQL, CSV, JSON, GraphQL	RDB, CSV, XML	RDB, NoSQL, CSV, JSON, XML	RDB, NoSQL, CSV, JSON, XML
SF4	ShExML	R2RML, SMS, SMS2	XRM	YARRRML, R2RML, RML	YARRRML, R2RML, RML
SF5	RML, RDF	RDF	R2RML, RML, CSVW, CARML	R2RML, RML, YARRRML	R2RML, RML, YARRRML
SF6	N/A	✓	N/A	×	✓
SF7	✓	✓	N/A	×	✓
SF8	✓	N/A	N/A	×	✓

Discussion. All systems provide a solid user experience and are -mostly- highly conformant with their corresponding serialization. ShExML is especially useful for integrating different data sources and formats, but lacks RDF-star support, writing functions results cumbersome and the translation to RML is incomplete. Stardog works smoothly with its proprietary Stardog databases, but managing several different sources becomes complex as a different mapping rule set is required per source. XRM is installed within a coding editor, and helps actively the writing process with suggestions and warnings. YARRRML-parser supports most of the functionalities that are also implemented in Yatter but still it does not support the latest RML features. YARRRML-parser translates functions to non-standard set of RML rules, while our implementation supports the specification proposed by the W3C CG on Knowledge Graph Construction[22].

5.3 Summary

This chapter addresses the second objective of this thesis *O2: To help knowledge engineers and domain experts with building declarative mappings in a user-friendly manner.* The first contribution within this objective is *C4: Design of a spreadsheet-based approach for writing mapping rules.* This contribution achieves a two-fold objective: (i) To ease the writing of mapping rules in a familiar environment (spreadsheets), while avoiding the need of learning a mapping language syntax; and (ii) to increase the interoperability among mapping languages, as the spreadsheet representation can be translated into more than one language. We test the developed approach in a user study with 30 volunteer practitioners, comparing it to a

[22] https://w3id.org/rml/fnml/spec

graphical approach (RMLEditor) and the baseline (writing mappings directly in the RML mapping language). The results suggest that participants manage to write mappings with higher quality and more complete than with the other approaches. Thus, this suggests that the spreadsheet-based approach is suitable for enhancing the mapping writing process for users.

The second contribution tackles *C5: Update of a user-friendly syntax with extended features.* We present the latest updates to a user-friendly syntax for the RML mapping language, YARRRML, one of the most popular and adopted for writing RML mappings. The updates include the generation of RDF-star graphs, dynamic datatypes and language tags, and a proposal for simplifying the syntax of join conditions. We qualitatively validate the updates proposed by comparing them to other user-friendly syntaxes to ensure maximum expressiveness coverage.

Finally, we present the implementations for both presented approaches, Mapeathor for generating [R2]RML and YARRRML mappings from spreadsheets, and Yatter for translating the updated YARRRML syntax into [R2]RML. The objective of the contributions presented in this chapter is to provide further support for user to ease the writing of mappings, aiming at reducing the barrier for adopting mappings to build knowledge graphs.

Chapter 6

Representation Impact on Knowledge Graph Refactoring from a Reification Perspective

Knowledge graphs are not immutable, as they are subject to modifications along with the ontology or network of ontologies that are related to. This need for evolution may be triggered by changes in the domain of knowledge or on the graph consumption requirements in downstream tasks. An example is the need to add information about a triple, thus making statements about statements (i.e., statement reification). For instance, Nanopublications (Groth et al., 2010) adopt a Named Graph-based (Carroll et al., 2005) model to publish minimal scientific statements along with their provenance and associated context. Meanwhile, in ontology engineering, N-Ary Relationships (Noy & Rector, 2006) are widely used as an ontology design pattern (Gangemi & Presutti, 2013). Moreover, the well-known knowledge base of Wikidata has implemented its own reification schema based on *qualifiers* (Erxleben et al., 2014).

It is not uncommon for an existing KG to strive for adopting one of these representations. Several studies throughout the years prove that these different representations impact the downstream KG consumption tasks; for query performance (Alocci et al., 2015; Das et al., 2014; Frey et al., 2019; Hernández et al., 2015; Nguyen et al., 2014; Orlandi et al., 2021) as well as for human consumption and machine learning-based tasks (Iglesias-Molina, Ahrabian, et al., 2023). Some examples of existing KGs that have changed their representation include the preliminar transition of Wikidata to the current qualifier-based representation (Erxleben et al., 2014); the publication of the DisGeNet KG as Nanopublications (Queralt-Rosinach et al., 2016), and the introduction of RDF-star in the European Union Agency for Railway KG[1].

When the situation comes that this kind of refactoring of the knowledge graph is needed, the following question arises: Is more convenient to construct the graph from the original input data sources again, or to re-construct it within the triplestore? This section analyses which of these approaches reports better performance and the parameters that influence

[1] https://data-interop.era.europa.eu/

the choice for a set of cases related to metadata representation approaches. To this end, we perform a comprehensive empirical study of this transformation in four representations (Standard Reification, N-Ary Relationships, Named Graphs and RDF-star) using (i) declarative mapping languages to construct each representation with KG construction engines and (ii) transformation per peers of representations with SPARQL `CONSTRUCT` queries in different triplestores. This chapter aims to unravel the aspects to consider when refactoring a knowledge graph from a reification perspective, to help in the decision on how to perform it and assess the role that mapping technologies can play in this kind of KG evolution.

6.1 Motivation

6.1.1 Re-construction Example

Consider the SemMedDB dataset (Kilicoglu et al., 2012), that contains entities extracted from biomedical texts, as well as the semantic types of these concepts and a timestamp of when they were extracted. A stable version of a knowledge graph represents this information, where the timestamp and semantic type are related to its correspondent entity with an N-Ary relationship. However, the maintainers of this knowledge graph want to reduce the size of the graph by reducing the number of nodes, as well as to represent this information with RDF-star to start adopting the RDF 1.2 specification.

The graph maintainers have access to the original source data, so they can change the original KG construction pipeline, that uses declarative mappings to construct the graph. However, since this graph is not going to be updated with new data, it is also possible to change the structure of the graph with queries within the triplestore where it is maintained. They know the time it takes to generate the graph in its original representation. However, they are not certain whether generating the graph in the new representation using this pipeline will take more time, or if with queries the transformation could be performed faster (Fig. 6.1). Hence, the objective of this work consist of assessing the aspects that influence this situation to help in making an informed decision.

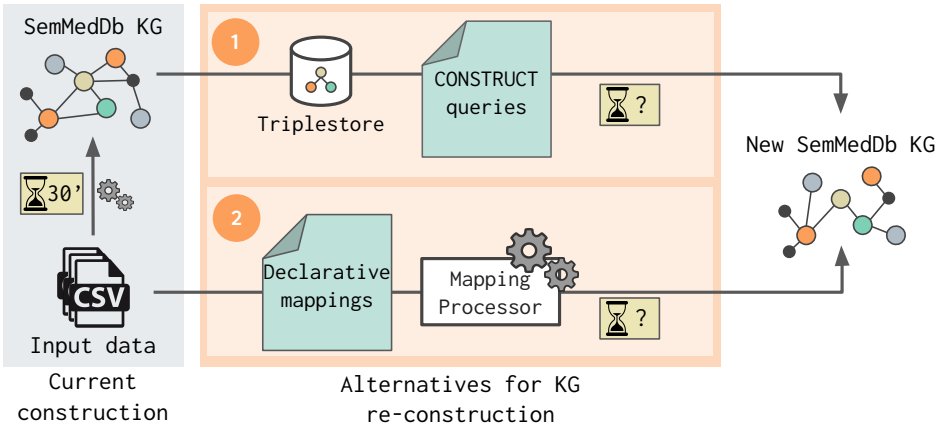


Figure 6.1: Alternatives for refactoring a pre-existing knowledge graph.

6.1.2 Representation Impact on Knowledge Graph Consumption

The different representations for adding additional information to triples have an impact on the performance of different KG consumption tasks. Thus, it becomes important to understand their differences and implications to choose the most suitable one depending on the particular needs.

Previous studies comparing knowledge representations have focused primarily on query performance (Alocci et al., 2015; Das et al., 2014; Frey et al., 2019; Hernández et al., 2015; Nguyen et al., 2014; Orlandi et al., 2021) and graph interoperability (Angles et al., 2019, 2020). For this scenario, the representations need to ensure efficiency to maximize performance. However, applications relating to exploration by end users and machine learning over KGs have not been taken into account (Hogan, 2020; Karger, 2014). Knowledge exploration scenarios, e.g., browsing, are impacted by representational choices, and therefore the selected representations should reduce the cognitive load and user expertise needed to explore, access, and acquire knowledge. Similarly, many embedding models-based tasks such as knowledge graph completion (Ren et al., 2022) require adequate representations to maximize the performance of the models on downstream predictive tasks.

Motivated by the lack of research on the representation impact on a broader set of KG consumption tasks, we carried out in (Iglesias-Molina, Ahrabian, et al., 2023) an evaluation to assess the fitness of four knowledge representation approaches (Standard Reification, N-Ary Relationships, Wikidata qualifiers, and RDF-Star) for the needs of the abovementioned scenarios. First, to understand user preferences in knowledge exploration tasks, we run a user study where participants interact with a web browser interface and a query endpoint to determine the representation that improves knowledge acquisition for real-world questions. Then, to assess the differential performance of the representations, we test several queries using synthetic and real-world data. Lastly, to estimate the impact on KG embedding model performance, we train and evaluate a selection of these models for the KG completion task with different representations.

This study found significant differences for particular scenarios, summarized in Table 6.1, drawing the following conclusions:

1. Standard Reification is the least suitable for users. Its anti-intuitive structure results time-consuming to navigate with, and it introduces additional complexity to retrieve complete information.

2. RDF-star still needs improved support in all studies scenarios, as it is underway of becoming part of the RDF 1.2 specification (Hartig et al., 2023). At the moment, it is risky to use it in high-demanding scenarios.

3. Qualifiers obtain steadily better results for retrieving results in high-demanding querying scenarios. Despite being restricted to Wikidata at the moment, its representation could be considered to be adopted in more knowledge graphs.

4. Analysing and understanding how each embedding model works is key to select a representation for graph completion (and hence, additional embedding-based tasks). While for the other scenarios all representations showed acceptable behaviour, here the

Table 6.1: Summary of the fitness for each representation evaluated in the studied scenarios, where ✓means suitable, ∽ is acceptable, ✕is to be avoided and * indicates that the value is not tested but equivalent to Qualifiers.

	User interaction	Simple graphs and queries	Large graphs and demanding queries	Graph completion
Qualifiers	✓	✓	✓	✓(RotatE)
RDF-star	✓*	✓	∽	-
N-Ary Rel.	✓	✓	∽	✓(TransE)
Std. Reif.	✕	✓	∽	✓(ComplEx)

decision is critical.

5. Promoting the use of interfaces such as browsers highly improves the user experience in knowledge exploration. These interfaces help mask the representation complexity and differences, which directly influences the adoption and usability of semantic resources, an aspect usually overlooked.

Overall, the lack of a good-for-all solution raises the need of improving the interoperability among representations, especially in cases where knowledge graphs are consumed for very different purposes. The study presented in this chapter looks into this interoperability issue when refactoring a KG in different representations.

6.2 Methodology

In this section, we present the methodology followed to analyze the impact of different approaches for re-constructing knowledge graphs from a reification perspective. More in detail, we compare two ways of re-structuring a KG to refactor it with a different reification approach, (i) with declarative mapping technologies from scratch (e.g., RML (Dimou et al., 2014; Iglesias-Molina, Van Assche, et al., 2023b)) and (ii) with `CONSTRUCT` queries in a triplestore from the previous version of the graph. All resources to reproduce the experiments are available on GitHub[2].

6.2.1 Dataset

We use the Semantic MEDLINE Database (SemMedDB) (Kilicoglu et al., 2012) in the experiments. This database consists of a repository with biomedical entities and relationships (subject-predicate-object) extracted from biomedical texts. The complete description of this dataset can be found in the *Biomedical Research Literature* use case in Section 4.3.2. Listings 6.1, 6.2 and 6.3 illustrate the columns used from the files with synthetic data. We use the CSV files to model the dataset as five annotated statements: Three assign *semantic*

[2]https://github.com/oeg-upm/kg-reconstruction-eval

Table 6.2: Number of triples of the SemMedDB graph in the selected representations for each scale.

	1K	10K	100K	1M
Standard Reification	25,000	249,997	2,499,966	24,999,607
Named Graphs	10,000	99,994	999,932	9,999,190
N-Ary Relationships	15,000	149,997	1,499,966	14,999,595
RDF-star	8,485	78,655	710,588	6,503,388

types to *subjects*, *objects*, and *entities* with a confidence score; and two provide the timestamp for the extraction of *subjects* and *objects* from text.

```
1 ENTITY_ID , SEMTYPE , SCORE
2 12345     , org     , 790
```

Listing 6.1: ENTITY.csv snippet.

```
1 PREDICATION_ID , SUBJ_SEMTYPE , SUBJ_NAME
2 13579          , Semtype      , SubjName
```

Listing 6.2: PREDICATION.csv snippet.

```
1 PREDICATION_AUX_ID , PREDICATION_ID , SUBJ_SCORE , TIMESTAMP
2 67890              , 13579          , 800        , 1651740766
```

Listing 6.3: PREDICATION_AUX.csv snippet.

To test the scalability of the evaluated approaches, we made subsets of this dataset in four different sizes, taking as input from the abovementioned CSV files with (i) 1K rows, (ii) 10K rows, (iii) 100K rows and (iv) 1M rows. The number of triples produced for each reification version for each size is shown in Table 6.2.

6.2.2 Mappings and Queries

The following resources are used: (i) a set of declarative mappings that are used for constructing the knowledge graph from the SemMedDB CSV files in the four selected reification representations; and (ii) a set of SPARQL queries that are used for re-constructing the graph within different triplestores. A summary of their characteristics is shown in Table 6.3.

We use two sets of mappings, one set written in the RML mapping language (Iglesias-Molina, Van Assche, et al., 2023b), and the other in SPARQL-Anything (Asprino et al., 2023). Each set of mappings is comprised of four mapping files, to construct the knowledge graphs in the four representations selected (i.e., Standard Reification, N-Ary Relationships, Named Graphs and RDF-star). Listings 6.4 and 6.5 show an RML and SPARQL-Anything mapping respectively that generates in RDF-star the annotation of the semantic types of *entities* with a score from the CSV file shown in Listing 6.1. These RML mappings use the RML-star module (described in Section 4.3). For these mappings, we show the number of sets of rules (i.e., *Triples Map*) and *Predicate Object Maps* specified (Table 6.3). For the SPARQL-Anything mappings, we show the number of triple patterns in this clause and additional SPARQL

Table 6.3: Characteristics of mappings in RML and SPARQL-Anything. #TM stands for number of Triples Map, #POM for Predicate Object Map, and #TP for Triple Patterns. The shown operators appear usually in the `WHERE` clause; the ones marked with ^{c} appear in the `CONSTRUCT` clause.

	RML		SPARQL-Anything	
	#TM	**#POM**	**#TP^c**	**Additional operators**
Standard Reification	9	20	25	UNION, BIND
Named Graphs	10	10	10	UNION, BIND, GRAPH^c
N-Ary Relationships	13	15	15	UNION, BIND
RDF-star	10	10	10	UNION, BIND

clauses (Table 6.3). All mappings contain the same number of triple patterns in the `WHERE` clause.

```
<#Entity>                          <#EntityScore>
 a rml:TriplesMap ;                  a rml:AssertedTriplesMap ;
 rml:logicalSource [                rml:logicalSource [
  rml:source "ENTITY.csv"            rml:source "ENTITY.csv"
 ];                                 ];
 rml:subjectMap [                   rml:subjectMap [
  rml:template ":{ENTITY_ID}"        rml:quotedTriplesMap <#Entity>;
 ];                                 ];
 rml:predicateObjectMap [           rml:predicateObjectMap [
  rml:predicate :semanticType;       rml:predicate :score ;
  rml:objectMap [                    rml:objectMap [
   rml:reference "SEMTYPE"            rml:reference "SCORE"
  ] ] .                              ] ] .
```

Listing 6.4: RML-star mapping snippet to create the RDF-star graph for *entity* from data in Listing 6.1.

```
CONSTRUCT {
  ?entity_id_iri :semanticType ?entity_semtype .
  << ?entity_id_iri :semanticType ?entity_semtype >> :score ?entity_score .
 } WHERE {
  {  SERVICE <x-sparql-anything:location=./data/entity.csv>
   { [] xyz:ENTITY_ID ?entity_id;
     xyz:SEMTYPE ?entity_semtype;
     xyz:SCORE ?entity_score;
    BIND(uri(concat(str("http://semmeddb.com/entity/"),
                        encode_for_uri(?entity_id))) as ?entity_id_iri)
  }  } }
```

Listing 6.5: SPARQL-Anything mapping snippet to create the RDF-star graph for *entity*

Table 6.4: Characteristics of the SPARQL queries used for the evaluation to transform the graph from a *source* representation to a *target* representation. #TP stands for the number of triple patterns. Fields marked with ^{c} appear in the `CONSTRUCT` clause, while ^{w} indicates the `WHERE` clause.

	Source	Target	#TP^w	#TP^c	UNION^w	BIND^w	FILTER^w	VALUES^w	GRAPH
Q1	**N-AryRel.**	**Std. Reif.**	6	10	✓			✓	
Q2		**RDF-Star**	6	4	✓			✓	
Q3		**Named Graphs**	6	4	✓			✓	✓^{c}
Q4	**Std. Reif.**	**RDF-star**	8	4	✓			✓	
Q5		**N-Ary Rel.**	20	15	✓		✓		
Q6		**Named Graphs**	8	4	✓			✓	✓^{c}
Q7	**RDF-star**	**Std. Reif.**	5	25	✓	✓	✓		
Q8		**N-Ary Rel.**	5	15	✓	✓	✓		
Q9		**Named Graphs**	5	10	✓	✓	✓		✓^{c}
Q10	**Named Graphs**	**Std. Reif.**	4	10	✓			✓	✓^{w}
Q11		**RDF-Star**	4	4	✓			✓	✓^{w}
Q12		**N-Ary Rel.**	4	15	✓			✓	✓^{w}

from data in Listing 6.1.

SPARQL queries use the `CONSTRUCT` clause to re-construct a given graph represented with one of the reifications into the other three representations. We use twelve queries to transform all pairs of representations, whose characteristics are shown in Table 6.4. We show for each query the number of triple patterns in the `WHERE` and `CONSTRUCT` clauses, and the additional operators used. Except for `GRAPH`, the rest of operators only appear within the `WHERE` clause. An example of a query is shown in Listing 6.6, which generates RDF-star from Named graphs.

```
1 CONSTRUCT  {
2   ?entity_id_iri :semanticType ?entity_semtype .
3   << ?entity_id_iri :semanticType ?entity_semtype >> :score ?entity_score .
4 } WHERE {
5   GRAPH ?entity_graph_iri { ?entity_id_iri :semanticType ?entity_semtype }
6   ?entity_graph_iri :score ?entity_score .
7 }
```

Listing 6.6: SPARQL query snippet to create the RDF-star graph from the Named Graphs representation for *entity*.

6.2.3 Engines

We choose a set of 2 mapping processors and 3 triplestores to be representative in our evaluation, while focusing on open-source tools. The selected mapping processors, Morph-KGC (v2.5.0) (Arenas-Guerrero, Chaves-Fraga, et al., 2024) and SPARQL-Anything (v0.8.1) (Asprino et al., 2023) are the systems with a stable version capable of producing RDF-star at the time of running these experiments. Regarding triplestores, we use the free version of GraphDB (v10.2.1), and the open-source Jena Fuseki (v4.8.0) and Oxigraph (v0.3.16). While GraphDB and Jena Fuseki store the graph in physical memory, Oxigraph performs the queries in memory. All engines perform duplicate removal in the results by default, with the exception of Oxigraph. For this triplestore, we add and measure a second step of duplicate removal with BASH commands.

We did attempt to include Virtuoso in the evaluation; however, several issues raised. The number of queries this triplestore can perform is limited in this evaluation (only 4 from 12): This triplestore does not implement RDF-star yet, and cannot produce named graphs declared inside the `CONSTRUCT` clause. In addition, Virtuoso limits the number of triples that can be produced with this clause to 1M.[3] When this limit is removed, an error appears that impedes the writing of the result in a file when it is large, which has been unsolved for years.[4]

6.2.4 Experimental Setup and Metrics

We perform the evaluation in two steps. First, the KG construction system evaluation is run, taking as input the SemMedDb dataset in CSV format and the mappings, and producing the corresponding RDF datasets in the four selected representations. Then, the triplestore evaluation is carried out, taking as input the produced datasets in the KG construction system evaluation, and producing RDF datasets in another representation. We perform a validation step afterwards to verify that the output graphs do not lack information.

We measure the *materialization time* to construct the RDF graph from the input data sources in KG construction systems. For triplestores, we measure *query execution time* as the total time from query execution until the complete answer is generated. We also report the geometric mean of all queries that generate the same reified RDF graph, similar to previously used by (Morsey et al., 2011; Schmidt et al., 2009). The *geometric mean* reports the central tendency of all execution times for a set of queries, reducing the effect of outliers. With this metric, we provide a general measurement of the performance of each triplestore when generating graphs in each representation. This allows an easier comparison with the KG construction systems. We run every experiment 5 times with a timeout of 24h, measure the time and calculate the median. We run all experiments over an Ubuntu 20.04 server with 32 cores Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz 102400 Mb RAM RDIMM, 3200 MT/s and 100 Gb HD SSD 6 Gb/s.

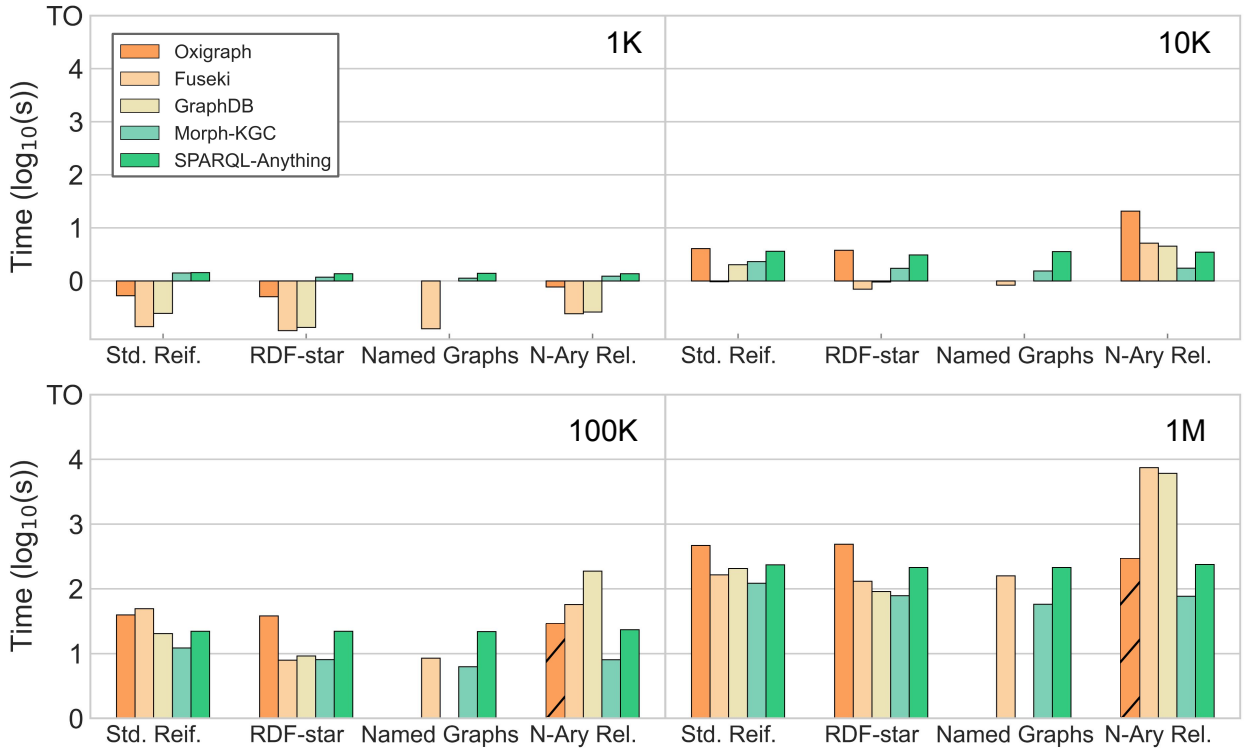


Figure 6.2: Execution time for KG construction engines with declarative mappings, and triplestores with SPARQL queries. The results of the triplestores are grouped by the *target* representation representing their geometric mean. Bars with diagonal pattern do no include some target representation times, as they report out-of-memory errors (details in Fig. 6.3). See values in detail in Table C.1 and Table C.2.

6.3 Results

The evaluation aims at assessing which is the re-construction approach that performs best for refactoring KGs, and to extract some insights about the impact of the different representations selected in the process. Fig. 6.2 reports the comparison between (KGC) systems using declarative mappings, and triplestores with SPARQL queries. Fig. 6.3 reports a fine-grained comparison of each triplestore performance on the proposed queries performing translations between each pair of representations. The reported results in detailed can be seen in Table C.1 and Table C.2.

Focusing on the comparison between triplestores and KGC systems (Fig. 6.2), we generally observe that for small data sizes, triplestores obtain better results, while KGC systems scale better as data size increases. However, except for producing N-Ary Relationships, Fuseki and GraphDB are competitive w.r.t. SPARQL-Anything or Morph-KGC. These triplestores obtain better results for the scales 1K and 10K, and similar ones for 100K and 1M. Additionally, despite the variety in mapping characteristics, neither Morph-KGC nor SPARQL-Anything

[3] https://lig-membres.imag.fr/rousset/publis/tess.pdf
[4] https://github.com/openlink/virtuoso-opensource/issues/11

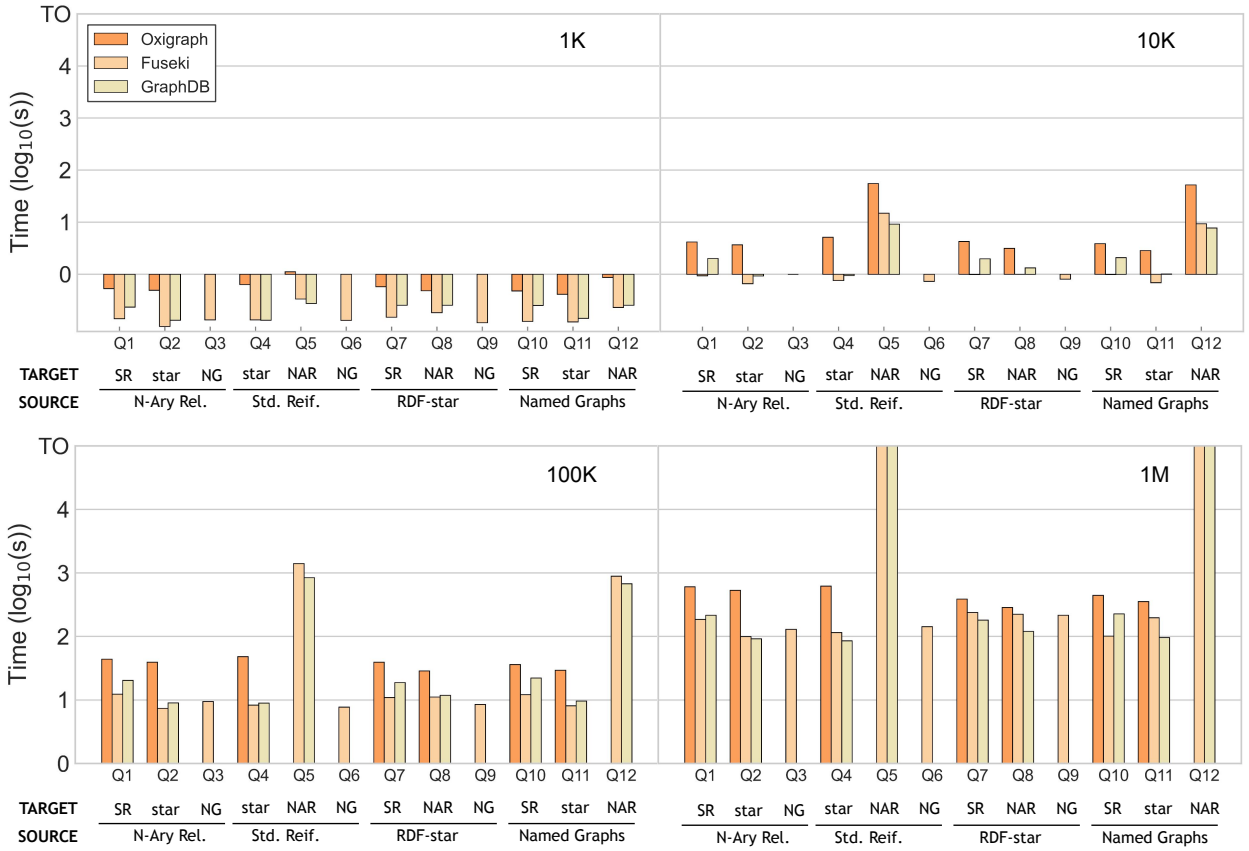


Figure 6.3: Execution time of triplestores with the SPARQL queries that perform translations for pairs of representations. See values in detail in Table C.2.

seem to be highly affected by the different representations, as opposed to the triplestores. The differences are not remarkable, but in general Morph-KGC generates the fastest Named Graphs, in contrast with SPARQL-Anything, which performs equally well with Named Graphs and RDF-star. For triplestores, producing Standard Reification and specially N-Ary Relationships is more costly than the other representations. The apparent good performance of Oxigraph in this case is due to out-of-memory errors in scales 100K and 1M. In addition, SPARQL 1.1 does not allow `GRAPH` clause within the `CONSTRUCT` operator. Hence, only Fuseki, which implements this extension natively, can generate the Named Graphs datasets.

Comparing the behavior of engines that successfully perform the same task, GraphDB overcomes Fuseki and Oxigraph for SPARQL queries, while Morph-KGC reports better results than SPARQL-Anything for KG construction from heterogeneous data sources, as was reported in a previous work (Arenas-Guerrero, Iglesias-Molina, et al., 2024). For small data sizes, GraphDB and Fuseki perform similarly, while Oxigraph reports higher query execution time and out-of-memory errors. This is because Oxigraph loads the complete RDF graph in memory and the physical data structures from Fuseki and GraphDB speed up query execution time. In larger datasets, GraphDB generally scales better than Fuseki, which for example, reports a timeout for generating N-Ary Rel. in scale 1M.

Table 6.5: Geometric mean of the query result times (s) from all triplestores grouping by *source* representation and *target* representation. The lowest times are highlighted in **bold**, while the highest are underlined.

	Representation	1K	10K	100K	1M
Source	**Std. Reif.**	0.283	4.126	56.612	1085.228
	RDF-Star	0.248	1.612	**15.951**	223.626
	Named Graphs	0.260	3.395	43.756	773.737
	N-Ary Rel.	**0.204**	**1.505**	16.040	**205.407**
Target	**Std. Reif.**	0.261	2.005	34.212	252.490
	RDF-Star	0.199	1.366	14.066	180.384
	Named Graphs	**0.126**	**0.836**	**8.524**	**158.678**
	N-Ary Rel.	0.365	7.896	67.886	2351.176

Looking into the particular differences of the pairs of translation performed with triplestores, we can observe how the different representations affect the performance of the graph re-construction with `CONSTRUCT` queries. Fig. 6.3 present the results grouping the queries in the legend by the *source* representation (i.e., from which representation the dataset is transformed). The behaviour reported when producing N-Ary Relationships stands out. It is more costly to produce than the other representations, except for when RDF-star is the *source* representation (Q8). Queries Q5 and Q12 report an out-of-memory error in Oxigraph for scales 100K and 1M, reaching a timeout in Fuseki and GraphDB in scale 1M. In contrast, Q8 requires a more complex `WHERE` clause, that avoids introducing a join that is needed in Q5 and Q12.

Table 6.5 shows a summary of the results from the triplestores grouped by *source* and *target* representation. In general, Named Graphs are the fastest representation to construct. N-Ary Relationships perform the fastest as the *source* representation, but require performing joins in the `CONSTRUCT` that make them the least suitable as a target representation. Standard Reification supposes the least suitable *source* representation, probably because of its larger size. Meanwhile, RDF-star performs consistently well acting as both *source* and *target* representation.

6.4 Discussion

In this chapter, we study how two graph re-construction approaches behave with different graph representations, and the aspects that influence their performance ensuring no information loss. We evaluate four representations with (i) KG construction systems, that construct the KG from heterogeneous data with declarative mappings; and (ii) using `CONSTRUCT` queries from KG stored in triplestores.

Our evaluation shows that KGC systems are more robust for increasing data size and interchange of metadata representation models. These systems are able to produce the four representations without major performance changes among them. Performing the re-

construction in triplestores is suitable for small data sizes, but it is more dependant on the target representation and query optimizations to offer competitive performance. In addition, this approach presents limitations as to which representations can be produced: not all existing triplestores process RDF-star, and only Fuseki can produce Named Graphs with `CONSTRUCT`. Thus, we can affirm that performing the re-construction with mappings is in general more reliable, as it is less affected by data size and target representation.

Regarding the representations, RDF-star and Named Graphs are a safe option to adopt in terms of system performance for both re-construction approaches. However, they are the ones that are currently less supported by both KGC systems (only for RDF-star) and triplestores. The other two representations are supported by a broader set of engines, some of which are not included in this study (e.g., RMLMapper, Virtuoso), but they present worse performance. Independently on the representation and re-construction approach, it is possible to re-construct a graph with no information loss.

The setup of the evaluation shows us the importance of optimizing SPARQL queries, which gains relevance as the triples to construct increase in number. For instance, the introduction of `UNION` clauses was needed to avoid costly cartesian products that prevented queries from finishing before the established timeout, or before reaching an out-of-memory error. This is particularly important when RDF-star is the *source* representation, as otherwise queries would often incur in erroneous output graphs. While this is well known in proficient SPARQL practitioners, it does not come as easily for non-expert users. The performance of SPARQL-Anything, relying almost entirely on SPARQL and Jena processing, is also affected by these different manners of writing the mapping. In contrast, RML mappings are less susceptible in this aspect, since how the user writes the mapping does not usually influence the performance of the compliant systems. SPARQL possess a flexibility and rich expressiveness that languages such as RML lack yet. Nevertheless, it poses the risk for non-expert users to hamper the result retrieval, incurring in suboptimal and erroneous queries. This opens up a challenge for improving the query processing, or even rewriting the queries automatically to be optimized, so as to reduce this accessibility gap of SPARQL for non-expert users.

In addition, this evaluation brings to light that the behavior of the `CONSTRUCT` clause is not as studied as other SPARQL operators. SPARQL benchmarks often overlook this clause, as it is considered as an extension of `SELECT` (Schmidt et al., 2009). Hence, it is assumed that their performance is comparable and not affected by the change of clause. However, we encountered `SELECT` queries that returned results in miliseconds, while taking several minutes with `CONSTRUCT`. In addition, all triplestores struggle when the `CONSTRUCT` clause includes several joins. This fact affects not only the metadata representations studied in this paper, but also all potential transformations for evolving knowledge graphs. Regarding KGC systems, the main challenge still consists of their adoption, as the learning curve for mapping languages is still steep despite efforts to lower it (Iglesias-Molina, Chaves-Fraga, et al., 2023).

6.5 Summary

This chapter addresses the third objective of the thesis *O3: To assess declarative knowledge graph construction technologies in terms of their benefits for supporting the creation and*

evolution of knowledge graphs. To this end, we study the re-construction of graphs with different technologies to implement different reification strategies. We evaluate, for a given KG where a refactoring of statement reification is needed, whether it is more efficient to construct the KG from the original data sources with the new representation, or to re-construct it within a triplestore. That is, to change the schema of a pre-existing knowledge graph without changing the data. To that end, we perform an empirical analysis using two KG construction engines and three triplestores, performing the re-construction with four data sizes in four different reification models. This comprises the main contribution associated with *O3, C6: Analysis of the scenarios in which declarative KG construction technologies play a valuable role in the refactoring of knowledge graphs.*

Chapter 7

Conclusions and Future Work

The general objective of this thesis is to improve the understanding and operational management of declarative KG construction languages, as stated in Section 3.1. To that end, we focus on three specific objectives: (i) understanding and gathering the current needs for KG construction, supporting the evolution of mapping languages to address them; (ii) helping knowledge engineers and domain experts to build mapping documents, providing the means for a user-friendly experience; and (iii) assessing the value of declarative KG construction technologies for supporting KGs in their evolution. These objectives are fulfilled with the contributions presented in the thesis.

In this chapter, we present a summary of the contributions, how they meet the objectives and the insights derived from them (Section 7.1), and next, we outline the future research lines (Section 7.2).

7.1 Contributions Summary and Conclusions

- Regarding the first objective *O1: To analyze the needs for declarative knowledge graph construction from heterogeneous data sources, in order to facilitate the support of advances in existing mapping languages to address the most relevant*, we first conducted an **extensive, fine-grained analysis of the expressiveness of the state-of-the-art mapping specifications**, presented as a comparison framework. We studied the characteristics of various mapping languages, a total of 16, based on different schemas (RDF and SPARQL mainly, among others), both pioneers and recent ones. We evaluated their features regarding data source and access description, triple generation and additional features (such as data transformation functions, linking rules, conditions, named graphs, etc.). This analysis sheds light on the understanding of these languages. It establishes the basis of the thesis, allowing us to identify the shared characteristics and their ultimate goal, their differential capabilities and the motivation behind them, how the needs have evolved and the new language releases and extensions with them, as well as their limitations. The framework analyses features not studied in detail so far and considers languages that have been excluded from other relevant related works, but that have had an impact in the field.

From this analysis, **a set of requirements for KG construction was extracted and enriched with the concerns of the community**, published as mapping challenges. These requirements were implemented in the Conceptual Mapping ontology, gathering the expressiveness of the analyzed languages and extending them with open challenges. This showcases how requirements can be implemented in RDF-based languages, and it opens the door to language interoperability. However, not all the features and capabilities of SPARQL-based languages could be implemented, as they can specify "instructions" for KG construction in a similar way as procedural languages do, and it is beyond the scope of representing them in an ontology. The ontology was evaluated by validating that the features provided in the language can address the set of requirements. The definition of the requirements and sets of features that a mapping language should be able to do has been taken as reference for different developments in the field, such as mapping translation (Alhazmi et al., 2023) and knowledge conversion between different data representations (Scrocca et al., 2024), where the Conceptual Mapping requirements defined have also been used for its evaluation.

The evolution of the set of requirements in KG construction led to the **development of RML-star, an extension of RML for constructing RDF-star graphs**. RML-star was developed along with a set of test cases, that are used to evaluate the conformance of an engine with respect to the language specification. The proposal was validated with two use cases, with data from (i) biomedical research literature and (ii) scientific software metadata extraction. The language could satisfy the needs of both use cases, thus reporting that it can successfully describe the generation of RDF-star graphs. RML-star has become a module part of the 2023 revision of the RML mapping language (Iglesias-Molina, Van Assche, et al., 2023b). Its relevance specially relies on the uptake of a new feature that will become part of the RDF 1.2 specification, currently under development (Hartig et al., 2023). It is already implemented by two systems, and a growing number of use cases, such as evaluations of statement reification (Iglesias-Molina, Toledo, et al., 2023; Kieffer et al., 2023), medical provenance and software metadata (Iglesias-Molina & Garijo, 2023).

Conclusion. Throughout the development of these contributions, we learned that, despite the wide variety of mapping languages, there is still a need to improve their capabilities. The declarative approach that they allow for constructing KGs brings a set of benefits, but they are still far from being as powerful as if the process used procedural languages (i.e., programming scripts) or SPARQL-based languages. Additionally, the semantic technologies are not static, they keep on evolving, and along with them, the requirements for KG construction. Hence, there is a continuous need to keep up-to-date with them. In the past few years, a community came together to tackle this issue, with the objective of identifying and overcoming current limitations in the languages to produce a versatile resource that can address more complex use cases, and that can be useful for a broader audience. This effort is materialized in the RML ontology (Iglesias-Molina, Van Assche, et al., 2023b), a revision of the RML mapping language that is in process of becoming a W3C standard. On the other hand, we learned how complex it is to define a good-for-all solution, and the implications of a technical implementation. The thorough analysis performed of mapping languages showed not only the capabilities of each language, but

also their adaptability for particular use cases, rationale and motivation behind each. While we believe that evolving the current techniques is needed, we also acknowledge that the heterogeneity of data and the complexity of data schema alignment is a long-term issue that may not be possible to solve with a single approach, however expressive it may be. Implementing interoperable and modular solutions can lead us to obtain a support environment of approaches suited for different use cases, yet able to communicate with each other, so that heterogeneity can become the solution, instead of the setback.

- For the second objective *O2: To help knowledge engineers and domain experts to build declarative mappings in a user-friendly manner*, we focus on improving the mapping writing process with two different approaches. The first involved the **development of a familiar spreadsheet-based environment to write mapping transformation rules**. This approach aimed at facilitating the task for any user, but in particular, profiles with no technical background that may struggle with the language's or serialization's syntax. The proposed spreadsheet design is an abstract representation of the transformation rules, which enables their translation into different languages. This approach was implemented in the tool Mapeathor, able to process Excel and Google Spreadsheets and generate mapping documents in R2RML, RML and YARRRML. The spreadsheet design was evaluated in a user study with the baseline, RML and a visual editor, RMLEditor. The pool of participants included 30 practitioners with heterogeneous backgrounds and expertise. The evaluation required participants to write some mapping rules in a fixed amount of time given the needed input. The results revealed that the proposed spreadsheet-based environment favored the writing in terms of accuracy and completeness compared to the other two. This motivates us to continue improving the spreadsheet design and implementation, to address the limitations raised by the participants, and to keep up to date with the currently evolving languages. This tool has been used to build knowledge graphs in soil ecological research (Le Guillarme & Thuiller, 2023), medical evidences (A. Á. Pérez et al., 2022) and scientific literature (Badenes-Olmedo & Corcho, 2023), among others.

  The second approach involved **extending the YARRRML serialization for RML, YARRRML-star**. These extensions included some of the latest advances in the new RML specification, RML-star among them. We also developed along with the serialization extensions a set of test cases, and the approach was validated in terms of expressiveness with other user friendly syntax (i.e., ShExML, SMS2 and XRM). A compliant implementation, Yatter, was developed along to translate bidirectionally YARRRML-star, RML and R2RML. This implementation successfully passed the proposed test cases, and serves towards the interoperability of these languages. Moreover, its lightweight design is adequate for use by other services, such as mapping documentation (Toledo et al., 2024), to process more languages.

  Conclusion. One of the biggest challenges to spread the adoption of the use of mapping languages for KG construction is their steep learning curve. The diversity of languages, an advantage for heterogeneity and expressiveness, supposes a setback in this regard, along with the general lack of interoperability. Despite the number of visual editors developed after the R2RML recommendation, none of these tools was widely adopted.

User-friendly serializations have been more successful, but they still impose a barrier for non-technical profiles. The stories and experiences of practitioners in the field led us to contribute to different approaches to this goal, trying to tackle the different needs of different profiles. Domain experts and users without technical skills are increasingly interested in using and producing knowledge graphs, while knowledgeable practitioners look for simplicity in the writing, forgetting about visual approaches that may slow down the process. These two contributions aim at lowering the learning barrier and facilitate the writing of these two profiles, so that mapping languages can be adopted and used more easily. However, we acknowledge that studies are missing that investigate why the success and failure of the different tools proposed over the years. For instance, why visual approaches, although they have been shown to suppose a lesser cognitive effort than plainly writing the mapping rules in a file (Crotti Junior et al., 2018), are not as widely used as user-friendly serializations, such as YARRRML. Not only that, but a deeper understanding in the main setbacks in the initial stage of the learning curve can greatly help with knowing how we can better communicate and teach to create mappings rules, which remains quite unclear.

- For the third objective *O3: To assess declarative knowledge graph construction technologies in terms of their benefits in supporting the creation and evolution of knowledge graphs*, we **evaluated the support of declarative KG construction technologies in the evolution of the schema of a knowledge graph**. We considered the changes in schema that come with the changes in the reification of triples, motivated from different real use cases for switching a metadata representation schema. These reification approaches model the same knowledge with different strategies, which affect the performance of downstream tasks. We evaluated the re-construction of a knowledge graph (i) from the original data sources with mapping technologies and (ii) pairwise from each representation with `CONSTRUCT` queries within a triplestore. The experiment was executed using different mapping languages along with their corresponding engines (RML with Morph-KGC and Facade-X with SPARQL-Anything) and triplestores (GraphDB, Jena Fuseki and Oxigraph). The results showed that for small amounts of data, triplestores were faster, but mapping technologies appeared as a more scalable approach as the data size increases, and robust to the changes of different representations. The re-construction process is also facilitated since how the `CONSTRUCT` queries are written highly influences the performance of triplestores, while it is indifferent in mapping documents.

  Conclusion. Mapping languages were developed for constructing KGs. The effort made to improve these declarative technologies has produced a set of mature resources optimized for construction, but also able to tackle more tasks within the knowledge graph life cycle (e.g., data pre-processing with transformation functions, metadata annotation). The contribution of this thesis in this regard is also to demonstrate its potential for long-term maintenance that supports the evolution of KG schemas. It shows the maturity level of declarative mapping-compliant technologies with respect to well-established resources such as triplestores, and how it can also bring benefits to users to perform the change of schema. The components and elements in a mapping influence the performance of the engines (Chaves-Fraga et al., 2019). Still, despite the

limited changes in schema evaluated in this contribution, we clearly observed that how mapping rules are written has a lesser impact on the performance in comparison to using SPARQL queries. We also stress the need for a wider implementation of the features for constructing graphs with these different metadata representations. Named graphs have been around for several years (Prud'hommeaux & Seaborne, 2008), yet the SPARQL specification does not allow to generate them with the `CONSTRUCT` clause. As for RDF-star, it is reported to outperform other approaches in certain scenarios, yet its adoption is the most limited at the time of writing.

7.2 Future Work

Following, we present some open challenges and issues that have not been addressed in the thesis, or that have appeared during or as a consequence of the advances proposed in it.

To be able to progress in declarative KGC technologies, there are three essential factors involved: (i) knowing which are the technical necessities for constructing KGs, (ii) acknowledging the needs of the target audience that is going to use them, and (iii) developing technologies and resources that can meet the needs gathered in (i) and (ii). Despite recent efforts to collect the technical necessities, this is an ongoing line of work that will continue in the future. While semantic technologies advance and evolve, there will always be the need to stay up-to-date to produce KGs capable of serving the tasks they are created for. The needs and preferences of users, especially with non-technical profiles, need to be put in the center. We need to learn from previous success and failure stories to put efforts into developments that are easier to adopt and continue to expand the community of users to a broader audience. Hence, it is essential in this regard to understand what made the most used resources successful, how we can better show their potential, and teach adequately to reduce the learning curve to ultimately ease and promote adoption.

Regarding mapping languages and their ability to fully address current KG construction needs, we believe that there is still room for improvement in terms of expressiveness. Specially RDF-based languages have limitations in how they can represent the transformation, a limitation that for SPARQL-based languages is softer. Currently, progress has been made in this regard with the collaboration with the Knowledge Graph Construction W3C Community Group to address some of the identified challenges. Participating in this international community opens the door to a wider engagement and agreement among different parties with heterogeneous perspectives. In addition to activities that are strictly related to the group, every year the KG Construction Workshop is held to gather insights and opinions from the community. This offers an opportunity to know the needs of practitioners using the developed resources, and to receive more feedback on which should be the roadmap in the mid-term.

There is also plenty of work to boost the adoption of declarative approaches for KG construction. Another key aspect on this issue is to demonstrate the overall benefits compared to ad-hoc approaches. In this regard, this thesis is focused on the evolution step of the KG life cycle in a restricted scenario: change of schema from different metadata representation approaches. There are studies showing the differential performance of KGC engines based on the characteristics of the mappings (Chaves-Fraga et al., 2019). We need to study these

changes further, considering also changes in the data and not only the KG schema, to assess their role in a wider set of evolution scenarios. In this direction, there is preliminary work on detecting the changes in the semantic resources involved in a KG life cycle, to propagate them to the affected resources automatically. Currently, this work focuses on propagating changes in the ontology to the mapping rules (Conde Herreros et al., 2024). Hence, there is still plenty future work to tackle more semantic resources (e.g., queries and validation shapes) and propagate the changes in any direction, not only stemming from ontologies. We are also interested in further evaluating how we can empower the transparency of workflows supporting the KG life cycle.

Lastly, the recent success of Large Language Models (LLM) with the release of ChatGPT has introduced them in the semantic web community to assess how they can improve current processes. These models have proven to be valuable for user interaction. This may be the main reason why they have had such success in so little time, despite the hallucinations produced in their responses (i.e., providing made-up facts that are false). These models can be suitable for enhancing different tasks during the KG life cycle, from the development of ontologies, to the KG construction and usage. In fact, there are already some initial work that investigate how they can help in writing declarative mappings (Hofer et al., 2024; Randles & O'Sullivan, 2024). This opens the door to keep looking into we can train and best interact with LLMs to help us create the mappings, to reduce the workload and effort for users. Thus, we need to assess whether this can be the solution towards automatic mapping creation, a research line that so far has not produced satisfying enough results.

References

Aisopos, F., Jozashoori, S., Niazmand, E., Purohit, D., Rivas, A., Sakor, A., Iglesias, E., Vogiatzis, D., Menasalvas, E., González, A. R., et al. (2023). Knowledge Graphs for Enhancing Transparency in Health Data Ecosystems. *Semantic Web*. https://doi.org/10.3233/SW-223294

Albertoni, R., Browning, D., Cox, S., González Beltrán, A., Perego, A., Winstanley, P., Maali, F., & Erickson, J. (2020). *Data Catalog Vocabulary (DCAT)* (W3C Recommendation 04 February 2020) (https://www.w3.org/TR/vocab-dcat-2/).

Alhazmi, A., Salas, J. O., & Konstantinidis, G. (2023). ForBackBench: From Database to Semantic Web Mappings and Back. *Proceedings of the ISWC 2023 Posters, Demos and Industry Tracks, co-located with 22nd International Semantic Web Conference (ISWC 2023), Athens, Greece, November 6-10, 2023*, *3632*.

Alobaid, A., Garijo, D., Poveda-Villalón, M., Santana-Perez, I., Fernández-Izquierdo, A., & Corcho, O. (2019). Automating ontology engineering support activities with ontoology. *Journal of Web Semantics*, *57*, 100472. https://doi.org/10.1016/j.websem.2018.09.003

Alocci, D., Mariethoz, J., Horlacher, O., Bolleman, J. T., Campbell, M. P., & Lisacek, F. (2015). Property graph vs RDF triple store: A comparison on glycan substructure search. *PloS ONE*, *10*(12), e0144578.

Angles, R., Thakkar, H., & Tomaszuk, D. (2019). RDF and Property Graphs Interoperability: Status and Issues. *Proceedings of the 13th Alberto Mendelzon International Workshop on Foundations of Data Management Asunción, Paraguay, June 3–7, 2019*, *2369*.

Angles, R., Thakkar, H., & Tomaszuk, D. (2020). Mapping RDF databases to property graph databases. *IEEE Access*, *8*, 86091–86110.

Arenas-Guerrero, J., Chaves-Fraga, D., Toledo, J., Pérez, M. S., & Corcho, O. (2024). Morph-KGC: Scalable Knowledge Graph Materialization with Mapping Partitions. *Semantic Web*, *15*, 1–20. https://doi.org/10.3233/SW-223135

Arenas-Guerrero, J., Iglesias-Molina, A., Chaves-Fraga, D., Garijo, D., Corcho, O., & Dimou, A. (2024). Declarative generation of RDF-star graphs from heterogeneous data. *Semantic Web (Under Review)*.

Aryan, P. R., Ekaputra, F. J., Kiesling, E., Tjoa, A. M., & Kurniawan, K. (2017). Rmlx: Mapping interface for integrating open data with linked data exploration environment. *2017 1st International Conference on Informatics and Computational Sciences (ICICoS)*, 113–118.

Asprino, L., Daga, E., Gangemi, A., & Mulholland, P. (2023). Knowledge Graph Construction with a Façade: A Unified Method to Access Heterogeneous Data Sources on the Web. *ACM Transactions on Internet Technology*, *23*(1). https://doi.org/10.1145/3555312

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). Dbpedia: A nucleus for a web of open data. *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007, Proceedings*, 722–735.

Badenes-Olmedo, C., & Corcho, O. (2023). Lessons Learned to Enable Question Answering on Knowledge Graphs Extracted from Scientific Publications: A Case Study on the Coronavirus Literature. *Journal of Biomedical Informatics*, *142*, 104382.

Barrasa, J., Corcho, ó., & Gómez-Pérez, A. (2004). R2O, an extensible and semantically based database-to-ontology mapping language. *Semantic Web and Databases: Second International Workshop, SWDB 2004, Toronto, Canada, August 29-30, 2004*, *14*. https://doi.org/10.1007/b106149

Barrasa, J., & Gómez-Pérez, A. (2006). Upgrading relational legacy data to the semantic web. *Proceedings of the 15th International Conference on World Wide Web, Edinburgh, Scotland, May 23 – 26, 2006*, 1069–1070.

Beckett, D., Berners-Lee, T., Prud'hommeaux, E., & Carothers, G. (2014). *RDF 1.1 Turtle: Terse RDF Triple Language* (W3C Recommendation 25 February 2014) (https://www.w3.org/TR/turtle/).

Berners-Lee, T., & Connolly, D. (2011). *Notation3 (N3): A readable RDF syntax* (W3C Team Submission 28 March 2011) (https://www.w3.org/TeamSubmission/n3/).

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American*, *284*(5), 34–43.

Bilbao-Arechabala, S., & Martínez-Rodríguez, B. (2022). A practical approach to cross-agri-domain interoperability and integration. *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, 1–6.

Birch, D., Lyford-Smith, D., & Guo, Y. (2018). The future of spreadsheets in the big data era. *arXiv preprint arXiv:1801.10231*.

Bischof, S., Decker, S., Krennwallner, T., Lopes, N., & Polleres, A. (2012). Mapping between RDF and XML with XSPARQL. *Journal on Data Semantics*, *1*(3), 147–185. https://doi.org/10.1007/s13740-012-0008-7

Bizer, C., & Seaborne, A. (2004). D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs. *Third International Semantic Web Conference (ISWC2004) Poster Track, Hiroshima, Japan, November 7–11, 2004*.

Blinkiewicz, M., & Bak, J. (2016). Square: A visual support for obda approach. *Proceedings of the 2nd International Workshop on Visualization and Interaction for Ontologies and Linked Data, Co-located with ISWC 2016, Kobe, Japan, October 17, 2016*, 41–53.

Bollacker, K., Cook, R., & Tufts, P. (2007). Freebase: A shared database of structured general human knowledge. *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence, Vancouver, Canada, July 22–26, 2007*, *2*, 1962–1963.

Bradbard, D. A., Alvis, C., & Morris, R. (2014). Spreadsheet usage by management accountants: An exploratory study. *Journal of Accounting Education*, *32*(4), 24–30.

Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodríguez-Muro, M., & Xiao, G. (2017). Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, *8*(3), 471–487.

Calvanese, D., Liuzzo, P., Mosca, A., Remesal, J., Rezk, M., & Rull, G. (2016). Ontology-based data integration in epnet: Production and distribution of food during the roman empire. *Engineering Applications of Artificial Intelligence*, *51*, 212–229.

Carothers, G., & Seaborne, A. (2014). *RDF 1.1 N-Triples: A line-based syntax for an RDF graph* (W3C Recommendation 25 February 2014) (https://www.w3.org/TR/n-triples/).

Carriero, V. A., Gangemi, A., Mancinelli, M. L., Marinucci, L., Nuzzolese, A. G., Presutti, V., & Veninata, C. (2019). ArCo: The Italian Cultural Heritage Knowledge Graph. *The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019*, *11779*, 36–52.

Carroll, J. J., Bizer, C., Hayes, P., & Stickler, P. (2005). Named Graphs, Provenance and Trust. *Proceedings of the 14th international conference on World Wide Web, Chiba, Japan, May 10 - 14, 2005*, 613–622.

Chaves, D., Gonzalez, M., Lopez, L., Doña, D., Guerrero, J. A., Ahmed, S., & Corcho, O. (2024). *Oeg-upm/yatter* (Zenodo) (https://doi.org/10.5281/zenodo.7024500).

Chaves, D., Iglesias, A., Garijo, D., & Guerrero, J. A. (2022, May). *Kg-construct/rml-star-test-cases: V1.1* (Zenodo) (https://doi.org/10.5281/zenodo.6518802).

Chaves Fraga, D. (2021). *Knowledge Graph Construction from Heterogeneous Data Sources exploiting Declarative Mapping Rules* [Doctoral dissertation, UPM].

Chaves-Fraga, D., Corcho, O., Yedro, F., Moreno, R., Olías, J., & De La Azuela, A. (2022). Systematic Construction of Knowledge Graphs for Research-Performing Organizations. *Information*, *13*(12), 562.

Chaves-Fraga, D., Endris, K. M., Iglesias, E., Corcho, O., & Vidal, M.-E. (2019). What are the Parameters that Affect the Construction of a Knowledge Graph? *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*, 695–713.

Chaves-Fraga, D., Priyatna, F., Cimmino, A., Toledo, J., Ruckhaus, E., & Corcho, O. (2020). GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain. *Journal of Web Semantics*, *65*, 100596. https://doi.org/10.1016/j.websem.2020.100596

Chaves-Fraga, D., Ruckhaus, E., Priyatna, F., Vidal, M.-E., & Corcho, O. (2021). Enhancing virtual ontology based access over tabular data with morph-csv. *Semantic Web*, *12*(6), 869–902.

Chávez-Feria, S., García-Castro, R., & Poveda-Villalón, M. (2022). Chowlk: From UML-based Ontology Conceptualizations to OWL. *The Semantic Web: 19th International Conference, ESWC 2022, Hersonissos, Crete, Greece, May 29 – June 2, 2022, Proceedings*, 338–352. https://doi.org/10.1007/978-3-031-06981-9_20

Chortaras, A., & Stamou, G. (2018). D2rml: Integrating heterogeneous data and web services into custom rdf graphs. *Workshop on Linked Data on the Web co-located with The Web Conference 2018 (LDOW@WWW 2018), Lyon, France, April 23, 2018*, *2073*.

Chue Hong, N. P., Katz, D. S., Barker, M., Lamprecht, A.-L., Martinez, C., Psomopoulos, F. E., Harrow, J., Castro, L. J., Gruenpeter, M., Martinez, P. A., & Honeyman, T.

(2021). FAIR Principles for Research Software (FAIR4RS Principles). https://doi.org/10.15497/RDA00068

Cimmino, A., & García-Castro, R. (2024). Helio: A Framework for Implementing the Life Cycle of Knowledge Graphs. *Semantic Web*, *5*(1), 223–249.

Cimmino, A., Poveda-Villalón, M., & García-Castro, R. (2020). eWoT: A Semantic Interoperability Approach for Heterogeneous IoT Ecosystems Based on the Web of Things. *Sensors*, *20*(3). https://doi.org/10.3390/s20030822

Conde Herreros, D., Chaves-Fraga, D., Poveda-Villalón, M., Pernisch, R., Stork, L., & Corcho, O. (2024). Propagating Ontology Changes to Declarative Mappings in Construction of Knowledge Graphs. *Proceedings of the 5th International Workshop on Knowledge Graph Construction co-located with 21th Extended Semantic Web Conference (ESWC 2024), Herssonissos, May 26–30, 2024*.

Corcho, O., Chaves-Fraga, D., Toledo, J., Arenas-Guerrero, J., Badenes-Olmedo, C., Wang, M., Peng, H., Burrett, N., Mora, J., & Zhang, P. (2021). A high-level ontology network for ICT infrastructures. *The Semantic Web – ISWC 2021: 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24–28, 2021, Proceedings*, 446–462.

Corcho, O., Priyatna, F., & Chaves-Fraga, D. (2020). Towards a new generation of ontology based data access. *Semantic Web*, *11*(1), 153–160. https://doi.org/10.3233/SW-190384

Creswell, J. W., & Creswell, J. D. (2017). *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications Ldt.

Crotti Junior, A., Debruyne, C., Brennan, R., & O'Sullivan, D. (2016). FunUL: a method to incorporate functions into uplift mapping languages. *iiWAS '16: Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services, Singapore, Singapore, November 28–30, 2016*, 267–275. https://doi.org/10.1145/3011141.3011152

Crotti Junior, A., Debruyne, C., Longo, L., & O'Sullivan, D. (2018). On the mental workload assessment of uplift mapping representations in linked data. *Second International Symposium, H-WORKLOAD 2018, Amsterdam, The Netherlands, September 20–21, 2018*, 160–179. https://doi.org/10.1007/978-3-030-14273-5_10

Crotti Junior, A., Debruyne, C., & O'Sullivan, D. (2017). Juma: An editor that uses a block metaphor to facilitate the creation and editing of r2rml mappings. *The Semantic Web: ESWC 2017 Satellite Events: ESWC 2017 Satellite Events, Portorož, Slovenia, May 28–June 1, 2017, Revised Selected Papers 14*, 87–92.

Cyganiak, R., Bizer, C., Garbers, J., Maresch, O., & Becker, C. (2012). *The D2RQ Mapping Language* [http://d2rq.org/d2rq-language].

Cyganiak, R., Wood, D., & Lanthaler, M. (2014). *RDF 1.1 Concepts and Abstract Syntax* (W3C Recommendation 25 February 2014) (https://www.w3.org/TR/rdf11-concepts/).

Daga, E., Asprino, L., Mulholland, P., & Gangemi, A. (2021, August). Facade-X: An Opinionated Approach to SPARQL Anything. In M. Alam, P. Groth, V. de Boer, T. Pellegrini, & H. J. Pandit (Eds.), *Volume 53: Further with knowledge graphs* (pp. 58–73, Vol. 53). IOS Press. https://doi.org/10.3233/ssw210035

Das, S., Srinivasan, J., Perry, M., Chong, E. I., & Banerjee, J. (2014). A Tale of Two Graphs: Property Graphs as RDF in Oracle. *Advances in Database Technology - EDBT 2014,*

17th International Conference on Extending Database Technology, Athens, Greece, March 24-28, 2014, Proceedings, 762–773.

Das, S., Sundara, S., & Cyganiak, R. (2012). *R2RML: RDB to RDF Mapping Language* (W3C Recommendation 27 September 2012) (www.w3.org/TR/r2rml).

De Meester, B., Dimou, A., Verborgh, R., & Mannens, E. (2016). An Ontology to Semantically Declare and Describe Functions. *The Semantic Web: ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29–June 2, 2016, Revised Selected Papers 13*, 46–49.

De Meester, B., Heyvaert, P., Verborgh, R., & Dimou, A. (2019). Mapping languages analysis of comparative characteristics. *1st International Workshop on Knowledge Graph Building and Large Scale RDF Analytics, co-located with the 16th Extended Semantic Web Conference (ESWC 2019), Portorož, Slovenia, June 3, 2019*, *2489*.

De Meester, B., Maroy, W., Dimou, A., Verborgh, R., & Mannens, E. (2017). Declarative data transformations for linked data generation: The case of DBpedia. *The Semantic Web: 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 – June 1, 2017, Proceedings, Part I*, *10250*, 33–48. https://doi.org/10.1007/978-3-319-58451-5_3

De Meester, B., Seymoens, T., Dimou, A., & Verborgh, R. (2020). Implementation-independent function reuse. *Future Generation Computer Systems*, *110*, 946–959.

De Meester, B., Van Assche, D., Iglesias-Molina, A., Jozashoori, S., & Chaves-Fraga, D. (2023). *RML-FNML Ontology: Functions* (Zenodo) (https://doi.org/10.5281/zenodo.7919856).

Debruyne, C., McKenna, L., & O'Sullivan, D. (2017). Extending R2RML with Support for RDF Collections and Containers to Generate MADS-RDF Datasets. In J. Kamps, G. Tsakonas, Y. Manolopoulos, L. S. Iliadis, & I. Karydis (Eds.), *Research and Advanced Technology for Digital Libraries - 21st International Conference on Theory and Practice of Digital Libraries, TPDL 2017, Thessaloniki, Greece, September 18-21, 2017, Proceedings* (pp. 531–536, Vol. 10450). Springer. https://doi.org/10.1007/978-3-319-67008-9_42

Debruyne, C., Michel, F., Iglesias-Molina, A., Van Assche, D., Chaves-Fraga, D., & Dimou, A. (2023). *RML-CC Ontology: Collections and Containers* (Zenodo) (https://doi.org/10.5281/zenodo.7919852).

Debruyne, C., & O'Sullivan, D. (2016). R2RML-F: towards sharing and executing domain logic in R2RML mappings. *LDOW@WWW 2016: Workshop on Linked Data on the Web co-located with 25th International World Wide Web Conference, Florence, Italy, April 11–15, 2016*, *1593*.

Delva, T., Arenas-Guerrero, J., Iglesias-Molina, A., Corcho, O., Chaves-Fraga, D., & Dimou, A. (2021). RML-star: A Declarative Mapping Language for RDF-star Generation. *International Semantic Web Conference (ISWC) 2021: Posters, Demos, and Industry Tracks, Virtual Event, October 24–28, 2021*, *2980*.

Delva, T., Van Assche, D., Heyvaert, P., De Meester, B., & Dimou, A. (2021). Integrating Nested Data into Knowledge Graphs with RML Fields. *Proceedings of the 2nd International Workshop on Knowledge Graph Construction co-located with 18th Extended Semantic Web Conference (ESWC 2021), Virtual Event, June 6–10, 2021*, *2873*, 1–16.

Dimou, A. (2020). High-Quality Knowledge Graphs Generation: R2RML and RML Comparison, Rules Validation and Inconsistency Resolution. *Applications and Practices in Ontology Design, Extraction, and Reasoning*, *49*, 55.

Dimou, A., Sande, M. V., Colpaert, P., Verborgh, R., Mannens, E., & Van De Walle, R. (2014). RML: A generic language for integrated RDF mappings of heterogeneous data. *Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (LDOW@WWW 2014), Seoul, Korea*, *1184*.

Dimou, A., Vander Sande, M., De Meester, B., Heyvaert, P., & Delva, T. (2020). *RDF Mapping Language (RML)* [https://rml.io/specs/rml/].

Dimou, A., Verborgh, R., Vander Sande, M., Mannens, E., & Van de Walle, R. (2015). Machine-interpretable dataset and service descriptions for heterogeneous data access and retrieval. *Proceedings of the 11th International Conference on Semantic Systems - SEMANTICS '15*. https://doi.org/10.1145/2814864.2814873

Dumontier, M., Callahan, A., Cruz-Toledo, J., Ansell, P., Emonet, V., Belleau, F., & Droit, A. (2014). Bio2RDF release 3: a larger connected network of linked data for the life sciences. *ISWC-PD'14: Proceedings of the 2014 International Conference on Posters & Demonstrations Track, Riva del Garda, Italy, October 19–23, 2014*, *1272*, 401–404.

Erxleben, F., Günther, M., Krötzsch, M., Mendez, J., & Vrandečić, D. (2014). Introducing Wikidata to the linked data web. *The Semantic Web – ISWC 2014: 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, 50–65.

Färber, M. (2019). The Microsoft Academic Knowledge Ggraph: A Linked Data Source with 8 Billion Triples of Scholarly Data. *The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019*, *11779*, 113–129.

Färber, M., Lamprecht, D., Krause, J., Aung, L., & Haase, P. (2023). Semopenalex: The scientific landscape in 26 billion rdf triples. *Proceedings of the 22nd International Semantic Web Conference, Athens, Greece, November 6–10, 2023*, *14265*, 94–112.

Fernández-Izquierdo, A., Cimmino, A., & García-Castro, R. (2021). Supporting Demand-Response strategies with the DELTA ontology. *2021 IEEE/ACS 18th International Conference on Computer Systems and Applications (AICCSA), Tangier, Morocco, November 30 – December 3, 2021*, 1–8. https://doi.org/10.1109/AICCSA53542.2021.9686935

Frey, J., Müller, K., Hellmann, S., Rahm, E., & Vidal, M.-E. (2019). Evaluation of metadata representations in RDF stores. *Semantic Web*, *10*(2), 205–229.

Gandon, F., & Schreiber, G. (2014). *RDF 1.1 XML Syntax* (W3C Recommendation 25 February 2014) (https://www.w3.org/TR/rdf-syntax-grammar/).

Gangemi, A., & Presutti, V. (2013). A Multi-dimensional Comparison of Ontology Design Patterns for Representing N-Ary Relations. *39th International Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 26-31, 2013, Proceedings*, 86–105.

García-González, H. (2021). A ShExML Perspective on Mapping Challenges: Already Solved Ones, Language Modifications and Future Required Actions. *Proceedings of the 2nd International Workshop on Knowledge Graph Construction co-located with 18th Extended Semantic Web Conference (ESWC 2021), Virtual Event, June 6–10, 2021*, *2873*, 1–14.

García-González, H. (2022). *Shape expressions mapping language (shexml)* [http://shexml.herminiogarcia.com/spec/].

García-González, H., Boneva, I., Staworko, S., Labra-Gayo, J. E., & Cueva-Lovelle, J. M. (2020). ShExML: improving the usability of heterogeneous data mapping languages for first-time users. *PeerJ Computer Science*, *6*, e318. https://doi.org/10.7717/peerj-cs.318

Garijo, D. (2017). WIDOCO: A Wizard for Documenting Ontologies. *The Semantic Web – ISWC 2017 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part I*, 94–102. https://doi.org/10.1007/978-3-319-68204-4_9

Garijo, D., Corcho, O., & Poveda-Villalón, M. (2021). FOOPS!: An Ontology Pitfall Scanner for the FAIR principles. *International Semantic Web Conference (ISWC) 2021: Posters, Demos, and Industry Tracks, Virtual Event, October 24–28, 2021*, *2980*.

Gonçalves, R. S., Horridge, M., Li, R., Liu, Y., Musen, M. A., Nyulas, C. I., Obamos, E., Shrouty, D., & Temple, D. (2019). Use of OWL and Semantic Web Technologies at Pinterest. *The Semantic Web–ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019*, *11779*, 418–435.

González-Gerpe, S., Cimmino, A., Bernardos, S., García-Castro, R., Poveda-Villalón, M., Katsigarakis, K., Lilis, G. N., & Rovas, D. (2022). An extension of thing descriptions from the web of things for digital twins. *Open Research Europe*, *2*(144), 144.

Grassi, M., Scrocca, M., Carenini, A., Comerio, M., & Celino, I. (2023). Composable semantic data transformation pipelines with chimera. *Proceedings of the 4th International Workshop on Knowledge Graph Construction co-located with 20th Extended Semantic Web Conference (ESWC 2023), Hersonissos, Greece, May 28 – June 1, 2023*, *3471*.

Groth, P., Gibson, A., & Velterop, J. (2010). The Anatomy of a Manopublication. *Information Services & Use*, *30*(1-2), 51–56.

Guha, R. V., Brickley, D., & Macbeth, S. (2016). Schema. org: Evolution of structured data on the web. *Communications of the ACM*, *59*(2), 44–51.

Gupta, S., Szekely, P., Knoblock, C. A., Goel, A., Taheriyan, M., & Muslea, M. (2012). Karma: A System for Mapping Structured Sources Into The Semantic Web. *The Semantic Web: Research and Applications 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012*, 430–434.

Harris, S., Seaborne, A., & Prud'hommeaux, E. (2013). *SPARQL 1.1 Query Language* (W3C Recommendation 21 march 2013) (https://www.w3.org/TR/sparql11-query/).

Hartig, O. (2017). Foundations of RDF* and SPARQL* (An Alternative Approach to Statement-Level Metadata in RDF). *Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7–9, 2017*, *1912*.

Hartig, O., Champin, P.-A., & Kellogg, G. (2023). *RDF 1.2 Concepts and Abstract Syntax* (W3C Working Draft) (https://www.w3.org/TR/2023/WD-rdf12-concepts-20230615/). W3C.

Herman, I., Adida, B., Sporny, M., & Birbeck, M. (2017). *RDFa 1.1 Primer - Third Edition* (W3C Working Group Note 17 March 2015) (https://www.w3.org/TR/rdfa-primer/).

Hernández, D., Hogan, A., & Krötzsch, M. (2015). Reifying RDF: What works well with Wikidata? *11th International Workshop on Scalable Semantic Web Knowledge Base Systems co-located with 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015*, *1457*, 32–47.

Hert, M., Reif, G., & Gall, H. C. (2011). A comparison of RDB-to-RDF mapping languages. *I-Semantics '11: Proceedings of the 7th International Conference on Semantic Systems, Graz, Austria, September 7–9, 2011*, 25–32. https://doi.org/10.1145/2063518.2063522

Heyvaert, P., Chaves-Fraga, D., Priyatna, F., Corcho, O., Mannens, E., Verborgh, R., & Dimou, A. (2019). Conformance test cases for the RDF mapping language (RML). *Knowledge Graphs and Semantic Web: First Iberoamerican Conference, KGSWC 2019, Villa Clara, Cuba, June 23–30, 2019, Proceedings*, 162–173.

Heyvaert, P., De Meester, B., Dimou, A., & Verborgh, R. (2018). Declarative Rules for Linked Data Generation at your Fingertips! *The Semantic Web: ESWC 2018 Satellite Events – ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers*, *11155*, 213–217. https://doi.org/10.1007/978-3-319-98192-5_40

Heyvaert, P., Dimou, A., Herregodts, A.-L., Verborgh, R., Schuurman, D., Mannens, E., & Van de Walle, R. (2016). RMLEditor: A Graph-based Mapping Editor for Linked Data Mappings. *The Semantic Web. Latest Advances and New Domains 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 – June 2, 2016, Proceedings*, 709–723.

Hofer, M., Frey, J., & Rahm, E. (2024). Towards self-configuring knowledge graph construction pipelines using llms – a case study with rml. *Proceedings of the 5th International Workshop on Knowledge Graph Construction co-located with 21th Extended Semantic Web Conference (ESWC 2024), Herssonissos, May 26–30, 2024*.

Hogan, A. (2020). The Semantic Web: Two Decades On. *Semantic Web*, *11*(1), 169–185.

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G. D., Gutierrez, C., Kirrane, S., Labra-Gayo, J. E., Navigli, R., Neumaier, S., et al. (2021). Knowledge graphs. *ACM Computing Surveys (CSUR)*, *54*(4), 1–37.

Iglesias, E., Chaves, D., Doña, D., Jozashoori, S., Rohde, P. D., Endris, K. M., Vidal, M.-E., Emonet, V., Assche, D. V., Leinweber, K., & Alexiev, V. (2024, May). *Sdm-tib/sdm-rdfizer: V4.7.4* (Zenodo) (https://doi.org/10.5281/zenodo.11191586).

Iglesias, E., Jozashoori, S., Chaves-Fraga, D., Collarana, D., & Vidal, M.-E. (2020). SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs. *Proceedings of the 29th ACM International Conference on Information and Knowledge Management, Virtual Event, October 19–23, 2020*, 3039–3046. https://doi.org/10.1145/3340531.3412881

Iglesias, E., Jozashoori, S., & Vidal, M.-E. (2023). Scaling up Knowledge Graph Creation to Large and Heterogeneous Data Sources. *Journal of Web Semantics*, *75*, 100755.

Iglesias, E., Vidal, M., Jozashoori, S., Collarana, D., & Chaves-Fraga, D. (2022). Empowering the SDM-RDFizer tool for scaling up to complex knowledge graph creation pipelines. *Semantic Web*.

Iglesias-Molina, A. (2022). *Resources for User Study of Mapeathor* (Zenodo) (https://doi.org/10.5281/zenodo.8154522).

Iglesias-Molina, A., Ahrabian, K., Ilievski, F., Pujara, J., & Corcho, O. (2023). Comparison of Knowledge Graph Representations for Consumer Scenarios. *Proceedings of the 22nd International Semantic Web Conference, Athens, Greece, November 6–10, 2023*, *14265*, 271–289.

Iglesias-Molina, A., Arenas-Guerrero, J., Delva, T., Dimou, A., & Chaves-Fraga, D. (2022, May). *RML-star* (W3C Draft Community Group Report) (https://w3id.org/rml/star/spec/).

Iglesias-Molina, A., Chaves-Fraga, D., Dasoulas, I., & Dimou, A. (2023). Human-Friendly RDF Graph Construction: Which One Do You Chose? *Web Engineering: 23rd International Conference, ICWE 2023, Alicante, Spain, June 6—9, 2023, Proceedings*, 262–277.

Iglesias-Molina, A., Chaves-Fraga, D., Priyatna, F., & Corcho, O. (2019). Enhancing the Maintainability of the Bio2RDF Project Using Declarative Mappings. *12th International Conference on Semantic Web Applications and Tools for Health Care and Life Sciences, Edinburgh, Scotland, December 10–11, 2019*, *2849*, 1–10.

Iglesias-Molina, A., Cimmino, A., & Corcho, O. (2022). Devising Mapping Interoperability with Mapping Translation. *Proceedings of the 3rd International Workshop on Knowledge Graph Construction co-located with 19th Extended Semantic Web Conference (ESWC 2022), Hersonissos, Greece, May 29 - June 2, 2022*, *3141*, 1–8.

Iglesias-Molina, A., Cimmino, A., Ruckhaus, E., Chaves-Fraga, D., García-Castro, R., & Corcho, O. (2024). An ontological approach for representing declarative mapping languages. *Semantic Web*, *15*(1), 191–221. https://doi.org/10.3233/SW-223224

Iglesias-Molina, A., & Garijo, D. (2023). Towards Assessing FAIR Research Software Best Practices in an Organization Using RDF-star). *Proceedings of the Posters and Demo Track of the 19th International Conference on Semantic Systems co-located with 19th International Conference on Semantic Systems (SEMANTiCS 2023) Leipzing, Germany, September 20 to 22, 2023*, *3526*.

Iglesias-Molina, A., Ruckhaus, E., Doña, D., & Chaves-Fraga, D. (2023). *Oeg-upm/mapeathor* (Zenodo) (https://doi.org/10.5281/zenodo.5973906).

Iglesias-Molina, A., Toledo, J., Corcho, O., & Chaves-Fraga, D. (2023). Re-construction impact on metadata representation models. *Proceedings of the 12th Knowledge Capture Conference 2023, Pensacola, USA, December 5 - 7, 2023*, 197–205.

Iglesias-Molina, A., Van Assche, D., Arenas-Guerrero, J., Chaves-Fraga, D., & Dimou, A. (2023a). *RML-star Ontology* (Zenodo) (https://doi.org/10.5281/zenodo.7919845).

Iglesias-Molina, A., Van Assche, D., Arenas-Guerrero, J., De Meester, B., Debruyne, C., Jozashoori, S., Maria, P., Michel, F., Chaves-Fraga, D., & Dimou, A. (2023b). The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF. *The Semantic Web – ISWC 2023: 22nd International Semantic Web Conference, Athens, Greece, November 6–10, 2023, Proceedings*.

Jozashoori, S., Chaves-Fraga, D., Iglesias, E., Vidal, M.-E., & Corcho, O. (2020). FunMap: Efficient Execution of Functional Mappings for Knowledge Graph Creation. *The Semantic Web - ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part I*, 276–293.

Jozashoori, S., & Vidal, M.-E. (2019). Mapsdi: A scaled-up semantic data integration framework for knowledge graph creation. *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*, 58–75.

Kaebisch, S., McCool, M., & Korkan, E. (2023). *Web of Things (WoT) Thing Description 1.1* (W3C Recommendation 05 December 2023) (https://www.w3.org/TR/wot-thing-description11/).

Karger, D. R. (2014). The semantic web and end users: What's wrong and how to fix it. *IEEE Internet Computing*, *18*(6), 64–70.

Kärle, E., Şimşek, U., & Fensel, D. (2017). Semantify. it, a platform for creation, publication and distribution of semantic annotations. *arXiv preprint arXiv:1706.10067*.

Kärle, E., Şimşek, U., Panasiuk, O., & Fensel, D. (2018). Building an Ecosystem for the Tyrolean Tourism Knowledge Graph. *Current Trends in Web Engineering: ICWE 2018 International Workshops, MATWEP, EnWot, KD-WEB, WEOD, TourismKG, Cáceres, Spain, June 5, 2018*, 260–267.

Kelley, A., & Garijo, D. (2021). A Framework for Creating Knowledge Graphs of Scientific Software Metadata. *Quantitative Science Studies*, 1–37. https://doi.org/10.1162/qss_a_00167

Kellogg, G., Champin, P.-A., & Longley, D. (2020). *JSON-LD 1.1: A JSON-based Serialization for Linked Data* (W3C Recommendation 16 July 2020) (https://www.w3.org/TR/json-ld11/).

Kieffer, M., Fakih, G., & Serrano Alvarado, P. (2023). Evaluating Reification with Multi-valued Properties in a Knowledge Graph of Licensed Educational Resources. In *Knowledge graphs: Semantics, machine learning, and languages* (pp. 94–109). IOS Press.

Kilicoglu, H., Shin, D., Fiszman, M., Rosemblat, G., & Rindflesch, T. C. (2012). SemMedDB: a PubMed-scale repository of biomedical semantic predications. *Bioinformatics*, *28*(23), 3158–3160. https://doi.org/10.1093/bioinformatics/bts591

Knublauch, H., & Kontokostas, D. (2017). *Shapes Constraint Language (SHACL)* (W3C Recommendation 20 July 2017) (https://www.w3.org/TR/shacl/).

Kyzirakos, K., Savva, D., Vlachopoulos, I., Vasileiou, A., Karalis, N., Koubarakis, M., & Manegold, S. (2018). GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings. *Journal of Web Semantics*, *52-53*, 16–32. https://doi.org/10.1016/j.websem.2018.08.003

Langegger, A. (2009). *XLWrap – Spreadsheet-to-RDF Wrapper* [https://xlwrap.sourceforge.io/].

Langegger, A., & Wöß, W. (2009). XLWrap–querying and integrating arbitrary spreadsheets with SPARQL. *The Semantic Web - ISWC 2009: 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009, Proceedings*, *5823*, 359–374. https://doi.org/10.1007/978-3-642-04930-9_23

Lassila, O., & Swick, R. R. (1999). *Resource Description Framework (RDF) Model and Syntax Specification* (W3C Recommendation 22 February 1999) (https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/).

Le Guillarme, N., & Thuiller, W. (2023). A practical approach to constructing a knowledge graph for soil ecological research. *European Journal of Soil Biology*, *117*, 103497.

Lefrançois, M., Zimmermann, A., & Bakerally, N. (2017). A SPARQL extension for generating RDF from heterogeneous formats. *The Semantic Web: 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 – June 1, 2017, Proceedings, Part I*, 35–50. https://doi.org/10.1007/978-3-319-58068-5_3

Lefrançois, M., Zimmermann, A., Bakerally, N., Khalfi, E. M., & Qawasmeh, O. (2022). *SPARQL-Generate - query and generate both RDF and text* [https://ci.mines-stetienne.fr/sparql-generate/index.html].

Lembo, D., Rosati, R., Ruzzi, M., Savo, D. F., Tocci, E., et al. (2014). Visualization and management of mappings in ontology-based data access (progress report). *Proceedings*

of the 27th International Workshop on Description Logics, DL 2014, Vienna, Austria, July 17–20, 2014, *1193*, 595–607.

Mao, A., Garijo, D., & Fakhraei, S. (2019). SoMEF: A Framework for Capturing Scientific Software Metadata from its Documentation. *2019 IEEE International Conference on Big Data, December 9–12, 2019, Los Angeles, California, USA*, 3032–3037. https://doi.org/10.1109/BigData47090.2019.9006447

Michel, F., Djimenou, L., Faron-Zucker, C., & Montagnat, J. (2015). Translation of Relational and Non-relational Databases into RDF with xR2RML. In V. Monfort, K. Krempels, T. A. Majchrzak, & Z. Turk (Eds.), *WEBIST 2015 - proceedings of the 11th international conference on web information systems and technologies, lisbon, portugal, 20-22 may, 2015* (pp. 443–454). SciTePress. https://doi.org/10.5220/0005448304430454

Michel, F., Djimenou, L., Zucker, C. F., & Montagnat, J. (2015). Translation of relational and non-relational databases into RDF with xR2RML. *Web Information Systems and Technologies: 11th International Conference, WEBIST 2015, Lisbon, Portugal, May 20-22, 2015, Revised Selected Papers*, 443–454. https://doi.org/10.5220/0005448304430454

Michel, F., Djimenou, L., Zucker, C. F., & Montagnat, J. (2017). *xR2RML: Relational and Non-Relational Databases to RDF Mapping Language*. https://www.i3s.unice.fr/~fmichel/xr2rml_specification_v5.html

Michel, F., Gandon, F., Ah-Kane, V., Bobasheva, A., Cabrio, E., Corby, O., Gazzotti, R., Giboin, A., Marro, S., Mayer, T., et al. (2020). Covid-on-the-Web: Knowledge graph and services to advance COVID-19 research. *The Semantic Web – ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part I*, 294–310. https://doi.org/10.1007/978-3-030-62466-8_19

Morsey, M., Lehmann, J., Auer, S., & Ngonga Ngomo, A.-C. (2011). DBpedia SPARQL benchmark–performance assessment with real queries on real data. *The Semantic Web–ISWC 2011: 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I 10*, 454–469.

Neto, L. E. T., Vidal, V. M. P., Casanova, M. A., & Monteiro, J. M. (2013). R2rml by assertion: A semi-automatic tool for generating customised r2rml mappings. *The Semantic Web: ESWC 2013 Satellite Events: ESWC 2013 Satellite Events, Montpellier, France, May 26-30, 2013, Revised Selected Papers 10*, 248–252.

Ngomo, A.-C. N., Auer, S., Lehmann, J., & Zaveri, A. (2014). Introduction to Linked Data and its Lifecycle on the Web. *Reasoning Web. Reasoning on the Web in the Big Data Era: 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings 10*, 1–99.

Nguyen, V., Bodenreider, O., & Sheth, A. (2014). Don't like RDF Reification? Making Statements about Statements Using Singleton Property. *Proceedings of the 23rd International Conference on World Wide Web*, 759–770. https://doi.org/10.1145/2566486.2567973

Noy, N., & Rector, A. (2006). *Defining N-ary Relations on the Semantic Web: Use With Individuals* (W3C Note) (https://www.w3.org/TR/swbp-n-aryRelations/). W3C.

Orlandi, F., Graux, D., & O'Sullivan, D. (2021). Benchmarking RDF Metadata Representations: Reification, Singleton Property and RDF. *2021 IEEE 15th International Conference on Semantic Computing (ICSC), Virtual Event, January 27–29, 2021*, 233–240.

Pellissier Tanon, T. (2023). *Oxigraph* (Zenodo) (https://doi.org/10.5281/zenodo.7749949).

Pellissier Tanon, T., Weikum, G., & Suchanek, F. (2020). Yago 4: A reason-able knowledge base. *The Semantic Web: 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31–June 4, 2020*, 583–596.

Pemberton, J. D., & Robson, A. J. (2000). Spreadsheets in business. *Industrial Management & Data Systems*, *100*(8), 379–388.

Pérez, A. Á., Iglesias-Molina, A., Santamaría, L. P., Poveda-Villalón, M., Badenes-Olmedo, C., & Rodríguez-González, A. (2022). EBOCA: Evidences for Biomedical Concepts Association Ontology. *International Conference on Knowledge Engineering and Knowledge Management (EKAW2022), Bozen-Bolzano, Italy, September 26-29, 2022*, 152–166.

Pérez, J., Arenas, M., & Gutierrez, C. (2009). Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, *34*(3), 1–45. https://doi.org/10.1145/1567274.1567278

Petrova-Antonova, D., & Tancheva, R. (2020). Data cleaning: A case study with openrefine and trifacta wrangler. *Quality of Information and Communications Technology: 13th International Conference, QUATIC 2020, Faro, Portugal, September 9–11, 2020, Proceedings 13*, 32–40.

Piñero, J., Ramírez-Anguita, J. M., Saüch-Pitarch, J., Ronzano, F., Centeno, E., Sanz, F., & Furlong, L. I. (2020). The DisGeNET knowledge platform for disease genomics: 2019 update. *Nucleic Acids Research*, *48*(D1), D845–D855.

Pinkel, C., Binnig, C., Haase, P., Martin, C., Sengupta, K., & Trame, J. (2014). How to best find a partner? an evaluation of editing approaches to construct r2rml mappings. *The Semantic Web: Trends and Challenges: 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings 11*, 675–690.

Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., & Rosati, R. (2008). Linking data to ontologies. In S. Spaccapietra (Ed.), *Journal on data semantics x* (pp. 133–173). Springer. https://doi.org/10.1007/978-3-540-77688-8_5

Polleres, A., Krennwallner, T., Lopes, N., Kopecky, J., & Decker, S. (2009). *XSPARQL Language Specification* [https://www.w3.org/Submission/xsparql-language-specification/].

Poveda-Villalón, M., Fernández-Izquierdo, A., Fernández-López, M., & García-Castro, R. (2022). LOT: An Industrial Oriented Ontology Engineering Framework. *Engineering Applications of Artificial Intelligence*, *111*, 104755. https://doi.org/10.1016/j.engappai.2022.104755

Poveda-Villalón, M., Gómez-Pérez, A., & Suárez-Figueroa, M. C. (2014). Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation. *International Journal on Semantic Web and Information Systems (IJSWIS)*, *10*(2), 7–34. https://doi.org/10.4018/ijswis.2014040102

Priyatna, F., Corcho, O., & Sequeda, J. (2014). Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph. *WWW '14: Proceedings of the 23rd International Conference on World Wide Web, Seoul, Korea, April 7–11, 2014*, 479–490. https://doi.org/10.1145/2566486.2567981

Prud'hommeaux, E., Labra Gayo, J. E., & Solbrig, H. (2014). Shape expressions: an RDF validation and transformation language. *SEM '14: Proceedings of the 10th International Conference on Semantic Systems, Leipzig, Germany, September 4–5, 2014*, 32–40. https://doi.org/10.1145/2660517.2660523

Prud'hommeaux, E., & Seaborne, A. (2008). *SPARQL Query Language for RDF* (W3C Recommendation 15 January 2008) (https://www.w3.org/TR/rdf-sparql-query/).

Queralt-Rosinach, N., Kuhn, T., Chichester, C., Dumontier, M., Sanz, F., & Furlong, L. I. (2016). Publishing DisGeNET as Nanopublications. *Semantic Web*, *7*(5), 519–528.

Radulovic, F., Poveda-Villalón, M., Vila-Suero, D., Rodríguez-Doncel, V., García-Castro, R., & Gómez-Pérez, A. (2015). Guidelines for Linked Data generation and publication: An example in building energy consumption. *Automation in Construction*, *57*, 178–187.

Randles, A., & O'Sullivan, D. (2024). R2[RML]-ChatGPT Framework. *Proceedings of the 5th International Workshop on Knowledge Graph Construction co-located with 21th Extended Semantic Web Conference (ESWC 2024), Herssonissos, May 26–30, 2024*.

Ren, H., Dai, H., Dai, B., Chen, X., Zhou, D., Leskovec, J., & Schuurmans, D. (2022). Smore: Knowledge graph completion and multi-hop reasoning in massive knowledge graphs. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14–18, 2022*, 1472–1482.

Rodríguez-Muro, M., & Rezk, M. (2015). Efficient sparql-to-sql with r2rml mappings. *Journal of Web Semantics*, *33*, 141–169. https://doi.org/10.1016/j.websem.2015.03.001

Rojas, J. A., Aguado, M., Vasilopoulou, P., Velitchkov, I., Van Assche, D., Colpaert, P., & Verborgh, R. (2021). Leveraging Semantic Technologies for Digital Interoperability in the European Railway Domain. *The Semantic Web – ISWC 2021: 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24–28, 2021, Proceedings*, 648–664.

Ryen, V., Soylu, A., & Roman, D. (2022). Building semantic knowledge graphs from (semi-) structured data: A review. *Future Internet*, *14*(5), 129.

Schmidt, M., Hornung, T., Lausen, G., & Pinkel, C. (2009). SP^{2}Bench: a SPARQL performance benchmark. *IEEE 25th International Conference on Data Engineering, Shanghai, China, March 29 – April 2, 2009*, 222–233.

Scrocca, M., Carenini, A., Grassi, M., Comerio, M., & Celino, I. (2024). Not Everybody Speaks RDF: Knowledge Conversion between Different Data Representations. *Proceedings of the 5th International Workshop on Knowledge Graph Construction co-located with 21th Extended Semantic Web Conference (ESWC 2024), Herssonissos, May 26–30, 2024*.

Sengupta, K., Haase, P., Schmidt, M., & Hitzler, P. (2013). Editing R2RML mappings made easy. *Proceedings of the ISWC 2013 Posters & Demonstrations Track a Track Within the 12th International Semantic Web Conference, Sydney, Australia, October 23, 2013*, *1035*.

Sicilia, Á., Nemirovski, G., & Nolle, A. (2017). Map-On: A web-based editor for visual ontology mapping. *Semantic Web*, *8*(6), 969–980.

Simsek, U., Angele, K., Kärle, E., Opdenplatz, J., Sommer, D., Umbrich, J., & Fensel, D. (2021). Knowledge Graph Lifecycle: Building and Maintaining Knowledge Graphs. *Proceedings of the 2nd International Workshop on Knowledge Graph Construction co-located with 18th Extended Semantic Web Conference (ESWC 2021), Virtual Event, June 6–10, 2021*, *2873*.

Slepicka, J., Yin, C., Szekely, P. A., & Knoblock, C. A. (2015). KR2RML: An Alternative Interpretation of R2RML for Heterogenous Sources. *Proceedings of the 6th International Workshop on Consuming Linked Data co-located with 14th International Semantic Web Conference (ISWC 2105), Bethlehem, Pennsylvania, USA, October 12th, 2015*, *1426*.

Soylu, A., Corcho, O., Elvesæter, B., Badenes-Olmedo, C., Blount, T., Yedro Martínez, F., Kovacic, M., Posinkovic, M., Makgill, I., Taggart, C., et al. (2022). TheyBuyForYou Platform and Knowledge Graph: Expanding Horizons in Public Procurement with Open Linked Data. *Semantic Web*, *13*(2), 265–291.

SPARQL-Anything. (2023). *SPARQL-Anything* [https://sparql-anything.readthedocs.io/en/latest/].

Stadler, C., Bühmann, L., Meyer, L.-P., & Martin, M. (2023). Scaling rml and sparql-based knowledge graph construction with apache spark. *Proceedings of the 4th International Workshop on Knowledge Graph Construction co-located with 20th Extended Semantic Web Conference (ESWC 2023), Hersonissos, Greece, May 28 – June 1, 2023*, *3471*.

Stadler, C., Unbehauen, J., Westphal, P., Sherif, M. A., & Lehmann, J. (2015). Simplified RDB2RDF mapping. *Proceedings of the Workshop on Linked Data on the Web co-located with the 24th International World Wide Web Conference (LDOW@WWW 2015), Florence, Italy, May 18–22, 2015*, *1409*.

Stardog. (2021). *SMS2 (Stardog Mapping Syntax 2)* [https://docs.stardog.com/archive/7.5.0/virtual-graphs/mapping-data-sources.html].

Stocker, M., Oelen, A., Jaradeh, M. Y., Haris, M., Oghli, O. A., Heidari, G., Hussein, H., Lorenz, A.-L., Kabenamualu, S., Farfar, K. E., et al. (2023). FAIR scientific information with the Open Research Knowledge Graph. *FAIR Connect*, *1*(1), 19–21.

Suárez-Figueroa, M. C., Gómez-Pérez, A., & Fernandez-Lopez, M. (2015). The NeOn Methodology framework: A scenario-based methodology for ontology development. *Applied ontology*, *10*(2), 107–145. https://doi.org/10.3233/AO-150145

Sundqvist, L. (2022). *Extending VKG Systems with RDF-star Support* (Master thesis, Free University of Bozen-Bolzano) (https://ontop-vkg.org/publications/2022-sundqvist-rdf-star-ontop-msc-thesis.pdf).

Szekely, P., Garijo, D., Bhatia, D., Wu, J., Yao, Y., & Pujara, J. (2019). T2WML: Table To Wikidata Mapping Language. *K-CAP'19: Proceedings of the 10th International Conference on Knowledge Capture, Marina del Rey, CA, USA, November 19–21, 2019*, 267–270. https://doi.org/10.1145/3360901.3364448

Tamašauskaitė, G., & Groth, P. (2023). Defining a Knowledge Graph Development Process Through a Systematic Review. *ACM Transactions on Software Engineering and Methodology*, *32*(1), 1–40.

TARQL. (2019). *TARQL: SPARQL for Tables* [http://tarql.github.io/].

Tennison, J., Kellogg, G., & Herman, I. (2015). *Model for Tabular Data and Metadata on the Web* (W3C Recommendation 17 December 2015) (https://www.w3.org/TR/tabular-data-model/).

Toledo, J., Iglesias-Molina, A., Chaves-Fraga, D., & Garijo, D. (2024). RMLdoc: Documenting Mapping Rules for Knowledge Graph Construction. *ESWC 2024 Posters and Demos Track*.

Van Assche, D., Delva, T., Haesendonck, G., Heyvaert, P., De Meester, B., & Dimou, A. (2023). Declarative RDF graph generation from heterogeneous (semi-)structured data: A Systematic Literature Review. *Journal of Web Semantics*, *75*, 100753.

Van Assche, D., Haesendonck, G., De Mulder, G., Delva, T., Heyvaert, P., De Meester, B., & Dimou, A. (2021). Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation. *Web Engineering, 21st International Conference, ICWE 2021,*

Biarritz, France, May 18–21, 2021, 337–352. https://doi.org/10.1007/978-3-030-74296-6_26

Van Assche, D., Iglesias-Molina, A., Dimou, A., De Meester, B., Chaves-Fraga, D., & Maria, P. (2023). *RML-Core Ontology: Generic Mapping Language for RDF* (Zenodo) (https://doi.org/10.5281/zenodo.7919848).

Van Assche, D., Iglesias-Molina, A., & Haesendonck, G. (2023). *RML-IO Ontology: Source and Target* (Zenodo) (https://doi.org/10.5281/zenodo.7919850).

Vidal, M.-E., Sakor, A., Jozashoori, S., Niazmand, E., Purohit, D., Iglesias, E., Aisopos, F., Vogiatzis, D., Menasalvas, E., Gonzalez, A. R., et al. (2023). Knowledge Graphs for Enhancing Transparency in Health Data Ecosystems. *Semantic Web*, *14*(5), 943–976.

Vila-Suero, D., Villazón-Terrazas, B., & Gómez-Pérez, A. (2013). datos.bne.es: A library linked dataset. *Semantic Web*, *4*(3), 307–313.

Villazón-Terrazas, B., & Hausenblas, M. (2012). *R2RML and Direct Mapping Test Cases* (W3C Working Group Note 14 August 2012) (http://www.w3.org/TR/rdb2rdf-test-cases/).

Vrandečić, D., & Krötzsch, M. (2014). Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, *57*(10), 78–85.

Vu, B., Pujara, J., & Knoblock, C. A. (2019). D-REPR: A Language for Describing and Mapping Diversely-Structured Data Sources to RDF. *K-CAP '19: Proceedings of the 10th International Conference on Knowledge Capture, Marina del Rey, CA, USA, November 19–21, 2019*, 189–196. https://doi.org/10.1145/3360901.3364449

Xiao, G., Lanti, D., Kontchakov, R., Komla-Ebri, S., Güzel-Kalaycı, E., Ding, L., Corman, J., Cogrel, B., Calvanese, D., & Botoeva, E. (2020). The virtual knowledge graph system ontop. *International Semantic Web Conference*, 259–277.

Zazuco. (2022). *Expressive RDF Mapper (XRM)* [https://zazuko.com/products/expressive-rdf-mapper/].

Annexes

A Conceptual Mapping Requirements

Table A.1: Requirements identified for declarative knowledge graph construction.

Identifier	Requirement
cm-r1	A mapping can describe data sources retrieved synchronously, asynchronously and as streams
cm-r2	A data source can describe data in different formats specifying its Media Type
cm-r3	A data source may have a specified data access service
cm-r4	A data access service can specify security terms
cm-r5	A data access service can specify up to one retrieval protocol
cm-r6	A data source can specify the encoding
cm-r7	A mapping can describe data sources and their access
cm-r8	A mapping specifies statement maps to declare the transformation of frames into RDF triples
cm-r9	Multiple data sources can be described with combined frames and nested frames
cm-r10	A source frame describes exactly one data source
cm-r11	A statement map can describe constant and dynamic subjects, predicates and objects
cm-r12	A statement map can describe constant and dynamically up to 1 named graph, datatype and language tag
cm-r13	A resource map can reference one or more data fields from one or more data sources
cm-r14	A resource map may contain data with different levels of iteration of a source
cm-r15	A resource map may contain functions
cm-r16	A subject can only be expressed as a blank node or an IRI
cm-r17	A predicate can only be expressed as an IRI
cm-r18	An object can only be expressed as a blank node, an IRI, a literal, a container or a list
cm-r19	A named graph can only be expressed as an IRI or blank node
cm-r20	A datatype tag can only be expressed as an IRI or a List
cm-r21	A language tag can only be expressed as a Literal or a List
cm-r22	A statement map can be subject to a condition
cm-r23	A statement map can be linked to another statement map with none, one or several conditions
cm-r24	The condition to link statements may be any boolean condition
cm-r25	A function may have nested functions
cm-r26	A frame can have nested frames to access multi-value references
cm-r27	A statement map can have objects of type literal using data from different sources using combined frames
cm-r28	A statement map can have datatype and language tags using data from different sources using combined frames

B Mapeathor Mappings

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rml: <https://w3id.org/rml/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix ex: <http://ex.com/>.
@prefix grel: <http://semweb.datasciencelab.be/ns/grel#>.
@base <http://example.com/>.

<#PERSON>
    a rml:TriplesMap;
    rml:logicalSource [
      rml:source "/home/user/data/people.csv";
      rml:referenceFormulation rml:CSV;
    ];
    rml:subjectMap [
      a rml:Subject ;
      rml:termType rml:IRI ;
      rml:template "http://ex.com/Person/{name}" ;
      rml:class ex:Person ;
      rml:class ex:Athlete ;
    ];
    rml:predicateObjectMap [
      rml:predicateMap  [ rml:constant ex:name];
      rml:objectMap [ rml:reference "name";
        rml:termType rml:Literal;
        rml:datatype xsd:string;
        rml:language "en" ]
    ];
    rml:predicateObjectMap [
      rml:predicateMap  [ rml:constant ex:plays ];
      rml:objectMap [
        rml:termType rml:IRI ;
        rml:parentTriplesMap  <#SPORT> ;
        rml:joinCondition [
          rml:child "sport_id" ;
          rml:parent  "id" ;
        ];
      ];
    ];
    rml:predicateObjectMap [
      rml:predicateMap  [ rml:constant ex:birthdate] ;
      rml:objectMap [
        rml:functionExecution <#Fun-date> ;
        rml:return grel:dateOut ;
      ];
    ].

```

```
<#SPORT>
    a rml:TriplesMap;
    rml:logicalSource [
      rml:source "/home/user/data/sports.json";
      rml:referenceFormulation rml:JSONPath;
        rml:iterator "$.*";
    ];
    rml:subjectMap [
      a rml:Subject ;
      rml:termType rml:IRI ;
      rml:template "http://ex.com/Sport/{sport}" ;
      rml:class ex:Sport ;
      rml:graphMap [ rml:constant "ex:SportsGraph" ] ;
    ];
    rml:predicateObjectMap [
      rml:predicateMap  [ rml:constant ex:name];
      rml:objectMap [ rml:reference "sport";
        rml:termType rml:Literal;
        rml:datatype xsd:string;
        rml:language "en" ]
    ];
    rml:predicateObjectMap [
      rml:predicateMap  [ rml:constant ex:code];
      rml:objectMap [ rml:reference "id";
        rml:termType rml:Literal;
        rml:datatype xsd:integer ]
    ].

<#Fun-date> a rml:FunctionExecution;
    rml:function grel:toDate ;
    rml:input [
      a rml:Input ;
      rml:parameter grel:valueParam1 ;
      rml:inputValueMap [ rml:reference "birthdate" ];
    ];
    rml:input [
      a rml:Input ;
      rml:parameter grel:valueParam2 ;
      rml:inputValueMap [ rml:constant "dd/MM/yyyy" ];
    ];
    rml:input [
      a rml:Input ;
      rml:parameter grel:valueParam3 ;
      rml:inputValueMap [ rml:constant "yyyy-MM-dd" ];
    ].
```

Listing B.1: RML mapping (2023 release) generated with Mapeathor from the spreadsheet

shown in Tables 5.1, 5.2, 5.3, 5.4 and 5.5.

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#>.
@prefix ql: <http://semweb.mmlab.be/ns/ql#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix ex: <http://ex.com/>.
@prefix grel: <http://semweb.datasciencelab.be/ns/grel#>.
@base <http://example.com/>.

<#PERSON>
    a rr:TriplesMap;
    rml:logicalSource [
      rml:source "/home/user/data/people.csv";
      rml:referenceFormulation ql:CSV;
    ];
    rr:subjectMap [
      a rr:Subject ;
      rr:termType rr:IRI ;
      rr:template "http://ex.com/Person/{name}" ;
      rr:class ex:Person ;
      rr:class ex:Athlete ;
    ];
    rr:predicateObjectMap [
      rr:predicateMap [ rr:constant ex:name];
      rr:objectMap  [ rml:reference "name"; rr:termType rr:Literal; rr:datatype
          xsd:string; rr:language "en" ]
    ];
    rr:predicateObjectMap [
      rr:predicateMap [ rr:constant ex:plays ];
      rr:objectMap  [
        rr:termType rml:IRI ;
        rr:parentTriplesMap <#SPORT>;
        rr:joinCondition  [
          rr:child  "sport_id";
          rr:parent "id";
        ];
      ];
    ];
    rr:predicateObjectMap [
      rr:predicateMap [ rr:constant ex:birthdate] ;
      rr:objectMap  [
        rml:functionExecution <#Fun-date> ;
        rml:return grel:dateOut ;
      ];
    ].
```

```
45
46 <#SPORT>
47     a rr:TriplesMap;
48     rml:logicalSource [
49       rml:source "/home/user/data/sports.json";
50       rml:referenceFormulation ql:JSONPath;
51         rml:iterator "$.*";
52     ];
53     rr:subjectMap [
54       a rr:Subject ;
55       rr:termType rr:IRI ;
56       rr:template "http://ex.com/Sport/{sport}" ;
57       rr:class ex:Sport ;
58       rr:graphMap [ rr:constant "ex:SportsGraph" ] ;
59     ];
60     rr:predicateObjectMap [
61       rr:predicateMap [ rr:constant ex:name];
62       rr:objectMap  [ rml:reference "sport"; rr:termType rr:Literal; rr:datatype
             xsd:string; rr:language "en" ]
63     ];
64     rr:predicateObjectMap [
65       rr:predicateMap [ rr:constant ex:code];
66       rr:objectMap  [ rml:reference "id"; rr:termType rr:Literal; rr:datatype xsd:
             integer ]
67     ].
68
69 <#Fun-date> a rml:FunctionExecution;
70     rml:function grel:toDate ;
71     rml:input [
72       a rml:Input ;
73       rml:parameter grel:valueParam1 ;
74       rml:inputValueMap [ rml:reference "birthdate" ];
75     ];
76     rml:input [
77       a rml:Input ;
78       rml:parameter grel:valueParam2 ;
79       rml:inputValueMap [ rr:constant "dd/MM/yyyy" ];
80     ];
81     rml:input [
82       a rml:Input ;
83       rml:parameter grel:valueParam3 ;
84       rml:inputValueMap [ rr:constant "yyyy-MM-dd" ];
85     ].
```

Listing B.2: RML mapping (2014 release) generated with Mapeathor from the spreadsheet shown in Tables 5.1, 5.2, 5.3, 5.4 and 5.5.

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix ql: <http://semweb.mmlab.be/ns/ql#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix ex: <http://ex.com/>.
@prefix grel: <http://semweb.datasciencelab.be/ns/grel#>.
@base <http://example.com/>.

<#PERSON>
    a rr:TriplesMap;
    rr:logicalTable [
      rr:tableName "PEOPLE"
    ];
    rr:subjectMap [
      a rr:Subject ;
      rr:termType rr:IRI ;
      rr:template "http://ex.com/Person/{name}" ;
      rr:class ex:Person ;
      rr:class ex:Athlete ;
    ];
    rr:predicateObjectMap [
      rr:predicateMap [ rr:constant ex:name];
      rr:objectMap  [ rr:column "name"; rr:termType rr:Literal; rr:datatype xsd:
          string; rr:language "en" ]
    ];
    rr:predicateObjectMap [
        rr:predicateMap [ rr:constant ex:plays ];
        rr:objectMap  [
            rr:parentTriplesMap <#SPORT>;
            rr:joinCondition  [
                rr:child  "sport_id";
                rr:parent "id";
            ];
        ];
    ];
.

<#SPORT>
    a rr:TriplesMap;
    rr:logicalTable [
      rr:tableName "SPORTS"
    ];
    rr:subjectMap [
      a rr:Subject ;
      rr:termType rr:IRI ;
      rr:template "http://ex.com/Sport/{sport}" ;
      rr:class ex:Sport ;
      rr:graphMap [ rr:constant "ex:SportsGraph" ] ;
```

```
48      ];
49      rr:predicateObjectMap [
50        rr:predicateMap [ rr:constant ex:name];
51        rr:objectMap  [ rr:column "sport"; rr:termType rr:Literal; rr:datatype xsd:
              string; rr:language "en" ]
52      ];
53      rr:predicateObjectMap [
54        rr:predicateMap [ rr:constant ex:code];
55        rr:objectMap  [ rr:column "id"; rr:termType rr:Literal; rr:datatype xsd:
              integer ]
56      ];
57  .
```

Listing B.3: R2RML mapping generated with Mapeathor from the spreadsheet shown in Tables 5.1, 5.2, 5.3, 5.4 and 5.5.

```
1  base: "http://example.com/"
2  prefixes:
3    rdf: "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4    ex: "http://ex.com/"
5    grel: "http://semweb.datasciencelab.be/ns/grel#"
6
7  mappings:
8    PERSON:
9      sources:
10       - [/home/user/data/people.csv~csv]
11     subjects: http://ex.com/Person/$(name)
12     graph: nan
13     po:
14       - [a, ex:Person]
15       - [a, ex:Athlete]
16       - [ex:name, $(name), xsd:string, en~lang]
17       - p: ex:plays
18         o:
19           - mapping: SPORT
20             condition:
21               function: equal
22               parameters:
23                 - [str1, $(sport_id)]
24                 - [str2, $(id)]
25
26   SPORT:
27     sources:
28       - [/home/user/data/sports.json~jsonpath, '$.*']
29     subjects: http://ex.com/Sport/$(sport)
30     graph: ex:SportsGraph
31     po:
```

```
32        - [a, ex:Sport]
33        - [ex:name, $(sport), xsd:string, en~lang]
34        - [ex:code, $(id), xsd:integer]
```

Listing B.4: YARRRML mapping generated with Mapeathor from the spreadsheet shown in Tables 5.1, 5.2, 5.3, 5.4 and 5.5.

C Supplementary Tables for KG Refactoring Evaluation

Table C.1: Execution time of KG construction engines with the declarative mappings. Values corresponding to Fig. 6.2.

Size	Target	Morph-KGC	SPARQL-Anything
1K	Std. Reif.	1.424	1.439
	RDF-star	1.180	1.370
	Named Graphs	1.134	1.391
	N-Ary Rel.	1.233	1.373
10K	Std. Reif.	2.324	3.631
	RDF-star	1.736	3.105
	Named Graphs	1.550	3.598
	N-Ary Rel.	1.753	3.507
100K	Std. Reif.	12.291	22.162
	RDF-star	8.121	22.205
	Named Graphs	6.286	21.913
	N-Ary Rel.	8.070	23.386
1M	Std. Reif.	121.845	234.606
	RDF-star	78.476	214.874
	Named Graphs	57.861	214.738
	N-Ary Rel.	76.821	238.764

Table C.2: Execution time of triplestores with the SPARQL queries that perform translations for pairs of representations. Values corresponding to Fig. 6.3.

Size	Source	Target	Query	Oxigraph	Fuseki	GraphDB
1K	N-Ary Rel.	Std. Reif.	Q1	0.531	0.140	0.233
		RDF-Star	Q2	0.491	0.099	0.131
		Named Graphs	Q3	-	0.133	-
	Std. Reif.	RDF-Star	Q4	0.632	0.133	0.131
		N-Ary Rel.	Q5	1.116	0.333	0.275
		Named Graphs	Q6	-	0.130	-
	RDF-Star	Std. Reif.	Q7	0.577	0.150	0.254
		N-Ary Rel.	Q8	0.481	0.183	0.253
		Named Graphs	Q9	-	0.117	-
	Named Graphs	Std. Reif.	Q10	0.477	0.124	0.252
		RDF-Star	Q11	0.413	0.121	0.142
		N-Ary Rel.	Q12	0.868	0.231	0.254
10K	N-Ary Rel.	Std. Reif.	Q1	4.172	0.931	2.007
		RDF-Star	Q2	3.687	0.659	0.927
		Named Graphs	Q3	-	0.995	-
	Std. Reif.	RDF-Star	Q4	5.144	0.758	0.944
		N-Ary Rel.	Q5	55.225	14.946	9.186
		Named Graphs	Q6	-	0.729	-
	RDF-Star	Std. Reif.	Q7	4.250	0.992	1.986
		N-Ary Rel.	Q8	3.151	1.002	1.328
		Named Graphs	Q9	-	0.806	-
	Named Graphs	Std. Reif.	Q10	3.879	0.990	2.088
		RDF-Star	Q11	2.851	0.690	1.011
		N-Ary Rel.	Q12	52.019	9.311	7.749
100K	N-Ary Rel.	Std. Reif.	Q1	43.932	12.347	20.366
		RDF-Star	Q2	39.344	7.392	9.021
		Named Graphs	Q3	-	9.424	-
	Std. Reif.	RDF-Star	Q4	48.300	8.286	8.949
		N-Ary Rel.	Q5	Out-of-Memory	1408.531	844.092
		Named Graphs	Q6	-	7.731	-
	RDF-Star	Std. Reif.	Q7	39.500	10.953	18.791
		N-Ary Rel.	Q8	28.758	11.145	11.862
		Named Graphs	Q9	-	8.500	-
	Named Graphs	Std. Reif.	Q10	36.160	12.108	22.210
		RDF-Star	Q11	29.420	8.094	9.635
		N-Ary Rel.	Q12	Out-of-Memory	890.306	676.530
1M	N-Ary Rel.	Std. Reif.	Q1	606.011	185.208	216.501
		RDF-Star	Q2	533.988	99.838	91.876
		Named Graphs	Q3	-	129.621	-
	Std. Reif.	RDF-Star	Q4	622.131	115.130	85.698
		N-Ary Rel.	Q5	Out-of-Memory	Timeout	Timeout
		Named Graphs	Q6	-	142.600	-
	RDF-Star	Std. Reif.	Q7	388.227	237.749	181.685
		N-Ary Rel.	Q8	285.841	223.663	120.688
		Named Graphs	Q9	-	216.149	-
	Named Graphs	Std. Reif.	Q10	443.620	101.384	227.556
		RDF-Star	Q11	353.570	197.742	96.191
		N-Ary Rel.	Q12	Out-of-Memory	Timeout	Timeout