Effectiviteitsverbetering van de creatie en uitvoering van kennisgraafgeneratieregels

Improving Effectiveness of Knowledge Graph Generation Rule Creation and Execution

Pieter Heyvaert

Promotoren: prof. dr. ir. R. Verborgh, dr. A. Dimou
Proefschrift ingediend tot het behalen van de graad van
Doctor in de ingenieurswetenschappen: computerwetenschappen

UNIVERSITEIT
GENT

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. K. De Bosschere
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2019 - 2020

ISBN 978-94-6355-289-9
NUR 988
Wettelijk depot: D/2019/10.500/97

Examination Board

Dr. Anastasia Dimou (PhD advisor)

Prof. Ruben Verborgh (PhD advisor)

Prof. Filip De Turck (Chair)

Prof. Steven Verstockt (Secretary)

Prof. Erik Mannens

Prof. Oscar Corcho

Dr. Steffen Lohmann

Dr. Mariano Rodriguez-Muro

Contents

Summary

More and more data is generated by an increasing number of agents (applications and devices) that are connected to the Internet, all contributing to data that is available on the Web. When this data is analyzed, combined, and shared powerful, new techniques can be designed and deployed, such as artificial intelligence applied by personal assistants (e.g., Apple's Siri and Amazon Alexa), improved search engines (e.g., Google Search and Microsoft's Bing), and decentralized data storages (as opposed to a large, central storage used by big companies).

Due to this increase in data, methods originally used to model data have become insufficient: they isolate the data, which resides in different data sources, and make it hard for agents to exchange data on the Web. Data cannot be easily exchanged between agents, because every agent uses a different data model to describe the concepts and relationships of the data they generate and use.

The Semantic Web offers a solution to this problem: knowledge graphs, which are directed graph that provides semantic descriptions of entities and their relationships. These graphs are often generated from other data sources, such as multiple databases and files that each can be organized and structured in different ways. For instance, the DBpedia knowledge graph is generated from Wikipedia, and the Google knowledge graph is generated from different sources, including data coming from Google's own services and Wikipedia.

A common way to generate these knowledge graphs is by using rules, which are created and executed. The rules attach semantic annotations, using concepts and relationships, to data in those sources. Which concepts and relationships are used and how they are applied to data sources is contained in a semantic model. Rules thereby determine how data sources are modeled using specific concepts and relationships during knowledge graph generation. The syntax and grammar of these rules are determined by a knowledge graph generation language, such as R2RML and RML. Thus, the rule creation is influenced by three aspects: (i) data sources, (ii) concepts and relationships, and (iii) semantic model.

Once (a subset of) the rules are created, they are executed to generate a knowledge graph. This is done via a processor: a software tool that, given a set of rules and data sources, generates knowledge graphs. Multiple processors can be developed for a single knowledge graph generation language with each a different set of features:

conformance to the specification of the rule language, API, programming language, scalability, and so on. Users can switch between them without changing the rules, and thus, the selection of the most suitable processor depends on the use case at hand.

In this dissertation, we look at three challenges and discuss our contributions to tackle them. The first challenge deals with the improvement of users' understanding of a knowledge graph creation's components. The second challenge deals with the avoidance and removal of inconsistencies introduced by concepts, relationships, and semantic model, which influences the rule creation. Both can lead to inconsistencies in the generated knowledge graphs. Inconsistencies in knowledge graphs are introduced when concepts and relationships are used without adhering to their restrictions, and this affects the graphs' quality. Possible root causes for these inconsistencies include: (i) semantic model that introduce new inconsistencies by, for example, not using the suitable concepts; and (ii) concept and relationship definitions that do not model the domain as desired. The third challenge deals with the selection of the most suitable processor for the use case at hand, which influences rule execution. A processor is used to generate knowledge graphs. If multiple processors are available, users need to select the most suitable one for the use case at hand. However, this is not trivial if each processor has a different set of features, such as conformance to the specification, API, programming language, scalability, and so on.

Our contributions to tackle the challenges are MapVOWL, the RMLEditor, Resglass, and the RML test cases. MapVOWL is a visual notation for knowledge graph generation rules. Relying on MapVOWL's unified graphical elements, rules can be created entirely using visual representations, while the rules in the underlying language's syntax and grammar are generated without user intervention. Our evaluation shows that MapVOWL is preferred over using a knowledge graph generation language, such as RML, directly. MapVOWL contributes to tackling the first and second challenge. In future research we want investigate how to visualize common data structures, such as lists and bags. Furthermore, the representation of these rules should not necessarily only happen via visualizations: there are users who desire a text-based approach. To this end, initial steps have been taken by introducing YARRRML, a human-readable text-based representation for knowledge graph generation rules.

The RMLEditor is a graphical user interface (GUI) for knowledge graph generation rules, using MapVOWL. It allows users to load data sources, create rules, and preview the resulting knowledge graphs. The GUI pays special attention to (i) the scalability issues that accompany the use of graphs for visualizations; (ii) the visualization of heterogeneous data sources, besides showing the raw data; and (iii) the integration of transformations of the data sources in the visualizations of the rules. Our evaluation shows that the RMLEditor is rated good by users and is preferred over using a form-based GUI, such as RMLx. The RMLEditor contributes to tackling the first and second challenge. In future research we want to combine MapVOWL with other representations, such as YARRRML, and add new features to improve the overall rule creation process, such importing ontologies.

Resglass is a rule-driven method for the resolution of inconsistencies in rules,

concepts, and relationships. The rules, concepts, and relationships are automatically ranked in order of inspection based on a score that considers the number of inconsistencies a rule, concept, or relationship is involved in. Our evaluation shows that Resglass' ranking improves the overlap with experts' with 41% for rules and 23% for concepts and relationships, compared to a random ranking. Resglass contributes to the second challenge. In future research, we want to investigate how to further assist users in both detecting and resolving inconsistencies, such as, for example, suggesting possible resolutions based on the used and other ontologies, and rules of other use cases.

The RML test cases allow (i) developers to determine how conformant their RML processors are to the RML specification, and (ii) users can use the test cases results to select the most appropriate processor for a specific use case. This is done towards designing test cases that are independent of RML and are applicable to all languages that generate knowledge graphs from heterogeneous data sources. The test cases contribute to the third challenge. In future research, we want to investigate the characteristics of test cases where data streams are used instead of static files or databases, and other metrics that further improve the selection of the most suitable processor, such as the speed, memory consumption, usability, and extensibility.

Samenvatting

Meer en meer data wordt gegenereerd door een steeds groter aantal agenten (applicaties en apparaten) die met het Internet verboden zijn en allemaal bijdragen tot de data die beschikbaar is op het Web. Wanneer deze data geanalyseerd, gecombineerd, en gedeeld wordt, dan kunnen krachtige en nieuwe technieken ontworpen en uitgerold worden, zoals artificiële intelligentie toegepast door persoonlijke assistenten (b.v., Apple's Siri en Amazon Alexa), verbeterde zoekmachines (b.v., Google Search en Microsoft's Bing), en gedecentraliseerde dataopslag (in tegenstelling tot grote, centrale opslag gebruikt door grote bedrijven).

Door deze toename aan data zijn methoden die origineel gebruikt werden om data te modelleren inefficiënt: ze isoleren de data, welke zich in verschillende databronnen bevindt, en maken het moeilijk voor agenten om data uit te wisselen op het Web. De data kan niet gemakkelijk uitgewisseld worden tussen agenten, omdat elke agent een verschillend data model gebruikt om de concepten en relaties te beschrijven van de data die het genereert en gebruikt.

Het Semantische Web biedt een oplossing voor dit probleem: kennisgrafen, welke gerichte grafen zijn die semantische beschrijvingen voorzien voor entiteiten en hun relaties. Deze grafen worden vaak gegenereerd van andere databronnen, zoals meerdere databanken en bestanden die elk georganiseerd en gestructureerd worden op een verschillende manier. Bijvoorbeeld, de DBpedia kennisgraaf wordt gegeneerd van Wikipedia, en de Google kennisgraaf wordt genereerd van verschillende bronnen, inclusief data komend van de Google's eigen diensten en Wikipedia.

Een gebruikelijke manier om deze kennisgrafen te genereren is door gebruik te maken van regels, welke gecreëerd en uitgevoerd worden. De regels voegen semantische annotaties toe aan de data in deze bronnen, gebruikmakend van concepten en relaties. Welke concepten en relaties gebruikt worden en hoe ze toegepast worden op de databronnen is bevat in een semantisch model. Regels bepalen, op de manier, hoe databronnen gemodelleerd worden gebruikmakend van specifieke concepten en relaties tijdens de kennisgraafgeneratie. De syntaxis en grammatica van deze regels wordt bepaald door de kennisgraafgeneratietaal, zoals R2RML en RML. Dus, de regelcreatie wordt beïnvloed door drie aspecten: (i) databronnen, (ii) concepten en relaties, en (iii) semantisch model.

Eenmaal (een deel van) de regels gecreëerd zijn, worden ze uitgevoerd om een

kennisgraaf te genereren. Dit gebeurt via een processor: een software tool dat, gebruikmakend van een reeks regels en databronnen, kennisgrafen genereert. Meerdere processoren kunnen ontwikkelend worden voor een één kennisgraafgeneratietaal met elke een verschillende reeks van eigenschappen: conformiteit aan de specificatie van de regeltaal, API, programmeertaal, schaalbaarheid, en zo verder. Gebruikers kunnen wisselen tussen deze processoren zonder de regels aan te passen, en dus, de selectie van de meest geschikte processor hangt af van een specifieke toepassing.

In dit proefschrift kijken we naar drie uitdagingen en bediscussiëren we onze bijdragen om deze aan te pakken. De eerste uitdaging behandelt de verbetering van gebruikers' begrip van de componenten van de kennisgraafgeneratie. De tweede uitdaging behandelt het vermijden en verwijderen van inconsistenties geïntroduceerd door concepten en relaties, en het semantisch model, welke de regelcreatie beïnvloeden. Beide kunnen leiden tot inconsistenties in de gegeneerde kennisgrafen. Inconsistenties in kennisgrafen worden geïntroduceerd wanneer concepten en relaties gebruikt worden zonder rekening te houden met hun restricties, en dit tast de graaf's kwaliteit aan. Mogelijke oorzaken voor deze inconsistenties zijn: (i) het semantisch model introduceert nieuwe inconsistenties door, b.v., niet gebruik te maken van de geschikte concepten; en (ii) de definities van concepten en relaties modelleren niet het domein zoals gewenst. De derde uitdaging behandelt de selectie van de meest geschikte processor voor een specifieke toepassing, welke de regeluitvoering beïnvloedt. Een processor wordt gebruikt om kennisgrafen te genereren. Als meerdere processoren beschikbaar zijn, moeten gebruikers de meeste geschikte kiezen voor hun specifieke toepassingen. Echter, dit is niet triviaal omdat elke processor een verschillende reeks van features heeft, zoals conformiteit aan de specificatie, API, programmeertaal, schaalbaarheid, en zo verder.

Onze bijdragen die deze uitdagingen aanpakken zijn MapVOWL, de RMLEditor, Resglass, en de RML testen. MapVOWL is een visuele notatie voor kennisgraafgeneratieregels. Via MapVOWL's eendrachtige grafische elementen, kunnen regels volledig gecreëerd worden via visuele voorstellingen, terwijl de regels in de onderliggende taal's syntaxis en grammatica gegenereerd worden zonder interventie van gebruikers. Onze evaluatie toont aan dat aan MapVOWL de voorkeur gegeven wordt in plaats van direct een kennisgraafgeneratietaal, zoals RML, te gebruiken. MapVOWL draagt bij tot het aanpakken van de eerste en tweede uitdaging. In toekomstig onderzoek willen we bekijken hoe alledaags datastructuren, zoals lijsten en sequenties, gevisualiseerd kunnen worden. Bovendien, deze regels moeten niet alleen voorgesteld worden via visualisaties: er zijn gebruikers die een tekst-gebaseerde aanpak wensen. Daartoe hebben we initiële stappen ondernemen door het introduceren van YARRRML, een menselijk-begrijpbare, tekst-gebaseerde voorstelling van kennisgraafgeneratieregels.

De RMLEditor is een grafische gebruikersinterface voor kennisgraafgeneratieregels, gebruikmakend van MapVOWL. Het staat gebruikers toe om databronnen in te laden, regels te creëren, en de resulteerde kennisgraaf vooraf te bezichtigen. De gebruikersinterface besteedt aandacht aan (i) de schaalbaarheidsproblemen die gepaard gaan met het gebruik van grafen voor visualisaties; (ii) de visualisatie van heterogene

databronnen, naast het tonen van de ruwe data; en (iii) de integratie van transformaties van de databronnen in de visualisatie van de regels. Onze evaluatie toont aan dat de RMLEditor een goede beoordeling krijgt van gebruikers en wordt verkozen over het gebruik van een formulier-gebaseerde gebruikersinterface, zoals RMLx. De RMLEditor draagt bij tot het aanpakken van de eerste en tweede uitdaging. In toekomstig onderzoek willen we MapVOWL combineren met andere voorstellingen, zoals YARRRML, en nieuwe functies toevoegen om het volledige proces om regels te creëren te verbeteren, zoals het importeren van ontologieën.

Resglass is een regel-gebaseerde methode om inconsistenties op te lossen in regels, concepten en relaties. De regels, concepten, en relaties worden automatisch gerangschikt in de volgorde van inspectie gebaseerd op een score die rekening houdt met het aantal inconsistenties in welke een regel, concept, of relatie betrokken is. Onze evaluatie toont aan dat de ordening van Resglass de overlap met die van experts verbetert met 41% voor regels en 23% voor concepten en relaties, in vergelijking met een willekeurige ordening. Resglass draagt bij tot het aanpakken van de tweede uitdaging. In toekomstig onderzoek willen we bekijken hoe we de gebruikers verder kunnen assisteren in het detecteren en vermijden van inconsistenties, zoals, b.v., het suggereren van mogelijke oplossing gebaseerd op gebruikte en andere ontologieën, en regels van andere toepassingen.

De RML testen laten (i) ontwikkelaars toe om te bepalen hoe conform hun RML processoren zijn aan de RML specificatie, en (ii) gebruikers toe om de testresultaten te gebruiken om de meeste geschikte processor voor hun toepassingen te selecteren. Dit is gedaan met het oog op het ontwerpen van testen die onafhankelijk zijn van RML en die van toepassing zijn op alle talen die kennisgrafen genereren van heterogene databronnen. De testen dragen bij tot het aanpakken van de derde uitdaging. In toekomstig onderzoek willen we bekijken wat de karakteristieken zijn van testen waarbij datastromen gebruikt worden in plaats van statische bestanden of databanken, en andere metrieken om de selectie van de meest geschikte processor nog te verbeteren, zoals snelheid, geheugengebruik, bruikbaarheid, en uitbreidbaarheid.

List of acronyms

API: Application Programming Interface

CSV: Comma Separated Values

EARL: Evaluation and Report Language

ESWC: Extended Semantic Web Conference

FWO: Fonds Wetenschappelijk Onderzoek

GUI: Graphical User Interface

IRI: Internationalized Resource Identifier

JSON: JavaScript Object Notation

KG: Knowledge Graph

KGSWC: Knowledge Graphs and Semantic Web Conference

MVC: Model-View-Controller

OWL: Web Ontology Language

R2RML: RDB to RDF Mapping Language

RBO: Ranked-Bias Overlap

RDB: Relational Database

RDF: Resource Description Framework

RML: RDF mapping language

SHACL: Shapes Constraint Language

SPARQL: SPARQL Protocol and RDF Query Language

SQL: Structured Query Language

SUS: System Usability Scale

VOWL: Visual Notation for OWL Ontologies

W3C: World Wide Web Consortium

XML: Extensible Markup Language

Chapter 1

Introduction

More and more data is generated by an increasing number of agents (applications and devices) that are connected to the Internet [1]. For example, Twitter users wrote 473,400 tweets per minute in 2018 [2], compared to 100,000 in 2013 [3], and Instagram users posted 46,740 photos per minute in 2018, compared to 3,600 in 2013. At the same time, there were 15,41 billion Internet connected devices installed in 2015, while it is expected that by 2025 there will be 75,44 billion installed [4]. They all contribute to data that is available on the Web.

When this data is analyzed, combined, and shared powerful, new techniques can be designed and deployed, such as artificial intelligence applied by personal assistants (e.g., Apple's Siri [5] and Amazon's Alexa [6]), improved search engines (e.g., Google Search [7] and Microsoft's Bing [8]), and decentralized data storages (as opposed to a large, central storage used by big companies).

Due to this increase in data, the representations originally used for data, such as simple JSON and XML documents, have become insufficient. They make it hard for agents to exchange and reuse data, residing in different data sources, on the Web, because arbitrary structures can be added to the representations without saying what these structures actually mean [9]. Thus, the problem is how can we represent data on the Web that makes it possible for agents to exchange this data with other agents and reuse it for their own purposes?

The Semantic Web offers a solution to this problem: it is an extension of the World Wide Web that allows data to be exchanged between multiple agents that generate and use data in different ways on a large scale, such as the Web [9]. Semantics is concerned with relationships between entities (something that exists apart from other things) and what they mean in reality[1]. The Semantic Web adds semantics to the Web through a set of formats and technologies that provide a formal description of concepts and relationships within a given domain. The main technology is knowledge graphs: directed graphs that provide semantic descriptions of entities and their relationships [10]. For

[1] `https://en.wikipedia.org/wiki/Semantics`

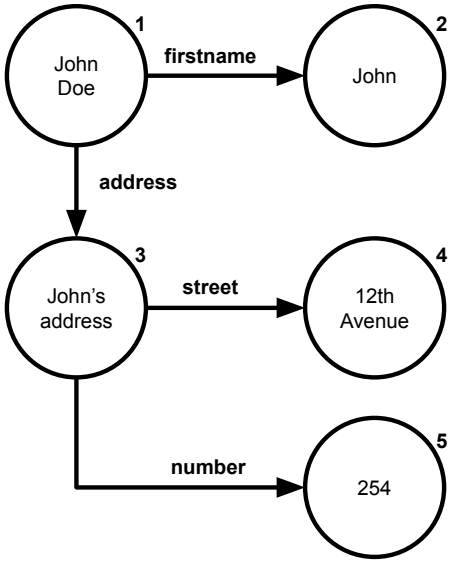


Figure 1.1: Knowledge graph about John Doe with his name and address. Nodes represent entities and arrows represent directed relationships.

example, in Figure 1.1 a knowledge graph describes John Doe and his address. The nodes represent entities and arrows directed relationships. John Doe's first name is "John" (an arrow from node 1 to 2) and his address (arrow from node 1 to 3) has street "12th Avenue" (arrow from node 3 to 4) and number "254" (arrow from node 3 to 5).

Knowledge graphs are often generated from other data sources, such as multiple databases and files that each represent their data in different ways. For instance, the DBpedia knowledge graph [11] is generated from Wikipedia [12]; the Google knowledge graph [13] is generated from different sources, including data coming from Google's own services and Wikipedia; and IBM's Watson Discovery Knowledge Graph [14] is generated from custom data sources. The DBpedia knowledge graph is generated by extracting the relevant data from the Wikipedia pages and assigning the corresponding concepts and relationships [11]. A custom Watson Discovery Knowledge Graph can be generated by extracting unstructured text from documents, classifying and tagging the text, correlate it with other text, filter the results, and representing the results as a knowledge graph [15].

A common way to generate these knowledge graphs is by using rules, which are created, refined, and executed (see Figure 1.2). The rules attach semantic annotations to data in those sources. Semantic annotations provide additional information about various entities that are described in the data source (e.g., things, people, companies, and so on). The semantic annotations are added using machine-understandable

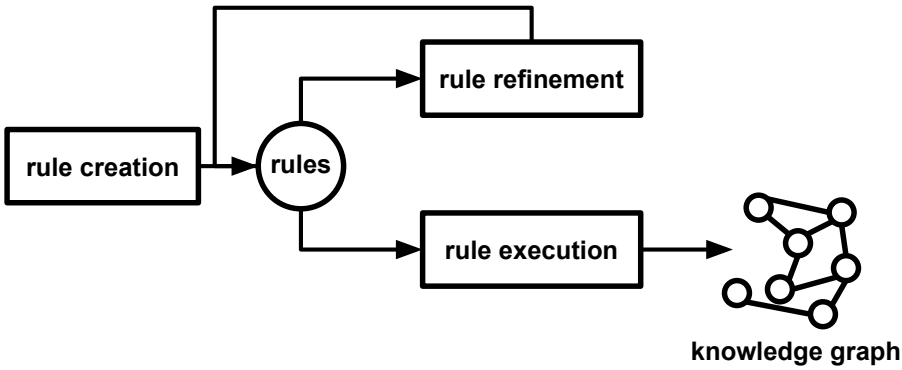


Figure 1.2: Rules are created, refined if needed, after which they are executed to generate a knowledge graph.

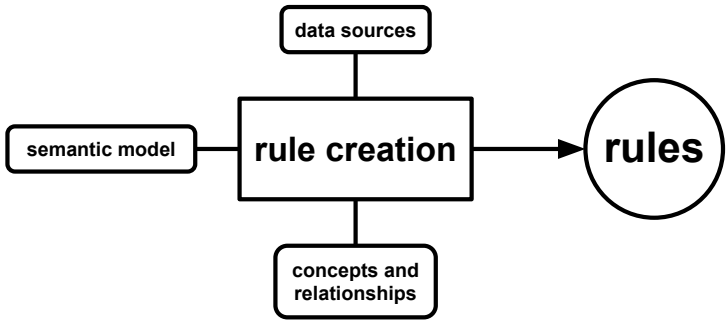


Figure 1.3: Rules are influenced by data in data sources, concepts, relationships, and semantic model.

concepts and relationships, which allow agents to share information in a domain. For example in the domain of families, concepts include child, mother, and brother, and relationships include "is sibling of" and "is parent of". Which concepts and relationships are used and how they are applied to data sources is contained in a semantic model. Rules thereby determine how data sources are modeled using specific concepts and relationships during knowledge graph generation. The syntax and grammar of these rules are determined by a knowledge graph generation language, such as R2RML [16] and RML [17]. For example, a language can say that if there are rules that describe how a relationship between two entities is generated then there also need to be rules that describe how these entities are generated. Thus, the rule creation is influenced by three aspects: (i) data sources, (ii) concepts and relationships, and (iii) semantic model (see Figure 1.3). In most cases rules are created manually by users, optionally assisted

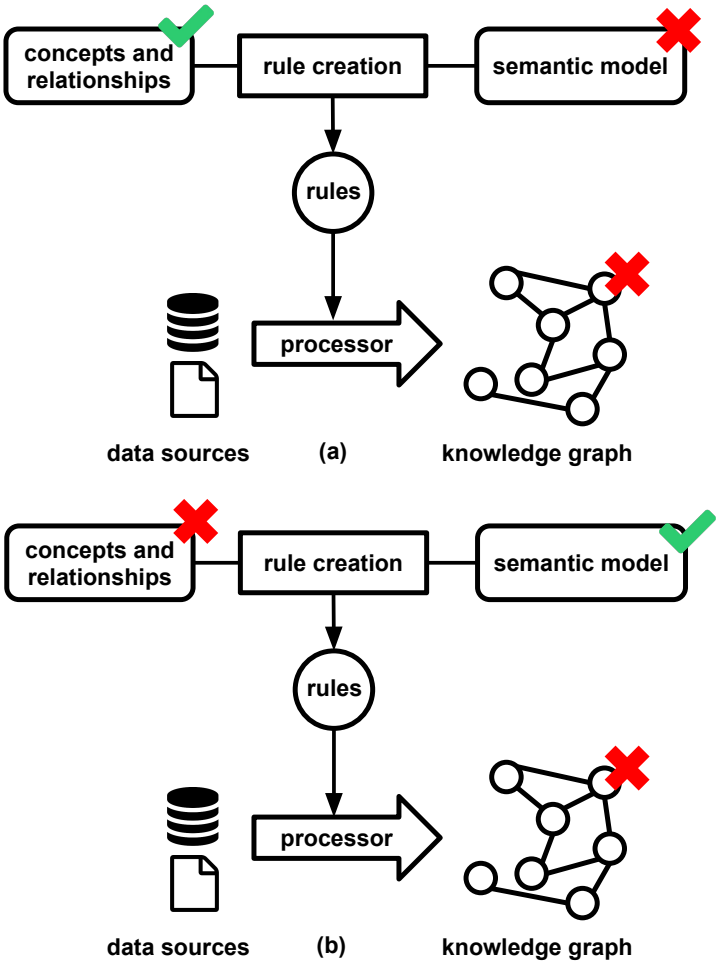


Figure 1.4: Knowledge graphs contain inconsistencies if they are introduced by the concepts, relationships, or semantic model during the rule creation.

by semi-automatic approaches [18].

Once (a subset of) the rules are created, they are refined to resolve inconsistencies, if there are any. Inconsistencies in knowledge graphs are introduced when concepts and relationships are used without adhering to their restrictions, and this affects the graphs' quality. For example, a person cannot be a sibling and a parent of another person at the same time. Possible root causes for these inconsistencies include: (i) semantic model that introduce new inconsistencies by, for example, not using the suitable concepts and relationships [19, 20] (see Figure 1.4a); and (ii) concept and relationship definitions

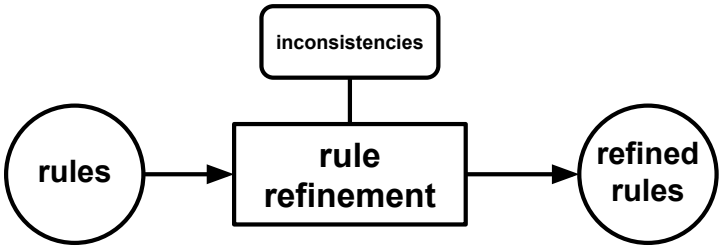


Figure 1.5: Rule refinement takes a set of existing rules as input, is influenced by inconsistencies, and outputs a set of refined rules.

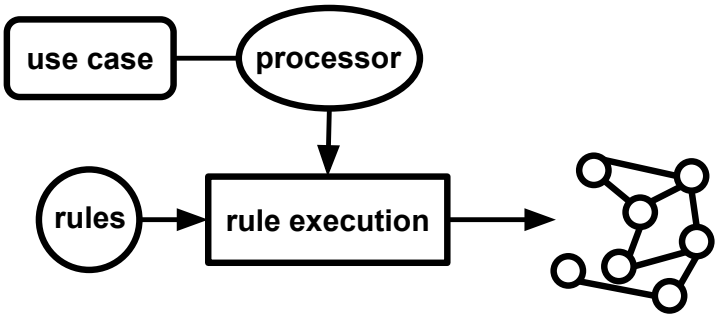


Figure 1.6: During rule execution a processor, selected based on the use case, execute the rules to generate the corresponding knowledge graph.

that do not model the domain as desired [20] (see Figure 1.4b). For example for the former, if a rule defines that all persons in a certain data source are both brothers and sisters then inconsistencies are introduced because a person is either a brother or a sister of someone, but never both. For the latter, if it is defined that if a person is someone's sister it is also that person's mother inconsistencies are introduced because if a person is someone's sister then it cannot be that person's mother at the same time and vice versa. Thus, rule refinement takes a set of existing rules as input, is influenced by inconsistencies, and outputs a set of refined rules (see Figure 1.5).

Once the rules are created and refined, they are executed to generate a knowledge graph. This is done via a processor: a software tool that, given a set of rules and data sources, generates knowledge graphs (see Figure 1.6). Multiple processors can be developed for a single knowledge graph generation language while each processor has a different set of features: conformance to the specification of the rule language, API, programming language, scalability, and so on. Users can switch between them without changing the rules (see Figure 1.7), and thus, the selection of the most suitable processor depends on the use case at hand (see Figure 1.6). For example, when a processor

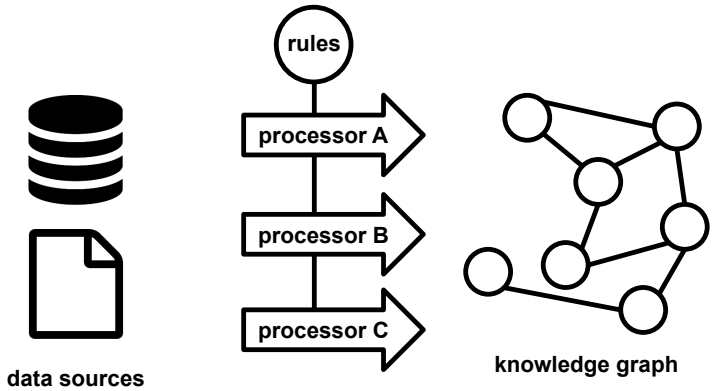


Figure 1.7: Different processors can be used for the execution of rules, without requiring changes to rules.

wants to be integrated in existing software as a library, then a processor's programming language and API affect whether this is possible or not. When a processor is needed to deal with huge amounts of data, then a processor's scalability determines if it can be used or not.

1.1 Research challenges

In this dissertation, we look at three research challenges of knowledge graph generation using rules: the first one deals with rule creation, the second one with rule refinement, and the third one with rule execution. Rule creation is influenced by users' understanding of the different components used during the creation: data sources, rules, concepts, relationships, and semantical model. This leads to the first challenge:

Challenge 1: *Improvement of users' understanding of the rule creation's components.*

Possible root causes for inconsistencies in knowledge graphs are the semantic model and the definitions of the concepts and relationship. These inconsistencies need to be removed, during rule refinement, or are ideally avoided in the first place. This leads to the following challenge:

Challenge 2: *Avoidance and removal of inconsistencies introduced by concepts, relationships, and semantic model.*

During rule execution a processor is used to generate knowledge graphs. If multiple processors are available, users need to select the most suitable one for the use case at hand. However, this is not trivial if each processor has a different set of features. This leads to the following challenge:

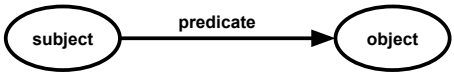


Figure 1.8: Each RDF triple consists of a subject, predicate, and object.

Challenge 3: *Selection of the most suitable processor for the use case at hand.*

1.2 Background

Knowledge graphs can be materialized using the *Resource Description Framework* (RDF) [21]. It is a framework for representing information on the Web using a directed graph-based model. A graph consists of zero or more triples and each triple consists of a subject, predicate, and object (see Figure 1.8). For example, when aligning with a natural language, the sentence "John has age 30" can be seen as a triple, where "John" is the subject, "has age" is the predicate, and "30" is the object.

In RDF, a subject is identified by an Internationalized Resource Identifier (IRI) [22] or blank node; a predicate an IRI; and an object an IRI, blank node or literal. IRIs and literals are used for entities and denote something in this world, including physical things, documents, abstract concepts, numbers, and text. The IRI is an internet protocol standard which uses a series of characters that unambiguously identifies a particular resource, typically on the Web. Examples are `https://google.com/` and `https://www.wikipedia.org/`. Literals are also a series of characters, but they are used for values such as strings, numbers, and dates. Examples are "John", "30", and "2019-05-21". Furthermore, they have datatypes or are language-tagged. Datatypes, which are also IRIs, define the possible values of a literal, such as numbers and dates. For example, a literal that represents the age of a person will have "number" as datatype and a literal that represents the birth date of a person will have "date" as datatype. Language-tagged literals denote a text in a human language. For example, "12th Avenue" denotes a street name in English. Blank nodes are used for entities for which no explicit identifier is required, opposed to IRIs.

An *ontology* includes machine-understandable definitions of basic concepts in the domain and relationships among them and encodes the implicit rules constraining the structure of a piece of reality [23]. Such implicit rules can be encoded as ontological axioms in OWL [24] and are henceforth referred to as restrictions. For example, the domains and ranges of properties are restricted to a set of classes. Such restrictions are either defined via the ontology term's definitions (e.g., a "Person" is a concept and "is sibling of" a relationship) or via the interpretation of the ontology's axioms (e.g., a person is either a brother or a sister of someone, but never both) as restrictions [25, 26].

In Figure 1.9, an example can be found of how RDF and ontologies are used to describe John Doe and his address. An IRI is used for the entity that represents John

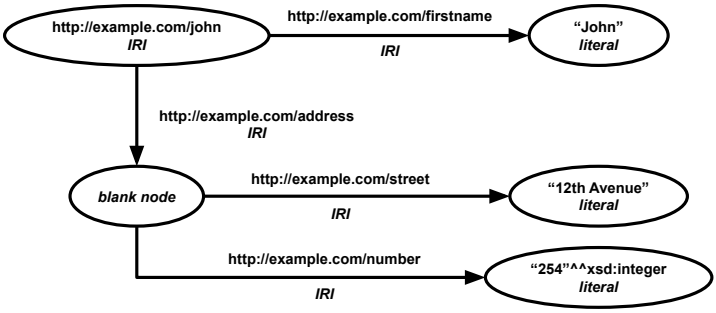


Figure 1.9: An RDF graph describing a person called John and his address.

Doe (`http://example.com/john`). A literal is used for his name and is related to John Doe via an IRI (`http://example.com/firstname`). A blank node is used for the entity that represents John Doe's address and is related to John Doe via an IRI (`http://example.com/address`). Literals are used for the address' street and number. For the latter, a datatype (`http://example.com/integer`) is used to make clear that the number only contains digits and is not, for example, a date.

A number of knowledge graph generation languages have been proposed, such as the RDB to RDF Mapping Language (R2RML) [16] and the RDF Mapping Language (RML) [17]. R2RML is a W3C recommendation that focuses on the generation of knowledge graphs from relational databases (RDBs). RML is an extension of R2RML and supports multiple, different data sources, such as databases, files, and Web APIs, and data formats, such as CSV, JSON, and XML. However, manually creating rules requires a substantial amount of human effort [27]. Therefore, a significant number of tools with a graphical user interface (GUI), henceforth referred to as rule editors, were implemented to help users to create rules [28, 29, 30]. They offer different features, which are not necessarily found in one single editor. For example, they hide the syntax and grammar of the underlying language, they visualize the generated knowledge graph, they allow to view the data sources, and they support data sources with different data formats.

Different processors have been developed for the existing knowledge graph generation languages[2]. The conformance of these processors to the language's specification is assessed based on whether the correct knowledge graph is generated for a set of rules and certain data source or not. Consequently, users are able to consider the processors' conformance to the specification, among other features, during their selection of the most appropriate processor for a certain use case. For example, do users want a pro-

[2] RMLMapper (`https://github.com/RMLio/rmlmapper-java`), CARML (`https://github.com/carml/carml`), Morph-RDB (`https://github.com/oeg-upm/morph-rdb`), R2RML Parser (`https://github.com/nkons/r2rml-parser`)

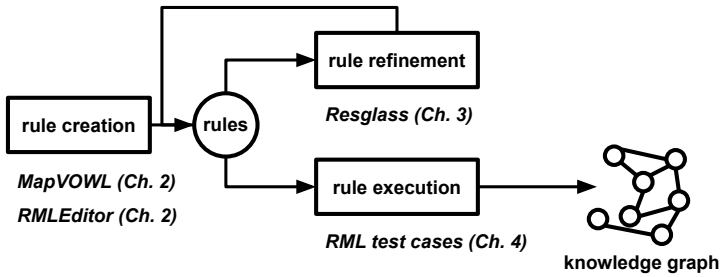


Figure 1.10: MapVOWL and the RMLEditor contribute to rule creation (Chapter 2), Resglass to rule refinement (Chapter 3), and the RML test cases to rule execution (Chapter 4).

cessor that supports the complete specification, or do they prefer a processor that does not support certain aspects of the specification, but executes the rules faster?

1.3 Research questions and hypotheses

In this section, we identify the research questions and hypotheses identified through the research challenges and existing work, and highlight the corresponding solutions developed during my PhD. Each solution either contributes to rule creation, rule refinement, or rule execution (see Figure 1.10).

Existing rule editors, such as RMLx [29] and Map-On [30], facilitate the creation of rules and thus, contribute to tackling Challenges 1 and 2. However, the design of these tools and their GUIs are not thoroughly investigated yet. This leads to the following two research questions:

Research Question 1: *How can we design visualizations that improve the cognitive effectiveness of visual representations of knowledge graph generation rules?*

Research Question 2: *How can we visualize the components of a knowledge graph generation process to improve its cognitive effectiveness?*

Note that cognitive effectiveness is defined as the speed, ease, and accuracy with which a representation can be processed by the human mind [31]. To address the first question, we introduce a visual notation for knowledge graph generation rules called MapVOWL. This leads to the following hypothesis:

Hypothesis 1: *MapVOWL improves the cognitive effectiveness of the generation rule visual representation to generate knowledge graphs compared to using RML directly.*

RML was chosen over other existing knowledge graph generation languages as it supports rules with data from multiple, heterogeneous data sources. To address the

second question, we developed a graph-based rule editor called the RMLEditor. This leads to the following hypothesis:

Hypothesis 2: *The cognitive effectiveness provided by the RMLEditor's GUI improves the user's performance during the knowledge graph generation process compared to RMLx.*

RMLx was chosen over other existing rule editors as it allows to annotate multiple, heterogeneous data sources and values. More, it also uses graph visualizations to represent the rules; however a form-based approach is used to edit the rules. Both MapVOWL and the RMLEditor contribute to rule creation (see Figure 1.10). They have been used by partners and different projects including COMBUST[3] and the Belgian Chapter of Open Knowledge Foundation[4].

In previous research efforts, a method to resolve inconsistencies by automatically refining the corresponding rules has been developed [19], tackling Challenge 2. However, this method assumes that ontologies do not contain inconsistencies. Furthermore, when a high number of rules are involved in inconsistencies users have no insights regarding the order in which rules should be inspected. This issue leads to the following research question:

Research Question 3: *How can we score and rank rules and ontology terms for inspection to improve the manual resolution of inconsistencies?*

To address this question, we developed a new method called Resglass that extends the existing rule-driven method [19]. The rules and ontology terms are automatically ranked in order of inspection based on a score that considers the number of inconsistencies a rule or ontology term is involved in. This leads to the following hypothesis:

Hypothesis 3: *The automatic inconsistency-driven ranking of Resglass improves, compared to a random ranking, by at least 20% the overlap with experts' manual ranking.*

A number of knowledge graph generation rule languages, such as RML, support multiple, heterogeneous data sources, instead of only supporting a single data source or data format, such as R2RML. However, for these languages no test cases are available that allow to determine the conformance of corresponding processors to the languages' specifications. This makes it hard for users to determine the most suitable processor for a certain use case. Furthermore, it has not been investigated yet what the characteristics are of such test cases and how they differ from test cases only focusing on a single data source or data format. This leads to the following research question, tackling Challenge 3:

[3] `https://www.imec-int.com/en/what-we-offer/research-portfolio/combust`
[4] `http://be.okfn.org/`

Research Question 4: *What are the characteristics of test cases for processors that generate knowledge graphs from heterogeneous data sources independent of languages' specifications?*

To address this question, we designed an initial set of conformance test cases for RML, based on the R2RML test cases. We use RML, because it supports heterogeneous data sources and is based on R2RML for which already test cases are designed. This is done towards designing test cases that are independent of RML and are applicable to all languages that generate knowledge graphs from heterogeneous data sources.

1.4 Publications

The work presented in this dissertation is based on four peer-reviewed publications in international, scientific journals and conference proceedings:

- Pieter Heyvaert et al. "RMLEditor: A Graph-Based Mapping Editor for Linked Data Mappings". In: *The Semantic Web – Latest Advances and New Domains (ESWC 2016)*. Vol. 9678. Lecture Notes in Computer Science. Springer, May 2016, pp. 709–723. ISBN: 978-3-319-34129-3. DOI: 10.1007/978-3-319-34129-3_43. URL: http://dx.doi.org/10.1007/978-3-319-34129-3_43

- Pieter Heyvaert et al. "Specification and Implementation of Mapping Rule Visualization and Editing: MapVOWL and the RMLEditor". In: *Journal of Web Semantics* 49 (Mar. 2018), pp. 31–50. ISSN: 1570-8268. DOI: 10.1016/j.websem.2017.12.003. URL: https://doi.org/10.1016/j.websem.2017.12.003

- Pieter Heyvaert et al. "Rule-driven inconsistency resolution for knowledge graph generation rules". In: *Semantic Web* (2019). URL: http://www.semantic-web-journal.net/system/files/swj2064.pdf

- Pieter Heyvaert et al. "Conformance Test Cases for the RDF Mapping Language (RML)". in: *Knowledge Graphs and Semantic Web (KGSWC 2019)*. Communications in Computer and Information Science. Springer, 2019, pp. 162–173. DOI: 10.1007/978-3-030-21395-4_12. URL: http://dx.doi.org/10.1007/978-3-030-21395-4_12

The following lists provide an overview of all publications I (co-)authored during my PhD.

1.4.1 Publications in international journals

- Pieter Heyvaert et al. "Specification and Implementation of Mapping Rule Visualization and Editing: MapVOWL and the RMLEditor". In: *Journal of Web Semantics* 49 (Mar. 2018), pp. 31–50. ISSN: 1570-8268. DOI: 10.1016/j.websem.2017.12.003. URL: https://doi.org/10.1016/j.websem.2017.12.003

- Pieter Heyvaert et al. "Rule-driven inconsistency resolution for knowledge graph generation rules". In: *Semantic Web* (2019). URL: http://www.semantic-web-journal.net/system/files/swj2064.pdf

1.4.2 Publications in international conference proceedings

- Pieter Heyvaert et al. "Merging and Enriching DCAT Feeds to Improve Discoverability of Datasets". In: *Proceedings of the 12th Extended Semantic Web Conference: Posters and Demos*. Vol. 9341. Lecture Notes in Computer Science. Springer, June 2015, pp. 67–71. DOI: 10.1007/978-3-319-25639-9_13

- Pieter Heyvaert et al. "Using EPUB 3 and the Open Web Platform for Enhanced Presentation and Machine-Understandable Metadata for Digital Comics". In: *Proceedings of the 19th International Conference on Electronic Publishing*. Sept. 2015, pp. 37–46. DOI: 10.3233/978-1-61499-562-3-37

- Pieter Heyvaert et al. "Linked Data-enabled Gamification in EPUB 3 for Educational Digital Textbooks". In: *Proceedings of the 10th European Conference on Technology Enhanced Learning*. Vol. 9307. Lecture Notes in Computer Science. Springer, Sept. 2015, pp. 587–591. DOI: 10.1007/978-3-319-24258-3_65. URL: http://dx.doi.org/10.1007/978-3-319-24258-3_65

- Pieter Heyvaert et al. "Towards a Uniform User Interface for Editing Mapping Definitions". In: *Proceedings of the 4th International Workshop on Intelligent Exploration of Semantic Data (IESD 2015)*. CEUR-WS.org, 2015. URL: http://ceur-ws.org/Vol-1472/IESD_2015_paper_4.pdf

- Pieter Heyvaert et al. "Towards Approaches for Generating RDF Mapping Definitions". In: *Proceedings of the 14th International Semantic Web Conference: Posters and Demos*. Oct. 2015. URL: http://ceur-ws.org/Vol-1486/paper_70.pdf

- Pieter Heyvaert et al. "Semantically Annotating CEUR-WS Workshop Proceedings with RML". in: *Proceedings of the 12th Extended Semantic Web Conference: Semantic Publishing Challenge*. Lecture Notes in Computer Science. Springer, Jan. 2016, pp. 165–176. DOI: 10.1007/978-3-319-25518-7_14. URL: http://dx.doi.org/10.1007/978-3-319-25518-7_14

- Pieter Heyvaert et al. "RMLEditor: A Graph-Based Mapping Editor for Linked Data Mappings". In: *The Semantic Web – Latest Advances and New Domains (ESWC 2016)*. Vol. 9678. Lecture Notes in Computer Science. Springer, May 2016, pp. 709–723. ISBN: 978-3-319-34129-3. DOI: 10.1007/978-3-319-34129-3_43. URL: http://dx.doi.org/10.1007/978-3-319-34129-3_43

- Pieter Heyvaert et al. "Graph-Based Editing of Linked Data Mappings using the RMLEditor". In: *Proceedings of the 13th Extended Semantic Web Conference: Posters and Demos*. Vol. 9989. Lecture Notes in Computer Science. Springer, June 2016, pp. 123–127. DOI: 10.1007/978-3-319-47602-5_25

- Pieter Heyvaert et al. "Linked Sensor Data Generation using Queryable RML Mappings". In: *Proceedings of the 15th International Semantic Web Conference: Posters and Demos*. Oct. 2016. URL: http://ceur-ws.org/Vol-1690/paper9.pdf

- Ruben Taelman et al. "Querying Dynamic Datasources with Continuously Mapped Sensor Data". In: *Proceedings of the 15th International Semantic Web Conference: Posters and Demos*. Oct. 2016. URL: http://ceur-ws.org/Vol-1690/paper6.pdf

- Anastasia Dimou et al. "Towards an Interface for User-Friendly Linked Data Generation Administration". In: *Proceedings of the 15th International Semantic Web Conference: Posters and Demos*. Vol. 1690. CEUR Workshop Proceedings. Oct. 2016. URL: http://ceur-ws.org/Vol-1690/paper98.pdf

- Pieter Heyvaert et al. "Data Analysis of Hierarchical Data for RDF Term Identification". In: *Proceedings of the 6th Joint International Semantic Technology Conference*. Nov. 2016. URL: https://biblio.ugent.be/publication/8504071/file/8504072

- Anastasia Dimou et al. "Modeling, Generating, and Publishing Knowledge as Linked Data". In: *EKAW 2016 Satellite Events*. Vol. 10180. Lecture Notes in Computer Science. Springer, May 2017. DOI: 10.1007/978-3-319-58694-6_1. URL: https://doi.org/10.1007/978-3-319-58694-6_1

- Pieter Heyvaert. "Ontology-Based Data Access Mapping Generation using Data, Schema, Query, and Mapping Knowledge". In: *Proceedings of the 14th Extended Semantic Web Conference: PhD Symposium*. May 2017. DOI: 10.1007/978-3-319-58451-5_15. URL: https://doi.org/10.1007/978-3-319-58451-5_15

- Pieter Heyvaert et al. "Semi-Automatic Example-Driven Linked Data Mapping Creation". In: *Proceedings of the Fifth International Workshop on Linked Data*

for Information Extraction. Oct. 2017. URL: http://ceur-ws.org/Vol-1946/paper-03.pdf

- Riccardo Tommasini et al. "Representing Dockerfiles in RDF". in: *Proceedings of the 16th International Semantic Web Conference: Posters and Demos*. Oct. 2017. URL: http://ceur-ws.org/Vol-1963/paper528.pdf

- Anastasia Dimou et al. "What Factors Influence the Design of a Linked Data Generation Algorithm?" In: *Proceedings of the 11th Workshop on Linked Data on the Web*. Apr. 2018. URL: http://events.linkeddata.org/ldow2018/papers/LDOW2018_paper_12.pdf

- Pieter Heyvaert et al. "Declarative Rules for Linked Data Generation at your Fingertips!" In: *Proceedings of the 15th ESWC: Posters and Demos*. Vol. 11155. Lecture Notes in Computer Science. Springer, June 2018, pp. 213–217. DOI: 10.1007/978-3-319-98192-5_40. URL: https://doi.org/10.1007/978-3-319-98192-5_40

- Bram Steenwinckel et al. "Automated extraction of rules and knowledge from risk analyses: a ventilation unit demo". In: *Proceedings of the 17th ISWC: Posters and Demos*. Oct. 2018. URL: http://ceur-ws.org/Vol-2180/paper-63.pdf

- Bram Steenwinckel et al. "Towards Adaptive Anomaly Detection and Root Cause Analysis by Automated Extraction of Knowledge from Risk Analyses". In: *Proceedings of the 9th International Semantic Sensor Networks Workshop*. Oct. 2018. URL: http://ceur-ws.org/Vol-2213/paper2.pdf

- Ben De Meester et al. "Towards a Uniform User Interface for Editing Data Shapes". In: *Proceedings of the 4th International Workshop on Visualization and Interaction for Ontologies and Linked Data*. Oct. 2018. URL: http://ceur-ws.org/Vol-2187/paper2.pdf

- Joachim Van Herwegen et al. "Knowledge Representation as Linked Data". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. Oct. 2018. DOI: 10.1145/3269206.3274275. URL: https://doi.org/10.1145/3269206.3274275

- Ben De Meester et al. "Mapping languages analysis of comparative characteristics". In: *Proceedings of the 1st Knowledge Graph Building Workshop*. 2019. URL: https://openreview.net/pdf?id=HklWL4erv4

- Pieter Heyvaert et al. "SAD Generator: eating our own dog food to generate KGs and websites for academic events". In: *Proceedings of the 16th ESWC: Posters and Demos*. 2019

- Pieter Heyvaert et al. "Conformance Test Cases for the RDF Mapping Language (RML)". in: *Knowledge Graphs and Semantic Web (KGSWC 2019)*. Communications in Computer and Information Science. Springer, 2019, pp. 162–173. DOI: 10.1007/978-3-030-21395-4_12. URL: http://dx.doi.org/10.1007/978-3-030-21395-4_12

1.5 Outline

The remainder of the dissertation consists of three chapters that are based on the four peer-reviewed publications that contribute to my PhD, and a conclusion chapter. In Chapter 2, we focus on rule creation and address Research Questions 1 and 2. More specific, we discuss the initial version of the RMLEditor: our software tool to create rules using graph-based visualizations; followed by MapVOWL: our visual notation for knowledge graph generation rules; and the updated version of the RMLEditor. In Chapter 3, we focus on rule refinement and address Research Question 3. More specific, we describe Resglass: our rule-driven method for the resolution of inconsistencies in ontologies and rules. In Chapter 4, we focus on rule execution and address Research Question 4. More specific, we elaborate on our initial set of conformance test cases for RML as exploratory step towards designing language-independent test cases for knowledge graph generation. In Chapter 5, we conclude the work of this dissertation and discuss future research options.

References

[1] Bernard Marr. *How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read*. https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#. Accessed: 2019-06-01.

[2] *Data Never Sleeps 6*. https://www.domo.com/learn/data-never-sleeps-6#/. Accessed: 2019-06-01.

[3] *Data Never Sleeps*. https://www.domo.com/learn/infographic-data-never-sleeps#/. Accessed: 2019-06-01.

[4] *Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)*. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. Accessed: 2019-06-01.

[5] *Siri – Apple*. https://www.apple.com/siri/. Accessed: 2019-06-01.

[6] *Amazon Alexa*. https://developer.amazon.com/alexa. Accessed: 2019-06-01.

[7] *Google*. https://www.google.com/. Accessed: 2019-06-01.

[8] *Bing*. https://www.bing.com/. Accessed: 2019-06-01.

[9] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web". In: *Scientific American* (2001). URL: https://www-sop.inria.fr/acacia/cours/essi2006/Scientific%5C%20American%5C_%5C%20Feature%5C%20Article%5C_%5C%20The%5C%20Semantic%5C%20Web%5C_%5C%20May%5C%202001.pdf.

[10] Heiko Paulheim. "Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods". In: *Semantic Web* 8.3 (2017), pp. 489–508. DOI: 10.3233/SW-160218. URL: https://doi.org/10.3233/SW-160218.

[11] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. "DBpedia–a Large-scale, Multilingual Knowledge Base Extracted from Wikipedia". In: *Semantic Web* 6.2 (2015), pp. 167–195. DOI: 10.3233/SW-140134. URL: https:/doi.org/10.3233/SW-140134.

[12] *Wikipedia*. https://www.wikipedia.org/. Accessed: 2019-06-01.

[13] Amit Singhal. *Introducing the Knowledge Graph: things, not strings*. https://www.blog.google/products/search/introducing-knowledge-graph-things-not/. Accessed: 2019-06-01.

[14] *Watson Discovery Knowledge Graph*. https://cloud.ibm.com/docs/services/discovery?topic=discovery-kg. Accessed: 2019-06-01.

[15] Neha Setia, Vishal Chahal, and Manjula Hosurmath. *Build a knowledge graph from documents*. https://developer.ibm.com/patterns/build-a-domain-specific-knowledge-graph-from-given-set-of-documents/. Accessed: 2019-06-01.

[16] Souripriya Das, Seema Sundara, and Richard Cyganiak. *R2RML: RDB to RDF Mapping Language*. Working Group Recommendation. W3C, Sept. 2012. URL: http://www.w3.org/TR/r2rml/.

[17] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data". In: *Workshop on Linked Data on the Web*. 2014. URL: http://events.linkeddata.org/ldow2014/papers/ldow2014_paper_01.pdf.

[19] Anastasia Dimou, Dimitris Kontokostas, Markus Freudenberg, Ruben Verborgh, Jens Lehmann, Erik Mannens, Sebastian Hellmann, and Rik Van de Walle. "Assessing and Refining Mappings to RDF to Improve Dataset Quality". In: *Proceedings of the 14th International Semantic Web Conference*. Vol. 9367. Lecture Notes in Computer Science. 2015, pp. 133–149. DOI: 10.1007/978-3-319-25010-6_8. URL: http://dx.doi.org/10.1007/978-3-319-25010-6_8.

[20] Heiko Paulheim. "Data-Driven Joint Debugging of the DBpedia Mappings and Ontology". In: *The Semantic Web*. Springer, 2017, pp. 404–418. DOI: 10.1007/978-3-319-58068-5_25. URL: https://doi.org/10.1007/978-3-319-58068-5_25.

[21] World Wide Web Consortium. *RDF 1.1 Concepts and Abstract Syntax*. Tech. rep. 2014. URL: https://www.w3.org/TR/rdf11-concepts/.

[22] Martin Dürst and Michel Suignard. *Internationalized resource identifiers (IRIs)*. Standard Track. IETF, Jan. 2005. URL: https://tools.ietf.org/html/rfc3987.

[23] Nicola Guarino, Stati Uniti, and Pierdaniele Giaretta. "Ontologies and knowledge bases: towards a terminological clarification". In: *Towards Very Large Knowledge Bases*. IOS Press, 1995.

[24] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. *OWL 2 Web Ontology Language*. Tech. rep. W3C, 2012. URL: https://www.w3.org/TR/owl2-syntax/.

[25] Dean Allemang and James Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. 2nd ed. Morgan Kaufmann Publishers Inc., 2011. URL: https://www.elsevier.com/books/semantic-web-for-the-working-ontologist/allemang/978-0-12-385965-5.

[26] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. "Test-driven Evaluation of Linked Data Quality". In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW '14. Seoul, Korea: ACM, 2014, pp. 747–758. DOI: 10.1145/2566486.2568002. URL: http://doi.acm.org/10.1145/2566486.2568002.

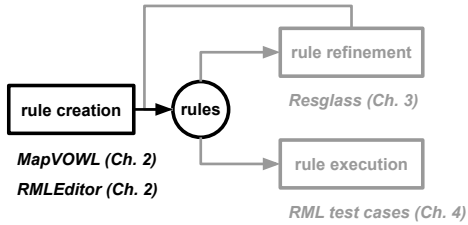
[27] Pieter Heyvaert, Anastasia Dimou, Aron-Levi Herregodts, Ruben Verborgh, Dimitri Schuurman, Erik Mannens, and Rik Van de Walle. "RMLEditor: A Graph-Based Mapping Editor for Linked Data Mappings". In: *The Semantic Web – Latest Advances and New Domains (ESWC 2016)*. Vol. 9678. Lecture Notes in Computer Science. Springer, May 2016, pp. 709–723. ISBN: 978-3-319-34129-3. DOI: 10.1007/978-3-319-34129-3_43. URL: http://dx.doi.org/10.1007/978-3-319-34129-3_43.

[28] Kunal Sengupta, Peter Haase, Michael Schmidt, and Pascal Hitzler. "Editing R2RML Mappings Made Easy". In: *Proceedings of the 2013th International Conference on Posters & Demonstrations Track*. Sydney, Australia: CEUR-WS.org, 2013, pp. 101–104. URL: http://dl.acm.org/citation.cfm?id=2874399.2874425.

[29] Peb R Aryan, Fajar J Ekaputra, Elmar Kiesling, A Min Tjoa, and Kabul Kurniawan. "RMLx: Mapping interface for integrating open data with linked data exploration environment". In: *1st International Conference on Informatics and Computational Sciences (ICICoS)*. IEEE, 2017, pp. 113–118. DOI: 10.1109/ICICOS.2017.8276347. URL: https://doi.org/10.1109/ICICOS.2017.8276347.

[30] Álvaro Sicilia, German Nemirovski, and Andreas Nolle. "Map-On: A web-based editor for visual ontology mapping". In: *Semantic Web* 8.6 (2017), pp. 969–980. DOI: 10.3233/SW-160246. URL: https://doi.org/10.3233/SW-160246.

[31] Jill H Larkin and Herbert A Simon. "Why a diagram is (sometimes) worth ten thousand words". In: *Cognitive science* 11.1 (1987), pp. 65–100.



Chapter 2

Visual notation and graphical user interface for rule creation

Manually creating knowledge graph generation rules requires a substantial amount of human effort [1]. Therefore, the use of rules by users remains complicated. To this end, a significant number of tools with a graphical user interface (GUI) were implemented to facilitate rules creation and updating, such as Map-On [2], the fluidOps Editor [3], and RMLx [4]. However, the design of these tools and their GUIs are not thoroughly investigated yet: (i) no visual specification is available, while this increases the visualization's adoption by other tools; none of the tools tackle (ii) the scalability issues that accompany the use of graphs for visualizations; (iii) the visualization of heterogeneous data sources, besides showing the raw data; and (iv) the integration of transformations of the data sources in the visualizations of the rules.

This chapter presents our following contributions to rule creation:

- the initial version of the RMLEditor (see Section 2.1): our software tool to create and update rules using graph-based visualizations;
- MapVOWL: our visual notation for knowledge graph generation rules (see Section 2.2); and
- the updated version of the RMLEditor (see Section 2.2).

We address Research Question 1 "How can we design visualizations that improve the cognitive effectiveness of visual representations of knowledge graph generation rules?" and Research Question 2 "How can we visualize the components of a knowledge graph generation process to improve its cognitive effectiveness?" and validate Hypothesis 1 "MapVOWL improves the cognitive effectiveness of the generation rule visual representation to generate knowledge graphs compared to using RML directly,"

and Hypothesis 2 “The cognitive effectiveness provided by the RMLEditor’s GUI improves the user’s performance during the knowledge graph generation process compared to RMLx.”

2.1 RMLEditor: a graph-based editor for knowledge graph generation rules

Pieter Heyvaert, Anastasia Dimou, Aron-Levi Herregodts, Ruben Verborgh, Dimitri Schuurman, Erik Mannens, and Rik Van de Walle

Published as "RMLEditor: A Graph-Based Mapping Editor for Linked Data Mappings" in *The Semantic Web: Latest Advances and New Domains (ESWC 2016)* Springer, 2016, pp. 709 – 723.

Abstract

Although several tools have been implemented to generate knowledge graphs from raw data, users still need to be aware of the underlying technologies and knowledge graph principles to use them. Knowledge graph generation languages enable to detach the rules from the implementation that executes them. However, no thorough research has been conducted on how to facilitate the editing of rules. We propose the RMLEditor, a visual graph-based user interface, which allows users to easily define the rules that deliver the RDF representation of the corresponding raw data. Neither knowledge of the underlying language nor the used technologies is required. The RMLEditor aims to facilitate the editing of rules, and thereby lowers the barriers to create knowledge graphs. The RMLEditor is developed for use by data specialists who are partners of (i) a companies-driven pilot and (ii) a community group. The current version of the RMLEditor was validated: participants indicate that it is adequate for its purpose and the graph-based approach enables users to conceive the linked nature of the data.

2.1.1 Introduction

Semantic Web technologies rely on data which is interlinked and whose semantically enriched representation is available. Such data can be made available as a knowledge graph, materialized using RDF. Most of the current knowledge graphs stems originally from (semi-)structured formats. Knowledge graph generation rules specify in a declarative way how a knowledge graph is generated from such raw data. Nevertheless, defining and executing them still remains complicated, despite the significant number of tools implemented for this scope. At first, most approaches incorporate the rules in the implementation that executes them. Thus, not only knowledge of Semantic Web technologies is required. However, also dedicated software development cycles for creating, updating and extending the implementations, whenever new or updated semantic annotations are desired, are needed. Knowledge graph generation languages, such as R2RML [5] and RML [6], enable *to detach the rules from the implementation that executes them*. Besides knowledge of the underlying language that is required to define the rules, manually editing and curating them requires a substantial amount of

human effort [7]. Moreover, data specialists are not Semantic Web experts or developers. Thus, the task of editing rules should be addressed independently, and disassociated from the corresponding language and/or underlying technology used. Facilitating the editing of rules further lowers the barriers of obtaining knowledge graphs and, thus stimulates the adoption of Semantic Web technologies. Nevertheless, dedicated environments that support users to intuitively edit rules were not thoroughly investigated yet, as it occurred with applications that actually execute them and deliver its RDF representation. *Step-by-step* wizards prevailed, e.g., fluidOps editor [3], as an easy-to-reach solution. However, such applications restrict data publishers' editing options, hamper altering parameters in previous steps, and detach rules from the overall knowledge modeling, since related information is separated in different steps. We propose the RMLEditor[1], an editing environment for specifying rules of raw data to their RDF representation based on graph visualizations, without requiring knowledge of the underlying language. The RMLEditor is developed to support partners of (i) a companies-driven pilot for sharing and integrating the RDF representations of their data and co-develop third-party applications, and of (ii) a community-group-driven bootstrap for showcasing the advantages of Semantic Web technologies. The tool is available to interested parties under custom licensing conditions. We performed an exploratory user validation of our proposed solution, which showed that the RMLEditor achieves the goal it was implemented for. 82% of the participants found the use of graphs beneficial for editing rules using the RMLEditor and 70% could better conceive that a relationship exists between multiple data sources. The remainder of the paper is structured as follows: Section 2.1.2 outlines existing knowledge graph languages and rules editors. Section 2.1.3 describes the generation process without and with the use of an editor. Section 2.1.4 presents our proposed solution, the RMLEditor. Section 2.1.5 outlines the use cases and explains the exploratory user validation and presents the results, Last, Section 2.1.6 discusses the results and presents the conclusions of our solution.

2.1.2 Related work

In this section, we discuss existing knowledge graph generation languages. Moreover, we elaborate on existing rule editors, with a distinction between editors either supporting homogeneous or heterogeneous data sources.

2.1.2.1 Knowledge graph generation languages

Knowledge graph generation languages specify in a declarative way how knowledge graphs are generated from raw data. At first, existing formalizations were considered as knowledge graph generation languages, such as XPath [8] or XQuery language [9]. Nevertheless, there were also languages defined for this particular task. R2RML [5] is

[1] `http://rml.io/RMLeditor`

the W3C-recommended language to define rules to generate RDF from data derived from relational databases. Besides R2RML, other *format-specific* languages were defined, such as X3ML[2] for XML data. There are also *query-oriented* languages such as XSPARQL [10], which combines XQuery and SPARQL to generate graphs from XML data, and Tarql[3], for data in CSV. However, these languages only support homogeneous data sources. A number of tools were developed supporting the generation of RDF graphs from heterogeneous data sources, such as Datalift[4], RDFizers[5] and Virtuoso Sponger[6]. However, those tools actually employ separate *source-centric* approaches for each format they support, which does not allow the interlinking between sources in different formats. The RDF Mapping Language (RML) [6] circumvents this, by enabling the generation of data in RDF representation based on multiple, heterogeneous data sources, e.g., XML and JSON.

2.1.2.2 Rule editors

Despite the significant number of knowledge graph generation languages, the number of corresponding editors that support users to define the rules is not comparable. Similar to the languages, a distinction can be made between tools supporting homogeneous data sources and tools supporting heterogeneous data sources.

2.1.2.2.1 Homogeneous data sources

The fluidOps editor [3] is a browser application that provides an intuitive user interface for editing rules. The underlying language is R2RML. The fluidOps editor relies on a single *step-by-step* workflow. There are six successive steps, similar to actually creating a set of rules. Although its Graphical User Interface (GUI) aims to hide the R2RML vocabulary, it still strongly focuses on concepts and terminology introduced by R2RML (e.g., Subject Maps and Object Maps). Therefore, knowledge of the language is required to use the editor. This decreases its adoption by non-Semantic Web experts and lowers the GUI's reusability potential. Additionally, only once the users reach the final step, they are able to preview the mappings in R2RML syntax and identify possible inconsistencies. Consequently, they need to restart the workflow to update the definitions of the previous steps. Pinkel et al. [7] adapted the original fluidOps editor to overcome flexibility limitations imposed by the database-driven step-by-step workflow. Their extension supports the *ontology-driven* approach. With the former approach creating the rules starts with the data in the databases and semantic annotations are added afterwards. With the latter approach the rules are created based on an existing ontology. Next, the rules are complemented with data fractions from the

[2] `https://github.com/delving/x3ml/blob/master/docs/x3ml-language.md`
[3] `https://tarql.github.io/`
[4] `http://datalift.org/`
[5] `http://simile.mit.edu/wiki/RDFizers`
[6] `http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtSponger`

databases. Sheet2RDF [11] is a platform that uses a Pearl [12] document to generate RDF graphs from data in spreadsheets. Its GUI allow users to view the source data, define the rules by editing the Pearl document directly, and view the resulting RDF through a tabular-structure. However, the adoption of the tool decreases because users need knowledge about Pearl to edit the rules. An alternative approach is proposed by Rodrıguez-Muro, Hardi, and Calvanese [13] who introduced -ontopPro-[7], a plugin for Protégé [14]. It allows users to generate RDF based on data from database(s) and an ontology. Tabs are provided to manage the databases and the mappings. Users need to write SPARQL-like templates to define how to generate a graph from the original data. However, data sources are limited to databases and users need to understand the template's custom syntax.

2.1.2.2.2 Heterogeneous data sources

Karma[8] differentiates from aforementioned tools because it supports heterogeneous sources, such as databases, delimited text files (e.g., CSV files), JSON, XML, Microsoft Excel and Web APIs. It uses Global-Local-As-View [15] rules to perform the rules. These rules can be exported using R2RML or D2RQ [16]. When displaying the rules to the users, Karma takes a data-centric approach: users can only view the input data. Users are not able to follow the ontology-driven approach, as it occurs with the fluidOps editor. DataOps [17] uses the latter to support heterogeneous data formats. However, users are still confronted with the syntax of the used languages. RDF123 [18] is similar to Sheet2RDF, however, it also supports CSV files, and it uses custom *map graphs* to represent the rules, which are converted to a custom function that produces RDF based on the input data. Consequently, users are not required to understand or know about the function to define rules, as it occurs with Pearl for Sheet2RDF. Additionally, they offer a Web service that uses a link to a Google Spreadsheet or a CSV file, and generates RDF based on the rules defined with the application. Therefore, the rules can be used outside the desktop application. However, their use is limited to that Web service, as a custom function is used and not a language. TopBraid Composer[9] supports heterogeneous sources. It allows data integration from databases, XML, UML, RSS, spreadsheets and RDF data backends. The data can be reconciliated with DBpedia [19]. However, as in the case of Karma, interlinking data from different sources is not possible. To set up the knowledge graph generation process, the GUI offers a *data-driven step-by-step* workflow. However, an ontology-driven approach is not possible. Similar to the original fluidOps editor, a more simplified wizard has been built on top of RML, instead of R2RML, called RMLx [4]. This form-based browser application supports heterogeneous data sources, because it has RML as its underlying language, compared to only homogeneous sources (i.e., databases) which are supported by the

[7] `http://ontop.inf.unibz.it/components/sample-page/`

[8] `http://www.isi.edu/integration/karma/`

[9] `http://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/`

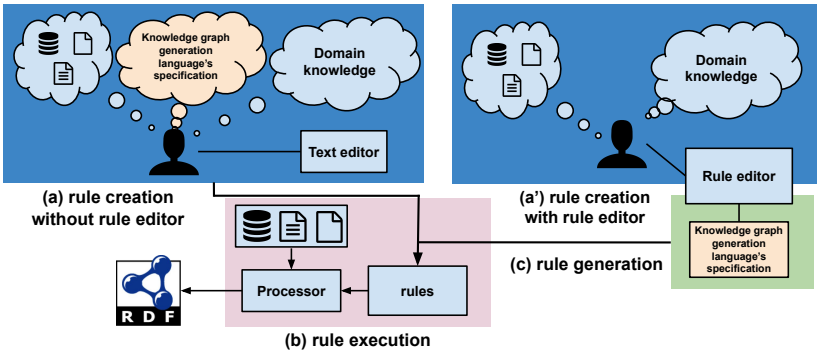


Figure 2.1: The difference in the knowledge graph generation process if during the rule creation a rule editor is used or not.

fluidOps editor. Last, OpenRefine[10] is a browser-based tool for cleansing raw data, changing the data format and incorporating external data using Web services. The RDF Refine extension[11] [20] allows users to export the data in RDF. A RDF graph is used to visualize the rules. However, the RDF graph is forced in a hierarchy-layout, which weakens the advantages of using a graph representation. Additionally, this extension supports reconciliation services that offer HTTP interfaces. User intervention is needed to assess the quality of each reconciliation. A preview of candidate entities is available to help the user.

2.1.3 Knowledge graph generation process

The knowledge graph generation process is a series of steps performed in order to generate a knowledge graph from raw data. There are two variations of the process, depending on whether a rule editor is used or not.

2.1.3.1 Rules without editor

Generating RDF from (semi-)structured data, using a knowledge graph generation language without a rule editor, consists of two consecutive steps: First, the rules are created (see Figure 2.1a). In this step, the user needs to be aware of the input data and has to have knowledge about the domain and the *knowledge graph generation language's specification*. The latter is, in principle, possessed by *Semantic Web experts*. In most cases, a text editor is used to create the rules. In the second step the rules together with the input data sources are used by a processor to generate the RDF triples (see

[10] `http://openrefine.org/`
[11] `http://refine.deri.ie/`

Figure 2.1b). A processor is a tool that knows how to interpret the language's specification to generate the RDF representation of the corresponding data, taking into consideration the rules derived in the previous step.

2.1.3.2 Rules with editor

When the process is done with a rule editor, the users do not longer need to be aware of the *language's specification* (see Figure 2.1a'), because the rules are represented in an abstract way. This allows *non-Semantic Web experts* to define the rules. The rule editor generates the rules according to the specification of the underlying language (see Figure 2.1c). Important to note is that in this step no user knowledge about the specification is needed. Subsequently, the rules are executed, and RDF triples are generated.

2.1.4 RMLEditor

The RMLEditor is a browser-based GUI with the goal to support users in production environments to define, in a uniform way, rules that specify how to generate knowledge graph, materialized using the prevalent RDF framework. In previous work [21], we listed 7 desired features of a GUI for uniform rule editors. First, it should be independent of the underlying knowledge graph generation language, so that users are able to create rules without knowledge of the language's syntax (Feature 1). Second, it should allow users to execute the rules outside of the editor, because it is only meant to create and edit the rules (Feature 2). Third, it should enable users to generate a knowledge graph from multiple data sources at the same time, as it might occur that data is spread across multiple sources (Feature 3). Fourth, the editor should support data sources in different data formats, as the generation of knowledge graphs should be independent of the original format (Feature 4). Fifth, as multiple ontologies and vocabularies can be used to create a rules, an editor should support the use of both existing and customs ontologies and vocabularies (Feature 5). Sixth, it should allow multiple alternative modeling approaches, as certain use cases might benefit from using a specific approach (Feature 6). Finally, by supporting non-linear workflows, users are able to keep an overview of the semantic model and its relationships (Feature 7). The GUI of the RMLEditor is designed to implement these features. The GUI uses graphs to visualize the rules. Manipulation of these graphs results in creating, updating and extending the rules, which can be done without any knowledge of the underlying knowledge graph generation language or other used technologies. The graphs express how the raw data will be represented as RDF. However, this expressiveness is independent of the language's expressiveness. The RMLEditor triggers a processor that executes the rules exported by the RMLEditor and generates RDF triples. For the RMLEditor, we chose RML which can support rules derived from a GUI that covers all of the aforementioned features. However, any other language could be used instead, if it allows to implement the features. In Section 2.1.4.1 we discuss the RMLEditor's

architecture. In Section 2.1.4.2, we elaborate on how the features are implemented in the GUI. In Section 2.1.4.3, we explain how the RMLProcessor, the processor for RML, is used with the RMLEditor. In Section 2.1.4.4, we present two real-life use cases of the RMLEditor.

2.1.4.1 Architecture

The RMLEditor's high-level architecture is based on the *multilayered architecture pattern* [22]. This allows to separate the presentation and the logic of the rules, using the *presentation layer* and *application layer*, respectively. The loading of rules and data sources is done using the *data access layer*. The latter only communicates with the application layer. Communication between the presentation layer and the data access layer is not possible, as the architecture prohibits communication between layers that are not directly under or above each other. For the presentation layer, the Webix JavaScript library[12] is used to build the GUI, in cooperation with the d3.js library [23] for the presentation of the graphs. The communication with application layer is facilitated by the *Model-View-Controller* pattern [24]. The Graph Markup Language (GraphML) [25] is used to represent the graph visualization of the rules independently of the underlying language. This allows users to export the graphs in an application-independent format. Additionally, the GraphML-version of the graphs are used to generate the corresponding RML rules. Users are able to load graphs by instructing the RMLEditor to load the corresponding GraphML document. When RML rules need to be loaded, the rules are first converted to a GraphML document, then interpreted as graph elements, and shown in the GUI.

2.1.4.2 Graphical user interface

The graphical user interface of the RMLEditor allows users to define rules for existing data. To implement the aforementioned features for the GUI, the RMLEditor offers three panels to the users: *Input Panel*, *Modeling Panel* and *Results Panel* (see Figure 2.2). They are aligned next to each other; however, when users want to focus on a specific panel, they are able to hide the other panels.

The *Input Panel* shows the data sources to users (Feature 3). Each data source is assigned with a unique color. Depending on the data format, an adequate visualization is chosen (Feature 4). The *Modeling Panel* shows the rules using a graph representation. The color of each node and edge depends on the data source that is used in that specific rule, if any. It offers the means to manipulate the nodes and edges of the graphs in order to update the rules. Semantic annotations can be added using multiple vocabularies and ontologies (Feature 5). The Linked Open Vocabularies[13] (LOV) can be consulted via the GUI to get suggestions on which classes, properties and datatypes to use. As the graphs offer a generic representation of the rules, because they do not

[12] `http://webix.com/`
[13] `https://lov.linkeddata.es/dataset/lov/`

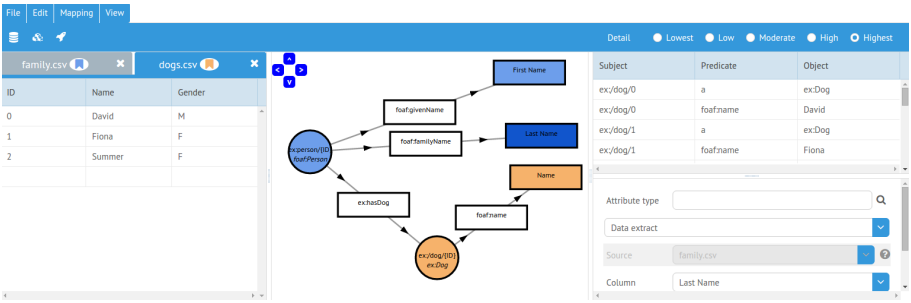


Figure 2.2: The RMLEditor with the *Input Panel* on the left, the *Modeling Panel* in the center and the *Results Panel* on the right

depend on the underlying language, this panel addresses Feature 1. Additionally, the graph representation and the RML rules can be exported (Feature 2), allowing the execution of the rules outside the RMLEditor. The *Results Panel* shows the resulting RDF dataset when the rules defined in the *Modeling Panel* are executed on the data in the *Input Panel*. For each RDF triple of the dataset it shows the subject, predicate and object. The functionality and the interaction between the panels supports the different rule generation approaches, as we described in previous work [21, 26] (Feature 6). The *data-driven* approach uses the input data sources as the basis to construct the rules. The classes, properties and datatypes of the schemas are then assigned to the rules. When users start with the vocabularies and ontologies to generate the rules, the *schema-driven* approach is followed. Next, data fractions from the data sources can be associated to the rules. Additionally, by not restricting users in when to interact with which panels – as would be the case for linear workflows – the RMLEditor supports non-linear workflows (Feature 7).

2.1.4.3 RMLProcessor server

As the RMLEditor uses RML, it needs the functionality of the RMLProcessor[14], a Java application which generates RDF based on provided RML rules. However, the processor is not needed to define the rules. For the RMLEditor's needs, the RMLProcessor's functionality is offered through a Web API. The Web API is developed using Node.js[15] and offers three functions: (i) executing a set of rules on a set of data sources; (ii) converting a GraphML document to RML to execute the rule using the RMLProcessor, and (iii) converting RML rules to a GraphML document to visualize the rules that are loaded in the RMLEditor.

[14] `https://github.com/RMLio/RML-Mapper`
[15] `https://nodejs.org/`

2.1.4.4 Real-life use cases

The RMLEditor is developed for a pilot (COMBUST[16]), initiated by companies in Flanders who aim to build their collaboration network on top of their interlinked data. The RMLEditor covers their need to generate the RDF representation of their raw data to be further used in other third-party applications. Moreover, the RMLEditor supports the partners of the Open Tourism working group of the Belgian Chapter of Open Knowledge Foundation (OKFN)[17] to semantically annotate their data with the Open Standard for Tourism Ecosystems Data[18] in the frame of the *Sustainable Mobile Guides for Tourism (in Flanders)* bootstrap. In both cases, the partners of the project have the raw data and the required knowledge about the domain. However, they have no understanding of knowledge graph generation languages, and without the use of the RMLEditor they need assistance of a Semantic Web expert. Additionally, for every update and execution of the rules the expert is consulted, which introduces significant overhead. We collected feedback during the deployment of the RMLEditor and concluded that they were able to create rules for their data using their own domain knowledge. This is confirmed during our exploratory user validation, which involved members of the aforementioned pilot (see Section 2.1.5). Furthermore, the RMLEditor is the topic of a number of workshops and tutorials for interested groups and users[19].

2.1.5 Exploratory user validation

We performed an exploratory user validation to assess the RMLEditor's adequacy to support users in defining rules that generate RDF datasets. In Section 2.1.5.1 we discuss the two use cases that the participants completed. In Section 2.1.5.2 we explain the two groups of participants, the apparatus and the procedure followed during the validation. In Sections 2.1.5.3 and 2.1.5.4 we elaborate on both the subjective and objective aspect of the validation. Finally, in Section 2.1.5.5 we discuss the results.

2.1.5.1 Use cases

In this section, two use cases are outlined and it is explained in more detail how the RMLEditor is used to create the rules. Each use case covers a different rule creation approach: *data-driven* or *schema-driven* [26]. The first use case involves data about employees and projects they work on, originally in two different data sources. The goal is to semantically annotate them, and link the employees to the projects they work on. The *data-driven* approach is recommended for editing the rules. Users start by loading the two data sources into the RMLEditor. Next, rules are created based on data fractions from the sources. Subsequently, users semantically annotate the rules.

[16] `https://www.imec-int.com/en/what-we-offer/research-portfolio/combust`
[17] `http://be.okfn.org/`
[18] `http://tourism.openknowledge.be:8080/spec/`
[19] `http://rml.io/RMLevents`

Identifying suitable classes, properties and datatypes is supported by LOV. The second use cases involves data about movies and their directors. The goal is to semantically annotate them, and to interlink each movie to the person that directed it. The *schema-driven* approach, supported by the RMLEditor, is recommended to the users for editing the rules. Users start by creating rules reusing movie concepts from the DBpedia Ontology[20] and person concepts from FOAF[21]. When the modeling is completed, the data sources are loaded. Subsequently, the rules are updated with the data fractions to be used for generating the final RDF dataset.

2.1.5.2 Method

2.1.5.2.1 Participants

15 participants from both Ghent University and the COMBUST network took part. They were divided in two major groups: (i) *Semantic Web experts* with 10 participants and (ii) *non-Semantic Web experts* with 5 participants. A *Semantic Web expert* is a user who has knowledge about Semantic Web technologies and standards, including knowledge graphs and RDF. A *non-Semantic Web expert* is not aware of the Semantic Web technologies, including the editor's underlying language RML. The *Semantic Web experts* are further distinguished in two sub-categories: (i) 5 experts with experience in publishing knowledge graphs, and (ii) 5 experts with no previous experience in publishing knowledge graphs.

2.1.5.2.2 Apparatus

The evaluation was carried out using the current version of the RMLEditor. Participants used their own computer with Chrome[22]. Both the RMLEditor and the RML-Processor were hosted on the same physical server. Each participant participated in both use cases described in Section 2.1.5.1.

2.1.5.2.3 Procedure

The following steps were followed during the evaluation:

Step 1 We gave a presentation[23] to the participants where basic concepts, such as knowledge graphs, RDF and schemas were explained. In order for them to understand the purpose of the RMLEditor. Additionally, the user interface's panels were described. However, no introductory tutorial regarding how to use the RMLEditor was given.

Step 2 We conducted a pre-assessment by asking the participants to fill in a questionnaire.

[20] `http://dbpedia.org/ontology/`
[21] `http://xmlns.com/foaf/0.1/`
[22] version 45.0.2454.101 or higher; `https://www.google.com/chrome/`
[23] `http://www.slideshare.net/secret/vI6AO0ywqzyk1m`

Step 3 Half of each group's participants started with Use Case A, and the other half with Use Case B. This was done to eliminate the influences of the use case-specific data and editing approach on the final results of the evaluation. Subsequently, they completed the other use case. During the execution of the use cases, the experts were able to ask questions when they had problems with the RMLEditor. It was recorded when intervention was needed. For the *non-Semantic Web experts*, closer observation was conducted and the think-aloud method [27] was taken into consideration. This allowed for the identification of specific usability issues related to the RMLEditor. After each use case, we conducted an assessment to inquire about the difficulty of the use case.

Step 4 (Experts with prior RML knowledge only) Half of the experts, with knowledge of the underlying language RML, also had to create rules using a plain text editor, to observe differences and preferences between use and non-use of the RMLEditor. However, this could not be done by all participants, as they do not possess the required knowledge.

Step 5 At the end of each use case, we collected the created rules. Additionally, a second and third questionnaire were filled in by the participants after their first and second use case, respectively.

Step 6 We conducted a post-assessment by providing the participants with a fourth questionnaire to fill in.

2.1.5.3 Subjective validation

We conducted a subjective validation of the RMLEditor by presenting the participants with four questionnaires during the validation in step 2, 5 and 6. With the pre-assessment we assess the participants expectations before interacting with the RMLEditor. For the pre-assessment's questionnaire, we used the System Usability Scale (SUS) scale [28]. The SUS statements were translated to the use of the RMLEditor. The intermediate questionnaires of step 5 allowed for a subjective self-assessment of the previously completed use case, and more specific the approach of the specific use case. With the post-assessment we assess the participants' experience with the RMLEditor and determine what the positive and negative aspects are of the RMLEditor. For the intermediate questionnaires and the post-assessment's questionnaire, we used the 7-point Likert scale [29] from "not difficult at all" to "very difficult" to measure difficulty, from "not useful at all" to "very useful" to measure usefulness, and from "not agree at all" to "very much agree" to measure the degree of agreement to a given statement.

2.1.5.4 Objective validation

After each completed use case, we collected each participant's rules. We conducted an objective validation of the RMLEditor by comparing all rules to the baseline rules

that we created ourselves. Additionally, if a participant also created rules using RML directly, it was compared with the participant's rules created using the RMLEditor.

2.1.5.5 Results

During the pre-assessment we measured that both groups of participants had high expectations regarding the ease-of-use and the improvements the RMLEditor would bring to the rule creation process. Interesting to note is that *Semantic Web experts* expected the tool to be hard to use if no introduction tutorial is provided, in contradiction to the *non-Semantic Web experts*.

Learning curve After having completed the first use case in row, we measured via the subjective validation that 60% of both the Semantic Web experts and non-experts found it difficult to use the RMLEditor to define rules that generate the RDF representation. After having completed the second use, all non-experts found it easier to use the RMLEditor. However, 30% of the experts still found it difficult to use the RMLEditor over all. This is partially due to the fact that they all had high expectations of the RMLEditor. All participants needed at least once help during the first in row use case. However, during the participants their second use case, 40% of the *Semantic Web experts* did not need any help at all anymore. However, all *non-Semantic Web experts* still needed help during their second use case. Nevertheless, overall, a significant lower level of intervention by us was needed. We deduced that the completeness and accuracy of the rules increased when users performed their second use case, compared to their first one. This shows that there is a learning curve to use the RMLEditor; however, once users learn how to use it, the RMLEditor is adequate for its scope. The latter is verified via subjective validation by the fact that 47% of the participants found that the overall usage of the RMLEditor was not difficult. However, still 20% found it too difficult to use. Again, this is partially due to the fact that the participants had high expectations of the RMLEditor. Even though the RMLEditor aims to eliminate language and RDF-specific terminology, through the observation during the validation we found that 40% of the participants had trouble with the used terminology in the GUI, such as "child" and "parent" when data sources need to be interlinked, and "templates" when the URIs of the entities are constructed based on a data fraction from the data source. However, once explained to the users, 90% of the users could use the terminology as intended.

Editing approaches 67% of the participants were able to start editing the rules using the given approach with no assistance at all. Through oral feedback during the evaluation, participants stated their preferred approach. Because of that preference, when they were asked to follow the other approach, they stated it was counterintuitive. If users prefer the *data-driven* approach, the *schema-driven* approach is counterintuitive, because no data is loaded initially. If users prefer the *schema-driven* approach, the *data-driven* is counterintuitive, because no predefined schema to be used is given.

Graph visualizations During the post-assessment, participants were asked to what extend they agree with the statement that the use of graph is beneficial for editing rules, and the statement that the graphs make the linked nature of the final RDF dataset clear. Through the subjective validation, we found that 82% of all participants found the use of graphs beneficial for editing rules, and that graph-based visualizations are adequate for conceiving how the final RDF dataset will be. The objective validation showed that in the case of *Semantic Web experts*, in only 15% of the total twenty use cases there were incomplete or inaccurate rules, regarding the modeling of the domain. In 67% of the use cases they were able to create rules as complete and accurate as expected.

Linking data sources In 33% of the use cases, participants missed the interlinking between multiple data sources when using RML, when we compared the rules created with the RMLEditor and the rules created directly using RML, via the objective validation. Through the post-assessment, we concluded that the links were missing because of the difficulties with the terminology of the language, or because the participants just forgot the interlinking. With the RMLEditor this only happened in 10% of the use cases. However, it still occurred because not all terminology was well covered in the GUI, as the participants made clear in the post-assessment.

2.1.6 Discussion & conclusions

During the assessment both groups of participants were able to generate a knowledge graph through the RMLEditor's GUI, with the graph-based visualization as the most important contributor. Therefore, the RMLEditor fulfills its initial goal to supports users to define, in a uniform way, rules that specify how to generate graphs. We present four main findings. First, non-Semantic Web experts are able to generate knowledge graphs when using the RMLEditor. Additionally, for experts, the quality of the knowledge graph is better when using the RMLEditor instead of RML directly, by looking at the rules created with the RMLEditor and the rules created by using RML directly. Therefore, second, when Semantic Web experts generate a knowledge graph using the RMLEditor, the resulting RDF dataset is of at least the same quality as the dataset generated when using directly RML, if quality metrics, defined by the Semantic Web experts, are kept constant. Furthermore, for example, when a relationship between two data sources exists; however, not made explicit by the user, this is reflected in the graph: a link between the resource nodes belonging to the sources is not present. Using RML directly, finding the missing relationship is more difficult. Therefore, third, graph-based visualizations of rules improve the interlinking between multiple data sources during the rule creation, compared to the use of no visualizations. Nevertheless, improvements to the RMLEditor, and more specifically to the GUI can be made during the next development sprints. As the main aspect of the RMLEditor is the use of graphs, it is important that manipulations on them are made as easy as possible. Although the required functionality for the manipulations is available, improving even

more its accessibility will benefit the rule creation process. Additionally, allowing users to determine the size of the graphs allows them to select the desired detail-level of their rules. As the participants where presented with two use cases, they were able to perceive a learning curve. Future evaluations need to be conducted to determine how steep this learning curve is. However, as the fourth finding, by providing a set of tutorials, where the features of the RMLEditor are discussed will already improve the initial use of the RMLEditor by new users. Topic of future research is the validation of these observations during further large-scale user testing. Moreover, the RMLEditor supports both the *data-driven* and *schema-driven* approach. Whether a user starts with the use case using the *data-driven* or the *schema-driven* approach, the second use case in row is less difficult. Therefore, there is no approach enforced on the user by the RMLEditor, and the preferred approach depends on the person and the circumstances [7]. Both Semantic Web experts and non-Semantic Web experts have different expectations from the RMLEditor. Non-Semantic Web experts expect that they can generate knowledge graphs without any understanding about a knowledge graph generation language or Semantic Web technology. Semantic Web experts can already generate knowledge graphs, as they can use RML directly. Therefore, they expect that they can at least do the same as with RML. Additionally, they have a set of requirements for their knowledge graphs, for which they look in the RMLEditor. These requirements improve, according to them, the quality of their generated graph. In the long run, we want to see these expectations united. Non-Semantic Web experts do not only generate knowledge graphs. They are also concerned with the quality of their knowledge graphs, in order to further improve its adoption.

2.2 Specification and implementation of knowledge graph rule visualization and creation: MapVOWL and the RMLEditor

Pieter Heyvaert, Anastasia Dimou, Ben De Meester, Tom Seymoens, Aron-Levi Herregodts, Ruben Verborgh, Dimitri Schuurman, and Erik Mannens

Published as "Specification and Implementation of Mapping Rule Visualization and Editing: MapVOWL and the RMLEditor" in *Journal of Web Semantics* 49, 2018, pp. 31 – 50.

Abstract

Visual tools are implemented to help users in defining how to generate knowledge graphs from raw data. This is possible thanks to knowledge graph generation languages which enable detaching rules from the implementation that executes them. However, no thorough research has been conducted so far on how to visualize such rules, especially if they become large and require considering multiple, heterogeneous raw data sources and transformed data values. In the past, we proposed the RMLEditor, a visual graph-based user interface, which allows users to

easily create rules for generating knowledge graphs from raw data. In this article, we build on top of our existing work: we (i) specify a visual notation for graph visualizations used to represent rules, (ii) introduce an approach for manipulating rules when large visualizations emerge, and (iii) propose an approach to uniformly visualize data fraction of raw data sources combined with an interactive interface for uniform data fraction transformations. We perform two additional comparative user studies. The first one compares the use of the visual notation to present rules to the use of a knowledge graph generation language directly, which reveals that the visual notation is preferred. The second one compares the use of the graph-based RMLEditor for creating rules to the form-based RMLx, which reveals that graph-based visualizations are preferred to create rules through the use of our proposed visual notation and uniform representation of heterogeneous data sources and data values.

2.2.1 Introduction

Nowadays knowledge graphs still stem from (semi-)structured formats. A few of the most well-known and larger knowledge graphs[24] [30] are: DBpedia dataset[25] [19], with approximately 1.39 billion triples derived from Wikipedia[26] where the data is originally represented in the wikitext syntax; Linked Geo Data[27] [31], with approximately 1.38 billion triples derived from Open Street Map planet files loaded in multiple databases; UniProt [32] (UniProtKB, Uniref and UniParc), with approximately 45 billion triples across 3 datasets derived from the UniProt Knowledgebase[28]; and Bio2RDF[29] [33], with approximately 11 billion triples across 35 datasets.

Overall, knowledge graph generation includes the following: (i) *multiple, heterogeneous raw data sources* whose data values might need transformation, (ii) *ontologies* used to annotate the RDF terms [34] that are generated from the data fractions of the different data sources, (iii) actual *rules* that define how data fractions are semantically annotated and used to generate RDF terms and triples, and (iv) a *generated knowledge graph*.

Several datasets are generated by tools that incorporate directly in their implementation how knowledge graphs are generated. This means when new or updated semantic annotations are needed, knowledge of Semantic Web technologies is required, as well as dedicated software development cycles for adjusting and extending the implementations.

To the contrary, rules may also be defined, according to a specified knowledge graph generation language's syntax and grammar, such as R2RML [5] or RML [6]. Knowledge graph generation languages define declaratively how terms are generated

[24] `http://stats.lod2.eu/rdfdocs?sort=triples`
[25] `http://dbpedia.org`
[26] `http://wikipedia.org`
[27] `http://linkedgeodata.org`
[28] `http://www.uniprot.org/help/uniprotkb`
[29] `http://bio2rdf.org/`

from corresponding raw data and annotated with ontology terms to form the desired knowledge graph. This way, rules are detached from the implementation that executes them. Nevertheless, knowledge of the underlying language is required to define rules, while manually editing and curating them requires a substantial amount of human effort [1]. Therefore, the creation of rules still remains complicated.

To this end, a significant number of rule editors were implemented to facilitate rules creation and editing, such as Map-On [2], the RMLEditor [1], and RMLx [4]. Only a few of them provide graph-based visualizations, although user evaluations suggest that such visualizations are suitable for supporting users to intuitively generate their desired knowledge graph [1].

Nevertheless, how such user interfaces should be designed is not thoroughly investigated so far: (i) a *visual notation* specification for rules does not exist. Such a specification would provide a formal description for rules and allows multiple tools to implement it, improving the accessibility for users across different tools; (ii) current rule editors do not uniformly present *multiple, heterogeneous data sources*. Therefore, creating rules that define relationships between heterogeneous data sources is not always straightforward; (iii) *data value transformations* cannot be defined in current visualization-based rule editors; (iv) *scalability* is not thoroughly addressed. Even if graph-based visualizations are used, large graphs cause difficulties to users when editing the corresponding rules.

In this work, we present and extend our ongoing work towards a uniform graphical user interface (GUI) to create and edit knowledge graph generation rules. Such a GUI is implemented in the RMLEditor, which we presented in the past. The RMLEditor offers a graph-based interface for specifying rules for raw data to generate knowledge graphs. Its target group of users have knowledge about both knowledge graphs and the domain of the data. The rules creation and editing is based on graph visualizations, without requiring knowledge of the underlying knowledge graph generation language. Our novel contributions include in particular:

- a rich graph-based visual notation for knowledge graph generation rule visualization;
- an approach for manipulating rules when large visualizations emerge;
- an approach to uniformly visualize data fractions of data sources combined with an interactive interface for uniform data fraction transformations;
- an implementation of these three contributions in the RMLEditor; and
- additional evaluations to compare the use of
  - the visual notation to present rules to the use of a language directly, which reveals that the visual notation is preferred; and

- the graph-based RMLEditor to create rules to the form-based RMLx, which reveals that the RMLEditor is preferred to create rules through its use of the visual notation and uniform representation of heterogeneous data sources and data values.

The remainder of the paper is structured as follows. In Section 2.2.2, we elaborate on knowledge graphs, rules, and RDF terms. In Section 2.2.3, we discuss related work. In Section 2.2.4, we introduce our research questions and hypotheses. We present in Section 2.2.5 our proposed visual notation for rules, and in Section 2.2.6, the RMLEditor. In particular, we elaborate in Section 2.2.6.5 on the manipulation of large graphs in the RMLEditor, and in Section 2.2.6.6 on how the RMLEditor deals with heterogeneous data sources. In Section 2.2.7, we present the evaluations and the results both for the visual notation and the new version of the RMLEditor. In Section 2.2.8, we summarize this work's conclusions.

2.2.2 Preliminary

Nowadays, *RDF* [34] is the prevalent framework to represent knowledge graphs. Most of the time, knowledge graphs originally stem from (semi-)structured formats (CSV, XML, and so on). Their RDF representation is obtained by repetitively applying rules according to an iteration pattern which specifies the extract of data that is considered during each iteration.

Rules define correspondences between data in different schemas [35]. In the case of knowledge graph generation, rules often define how RDF terms, i.e., IRIs, literals or blank nodes [34], are generated from data fractions derived from one or more data sources which are annotated with ontologies. These RDF terms are used to form RDF triples.

With the term *data fractions*, we do not only mean *raw data values* as they are in the original data source, but also *transformed data values* that result from a raw data value after applying a function to process the original data values. Data value transformations are needed to support changes in the structure, representation or content of data, such as string transformations. For instance, when a data fraction contains a date in the format "DD-MM-YYYY", it might be needed to be transformed to the format "YYYY-MM-DD".

References to raw or transformed data fractions might be combined with constant values (*template-valued* [5]) or only constant data values (*constant-valued* [5]) might be used to form the desired RDF terms. As such, a complete rule includes (i) a reference to zero or more raw data fractions and (ii) one or more ontology terms. Therefore, when visualizing a rule, its interrelation with the raw data as well as with its semantic annotation with an ontology term should be visualized when presented to and edited by users.

2.2.3 State of the art

The creation and editing of rules, as well as their visualization is closely related to knowledge graphs, ontologies, and query visualizations (Section 2.2.3.1), and is performed using different types of rule editors (Section 2.2.3.2).

2.2.3.1 Interfaces for Semantic Web visualizations

In this section, we elaborate on visualizations for different aspects of the Semantic Web: knowledge graphs, ontologies, and queries which become relevant. We discuss the characteristics of each aspect and their visualizations that are implemented in existing tools or formalized as a specification.

2.2.3.1.1 Knowledge graph visualizations

As rules resample how knowledge graphs will eventually be generated [1], visualizations that are applicable for knowledge graphs would be expected to be applicable for rules visualizations too.

Efforts to improve knowledge graph accessibility, resulted in tools offering either (i) *text-based* presentations, or (ii) *visualizations*. Dadzie and Rowe [36] conducted a survey on both approaches. They concluded that text-based solutions, such as Sig.ma [37], (i) fail to provide an overview of the available information, and (ii) are only suitable for users with understanding of the underlying technologies, whereas lay users need additional support.

Visual presentations lead to approaches relying on network maps [38], diagrams [38], geographic maps [39], timelines [39], charts [40], and graphs [41, 42, 43]. The latter was the default in the past according to Dadzie and Pietriga [44], because (i) ontologies are often hierarchically structured and used to annotate knowledge graphs; (ii) RDF's data model is a directed labeled graph [34]; and (iii) network analysis is one of the most common visualization-driven tasks carried out within the field, to explore, e.g., collaborations and other interrelationships between researchers within research data, and social networks at large. However, they fail to provide meaningful visualizations when graphs become large, even when styled to better convey the resources' and properties' semantics [45]. Furthermore, exposing the data's graph structure is not always needed, because the model may be of little importance to users [44]. Therefore, current efforts are more focused on user interface components designed for different types of data, such as temporal and geographical data. Nonetheless, graph visualizations can still be used, but they are no longer the main component to be represented.

Regardless of the used visualization, research is being done to improve the support for large datasets. For example, Bikakis et al. [46] introduce a generic model for organizing, and analyzing numeric and temporal data in a multilevel fashion to deal with the challenges that come with the use of large datasets, such as information overload.

The model is not tied to a specific visualization and, thus, it can be used with any of the aforementioned tools to improve the support for such datasets.

2.2.3.1.2 Ontology visualizations

Ontology visualizations are also close to knowledge graph generation rule visualizations. Rules define the modeling, namely the conceptualization, of raw data as knowledge graphs, using ontologies. Thus, approaches applicable for ontology representation which visualize the schema could be adjusted for visualizing rules too. To improve the ontology presentation and editing, a number of tools were developed. Such tools offer visualizations that represent ontology-specific elements. Only a limited number of efforts defined a specification for such visual notations that can be implemented by any other tool.

2.2.3.1.2.1 Tools

Graphs offer a natural way to depict the structure of ontological elements and relationships [47]. Therefore, graph-based visualizations are present in a high number of visualization tools for ontologies, such as GrOWL [48], OWLViz [49] and KC-Viz [50]. In these cases, classes and datatypes are represented as nodes and properties as edges. However, when similar graphical notations are used for different ontological elements it is difficult for users to distinguish them.

Furthermore, a number of graph-based visualizations are focused on a single task, e.g., providing insights regarding the class hierarchy, and neglect other aspects of ontologies, including the properties and datatypes. This makes them less suitable for other tasks [47], such as gaining insights in the relationships between the classes via the different properties. When visualizing large ontologies the graphs might become too large for users to cope with. Different approaches, applied by several tools [50, 51, 52], were developed to tackle this challenge: (i) showing detailed information only on demand, i.e., selecting a specific node results in displaying more detailed information about that node; (ii) displaying the parts of the graphs that are selected by a given filter; (iii) zooming in and out on the graphs, enabling them to gain more information about a specific region of interest of the graph.

So far, ontology visualizations were focused on presenting the ontologies. Nevertheless, more and more tools are built to facilitate editing of ontologies [14, 53], as ontologies are subject to change. This leads to the development of visualization plugins that show the (updated) ontology inside the tool [54]. However, such an approach disconnects the visualization of the ontology from its editing and requires users to understand both the visualization and the interface used to perform the updates. This is overcome when graph-based visualizations allow users to update the nodes and edges [48, 55]. The updates will result in the corresponding changes in the ontology. TurtleEditor [55], for instance, uses both methods: text editor and visualization to present and update the ontology, leaving the choice to the users.

2.2.3.1.2.2 Specifications for visual notations for ontologies

In most cases, tools create a new stand-alone visualization to present the ontology [50, 52]. However, when users switch between tools they need to learn a new visualization to work with the new tool. Recent efforts result in the creation of specifications for visual notations. They provide a formal description of the visual elements used to convey the required information to users. These specifications are independent of the tools that implement them, and, thus, can be applied by multiple tools. The latter allows to improve the user's ability to switch between these tools as they provide the same predefined visualization [56]. The most significant specifications are Graffoo [56] and VOWL [47].

The Graphical Framework For OWL Ontologies (Graffoo) [56] defines a graph-based visual notation for ontologies. Classes are represented with yellow rectangles and datatypes with green parallelograms. The properties are represented with lines connecting the rectangles or parallelograms, and have different colors depending on the property's type, i.e., object and datatype property. Graffoo is implemented in yEd[30], a diagram editor. However, it does not propose an intuitive ontology visualization that is immediately understandable to casual users due to its diagrammatic approaches [47].

The Visual Notation for OWL Ontologies[31] (VOWL) defines a visual language for user-oriented representation of ontologies and provides graphical depictions for elements of the Web Ontology Language (OWL) [47]. The initial specification of VOWL [57] focused on the visualization of ontology elements (i.e., classes, properties, and datatypes), together with the dataset's instances, the so-called TBox and ABox in Description Logic, respectively. However, based on their user study, they concluded that even when visualizing only a few instances, providing additional information already leads to difficulties for understanding the visualization.

Therefore, VOWL2 focuses on visualizing only the TBox. It relies on directed graphs to visualize ontologies, and is implemented in WebVOWL [51], a web application. Classes are represented by blue, circular nodes, and datatypes by yellow, rectangular nodes. Links are used to visualize how classes are related to other classes or datatypes through properties. Graphical elements are added to links between classes to represent characteristics, such as disjoint and union, while the use of colors helps identifying the different elements.

2.2.3.1.3 Query visualizations

Query visualizations aim to hide the query language to end users, whereas rule visualizations aim to hide the rule language. Thus, approaches applicable for query representation could be adjusted for visualizing rules.

Query formulation is important for the retrieval of information available as knowledge graphs. SPARQL [58] is the standard query language for knowledge graphs de-

[30] `http://www.yworks.com/products/yed`
[31] `http://purl.org/vowl/spec/`

scribed using the RDF framework. However, besides users from the Semantic Web community, lay users and domain experts from different areas, i.e., users without knowledge about this language, need support to define valid queries that provide the desired results [59]. Tools, such as NITELIGHT [60], RDF-GL [61], and FedViz [62], were developed to make it easier to work with a query language by removing the burden of dealing with the language's syntax. They apply graph-based visualizations to represent the data's underlying RDF structure. However, knowledge about SPARQL is still required, which makes them less usable for lay users, with exception of FedViz. This tool allows to browse knowledge graphs through a graph-based visualization (see Section 2.2.3.1.1). Once the users have found the desired data, FedViz generate the corresponding SPARQL query.

Nonetheless, visually spotting the difference between different aspects of a query or knowledge graphs is difficult as the same graphical notation is used to denote different aspects. QueryVOWL [59] addresses this ambiguity. It is a specification for a visual notation for queries that uses graph visualizations, as it is based on VOWL. QueryVOWL specifies a visual query language, with the goal to be accessible for lay users while still preserving most of SPARQL's expressiveness. Referents are represented as circular nodes and literal values as rectangular nodes. A referent's class is added inside the node as text. The same is done for the datatype of a literal value. Links represent the relationships between nodes, using properties. Different colors are used to make distinction between classes, instances, and literals. Icons represent the update actions to the queries, such as add and delete.

As queries can become large, it is required to support large graphs. Similar to ontology visualizations, different approaches were proposed to tackle the corresponding challenges. For example, details are presented on-demand when users click on a part of the query [59], specific details are shown by applying filters on the query's results [59], or users can zoom in on specific parts of the query through the corresponding buttons [60].

Specific tools are also developed for specific cases, such as SPEX [39] for spatial and temporal data. Their goal is to provide users with a GUI to explore the data and, subsequently, to help with query construction. Scheider et al. [39] introduced design principles to achieve this goal, such as the use of the results' feedback into the construction and the automatically handling of space and time data. SPEX, which follows these principles, includes also a graph-based visualization, similar to the aforementioned tools. However, these tools do not follow the design principles making them less usable than SPEX.

2.2.3.2 Rule editors

Research efforts to facilitate the rule creation and editing to generate knowledge graphs resulted in the development of two types of graphical user interface (GUI) tools: (i) *step-by-step wizards* and (ii) *visualization tools*.

Table 2.1: Comparison of rule visualization and editing of RMLx, Map-On, and the RMLEditor

Rule editor	**Rule visualization**	**Rule editing**
RMLx	Rules visualized as knowledge graph	Step-by-step wizard
Map-On	Separate visualizations for data and ontologies	Align both visualizations
RMLEditor	One visualization for both data and ontologies	Edit visualization

Table 2.2: Comparison of knowledge graph visualization and support for multiple, heterogeneous data sources of RMLx, Map-On, and the RMLEditor

Rule editor	**Knowledge graph visualization**	**Multiple, heterogeneous data sources**
RMLx	No visualization	Yes
Map-On	No visualization	No
RMLEditor	List of RDF triples	Yes

In the past, **step-by-step wizards** prevailed as an easy-to-reach solution, such as the fluidOps editor [3]. However, such applications restrict data publishers' editing options, hamper altering parameters in previous steps, and detach rules from the overall knowledge modeling, since related information is separated in different steps. These tools circumvent dealing with the underlying languages' syntax, but users are still required to have knowledge of the language's terminology, limiting thus the support for users who do not have this knowledge.

More recent tools incorporate **graph-based visualizations** to present rules. A limited number of them also apply these visualizations to edit the rules, such as Map-On [2] and the RMLEditor [1] (see Tables 2.1 and 2.2). We distinguish three prevalent approaches for rule visualizations: (i) visualizing rules as knowledge graphs; (ii) separate graph visualizations for the raw data and ontology; (iii) single graph visualization for both raw data and ontologies.

The first approach is applied by RMLx [4]. It visualizes rules by using graphs as if the rules are knowledge graphs, aiming to present them to the users rather than editing them. This is possible when the rules are defined in RDF syntax, which is the case for R2RML and RML, while editing is still performed relying on a *step-by-step wizard*. Consequently, users (i) need to understand the language's terminology to correctly interpret these graph visualizations; (ii) can only view and not edit the graphs, so rules can only be edited using forms; (iii) need to "imagine" how the triples will look like as the visualization does not communicate how the rules result in the corresponding RDF triples.

The second approach is applied by Map-On. Two graphs are created: one that represents the raw data structure and one that represents the target ontology which is predefined and cannot be adjusted after being loaded. Rules are created by aligning the two graphs, i.e., aligning the data fractions with the corresponding ontology elements. The two graphs are distinguished from each other using different colors. However, by visualizing both the raw data and ontology in the same visualization, the graphs quickly become cluttered and which aggravates when both of them are large.

The third approach is applied by the RMLEditor. The visualization aims to represent the RDF triples that the rules generate. The nodes represent how RDF terms of subjects and objects are generated, while the edges represent the predicates' RDF terms. More details regarding the RMLEditor are available in Section 2.2.6.

2.2.4 Problem statement

Based on the aforementioned, we notice that *graph-based visualizations* is the dominant approach when tools present knowledge graphs, ontologies, and queries to users, while these visualizations also have revealed benefits for visualizing rules. Knowledge graph and ontology visualizations mainly aim to present to the users, whereas queries require the users to actively interact with and shape the visualized object, as it also occurs with rules editing. Moreover, so far, Semantic Web related visualizations present only one of the components involved in knowledge graph generation, namely either knowledge graphs or ontologies, which do not require to interrelate multiple components. However, the rule definition involves the interrelation of the different involved components.

A number of open issues remain regarding rules visualization and editing: (i) the definition of a *visual notation specification for graph-based visualizations of rules*, (ii) *scalability* issues when large graphs emerge, (iii) *uniform representation for heterogeneous data sources*, and (iv) *data values transformation* integration with the presentation of these data sources. More details are presented in Section 2.2.4.1, followed by our research questions and hypotheses in Section 2.2.4.2.

2.2.4.1 Open issues

Specification The development of a visual notation specification increases the visualization's adoption by other tools. Current research efforts and rule editors neither specify nor implement such a specification. This impedes the users accessibility.

Scalability Dealing with large graphs is an important challenge for graph-based visualizations. The same occurs when rules are edited by using graphs. However, none of the existing tools tackle this challenge, besides barely applying zooming. The latter is not always sufficient as for users it is not always straightforward to know on which part of the graph they need to zoom in.

Heterogeneity Knowledge graphs might originally stem from multiple data sources with heterogeneous data formats. Therefore, rules and their corresponding visualizations need to support these data sources. Most existing tools support neither multiple nor heterogeneous data sources. The RMLEditor and RMLx are the only ones that support multiple, heterogeneous data sources. The latter does not show the raw data, while the former does. Nevertheless, even only showing the raw data makes it difficult to derive the data model and especially its data fractions.

Data transformations Although, transformations are required when raw data values are desired to be altered to generate RDF terms, only RMLx provides this functionality. In other cases, they are often implemented as custom solutions, or addressed by dedicated applications, thus the range of possible transformations is limited, and cannot be visualized uniformly to the users.

2.2.4.2 Research questions and hypotheses

Given these open issues, we define the following two research questions:

Research Question 1: *How can we design visualizations that improve the cognitive effectiveness of visual representations of knowledge graph generation rules?*

Research Question 2: *How can we visualize the components of a knowledge graph generation process to improve its cognitive effectiveness?*

This question has two sub-questions:

- How to deal with large rules graphs?
- How to uniformly visualize heterogeneous data fractions and their transformations?

Note that cognitive effectiveness is defined as the speed, ease, and accuracy with which a representation can be processed by the human mind [63].

To address Research Question 1, we introduce a visual notation for rules called MapVOWL (see Section 2.2.5). To address Research Question 2 and its sub-questions, we introduce an approach to improve the understanding of large graph-based visualizations (see Section 2.2.6.5) and an approach to deal with heterogeneous data sources and data values (see Section 2.2.6.6). These approaches are implemented in the RMLEditor. Furthermore, these questions lead to two corresponding hypotheses which are validated through two comparative user studies (see Section 2.2.7):

Hypothesis 1: *MapVOWL improves the cognitive effectiveness of the generation rule visual representation to generate knowledge graphs compared to using RML directly.*

Hypothesis 2: *The cognitive effectiveness provided by the RMLEditor's GUI improves the user's performance during the knowledge graph generation process compared to RMLx.*

RML was chosen over other knowledge graph generation languages for Hypothesis 1 as it supports rules with data from multiple, heterogeneous data sources. RMLx was chosen over other existing rule editors for Hypothesis 2 as it allows to annotate multiple, heterogeneous data sources and values. More, it also uses graph visualizations to represent the rules; however a form-based approach is used to edit the rules.

2.2.5 Visual notation for knowledge graph generation rules

In the past, we showed that graph-based visualizations are applicable for presenting knowledge graph generation rules [1]. It was also confirmed to be an intuitive way to represent the structure of ontologies in a comparative evaluation [47]. So, we rely on *graph-based ontology visualizations* and we investigated how to apply their approaches on corresponding *graph-based rule visualizations*. Therefore, we based the specification for a rule visualization on VOWL, leading to MapVOWL.

2.2.5.1 Requirements

The notation needs to support rules visualization that generate triples which, on their own turn, conform with the RDF specification. Each triple consists of three elements: subject, predicate, and object, which are RDF terms: IRI, blank node, or literal. A literal might have a datatype or language.

These terms can be generated based on fractions from the raw data, constant values, or a combination of the two. The data can be annotated by classes, properties, and datatypes from different ontologies.

Furthermore, a notation has to be cognitively effective. In the case of rules this includes the understanding of the different components of the knowledge graph generation process and their relationships: multiple, heterogeneous raw data sources, ontologies, rules, and the generated knowledge graphs. Therefore, a cognitive effective notation needs to support rules that define how to generate

Requirement 1: *a RDF term: IRI, blank node, and literal;*

Requirement 2: *a triple's subject (IRI/blank node), predicate (IRI), and object (IRI/blank node/literal).*

More, a notation should support references to

Requirement 3: *data values of data sources*

Requirement 4: *constant sources*

Requirement 5: *a combination of data and constant values*

Requirement 6: *ontological elements from various ontologies.*

In knowledge graph visualizations similar requirements were fulfilled in order to depict the RDF's data model, together with the used ontologies. However, they did not include the need to visualize the relationships with the raw data.

2.2.5.2 MapVOWL

We introduce MapVOWL, a user-oriented visual notation for rules which defines how knowledge graphs are generated from raw data. MapVOWL builds on top of VOWL's graphical elements [47]. Relying on MapVOWL's unified graphical elements, the rules can be created and edited entirely using visual representations, while the rules in the underlying language's syntax are generated by the rule editor without user intervention. This way, MapVOWL hides the underlying language, as QueryVOWL hides the query language.

Overall, MapVOWL aims to make rules creation and editing for generating knowledge graphs more accessible to knowledge graph experts, even more for those who already know VOWL or QueryVOWL. A comprehensive and uniform visual representation would be beneficial and would have great impact particularly on users with knowledge about the underlying language, as it occurs in the case of ontology visualizations [64].

The MapVOWL specification can be found at `http://rml.io/mapvowl`. Moreover, we applied MapVOWL to the RMLEditor. You may access a demo instance of the RMLEditor with MapVOWL implemented at `http://rml.io/jws-demo`. Moreover, you may find a screencast explaining the graph elements of MapVOWL as adopted in the RMLEditor at `http://rml.io/jws-screencast`.

In Section 2.2.5.2.1, we present the graphical primitives. In Section 2.2.5.2.2, we discuss the color scheme. In Section 2.2.5.2.3, we elaborate on the visual elements. In Section 2.2.5.2.4, we discuss how MapVOWL improves the cognitive effectiveness.

A user study between MapVOWL and RML compares the use of the visual notation to present rules to the use of a language directly (see Section 2.2.7.1), while a user study between the RMLEditor implementing MapVOWL and RMLx compares the use of a *graph-based* GUI to a *form-based* GUI to create rules (see Section 2.2.7.2).

2.2.5.2.1 Graphical primitives

MapVOWL provides a small set of unambiguous graphical primitives based on VOWL, as shown in Table 2.3. It is designed to be modular and, thus, it distinguishes the

Table 2.3: Graphical primitives

Primitive	Application
	referents
	blank nodes
	relationship labels and literal values
	relationships
	relationship direction
text	textual information about rules

primary elements that are crucial for the knowledge graph generation, such as the RDF terms to be generated, from elements which provide supplementary details, such as their (data)types.

Referents and literals are depicted as nodes of the rule graph visualization, whereas the relationships between them form the graph's edges. This choice is made because the shape plays a special role in discriminating between symbols as it represents the primary basis on which we identify objects in the real world; the shape is the primary visual variable for distinguishing between different constructs.

Shape In the case of VOWL, the shape is the graphical primitive used to distinguish the classes from datatypes which are the two fundamental constructs of ontologies. Following the same principle, in the case of rule visualizations, the type of the RDF term which will be generated is the most fundamental construct. Thus, deviation in the node's shape determines whether a referent (IRI or blank node) or a literal will be generated.

In the case of rule visualizations, the constructs which can have a class assigned to them are depicted as circles (Requirement 1), whereas the rest has a rectangular shape (Requirement 1). This follows VOWL where classes are depicted as circles and datatypes as rectangles. For the classes, there is no restriction on which ontologies to use (Requirement 6). Constructs that are not meant to represent an entity, but are an attribute of an entity are depicted as rectangles, similarly to datatypes in VOWL and might be assigned a datatype or have the default one.

Edge MapVOWL considers directed edges whose label is defined in a rectangle to associate referents with each other or with literals. MapVOWL uses edges as VOWL does to represent the properties which will be generated or (re)used to associate the entities among each other or the entities with the attributes (Requirement 2). However, even though in VOWL the property's label is detached from the edge, MapVOWL incorporates the label in the edge using a rectangle. This way, we make it explicit that a property is another RDF term as a referent or a datatype valued node and it could be defined under the same conditions as the rest of them. Namely, we do not distinguish generation of a property from the other RDF terms. Last, a property's direction is indicated by an arrowhead, as it is for VOWL too. This states the property's subject and object (Requirement 2).

Text While nodes and edges depict rules, text is used to provide additional information about these rules. Text is added to a circle to denote (i) its class, and (ii) how the IRIs are formed, either template-valued or constant-valued (see Section 2.2.2; Requirements 3 to 5), and to a rectangle to denote (i) the datatype or language tag (Requirement 1), and (ii) how the literal values are formed, either template-valued or constant-valued (Requirements 3 to 5). For the datatype there is no restriction on which ontologies to use (Requirement 6). When a language is specified, the language tag is preceded with "@" and the datatype is automatically set to `rdf:langString` [34]

Border As borders can convey meaning, they are considered as a distinct graphical primitive used to represent a certain notion. A border is only present when an entity is a blank node and then it is dashed, following VOWL's choice for using dashed borders and properties' lines for special classes and properties. In the case of VOWL, dashed circles are used when circles represent `owl:Thing` and, thus, they do not carry relevant domain information. Similarly, blank nodes might be generated and annotated; however, they differ from typical entities, as the ones related to blank nodes do not receive IRIs.

Size Similarly to VOWL, MapVOWL also recommends to vary the node size if possible and desired, without determining the scaling method though. VOWL associates the node size with the number of individuals which are members of a class. However, this is not possible in the case of rule visualizations as the individuals are not generated yet. Although, this number may be estimated by analyzing the data source for the specified data fraction(s) from which the individuals are generated in the end. If the analysis is not possible or desired, all nodes should have the same predefined size.

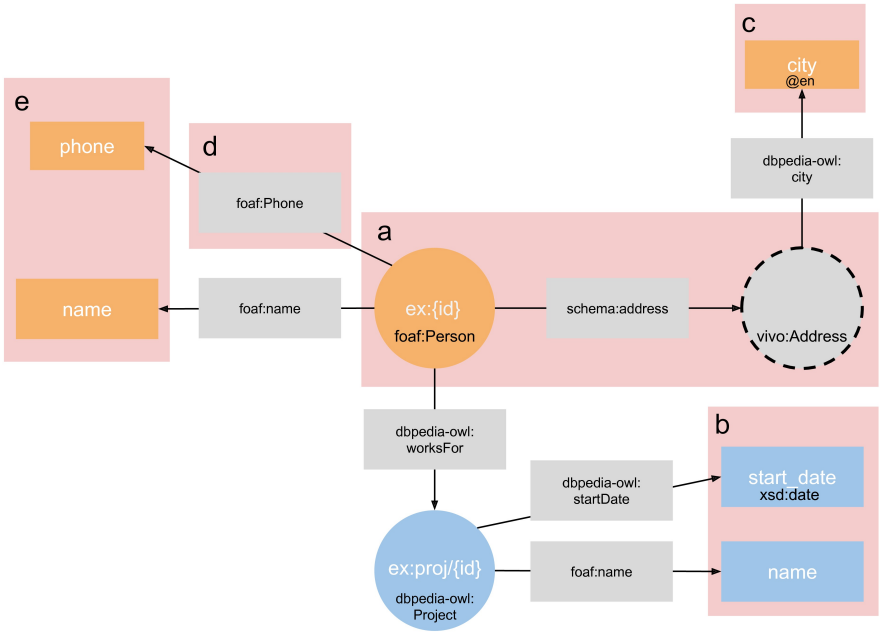


Figure 2.3: Examples of visual elements

2.2.5.2.2 Color Scheme

Color A color scheme is employed to interrelate data sources and rules. In contrast to VOWL where the color indicates different types of classes, in MapVOWL the color depends on the data source the term is (partially) derived from. If the color of a node is grey, it is not associated to any data source (yet). A color scheme is recommended by MapVOWL, but any other color scheme may be considered. Alternatively, a schema that relies on texture usage instead of different colors may be considered to support color-blinded people if needed.

Brightness In addition to color schema, the brightness level is employed to make the visualization more comprehensive. Changes on the brightness level occurs when a user interacts with the graph. To be more precise, the brightness is higher when a graph element is not selected and darker once the element is selected by the user.

2.2.5.2.3 Visual Elements

MapVOWL defines visual elements for rules. These elements are based on the aforementioned graphical primitives and color scheme. Figure 2.3 shows examples of these

elements.

Rules can be created to generate referents, i.e., they will be represented by an IRI or a blank node which can be annotated with a class to determine its type. For instance, in the examples, referents are generated for employees and their addresses (see Figure 2.3a). For each employee a IRI is generated, using a template, and for each address a blank node, thus its node remains gray as its definition does not depend on any data source. Their classes are `foaf:Person` and `vivo:Address`.

Rules can be created to generate literals, i.e., they are represented with the default datatype or a user-specified datatype. For instance, in our example, literals are generated for the names of projects. It has the default datatype `xsd:string`. The start date of a project has the user-specified datatype `xsd:date` (Figure 2.3b). When a literal value is a string, users can denote the value's language. For instance, in our example, the city of an address is provided in English (Figure 2.3c).

Rules can be created to generate properties, which can be represented by an IRI only. In our example, properties are used to denote the relationship between employees and their corresponding literals, such as their phone number using the property `foaf:phone` (see Figure 2.3d).

Nodes have different size based on the available data in the data source, unless the information is not available (either deliberately or ignored). In our example, all employees have a name; however, not all employees provided their phone number. This is reflected in the size of the nodes that represent these literals (see Figure 2.3e).

The overall visualization is combined to a graph that represents the envisaged model for the knowledge graph. The position of the graph elements is imposed by a force-directed graph layout algorithm. This means that the highly connected nodes are placed more together, which puts more focus on their relationships.

2.2.5.2.4 Conform to Physics of Notations

The notation also needs to be designed to be cognitively effective. To achieve this we rely on the design theory "Physics of Notations" by Moody [65]. The theory presents a set of principles to which a notation should adhere to be effective. The principles are semiotic clarity, perceptual discriminability, semantic transparency, complexity management, visual expressiveness, dual coding, graphic economy, coding fit, and cognitive integration. As a result the notation does not just only fulfill the requirements, but does this in a cognitive effective manner to improve the usability.

2.2.5.2.4.1 Semiotic clarity

When specifying a visual notation it is important to consider the principle of semiotic clarity as it impacts the notation's effectiveness of addressing the problem it tries to solve [65]. Semiotic clarity is determined by the correspondence between the semantic constructs and the graphical notations. In the case of MapVOWL, the nodes and edges of the graphs form the graphical notations, and the possible combinations

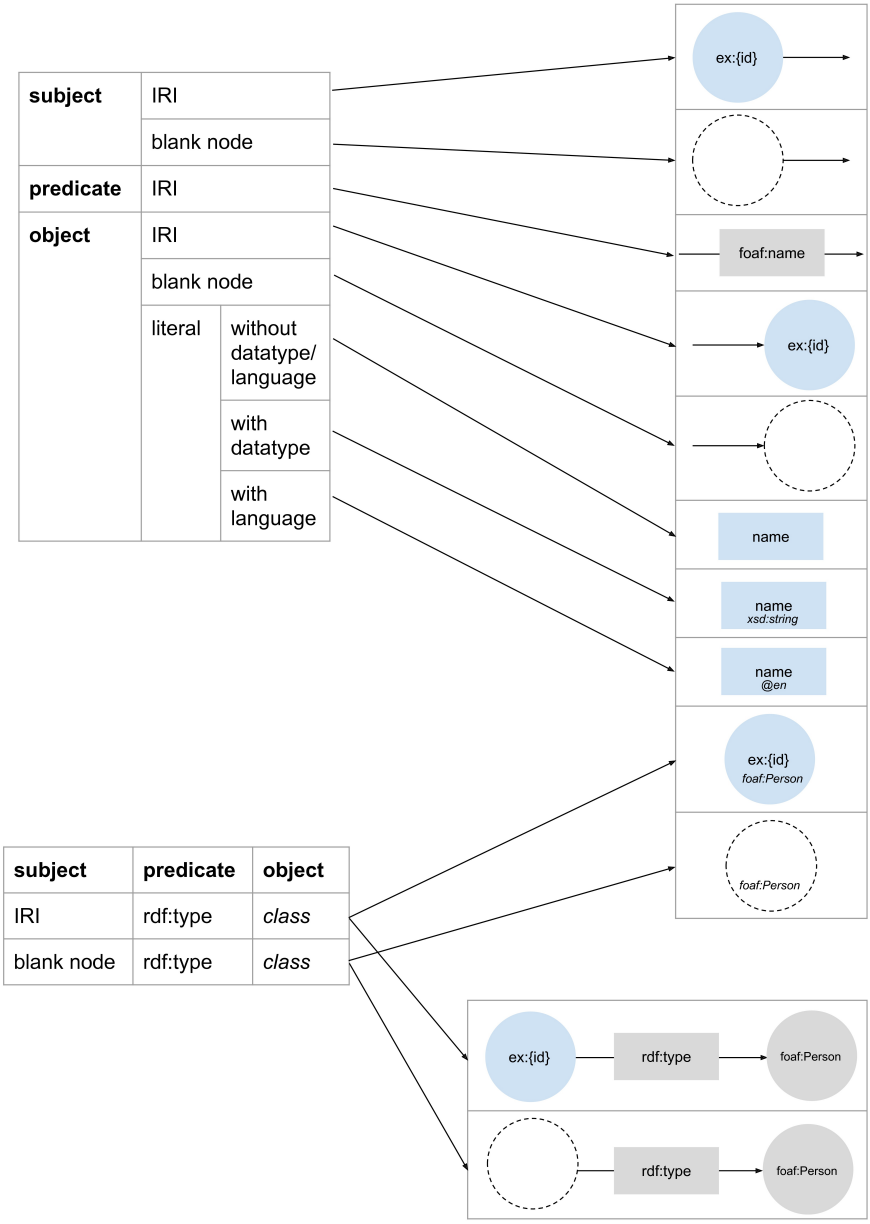


Figure 2.4: Semantic constructs (left) and graphical notations (right)

of RDF terms, which the user can create to form the generated triples, are the semantic constructs (see Figure 2.4). The semiotic clarity is determined by the presence of four anomalies: (i) symbol redundancy, (ii) symbol overload, (iii) symbol excess, and (iv) symbol deficit.

2.2.5.2.4.1.1 Symbol redundancy When a semantic construct can be represented by multiple graphical notations, called symbol redundancy, it has two negative effects on the user. On one hand, users have difficulties deciding which notation to use. On the other hand, users need to remember multiple notations for a single construct.

In the case of MapVOWL, a subject, which might be an IRI or a blank node, is represented by a circle from which an arrow starts. An IRI is different from a blank node, because no border is used, while a dashed border is used for a blank node, i.e., two distinct notations for subject. A predicate, which is always an IRI, is represented by an arrow with a rectangle. When an object is an IRI or a blank node, it is represented by a circle where an arrow ends, which distincts it from a subject. When an object is a literal, a rectangle is used. The use of a datatype or language is differently visualized: the datatype is added at the bottom of the rectangle, and the language is added at the bottom of the rectangle preceded with "@".

The class of an entity can be represented with two different notations: (i) a circle with the class at the bottom, as datatype is for literals, and (ii) a circle with an arrow and rectangle to another circle with the class. The former occurs in most cases, when the class is constant. To support rare cases where the class might depend on raw data values, the longer notation can be used, without excluding the support for a constant value. This might be a small symbol redundancy, nevertheless, it leads to a shorter notation in most cases.

2.2.5.2.4.1.2 Symbol overload When a single graphical notation can represent multiple semantic constructs, called symbol overload, it leads to ambiguity and potential misinterpretation. In the case of MapVOWL, no symbol overload is observed. Each graphical notation aligns with a single semantic construct, as can be observed in Figure 2.4 where only a single arrow arrives for each graphical notation on the right. In detail:

Subject A (dashed) circle from which an arrow starts aligns with a subject, corresponding with either an IRI or blank node.

Predicate A rectangle with an arrow through it aligns with a predicate.

Object A (dashed) circle where an arrow arrives aligns with an object, corresponding with either an IRI or blank node; a rectangle without text at the bottom with a literal without datatype or language; a rectangle with text about the datatype with a literal with a datatype; a rectangle with text preceded with "@" with a literal with a language.

Class A circle with text at the bottom aligns with an IRI as subject, `rdf:type` as predicate, and a class as object. If this circle has a dashed border, then it aligns with a blank node as subject instead of IRI. A circle with an arrow and rectangle to a circle

with text in the center aligns with an IRI as subject, `rdf:type` as predicate and a class as object. If this notation has dashed border for the first circle, then it aligns with a blank node as subject instead of IRI.

2.2.5.2.4.1.3 Symbol excess When a graphical notation is used that does not represent a semantic construct, called symbol excess, it clutters the visualization and increase the complexity of the visualization. Both have a negative impact on the understanding by users. In the case of MapVOWL, no symbol excess is present as for each graphical notation there is a corresponding semantic construct, as explained for symbol overload (at least one arrow arrives for each graphical notation one the right in Figure 2.4).

2.2.5.2.4.1.4 Symbol deficit As opposed to the aforementioned anomalies, symbol deficit is desired to limit the diagrammatic complexity. Symbol deficit occurs if not all semantic constructs are represented by a corresponding graphical notation. The deficit is desired when representing all semantic constructs would reduce, instead of improve, the notation's cognitive effectiveness. In the case of MapVOWL, no symbol deficit was introduced as the graphical notations required to represent all constructs do not lead to a complex notation (Figure 2.4).

2.2.5.2.4.2 Perceptual discriminability

Perceptual discriminability is the ease and accuracy with which graphical notations can be differentiated from each other [65]. For rules, it is important to perceive the difference between (i) the components of a triple (subject, predicate, and object), and (ii) the RDF terms (IRI, blank node, and literal). The former is done by using directed graphs, thus, an edge has an explicit direction, where the start is the subject and the end is the object. The latter is done by using a circle without a border for an IRI, and a circle with a dashed border for a blank node, regardless of whether they are subject or object. A rectangle is used on the edge between two nodes to represent the IRI, which is the only option for a predicate. A rectangle is also used for literals, but it is distinct since it can only be at the end position of the edge as it can only be the object of a triple.

2.2.5.2.4.3 Semantic transparency

Semantic transparency is defined as the extent to which the meaning of a notation can be inferred from its appearance [65]. When knowledge graphs are presented using RDF, RDF graphs represent the model of the data. By using graph-based visualizations for rules, users already see the model that will be used for the generated RDF triples, which form an RDF graph. Nodes in the rule graphs that are the start/end of an edge will result in subjects/objects of triples in the RDF graph. The edges of the graphs will result in the predicates of the triples in the RDF graph.

Table 2.4: Visual variables

Variable	Power	Capacity
horizontal position	internal	10-15
vertical position	interval	10-15
size	interval	20
brightness	ordinal	6-7
colour	nominal	7-10
texture	nominal	2-5
shape	nominal	unlimited
orientation	nominal	4

2.2.5.2.4.4 Complexity management

Complexity management refers to the ability of a visual notation to represent information without overloading the human mind [65]. Graph-based visualizations can be difficult to understand by users when they become too large. Therefore, when the application that implements the visual notation for rules fails to address it, it might overload its users.

2.2.5.2.4.5 Visual expressiveness

Visual expressiveness is defined as the number of visual variables used in a notation [65]. The number is proportional to the understanding of a notation, as it exploits multiple visual communication channels and maximizes computational offloading. In total, there are eight visual variables: horizontal position, vertical position, size, brightness, color, texture, shape and orientation (see Table 2.4). Each variable has a power and capacity. The power denotes which type of information can be used: interval, ordinal, or nominal. The capacity denotes how many perceptible steps are needed to understand the variable.

MapVOWL uses five out of the eight visual variables, as it uses size, brightness, color, texture, and shape (see Section 2.2.5.2.1). They are used in accordance to their power and capacity. Size denotes how often a data fraction is used in a data sources, ranging from 0% to 100% (interval), with steps of 10% resulting in 10 perceptible steps (< 20 (capacity in Table 2.4)). Brightness denotes which graph elements is selected, which is either true or false (ordinal, < 6). Color denotes the different data sources (nominal), and in most cases the number of data sources is limited to less than 5 (< 7). The texture is an alternative to denote a data sources (nominal, < 5). Shape denotes the limited graph elements, and corresponding rules (nominal, unlimited). Even though users are able to adjust the horizontal and vertical position, and orientation, which might improve their understanding, it does not carry a specific meaning in MapVOWL, because MapVOWL does not have a symbol deficit that would benefit from using these variables.

2.2.5.2.4.6 Dual coding

According to dual coding theory [66], using text and graphics together to convey information is more effective than using either on their own. MapVOWL does not specify the use of dual coding. However, in the RMLEditor it is used. Information about the used data source for an RDF term is available through the color of the nodes and edges (graphics), and when clicking on the details icon (text). Other details about the rules are only available as either text or graphics, because they do not have a corresponding graphical notation or because that would introduce the use of (language) terminology while MapVOWL's goal is to avoid that (see Section 2.2.5.2).

2.2.5.2.4.7 Graphic economy

Graphic complexity is defined by the number of graphical notations: the size of its visual vocabulary [65]. The principle of graphic economy states that this number should be cognitively manageable. In the case of MapVOWL, the number is low. There are six different graphical primitives (see Section 2.2.5.2.1).

2.2.5.2.4.8 Cognitive fit

Cognitive fit means a visual notation should include different representations to support users with skills ranging from novice to expert, and the representational medium, such as whiteboards, paper and computer screens [65], which is influenced by the perceptual discriminability, semantic transparency, and visual expressiveness.

The creation of MapVOWL has as main goal to support users who have knowledge about knowledge graphs and the data domain. Therefore, a different representation for others is not defined. MapVOWL is designed to be displayed on a screen. However, the perceptual discriminability is high enough as MapVOWL relies on differently shaped graphic notation, i.e., circles, rectangles, and lines. The semantic transparency is high enough as MapVOWL uses basic geometric shapes. The visual expressiveness might not be high enough though. For example, MapVOWL only uses colors to denote the different data sources. However, on paper easy-to-draw symbols can be used to compensate this.

2.2.6 RMLEditor

The RMLEditor is a graph-based visualization tool for rules that define how knowledge graphs are generated from multiple, heterogeneous data sources [1]. It allows users to load raw data sources, create and edit rules, and preview the resulting RDF triples. Its main goal is to allow users who have domain and knowledge graph understanding, but no knowledge about the knowledge graph generation language, to create rules (see Figure 2.5).

In this section, we discuss the GUI requirements for the creation of knowledge graph generation rules (see Section 2.2.6.1), the extended RMLEditor's architec-

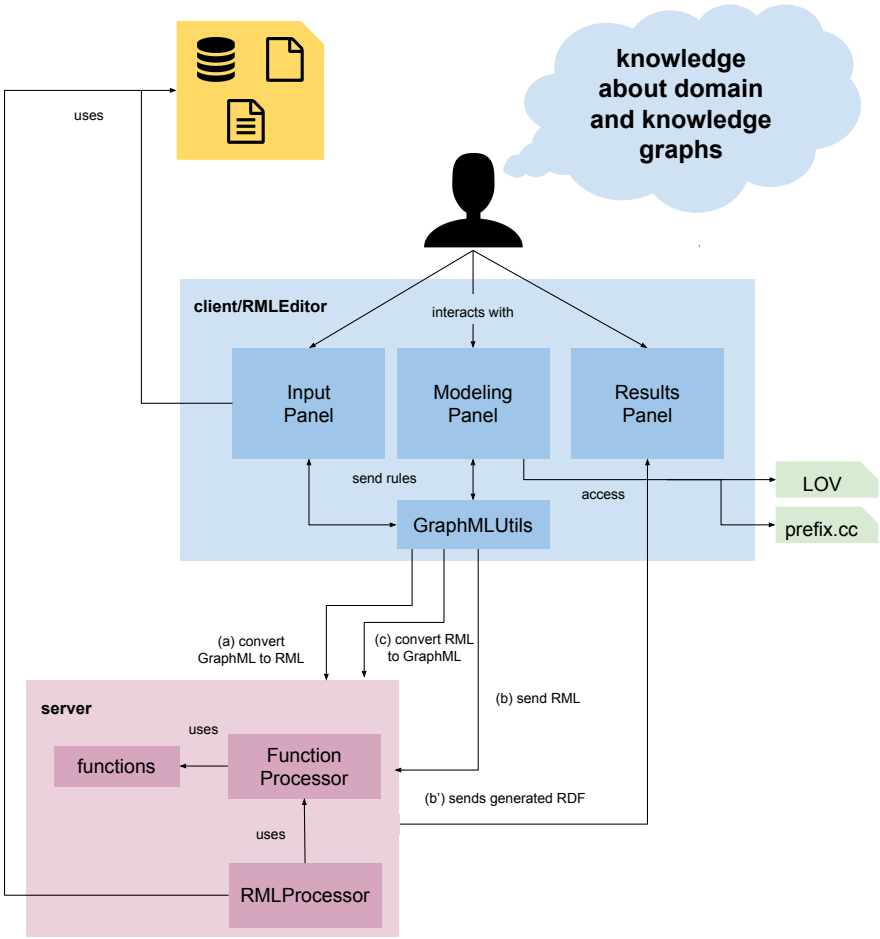


Figure 2.5: Overview of the RMLEditor

ture (see Section 2.2.6.2), the GUI's design and features (see Section 2.2.6.3), and how the rules are executed (see Section 2.2.6.4).

2.2.6.1 Requirements

In previous work [21], we introduced a list of desired GUI features for the creation of rules.

Requirement 7: *Independent of the underlying knowledge graph generation language*

The visualization of rules should be independent of the underlying language to execute the rules. This is to remove dependencies on the language and to support the editor to switch the underlying language for one that might better suit the use case at hand.

Requirement 8: *Support multiple data sources*

The GUI should support multiple data sources, as a single knowledge graph might be derived from several.

Requirement 9: *Support heterogeneous data formats*

The original data from which knowledge graph are generated might stem from different data formats, such as CSV, JSON, and XML. Thus, the GUI should support the creation of rules on top of different formats. Even more, it should be independent of the format.

Requirement 10: *Support multiple ontologies*

Different ontologies, which model complementary or overlapping aspects of domain-level knowledge, are available. The GUI should support the creation of a set of rules that uses different ontologies at the same time.

Requirement 11: *Support multiple alternative modeling approaches*

When users create rules they can follow different modeling approaches [26]. The used approach will be dependent on the user's preference and use case at hand. The GUI should enable using different approaches to support users and use cases.

Requirement 12: *Support non-linear workflows*

During the creation of rules different factors are involved, including data, ontologies, vocabularies, rules, and resulting knowledge graphs. When these factors are presented to the user in isolation, as done with non-linear workflows, the relationships betweens these factors are obscured. Thus, the GUI should support non-linear workflows to enable users to keep an overview of the different factors.

Requirement 13: *Independent of rule execution*

The rules definition and execution are different aspects. An editor and its GUI should not restrict the execution to a specific library or tool. This improves the rules' interoperability and reusability.

2.2.6.2 Architecture

The RMLEditor's high-level architecture is based on the multilayered architecture pattern [67]. This allows to separate the presentation of the rule components, the logic of the rule definition process and the access to data and external APIs, using the presentation, application, and data access layer, respectively. The latter only communicates with the application layer. Communication between the presentation and data access layer is not possible, as the architecture prohibits communication between layers not directly under or above each other.

The presentation layer consists of several panels with which users interact. The application layer consists of the modules to process the interactions triggered through the panels. The data access layer handles the requests that require data or rules from outside the RMLEditor. RML [6] is the underlying language of the RMLEditor, because RML supports rules with data from multiple, heterogeneous data sources.

The communication with the application layer is facilitated by the command pattern [68] and the flux pattern [69]. The flux pattern replaced the Model-View-Controller (MVC) pattern [68] that was used in the initial version of the RMLEditor. The MVC pattern causes problems in both performance and development, because of the bidirectional communication, where one change can loop back and have cascading effects across the codebase [69]. The flux pattern dictates an unidirectional flow. This flow is stricter than the flow in MVC, because it does not allow to start new flows when another flow is still being executed. For the presentation layer, the Webix JavaScript library[32] is used to build the GUI and the d3.js library [23] to build the graphs.

The graph-based visualizations are encoded using an XML document, based on Graph Markup Language (GraphML) [70] with custom extensions, to represent them independently of the underlying language. This allows users to export the graphs, besides the rules, in an application-independent format. Additionally, the graphs' GraphML representation is used to generate the corresponding RML statements (see Figure 2.5a). When users want to load rules into the RMLEditor, they can load (i) a GraphML document, or (ii) a set of RML rules, which is converted to GraphML and loaded (Figure 2.5c). The corresponding graphs are shown in the GUI.

Data transformation descriptions are also encoded in the GraphML representation and thus they are also present in the resulting RML document which is enriched in this case with descriptions based on the Function Ontology for the data transformations [71, 72]. When an RML document is being processed by the extended RMLProcessor server-side, the raw data values are extracted by the RMLProcessor, and data transformations are handled by passing the raw data values and the function which is desired to be applied to the raw data values to the Function Processor.

[32] `http://webix.com/`

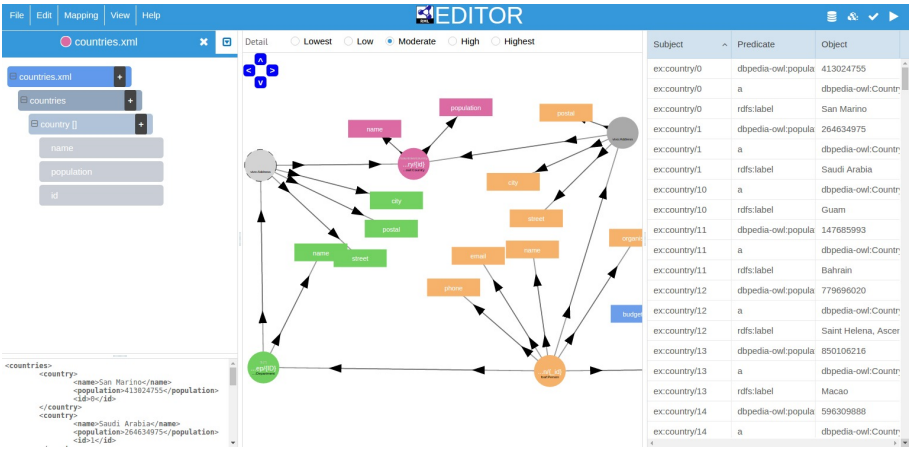


Figure 2.6: Overview of the RMLEditor

2.2.6.3 Graphical user interface

In this section, we discuss the RMLEditor's GUI and its interaction elements to manipulate the graph-based visualizations.

Panels The GUI consists of three views: Input Panel, Modeling Panel and Results Panel (Figures 2.5 and 2.6) [1]. They are aligned next to each other, but when users want to focus on a specific panel, they can hide the Input or Results Panel.

The **Input Panel** shows the data sources (the left panel in Figure 2.6). Multiple data sources can be loaded and they can be in different data formats, such as CSV, JSON, and XML . Each data source is assigned with a unique color to be used in the graphs. In the previous version of the RMLEditor, this panel only shows the raw data, thus, providing a different representation for each data format.

The different representations for each format were circumvented. Now, the Input Panel is divided into two subpanels. The top panel displays the data sources structure. This makes it independent of the data format. The structure can be manipulated if data values need to be transformed. The bottom panel shows the raw data of the source. This allows users to view both the structure and data values.

The **Modeling Panel** shows the rules using MapVOWL (the middle panel in Figure 2.6). The color of each node and edge depends on the data source that is used in a specific rule, if any. It offers the means to manipulate the nodes and edges of the graphs to update the rules. Semantic annotations can be added using multiple ontologies, which can be either defined locally or online. Linked Open Vocabularies [73] can be consulted via the GUI to get suggestions on which classes, properties and datatypes to use. It is possible to use prefixes instead of using a full IRI. These prefixes can

Table 2.5: Interaction elements

Element	Application
	show/edit details
	create relationship
	delete node/edge
	collapse graph
	expand graph

be customized and defined for both local and online ontologies. Users can consult `http://prefix.cc` to search for well-known prefixes and their namespaces.

In the previous version of the RMLEditor, scalability was only addressed using geometric zooming [74], but interactive filtering [75] was introduced to address usability issues with large graphs (more details are available at Section 2.2.6.5). Geometric zooming and interactive filtering are two established methods to address large graphs.

The **Results Panel** shows the resulting RDF triples when rules created in the Modeling Panel are executed on the data in the Input Panel (right panel in Figure 2.6). For each RDF triple, it shows the subject, predicate, and object.

Furthermore, the use of the three panels allows to follow different rule creation approaches [1, 26]. The data-driven approach uses the input data sources as the basis to construct the rules. Classes, properties, and datatypes of the schemas are then assigned to the rules. When users start with the ontologies to generate the rules, the schema-driven approach is followed. Next, data fractions from the data sources can be associated to the rules.

The RMLEditor implements the aforementioned requirements for a GUI for knowledge graph generation rules. The visualization of the rules is independent of the underlying language through the use of MapVOWL in the Modeling Panel (Requirement 7). Multiple, heterogeneous data sources are supported via the Input Panel (Requirements 8 and 9). Multiple ontologies are supported via the Modeling Panel (Requirement 10). The collaboration between the three panels supports multiple alternative modeling approaches (Requirement 11) and non-linear workflows (Requirement 12). The independence of rule execution is implemented because the rules can be exported as both GraphML and RML rules (Requirement 13).

Interaction Besides visually understanding the rules, users should also be able to interact with the graphs, which results in updating the corresponding rules. QueryVOWL provides this functionality for queries, but its approach can also be applied to rules. Therefore, similar to QueryVOWL, the RMLEditor uses a set of interaction elements (see Table 2.5): icons placed on the border of a node or edge (see

Figure 2.7: Interaction elements on a node

Figure 2.7). They correspond with five actions: edit details of a rule, create relationship, delete node/edge, and collapse/expand graph. The different icons were chosen to be simple, but concrete and distinctive to increase the user performance [76]. Every action has a distinctive icon, with exception for collapse and expand graph, but they are never shown at the same time. The icons are only shown when users hover over the node or edge to reduce the visual overload.

2.2.6.4 Rule execution

Once the rules are created, they can be executed by the RMLProcessor to generate knowledge graphs. This might happen also via the RMLEditor. We developed a Web API, using Node.js[33], to separate this functionality from the RMLEditor (see Figure 2.5). This makes it also easy to switch between different tools that execute rules that support multiple, heterogeneous data sources.

The API offers three functions: (i) executing a set of rules on a set of data sources (see Figures 2.5b and 2.5b'); (ii) converting a GraphML-based document to RML to execute the rules using the RMLProcessor (see Figure 2.5a), and (iii) converting RML statements to a GraphML-based document to visualize the loaded rules (see Figure 2.5c). Another language and corresponding processor can be used, only leading to adjustments to the RMLEditor's code that converts the visualization to language-specific rules, while no adjustments to the GUI are required.

2.2.6.5 Manipulation of large graphs

When applying graph-based visualizations, large graphs are inevitable. Users have difficulties editing and keeping an overview of the rules when large-scale graphs are not addressed. Geometric zooming [74] and interactive filtering [75] are two established methods to address large graphs. The former provides enlargement of the graphs. The zoom level can be set through, e.g., scrolling, buttons, or keyboard shortcuts. The latter allows users to filter out the relevant elements of the graphs.

VOWL does not specify any methods on how to deal with large graphs. However, WebVOWL, which implements VOWL, applies both aforementioned methods. Geometric zooming is done through scrolling. Interactive filtering is applied through a

[33] `https://nodejs.org`

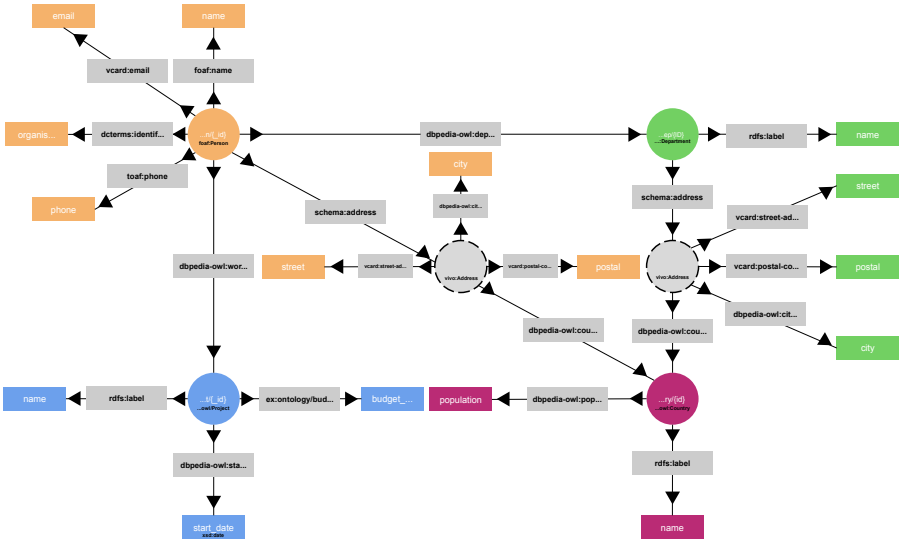


Figure 2.8: Highest detail level in the RMLEditor

filter feature which gives users access to four options that hide specific information or graphical notations and reduce the size of the graph: hide datatype properties, solitary subclasses, information about disjointness, and set operators.

Similar to VOWL, geometric zooming and interactive filtering is not specified by MapVOWL. Geometric zooming is implemented in the RMLEditor via scrolling, as in WebVOWL. Interactive filtering is implemented via the use of detail levels. Interactive filtering requires to define which filters can be applied on the graphs. Instead of letting the users select the filters, as WebVOWL does, we opted to group a set of appropriate filter together in a so-called detail level. The detail levels are arranged from highest to lowest, where each level applies one or more additional filters to the higher level.

Highest level The highest level shows all visual elements of MapVOWL. This level is used when users want to inspect and edit all details of the rules, and when the general overview of the rules is less important (see Figure 2.8).

High level The high level hides the literals' datatypes and languages (see Figure 2.9). This level focuses more on how different graph elements are related to each other, hiding the specifics, which might be the most redundant for the users.

Moderate level The moderate level hides the properties of the edges (see Figure 2.10). This puts more focus on which subjects and objects are related to each other,

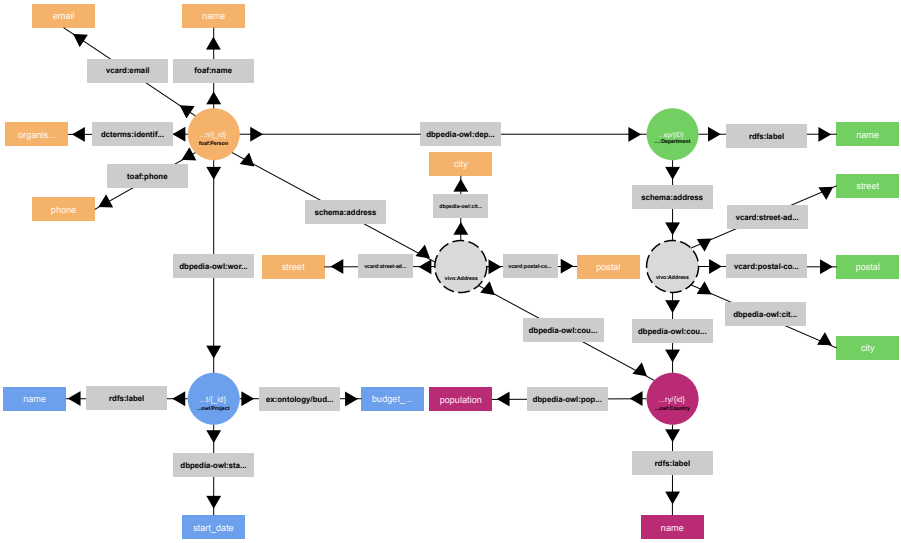


Figure 2.9: High detail level in the RMLEditor

instead of how, i.e., only the relationship, without a specific predicate. Furthermore, the graphs become less dense, leading to a better overview of the rules.

Low level The low level hides the literals (see Figure 2.11). This puts more focus on how the different entities are related, and more specific how the different data sources are connected. Furthermore, the graphs become less dense, leading to a better overview of the rules.

Lowest level The lowest level hides the blank nodes (see Figure 2.12). This only shows entities that have an IRI assigned, which allows the users to deal only with entities of the different data sources and how they are directly related, to improve their understanding of the links between the data.

More, when users are at a certain detail level, they might want to show or hide details of a specific subgraph, without the desire to change the overall level. In the RMLEditor, this is accomplished using the collapse and expand actions (see Section 2.2.6.3).

When a set of rules is loaded in the RMLEditor an appropriate detail level is selected based on the graphs size, i.e., the amount of nodes is higher than a specific threshold, to provide users with the rules overview. If the level is too high for the size of the graph, users lose the overview of the rules, which makes it difficult to navigate to the part of the graph which is desired to be edited.

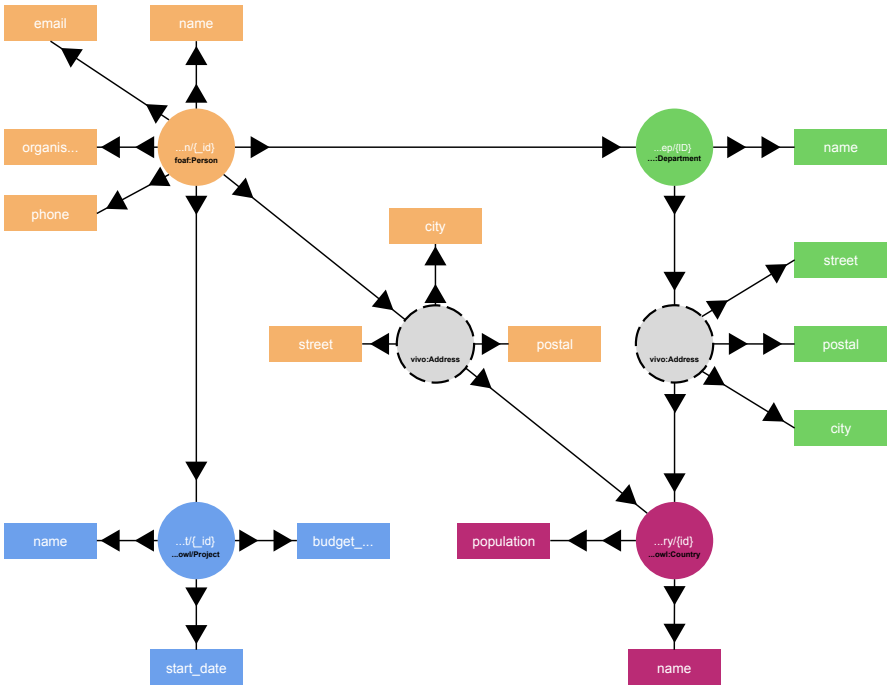


Figure 2.10: Moderate detail level in the RMLEditor

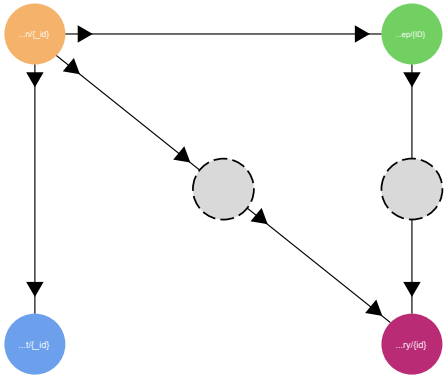


Figure 2.11: Low detail level in the RMLEditor

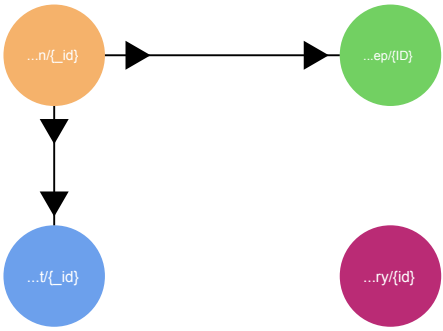


Figure 2.12: Lowest detail level in the RMLEditor

2.2.6.6 Heterogeneous data values manipulation

Data sources might have heterogeneous data formats. Thus, user interfaces for rules need to support multiple data formats. Moreover, presenting both the data structure and the data values allows users to gain insights in both the data model, and corresponding data. Transforming one or more data fractions when generating knowledge graphs might also be required. For example, if a data source has a data fraction that contains a date in the format "DD/MM/YYYY". A transformation is needed to convert it to "YYYY-MM-DD" when the datatype `xsd:date` is used for the corresponding literal.

In the RMLEditor, the support for manipulating heterogeneous data values and data sources is facilitated by the Input Panel (see Figure 2.6). It features both the structure of the data, and the raw data. The structure includes data fractions that users may use to generate the desired knowledge graph and can be updated to reflect also the required data transformations. This shows that the transformed data values can be used as any other data value.

2.2.7 Evaluation

We conducted two comparative studies to validate our two hypotheses (see Section 2.2.4): one study compares the representation of rules via MapVOWL and via RML directly, and another study compares the creation of rules via the RMLEditor and RMLx.

2.2.7.1 MapVOWL vs. RML

With this study, we aim to validate Hypothesis 1. We compared what knowledge users are able to extract from a set of rules that are represented via both MapVOWL and RML directly. This knowledge aligns directly with the six requirements for a visual notation (see Section 2.2.5.1). RML was chosen as it supports rules with data from multiple, heterogeneous data sources. Furthermore, we evaluated which representation

Table 2.6: Blocks of MapVOWL vs. RML questionnaire

Block	**Description**
introduction	Questions about participants' socio-demographics & knowledge graph expertise
MapVOWL vs. RML	MapVOWL-specific questions followed by four test cases to compare MapVOWL & RML
post-assessment	Questions on preference of MapVOWL vs RML and the use of MapVOWL for editing rules

they prefer. In Section 2.2.7.1.1, we discuss the method, namely the procedure and participants. In Section 2.2.7.1.2, we discuss the results, which can also be found at `https://w3id.org/mapvowl/eval17/results`, and, in Section 2.2.7.1.3, the corresponding derived insights.

2.2.7.1.1 Method

Procedure Participants with RML knowledge were directly contacted by the author. Those who agreed to partake in the test had to read an introduction to MapVOWL[34] and complete an online questionnaire. The introduction explains the different elements of MapVOWL through the use of an example which was provided to assure that participants have a basic understanding of MapVOWL, too. The participants completed an online questionnaire, which can be found at `https://w3id.org/mapvowl/eval17/survey`. The questionnaire consists of three main building blocks (see Table 2.6):

(i) The first block questioned the participants' main **sociodemographic traits**, such as year of birth, gender, level of education, and employment status. Then, it measured the participants' **knowledge graph expertise** through a self-assessment and a study of their familiarity with the topic and tools.

(ii) The second and essential block provides a **test with eleven questions** about rules that were presented as a MapVOWL visualization to access the understanding of the MapVOWL elements and *ten questions about four test cases* to compare a MapVOWL representation with an RML representation. Three of the use cases are real-world examples and one is artificially created for the study. Half of the participants answered the questions of the first two use cases via the MapVOWL representation and the last two via RML. The other half of the participants answered the questions of the

[34] `https://w3id.org/mapvowl/eval17/intro`

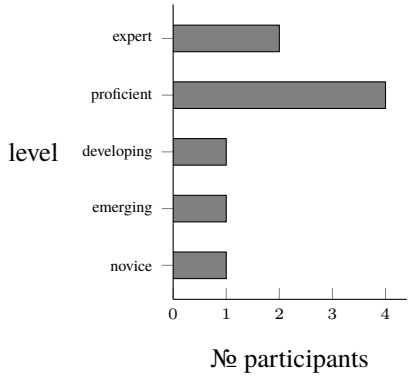


Figure 2.13: Level of knowledge graph expertise of participants of MapVOWL user study

first two use cases via the RML representation and the last two via MapVOWL. Participants were randomly assigned to one of the two aforementioned groups.

(iii) The **post-assessment** consists of three questions and gathers information about the participants' preference regarding the use of MapVOWL versus RML to answer the questions, and whether they want to use MapVOWL to also edit rules, besides only to visualize them.

Participants The online questionnaire was sent out to potential participants in August 2017. Nine participants have eventually taken part in the experiment, their age range was 20 to 38. All were highly educated: three of the contributors had a PhD, four a master's degree and two a bachelor's degree. They were highly familiar with knowledge graphs as well: 6 of the participants claimed they already generated knowledge graphs and only one declared to have merely a basic understanding of knowledge graphs (see Figure 2.13).

2.2.7.1.2 Results

Rules When answering the eleven questions about the elements of MapVOWL, four questions are answered correctly by all participants: determining the number of literals per entity and relationships, the used ontology and language. Two questions are answered correctly by 8 participants: detecting entities without a class and determining how IRIs are generated. Two questions are answered correctly by 7 participants: detecting the number of literals and their datatypes. Two questions are answered correctly by 6 participants: determining the number of distinct data sources and the number of entities that are not associated with a data source. One question is answered correctly

Table 2.7: № correct answers regarding MapVOWL

Questions	**Correct answers (max. 9)**
№ literals per entity/relationships used ontology/language	9
entities without a class how IRIs are generated	8
№ literals and their datatypes datatypes of literals	7
№ distinct data sources № entities w/ data source	6
№ relationships ind. of data source	5

Table 2.8: № correct answers when using MapVOWL and RML

Questions	**Correct answers**	
	RML	MapVOWL
types of entities № literals relationships b/t entities	51	**61**
№ (unlinked) data sources the language of literals	**65**	50
use of templates use of datatypes	24	24

by 5 participants: determining the number of relationships that do not depend on a data source. This is summarized in Table 2.7.

Regarding the questions about the use cases, the number of correctly answered questions is slightly higher for RML than MapVOWL (140 correct answers vs. 135), as summarized in Table 2.8. Questions regarding the types of entities, number of literals, and relationships between entities are answered more often correctly via MapVOWL than RML (61 correct answers vs. 51). Questions regarding the number of (unlinked) data sources and the language of literals are answered more often correctly via RML than MapVOWL (65 correct answers vs. 50). Questions regarding the use of templates and datatypes are answered correctly as often via MapVOWL as via RML (24 correct answers).

Post-assessment 7 participants preferred the application of the MapVOWL notation for *viewing* the presented cases. 1 participant preferred using RML directly over MapVOWL and 1 was neutral (see Figure 2.14). Similarly, when they were

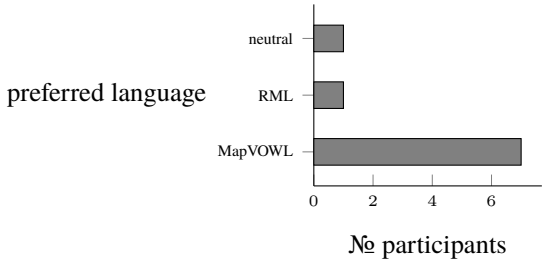


Figure 2.14: Preferred language for viewing the rules

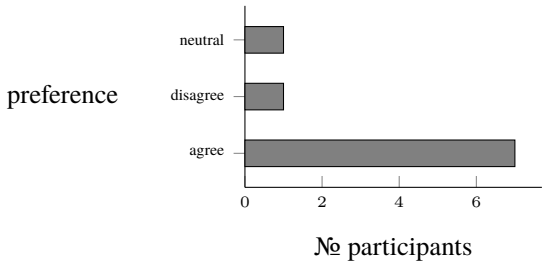


Figure 2.15: Participants' agreement to use MapVOWL to edit the rules

asked which they prefer to use for *editing* rules, 7 participants agreed that they prefer MapVOWL, 1 slightly disagreed and 1 remained neutral (see Figure 2.15).

2.2.7.1.3 Insights

The answers to MapVOWL-specific questions provide evidence that **the visual notation is cognitive effective**. When comparing the answers of questions that are both answered using MapVOWL and RML, it provides evidence that both options are possible candidates to represent rules. However, the post-assessment shows that **users prefer MapVOWL** over RML to visualize rules, even though RML results in a slightly higher number of correct answers.

2.2.7.2 RMLEditor vs. RMLx

With this study, we aim to validate Hypothesis 2 by comparing the use of the RMLEditor and RMLx to edit rules. Specifically, the visualization of the knowledge graph generation process components with special attention to the rules is studied. RMLx was chosen, as it allows to annotate multiple, heterogeneous data sources and values. More, it also uses graph visualizations to represent the rules; however a form-based approach is used to edit the rules.

Table 2.9: Steps of the RMLEditor vs. RMLx user study

Step	Description
introduction	Questions on participants' socio-demographics & Knowledge graph expertise
use cases	2 use cases per participant on rules creation
additional RMLEditor test	Questions on heterogeneous data sources & data values, & RMLEditor's large graphs
post-assessment	Questions on participants' experience with the used tool

In Section 2.2.7.2.1, we discuss the participants and procedure. In Section 2.2.7.2.2, we discuss the results, and in Section 2.2.7.2.3, the derived insights.

2.2.7.2.1 Method

Procedure This second study is modeled according to the following procedure[35,36]. Potential participants from imec and Ghent University were approached. Those agreeing to contribute to the user evaluations then had to read an introduction[37] to the RMLEditor and RMLx and work through the different steps of the study together with one of the authors. More details about the experimental setting of the latter can be found at `https://w3id.org/rml/editor/eval17/setting`. The introduction explains the different elements of both tools. For the RMLEditor, it also includes the introduction to MapVOWL, because it implements MapVOWL. The user study is divided into four steps (see Table 2.9):

(i) The participants are presented with questions about their **socio-demographics** and knowledge graph **expertise**, identical to the ones asked to participants of the MapVOWL vs. RML study (Section 2.2.7.1).

(ii) Two **use cases** are presented to the users for which rules need to be created. The first use case deals with data about employees and the projects they are working on. Both data sources are provided as CSV files. This use case is *data-driven*: every data fraction should be represented in the resulting knowledge graph. The second use case deals with data about movies and their directors. The movie data source is provided as a CSV file and the director data as a JSON file. This use case is *schema-driven*: based on a given set of classes, properties, and datatypes, knowledge graphs need to be generated using the provided raw data. Half of the participants completed the first

[35] `https://w3id.org/rml/editor/eval17/survey1`
[36] `https://w3id.org/rml/editor/eval17/survey2`
[37] `https://w3id.org/rml/editor/eval17/intro`

use case with the RMLEditor and the second use case with RMLx. The other half completed the first use case with RMLx and the second use case with the RMLEditor. Participants were randomly assigned to one of the two groups. The outcomes were assessed based on the generated knowledge graphs' correctness and the experience with each tool was recorded.

(iii) An **additional test** was completed for the RMLEditor. Here, participants got presented with an online survey, containing specific questions regarding heterogeneous data sources and values, and large graphs. Questions about the data sources included determining the different data fractions and data transformations that are available in the data sources and are used in the rules. Questions about the large graphs included determining the number of rules related to a specific RDF concept and asking about their preference with respect to the different detail levels. During the second and third step of the user study, each participant was supervised by one of the authors making use of the think-aloud protocol. Widely used by usability professionals, think-aloud originated out of the incapability to know what users think when completing tasks [27]. In the concurrent think-aloud process as implemented in the current usability study, users are probed to specify their thoughts and actions as they occur. This allows the attending author to discern when and why a participant faces difficulties.

(iv) After each use case, a **post-assessment** gathers information about the participants' experience with the used tool. Included are questions about the perceived difficulty of the tasks and confidence in a successful completion. A central part of the post-assessment is assigned to the System Usability Scale (SUS) [28], a procedure to quickly collect users' usability ratings of a technology. Benefits of applying SUS are its conciseness, its transferability over several technologies, and applicability with small sample sizes [77, 78]. As proposed by Bangor, Kortum, and Miller [77], an adjective rating using a 7-point Likert scale, which assesses the tools user-friendliness from worst imaginable to best imaginable, was also added to the questionnaire. It should be sharply noted though that the outcome of the adjective rating and the system usability scale should be supported by observations by the attending supervisor.

Participants The participants of our study were people from imec and Ghent University, that were contacted directly by the authors. This ensured that they had a sufficient level of knowledge graph expertise, required for a meaningful participation in the user study. A total of 10 participants, age range 27 to 39, were recruited. Their level of education was high: they all hold a master's degree, have conducted advanced graduate work, or hold a PhD. 8 participants assessed themselves as having at least a proficient level of knowledge graph expertise, 1 considered himself to be a novice and 1 observed his knowledge graph expertise as emerging (see Figure 2.16). They all had heard of the RMLEditor before, 2 participants had used it; whereas 6 participants had taken notice of RMLx before.

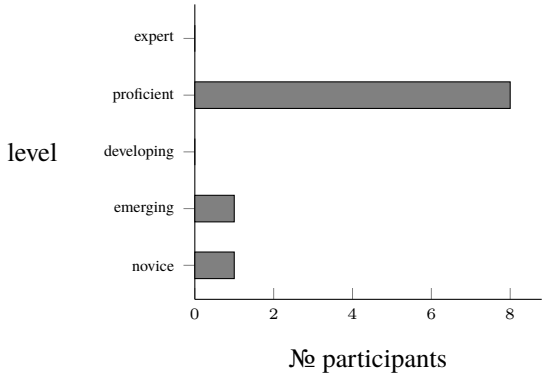


Figure 2.16: Level of knowledge graph expertise of participants of RMLEditor vs. RMLx user study

Table 2.10: Completeness of rules and SUS-score of RMLEditor and RMLx

	RMLEditor	**RMLx**
Completeness of rules (%)		
use case 1	**91**	83
use case 2	**98**	82
SUS-score		
	82.75 (good)	42 (poor)

2.2.7.2.2 Results

Rules An analysis of the created rules by the participants, which can be found at `https://w3id.org/rml/editor/eval17/results/mappings`, revealed several aspects that can be circumvented by updating the GUI of the tools. For the first use case 91% of the expected rules were present when using the RMLEditor. In the case of RMLx, it was only 83%. The same is observed for the second use case with 98% for the RMLEditor and 82% for RMLx (see Table 2.10).

In more details, the iterator was forgotten by 3 participants using RMLx, but this was not an issue with the RMLEditor. The use of constants, reference, and templates was incorrect for 2 participants using RMLx, but this was not an issue with the RMLEditor.

All participants added the data transformation with both tools, but 4 participants forgot to use the output of the transformation as an object using RMLx, while only 2 using the RMLEditor. This lead to triples that did not contained the transformed data value, but the original value. When interlinking datasets, all but one participant created the required rule using RMLx. Using the RMLEditor, all participants created the rule, but 3 participants did not provide the required join condition which interlinks

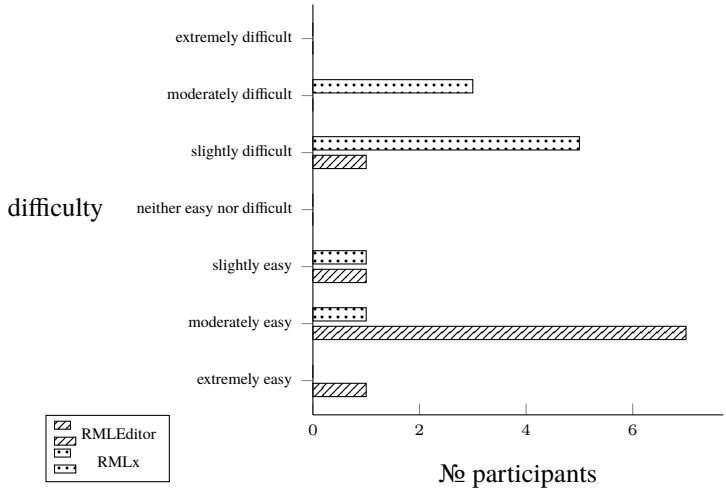


Figure 2.17: Difficulty of the RMLEditor vs. RMLx

entities that are not related. This was not an issue using RMLx.

Furthermore, we pay special attention to new features of the RMLEditor introduced in this work. Users are able to distinguish the different data sources, but when they have to count the total number of data sources that are loaded 4 participants forget to count the currently selected data source. 9 participants were able to correctly determine how many data transformations are applied when presented with a new set of rules. They were also able and perceived it as easy to determine which data transformation and data fractions are used. When users create and edit rules that link entities in large graph visualizations, 7 of them use the lower detail levels. When users create and edit rules that generate literals the preferred detail levels varies from the highest (2 participants) to the lowest level (1 participant), and 2 participants even stated no preference.

Post-assessment The first part of the post-assessment questioned the participants with regard to the **difficulty** of performing the use case with the tools and their confidence in a successful completion of the tasks. The results of the post-assessment can be found at `https://w3id.org/rml/editor/eval17/results/post`. The difficulty of performing the use cases was rated on a 7-point likert scale from extremely difficult to extremely easy.

Regarding *difficulty*, 1 participant rated completing the tasks with the RMLEditor as “slightly difficult”, all others assessed it as “neither easy nor difficult” or higher, with 3 participants valuating it at “extremely easy” (see Figure 2.17). For RMLx, all participants except 1 consistently rated executing the tasks more difficult than with the RMLEditor. 5 participants scored it as “slightly difficult” and the other 3 as “moder-

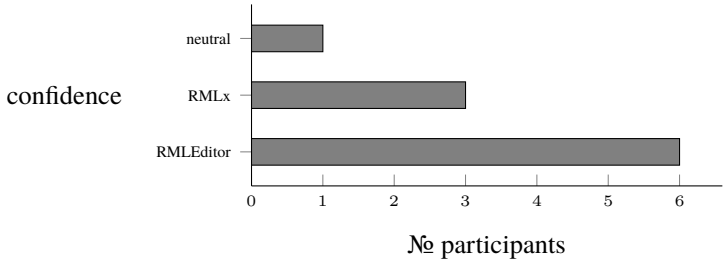


Figure 2.18: Confidence of using the RMLEditor vs. RMLx

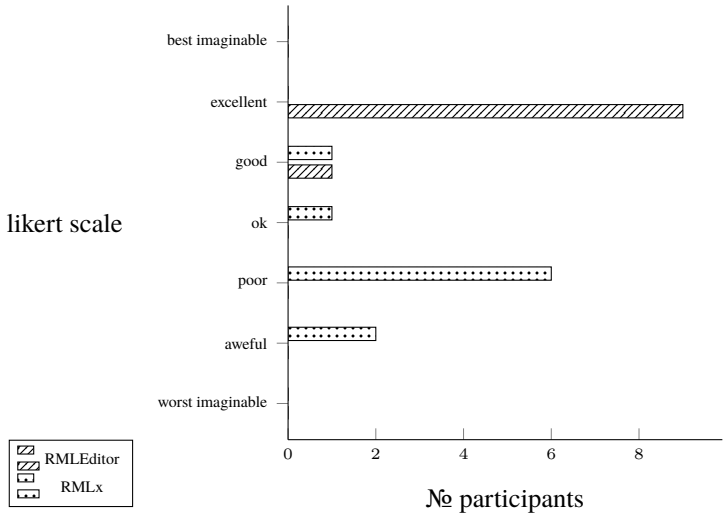


Figure 2.19: User-friendliness of the RMLEditor vs. RMLx

ately difficult".

The *confidence* for having correctly executed the tasks was positively rated for both tools. 6 participants were more confident in performing the tasks with the RMLEditor, 1 rated both tools equal and 3 were more confident with RMLx (see Figure 2.18).

The second part of the post-assessment applied the SUS and the accompanying **user-friendliness** rating proposed by Bangor, Kortum, and Miller [77]. The user-friendliness of the RMLEditor was rated on a 7-point likert scale. 9 participants scored it as excellent, the other 1 rated it as good. In contrast, the user-friendliness of RMLx was valuated as "good" by 1 respondent, "ok" by 1 respondent, "poor" by 6 respondents and "awful" by the remaining 2 (see Figure 2.19).

More, all except 1, who rated both as good, assessed the user-friendliness of the RMLEditor higher than that of RMLx. The average obtained mean SUS-score of the

RMLEditor is 82.75, whereas for RMLx, it is 42 (see Table 2.10). When translating these scores on the spectrum created by Bangor, Kortum, and Miller [77], the usability of the RMLEditor is identified as good, but that of RMLx as poor.

Two remarks: (i) one of the authors was present during the user study, which could result in a moderator acceptance bias. This happens when participants want to gratify the present author; (ii) a single metric should not be used to make absolute statements about a systems' usability [77]. The observations done by one of the authors and elaborated in the next paragraph give an enhanced understanding of the process of utilizing both tools.

Observations During the study, participants were supervised by one of the authors. The results can be found at `https://w3id.org/rml/editor/eval17/results/obs`. Participants could ask questions and receive feedback while using both tools. Although, users were provided with an introduction to both tools, which they could consult during the study, they still asked questions about how certain actions should be performed.

The participants referred to the *introduction* to find the required information. This happened for 8 participants when using the RMLEditor and for 9 participants when using RMLx.

RMLx does not allow to visualize the data sources. Therefore, every participant used text editors and spreadsheet tools to view the data sources. In the case of the RMLEditor only 1 participant first viewed the data sources externally before loading them in the RMLEditor.

Participants required additional information about the use of *parameters* to create a data transformation. This happened for 3 participants using RMLx and 6 participants using the RMLEditor. The difference occurs because the default parameter was already set in RMLx, whereas no default parameter is set for the RMLEditor. Nevertheless, still 3 participants have trouble understanding these parameters as shown via RMLx.

5 participants required additional information about the use of a *baseURI* and how it is defined in the RMLEditor. Although it is perceived as useful, the participants indicated that a clearer presentation in the GUI is required. RMLx does not offer this functionality.

A common issue with both tools is the use of constants, references, and templates to generate subjects, predicates, and objects. 9 participants for both tools required additional information to understand how they work and when they should be used. Although the RMLEditor selects a default depending on the element in the graph, additional information was still required for the users to fully understand these three options.

8 participants required additional information to understand how to *interlink* entities originating from different data sources using RMLx. "Parent mapping", "parent value", and "child value" were terms difficult to be understood, and participants had

trouble understanding how they would affect the resulting knowledge graph. This was not an issue during the use of the RMLEditor, because of the use of MapVOWL.

2 participants required information to understand that an *iterator* for a CSV file is not required when using RMLx. This was not an issue using the RMLEditor, because an iterator cannot be set when using a CSV file as data source, considering the iteration always happens per row.

3 participants required information about the use of the "output var"-field in RMLx. This field represents the output of a data transformation and the string can be used in other rules to use the output of a data transformation. This was not an issue using the RMLEditor, because data transformations are integrated in the original data fractions in the input panel.

2.2.7.2.3 Insights

The resulting rules provide evidence that the use of the RMLEditor leads to a **higher completeness** of rules, and thus of the knowledge graphs, compared to RMLx. For both tools updates to the GUI could improve this completeness, such as the inclusion of checks for the use of transformed values and iterators. The RMLEditor was rated as good and RMLx as poor by the participants. This is in line with the completeness of the rules. Important to note is that the preferred detail level, to deal with large graphs, is highly dependent on the individual participant as no single level was preferred more than the others.

The observations during the user study made clear that specific **terminology**, such as baseURI, reference, and template, needs further **clarification**. This could be done, for example, through tool tips in the GUI or by avoiding the use of the terminology. The observations provided evidence for the use of the RMLEditor over RMLx when interlinking entities due to the use of the visual notation in the former and the use of forms and specific terminology in the latter, as explicitly mentioned by the participants.

2.2.8 Conclusion

Visual tools are implemented to help users in defining how to generate knowledge graphs from raw data. This is possible thanks to knowledge graph generation languages which enable detaching rules from the implementation that executes them. However, no thorough research has been conducted so far on how to visualize such rules, especially if they become large and require considering multiple, heterogeneous data sources and transformed data values. In this article we introduced MapVOWL, a graph-based visual notation for rules; and an approach for manipulating rules when large visualizations emerge; an approach to uniformly visualize data fractions of raw data sources combined with an interactive interface for uniform data fraction transformations. MapVOWL and the two approaches were implemented in the RMLEditor to provide users with a uniform GUI to create and edit rules that is cognitive effective.

The results of the first study (Section 2.2.7.1) show that the use of MapVOWL is preferred over RML for representing rules. Answering questions about these rules leads to the same correctness (accuracy in cognitive effectiveness) by either using MapVOWL or RML for users who understand RML. Nevertheless, most participants indicated that they preferred to use MapVOWL as representation for rules, and they would also use MapVOWL to create and edit rules. This shows that MapVOWL is easier to use than RML directly. Therefore, this study provides evidence towards the acceptance of Hypothesis 1 "MapVOWL improves the cognitive effectiveness of the generation rule visual representation to generate knowledge graphs compared to using RML directly."

The results of the second study (Section 2.2.7.2) show that the RMLEditor is preferred over RMLx for creating and editing rules. Participants were able to create a high percentage of the required rules using both tools (high accuracy). The RMLEditor has a better usability compared to RMLx, as shown through the SUS-score (82.75 vs 42), making it easier for users to create and edit rules. This is due to the use of MapVOWL, which is independent of the underlying language, as pointed out by the participants. However, through the use of think-aloud, we conclude that updates to both GUIs would improve the knowledge graph generation process, with specific focus on the creation of the rules. RMLx's form-based GUI is strongly influenced by its underlying language RML, which leads to issues for users that are unfamiliar with RML. These issues can only be resolved either by acquiring knowledge about RML or avoiding to rely on knowledge of RML, as done by the RMLEditor. Therefore, this study provides evidence towards the acceptance of Hypothesis 2 "The cognitive effectiveness provided by the RMLEditor's GUI improves the user's performance during the knowledge graph generation process compared to RMLx."

MapVOWL fills the important gap between uniform rule visualizations and the different rule editors. It allows multiple tools to incorporate the same visualization, which increases the accessibility for users across these tools. The different semantic constructs of the rules align with the graphical notations of the graph-based visualization. This leads to an effective representation for users. Furthermore, it combines the different components of the knowledge graph generation process, providing users with insights on the interrelation of these components. Users are able to define and deduct which data fractions of which data sources are used in which rules, and deduct how the rules correspond with the generated knowledge graph.

Moreover, MapVOWL and the RMLEditor overcome two issues of current rule editors, namely, the uniform presentation of heterogeneous data sources and data transformations within knowledge graph generation. Multiple data sources are supported and color is an appropriate method to denote them. Distinguishing the data fractions that users rely on to generate knowledge graphs from the original raw data allows to create an abstraction layer. Such an abstraction layer not only enables to easily manipulate the data fractions but also incorporates data transformations uniformly for heterogeneous data sources. This allows users to handle transformed data values just as raw values.

Large graphs might result in a reduction of the graph-based visualizations' effi-

ciency. However, the second study shows that detail levels are appropriate to compensate this for rules. It is efficient for the users to change the detail levels based on the task at hand, and personal preference.

Overall, if users need to create and edit rules, the RMLEditor is a valid choice through the use of the proposed MapVOWL and the approaches to deal with large visualizations and heterogeneous data sources and values. Although, improvements to the GUI are required, which will be addressed in future work, the GUI's core elements have been proven to be effective via the two user studies.

Acknowledgements

The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (AIO), the Research Foundation – Flanders (FWO), and the European Union.

References

[1] Pieter Heyvaert, Anastasia Dimou, Aron-Levi Herregodts, Ruben Verborgh, Dimitri Schuurman, Erik Mannens, and Rik Van de Walle. "RMLEditor: A Graph-Based Mapping Editor for Linked Data Mappings". In: *The Semantic Web – Latest Advances and New Domains (ESWC 2016)*. Vol. 9678. Lecture Notes in Computer Science. Springer, May 2016, pp. 709–723. ISBN: 978-3-319-34129-3. DOI: 10.1007/978-3-319-34129-3_43. URL: http://dx.doi.org/10.1007/978-3-319-34129-3_43.

[2] Álvaro Sicilia, German Nemirovski, and Andreas Nolle. "Map-On: A web-based editor for visual ontology mapping". In: *Semantic Web* 8.6 (2017), pp. 969–980. DOI: 10.3233/SW-160246. URL: https://doi.org/10.3233/SW-160246.

[3] Kunal Sengupta, Peter Haase, Michael Schmidt, and Pascal Hitzler. "Editing R2RML Mappings Made Easy". In: *Proceedings of the 2013th International Conference on Posters & Demonstrations Track*. Sydney, Australia: CEUR-WS.org, 2013, pp. 101–104. URL: http://dl.acm.org/citation.cfm?id=2874399.2874425.

[4] Peb R Aryan, Fajar J Ekaputra, Elmar Kiesling, A Min Tjoa, and Kabul Kurniawan. "RMLx: Mapping interface for integrating open data with linked data exploration environment". In: *1st International Conference on Informatics and Computational Sciences (ICICoS)*. IEEE, 2017, pp. 113–118. DOI: 10.1109/ICICOS.2017.8276347. URL: https://doi.org/10.1109/ICICOS.2017.8276347.

[5] Souripriya Das, Seema Sundara, and Richard Cyganiak. *R2RML: RDB to RDF Mapping Language*. Working Group Recommendation. W3C, Sept. 2012. URL: http://www.w3.org/TR/r2rml/.

[6] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data". In: *Workshop on Linked Data on the Web*. 2014. URL: http://events.linkeddata.org/ldow2014/papers/ldow2014_paper_01.pdf.

[7] Christoph Pinkel, Carsten Binnig, Peter Haase, Clemens Martin, Kunal Sengupta, and Johannes Trame. "How to best find a partner? An evaluation of editing approaches to construct R2RML mappings". In: *The Semantic Web: Trends and Challenges*. Springer, 2014. DOI: 10.1007/978-3-319-07443-6_45. URL: https://doi.org/10.1007/978-3-319-07443-6_45.

[8] Jonathan Robie, Michael Dyck, and Josh Spiegel. *XML path language (XPath) 3.1*. Tech. rep. W3C, 2017. URL: https://www.w3.org/TR/2017/REC-xpath-31-20170321/.

[9] Jonathan obie, Michael Dyck, and Josh Spiegel. *XQuery 1.0: An XML query language*. 2017. URL: https://www.w3.org/TR/2017/REC-xquery-31-20170321/.

[10] Stefan Bischof, Stefan Decker, Thomas Krennwallner, Nuno Lopes, and Axel Polleres. "Mapping between RDF and XML with XSPARQL". In: *Journal on Data Semantics* 1.3 (2012), pp. 147–185. ISSN: 1861-2032. DOI: 10.1007/s13740-012-0008-7. URL: https://doi.org/10.1007/s13740-012-0008-7.

[11] Manuel Fiorelli, Tiziano Lorenzetti, Maria Teresa Pazienza, Armando Stellato, and Andrea Turbati. "Sheet2RDF: a Flexible and Dynamic Spreadsheet Import&Lifting Framework for RDF". In: *Current Approaches in Applied Artificial Intelligence*. Springer, 2015. DOI: 10.1007/978-3-319-19066-2_13. URL: https://doi.org/10.1007/978-3-319-19066-2_13.

[12] Maria Teresa Pazienza, Armando Stellato, and Andrea Turbati. "PEARL: ProjEction of Annotations Rule Language, a Language for Projecting (UIMA) Annotations over RDF Knowledge Bases." In: *LREC*. 2012. DOI: 10.1.1.671.1240. URL: https://doi.org/10.1.1.671.1240.

[13] Mariano Rodrıguez-Muro, Josef Hardi, and Diego Calvanese. "Quest: efficient SPARQL-to-SQL for RDF and OWL". In: *11th International Semantic Web Conference: Posters and Demos*. 2012, pp. 53–56. URL: https://pdfs.semanticscholar.org/4d8e/acd402681fcb9aadb8cf225a226f8cccb52d.pdf.

[14] Natalya F Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W Fergerson, and Mark A Musen. "Creating Semantic Web Contents with Protégé-2000". In: *IEEE Intelligent Systems* 16.2 (2001), pp. 60–71. DOI: 10.1109/5254.920601. URL: https://doi.org/10.1109/5254.920601.

[15] Marc Friedman, Alon Y Levy, Todd D Millstein, et al. "Navigational plans for data integration". In: *AAAI/IAAI* 1999 (1999).

[16] Christian Bizer and Andy Seaborne. "D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs". In: *Proceedings of the 3rd International Semantic Web Conference*. 2004. URL: http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/Bizer-D2RQ-ISWC2004.pdf.

[17] Christoph Pinkel, Andreas Schwarte, Johannes Trame, Andriy Nikolov, Ana Sasa Bastinos, and Tobias Zeuch. "DataOps: Seamless End-to-End Anything-to-RDF Data Integration". In: *The Semantic Web: ESWC 2015 Satellite Events*. Springer, 2015, pp. 123–127. DOI: 10.1007/978-3-319-25639-9_24. URL: https://doi.org/10.1007/978-3-319-25639-9_24.

[18] Lushan Han, Tim Finin, Cynthia Parr, Joel Sachs, and Anupam Joshi. *RDF123: from Spreadsheets to RDF*. Springer, 2008. DOI: 10.1007/978-3-540-88564-1_29. URL: https://doi.org/10.1007/978-3-540-88564-1_29.

[19] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. "DBpedia - A crystallization point for the Web of Data". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 7.3 (2009), pp. 154–165. ISSN: 1570-8268. DOI: 10.1016/j.websem.2009.07.002. URL: http://dx.doi.org/10.1016/j.websem.2009.07.002.

[20] Fadi Maali, Richard Cyganiak, and Vassilios Peristeras. "Re-using Cool URIs: Entity Reconciliation Against LOD Hubs". In: *Linked Data on the Web*. 2011. URL: http://ceur-ws.org/Vol-813/ldow2011-paper11.pdf.

[21] Pieter Heyvaert, Anastasia Dimou, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "Towards a Uniform User Interface for Editing Mapping Definitions". In: *Proceedings of the 4th International Workshop on Intelligent Exploration of Semantic Data (IESD 2015)*. CEUR-WS.org, 2015. URL: http://ceur-ws.org/Vol-1472/IESD_2015_paper_4.pdf.

[22] Mark Richards. *Software Architecture Patterns*. O'Reilly, 2015.

[23] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. "D^3 Data-Driven Documents". In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 2301–2309.

[24] Addy Osmani. *Learning JavaScript Design Patterns*. O'Reilly Media, 2012.

[25] Ulrik Brandes, Markus Eiglsperger, Ivan Herman, Michael Himsolt, and M Scott Marshall. "GraphML progress report structural layer proposal". In: *Graph Drawing*. Springer. 2002. DOI: 10.1007/3-540-45848-4_59. URL: https://doi.org/10.1007/3-540-45848-4_59.

[26] Pieter Heyvaert, Anastasia Dimou, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "Towards Approaches for Generating RDF Mapping Definitions". In: *Proceedings of the 14th International Semantic Web Conference: Posters and Demos*. Oct. 2015. URL: http://ceur-ws.org/Vol-1486/paper_70.pdf.

[27] Erica L Olmsted-Hawala, Elizabeth D Murphy, Sam Hawala, and Kathleen T Ashenfelter. "Think-aloud protocols: a comparison of three think-aloud protocols for use in testing data-dissemination web sites for usability". In: *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM. 2010, pp. 2381–2390.

[28] John Brooke. "SUS – A quick and dirty usability scale". In: *Usability Evaluation in Industry* (1996), pp. 189–194.

[29] Rensis Likert. "A technique for the measurement of attitudes." In: *Archives of Psychology* 22 (1932).

[30] Ivan Ermilov, Jens Lehmann, Michael Martin, and Sören Auer. "LODStats: The Data Web Census Dataset". In: *Proceedings of 15th International Semantic Web Conference - Resources Track (ISWC'2016)*. 2016. URL: http://svn.aksw.org/papers/2016/ISWC_LODStats_Resource_Description/public.pdf.

[31] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. "LinkedGeoData: A Core for a Web of Spatial Open Data". In: *Semantic Web* 3.4 (2012), pp. 333–354. DOI: 10.3233/SW-2011-0052. URL: https://doi.org/10.3233/SW-2011-0052.

[32] The UniProt Consortium. "UniProt: a hub for protein information". In: *Nucleic Acids Research* 43.D1 (2015), p. D204. DOI: 10.1093/nar/gku989. URL: http://dx.doi.org/10.1093/nar/gku989.

[33] François Belleau, Marc-Alexandre Nolin, Nicole Tourigny, Philippe Rigault, and Jean Morissette. "Bio2RDF: Towards a mashup to build bioinformatics knowledge systems". In: *Biomedical Informatics* 41.5 (2008), pp. 706–716. DOI: 10.1016/j.jbi.2008.03.004. URL: https://doi.org/10.1016/j.jbi.2008.03.004.

[34] World Wide Web Consortium. *RDF 1.1 Concepts and Abstract Syntax*. Tech. rep. 2014. URL: https://www.w3.org/TR/rdf11-concepts/.

[35] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer Berlin Heidelberg, 2013. DOI: 10.1007/978-3-642-38721-0. URL: https://doi.org/10.1007/978-3-642-38721-0.

[36] Aba-Sah Dadzie and Matthew Rowe. "Approaches to visualising linked data: A survey". In: *Semantic Web* 2.2 (2011), pp. 89–124. DOI: 10.3233/SW-2011-0037. URL: https://doi.org/10.3233/SW-2011-0037.

[37] Giovanni Tummarello, Richard Cyganiak, Michele Catasta, Szymon Danielczyk, Renaud Delbru, and Stefan Decker. "Sig.ma: Live views on the Web of Data". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 8.4 (2010), pp. 355–364. ISSN: 1570-8268. DOI: 10.1016/j.websem.2010.08.003. URL: http://dx.doi.org/10.1016/j.websem.2010.08.003.

[38] Muhammad Javed, Sandy Payette, Jim Blake, and Tim Worrall. "VIZ–VIVO: Towards Visualizations-driven Linked Data Navigation". In: *Proceedings of the Second International Workshop on Visualization and Interaction for Ontologies and Linked Data*. 2016, p. 80.

[39] Simon Scheider, Auriol Degbelo, Rob Lemmens, Corné van Elzakker, Peter Zimmerhof, Nemanja Kostic, Jim Jones, and Gautam Banhatti. "Exploratory querying of SPARQL endpoints in space and time". In: *Semantic Web* 8.1 (2017), pp. 65–86. DOI: 10.3233/SW-150211. URL: https://doi.org/10.3233/SW-150211.

[40] Nikos Bikakis, Melina Skourla, and George Papastefanatos. "rdf:SynopsViz – A Framework for Hierarchical Linked Data Visual Exploration and Analysis". In: *The Semantic Web: ESWC 2014 Satellite Events*. Springer, 2014, pp. 292–297. ISBN: 978-3-319-11955-7. DOI: 10.1007/978-3-319-11955-7_37. URL: http://dx.doi.org/10.1007/978-3-319-11955-7_37.

[41] Philipp Heim, Sebastian Hellmann, Jens Lehmann, Steffen Lohmann, and Timo Stegemann. "RelFinder: Revealing relationships in RDF knowledge bases". In: *Semantic Multimedia*. Springer Berlin Heidelberg. 2009, pp. 182–187. DOI: 10.1007/978-3-642-10543-2_21. URL: https://doi.org/10.1007/978-3-642-10543-2_21.

[42] Marc Weise, Steffen Lohmann, and Florian Haag. "LD-VOWL: Extracting and Visualizing Schema Information for Linked Data". In: *Proceedings of the Second International Workshop on Visualization and Interaction for Ontologies and Linked Data* (2016), pp. 120–127. URL: http://ceur-ws.org/Vol-1704/paper11.pdf.

[43] Laurens De Vocht, Selver Softic, Ruben Verborgh, Erik Mannens, and Martin Ebner. "ResXplorer: Revealing relations between resources for researchers in the Web of Data". In: *Computer Science and Information Systems* 14.1 (2017), pp. 25–50.

[44] Aba-Sah Dadzie and Emmanuel Pietriga. "Visualisation of Linked Data – Reprise". In: *Semantic Web* 8.1 (2017), pp. 1–21. DOI: 10.3233/SW-160249. URL: https://doi.org/10.3233/SW-160249.

[45] Emmanuel Pietriga. "Semantic Web Data Visualization with Graph Style Sheets". In: *Proceedings of the 2006 ACM Symposium on Software Visualization*. SoftVis '06. Brighton, United Kingdom: ACM, 2006, pp. 177–178. ISBN: 1-59593-464-2. DOI: 10.1145/1148493.1148532. URL: https://doi.org/10.1145/1148493.1148532.

[46] Nikos Bikakis, George Papastefanatos, Melina Skourla, and Timos Sellis. "A Hierarchical Aggregation Framework for Efficient Multilevel Visual Exploration and Analysis". In: *Semantic Web* 8.1 (2017), pp. 139–179. DOI: 10.3233/SW-160226. URL: https://doi.org/10.3233/SW-160226.

[47] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. "Visualizing ontologies with VOWL". In: *Semantic Web* 7.4 (2016), pp. 399–419. DOI: 10.3233/SW-150200. URL: https://doi.org/10.3233/SW-150200.

[48] Sergey Krivov, Richard Williams, and Ferdinando Villa. "GrOWL: A tool for visualization and editing of OWL ontologies". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 5.2 (2007), pp. 54–57. ISSN: 1570-8268. DOI: 10.1016/j.websem.2007.03.005. URL: http://dx.doi.org/10.1007/978-3-319-11955-7_37.

[49] Matthew Horridge. *OWLViz*. 2010. URL: http://protegewiki.stanford.edu/wiki/OWLViz.

[50] Enrico Motta, Silvio Peroni, José Manuel Gómez-Pérez, Mathieu d′Aquin, and Ning Li. "Visualizing and navigating ontologies with KC-Viz". In: *Ontology Engineering in a Networked World*. Springer Berlin Heidelberg, 2012, pp. 343–362. DOI: 10.1007/978-3-642-24794-1_16. URL: https://doi.org/10.1007/978-3-642-24794-1_16.

[51] Steffen Lohmann, Vincent Link, Eduard Marbach, and Stefan Negru. "WebVOWL: Web-based Visualization of Ontologies". In: *Knowledge Engineering and Knowledge Management*. Springer, 2014, pp. 154–158. DOI: 10.1007/978-3-319-17966-7_21. URL: https://doi.org/10.1007/978-3-319-17966-7_21.

[52] Ajaz Hussain, Khalid Latif, A Rextin, Amir Hayat, and Masoon Alam. "Scalable Visualization of Semantic Nets using Power-Law Graphs". In: *Applied Mathematics & Information Sciences* 8.1 (2014), pp. 355–367. DOI: 10.12785/amis/080145. URL: http://dx.doi.org/10.12785/amis/080145.

[53] Karlis Cerans, Julija Ovcinnikova, Renars Liepins, and Arturs Sprogis. "Advanced OWL 2.0 Ontology Visualization in OWLGrEd". In: *Databases and Information Systems VII*. IOS Press, 2013, pp. 41–54. DOI: 10.3233/978-1-61499-161-8-41. URL: https://doi.org/10.3233/978-1-61499-161-8-41.

[54] Steffen Lohmann, Stefan Negru, and David Bold. "The ProtégéVOWL Plugin: Ontology Visualization for Everyone". In: *The Semantic Web: ESWC 2014 Satellite Events*. Springer. 2014, pp. 395–400. DOI: 10.1007/978-3-319-11955-7_55. URL: https://doi.org/10.1007/978-3-319-11955-7_55.

[55] Alexandra Similea, Niklas Petersen, Christoph Lange, and Steffen Lohmann. "TurtleEditor 2.0: A Synchronized Visual and Text Editor for RDF Graphs". In: *IEEE 11th International Conference on Semantic Computing*. IEEE, 2017. DOI: 10.1109/ICSC.2017.90. URL: https://doi.org/10.1109/ICSC.2017.90.

[56] Riccardo Falco, Aldo Gangemi, Silvio Peroni, David Shotton, and Fabio Vitali. "Modelling OWL Ontologies with Graffoo". In: *The Semantic Web: ESWC 2014 Satellite Events*. Springer, 2014, pp. 320–325. DOI: 10.1007/978-3-319-11955-7_42. URL: https://doi.org/10.1007/978-3-319-11955-7_42.

[57] Stefan Negru and Steffen Lohmann. "A Visual Notation for the Integrated Representation of OWL Ontologies". In: *Proceedings of the 9th International Conference on Web Information Systems and Technologies (WEBIST '13)*. SciTePress, 2013, pp. 308–315. DOI: 10.5220/0004373003080315. URL: https://doi.org/10.5220/0004373003080315.

[58] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF*. Tech. rep. URL: http://www.w3.org/TR/rdf-sparql-query/.

[59] Florian Haag, Steffen Lohmann, Stephan Siek, and Thomas Ertl. "QueryVOWL: Visual Composition of SPARQL Queries". In: *The Semantic Web: ESWC 2015 Satellite Events*. Springer, 2015, pp. 62–66. DOI: 10.1007/978-3-319-25639-9_12. URL: https://doi.org/10.1007/978-3-319-25639-9_12.

[60] Alistair Russell, Paul R. Smart, Dave Braines, and Nigel R. Shadbolt. "NITELIGHT: A Graphical Tool for Semantic Query Construction". In: *Proceedings of the Fifth International Workshop on Semantic Web User Interaction (SWUI 2008)*. 2008. URL: https://eprints.soton.ac.uk/264975/1/CHI_Paperv7.pdf.

[61] Frederik Hogenboom, Viorel Milea, Flavius Frasincar, and Uzay Kaymak. "RDF-GL: A SPARQL-Based Graphical Query Language for RDF". In: *Emergent Web Intelligence: Advanced Information Retrieval*. Springer, 2010, pp. 87–116. DOI: 10.1007/978-1-84996-074-8_4. URL: https://doi.org/10.1007/978-1-84996-074-8_4.

[62] Syeda Sana e Zainab, Muhammad Saleem, Qaiser Mehmood, Durre Zehra, Stefan Decker, and Ali Hasnain. "FedViz: A Visual Interface for SPARQL Queries Formulation and Execution". In: *Proceedings of the International Workshop on Visualizations and User Interfaces for Ontologies and Linked Data*. 2015, pp. 49–60. URL: http://svn.aksw.org/papers/2015/ISWC-VOILA_FedViz/public.pdf.

[63] Jill H Larkin and Herbert A Simon. "Why a diagram is (sometimes) worth ten thousand words". In: *Cognitive science* 11.1 (1987), pp. 65–100.

[64] Martin Dzbor, Enrico Motta, Carlos Buil, Jose Gomez, Olaf Goerlitz, and Holger Lewen. "Developing ontologies in OWL: An observational study". In: *OWL: Experiences and Directions 2006*. Nov. 2006. URL: http://oro.open.ac.uk/6241/.

[65] Daniel Moody. "The "physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering". In: *IEEE Transactions on Software Engineering* 35.6 (2009), pp. 756–779. DOI: 10.1109/TSE.2009.67. URL: https://doi.org/10.1109/TSE.2009.67.

[66] Allan Paivio. *Mental Representations: A Dual Coding Approach*. Oxford University Press, 1990.

[67] Mark Richards. *Software Architecture Patterns*. O'Reilly Media, 2015.

[68] Addy Osmani. *Learning JavaScript Design Patterns*. O'Reilly Media, 2012.

[69] Adam Boduch. *Flux Architecture*. Packt Publishing, 2016.

[70] Ulrik Brandes, Markus Eiglsperger, Ivan Herman, Michael Himsolt, and M Scott Marshall. "GraphML Progress Report Structural Layer Proposal". In: *International Symposium on Graph Drawing*. Springer Berlin Heidelberg, 2001, pp. 501–512.

[71] Ben De Meester, Anastasia Dimou, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "An Ontology to Semantically Declare and Describe Functions". In: *The Semantic Web; ESWC 2016 Satellite Events*. Vol. 9989. Lecture Notes in Computer Science. Springer, Oct. 2016, pp. 46–49. DOI: 10.1007/978-3-319-47602-5_10. URL: https://doi.org/10.1007/978-3-319-47602-5_10.

[72] Ben De Meester and Anastasia Dimou. *The Function Ontology*. Unofficial Draft. http://users.ugent.be/ bjdmeest/function/. Oct. 2016.

[73] Pierre-Yves Vandenbussche, Ghislain A Atemezing, María Poveda-Villalón, and Bernard Vatant. "Linked Open Vocabularies (LOV): a gateway to reusable semantic vocabularies on the Web". In: *Semantic Web* 8.3 (2017), pp. 437–452. DOI: 10.3233/SW-160213. URL: https://doi.org/10.3233/SW-160213.

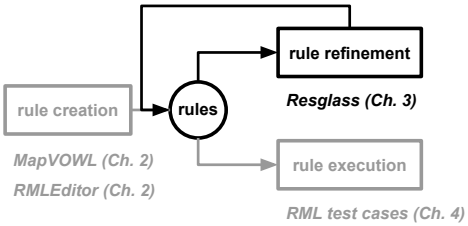
[74] Ivan Herman, Guy Melançon, and M Scott Marshall. "Graph Visualization and Navigation in Information Visualization: A Survey". In: *IEEE Transactions on Visualization and Computer Graphics* 6.1 (2000), pp. 24–43. DOI: 10.1109/2945.841119. URL: https://doi.org/10.1109/2945.841119.

[75] Ben Shneiderman. "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations". In: *Proceedings 1996 IEEE Symposium on Visual Languages*. IEEE. 1996, pp. 336–343. DOI: 10.1109/VL.1996.545307. URL: https://doi.org/10.1109/VL.1996.545307.

[76] Siné JP McDougall, Oscar de Bruijn, and Martin B Curry. "Exploring the effects of icon characteristics on user performance: the role of icon concreteness, complexity, and distinctiveness". In: *Journal of Experimental Psychology: Applied* 6.4 (2000), p. 291.

[77] Aaron Bangor, Philip Kortum, and James Miller. "Determining what individual SUS scores mean: Adding an adjective rating scale". In: *Journal of usability studies* 4.3 (2009), pp. 114–123. URL: http://uxpajournal.org/wp-content/uploads/sites/8/pdf/JUS_Bangor_May2009.pdf.

[78] Thomas S Tullis and Jacqueline N Stetson. "A comparison of questionnaires for assessing website usability". In: *Usability professional association conference*. 2004, pp. 1–12. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.396.3677&rep=rep1&type=pdf.



Chapter 3

Rule-driven inconsistency resolution for rule refinement

Inconsistencies are introduced in graphs when ontology terms are used without adhering to restrictions given by the ontologies, and this affects the graphs' quality. Possible root causes for these inconsistencies include: (i) semantic model that introduces new inconsistencies by, for example, not using the suitable ontology terms [1, 2]; and (ii) ontology definitions that do not model the domain as desired [2]. In previous research efforts, a method to resolve inconsistencies by automatically refining the corresponding rules has been developed [1]. However, this method assumes that used ontologies align with the user's envisioned semantic model, which is not always the case [2]. More, when a high number of rules are involved in inconsistencies users have no insights regarding the order in which rules should be inspected.

Whereas Chapter 2 contributes to rule creation, this chapter contributes to rule refinement, by describing Resglass: our rule-driven method for the resolution of inconsistencies in ontologies and rules. We address Research Question 3 "How can we score and rank rules and ontology terms for inspection to improve the manual resolution of inconsistencies?" and validate Hypothesis 3 "The automatic inconsistency-driven ranking of Resglass improves, compared to a random ranking, by at least 20% the overlap with experts' manual ranking."

*

Pieter Heyvaert, Ben De Meester, Anastasia Dimou, and Ruben Verborgh

Published as "Rule-driven inconsistency resolution for knowledge graph generation rules" in *Semantic Web Journal*, 2019.

Abstract

Knowledge graphs, which contain annotated descriptions of entities and their interrelations, are often generated using rules that apply semantic annotations to certain data sources. (Re)using ontology terms without adhering to the axioms defined by their ontologies results in inconsistencies in these graphs, affecting their quality. Methods and tools were proposed to detect and resolve inconsistencies, the root causes of which include rules and ontologies. However, these either require access to the complete knowledge graph, which is not always available in a time-constrained situation, or assume that only generation rules can be refined but not ontologies. In the past, we proposed a rule-driven method for detecting and resolving inconsistencies without complete knowledge graph access, but it requires a predefined set of refinements to the rules and does not guide users with respect to the order the rules should be inspected. We extend our previous work with a rule-driven method, called Resglass, that considers refinements for generation rules as well as ontologies. In this article, we describe Resglass, which includes a ranking to determine the order with which rules and ontology elements should be inspected, and its implementation. The ranking is evaluated by comparing the manual ranking of experts to our automatic ranking. The evaluation shows that our automatic ranking achieves an overlap of 80% with experts ranking, reducing this way the effort required during the resolution of inconsistencies in both rules and ontologies.

3.1 Introduction

Knowledge graphs use ontologies to provide annotated descriptions of entities and their interrelations [3]. The graphs can be published as Linked Data [4] using the Resource Description framework (RDF) [5] as data representation format.

Knowledge graphs are often generated from other sources. For instance, the DBpedia knowledge graph is generated from Wikipedia. A common way to generate these knowledge graphs is by using rules. The rules attach semantic annotations to data in those sources. The semantic annotations are added using terms defined in ontologies, such as *classes*, *properties*, and *datatypes*. These rules thereby determine how individual data fragments from the sources are modeled using specific ontology terms during knowledge graph generation. The syntax and grammar of the rules are determined by a knowledge graph generation language, such as R2RML [6], RML [7], and SPARQL-Generate [8].

An *ontology* is a conceptualization, an intensional semantic structure, which encodes the implicit rules constraining the structure of a piece of reality [9]. Such implicit

rules can be encoded as ontological *axioms* in OWL [10] and are henceforth referred to as *restrictions*. For example, the domains and ranges of properties are restricted to a set of classes. Such restrictions are either defined via the ontology term's definitions, e.g., a term is a class or a property, or via the interpretation of the ontology's axioms, e.g, domain and range of a property, as restrictions [11, 12].

Inconsistencies are introduced in graphs when ontology terms are used without adhering to restrictions, and this affects the graphs' quality. Possible root causes for these inconsistencies include: (i) *raw data* that contain inconsistencies [13]; (ii) *rules* that introduce new inconsistencies by, for example, not using the suitable ontology terms [2, 1]; and (iii) *ontology definitions* that do not model the domain as desired [2]. In this work, we focus on the latter two root causes which are related to the intrinsic dimension of knowledge graph quality [7].

Previous research efforts introduced methods and tools to identify inconsistencies in knowledge graphs [12, 14]. Although this enables resolving the inconsistencies in the graph itself, it does not fix the root cause. The same inconsistencies reappear when a new version of a knowledge graph is generated. Thus, methods and tools were developed to identify inconsistencies in knowledge graph generation rules [1, 2]. Methods applied to generation rules find inconsistencies in less time compared to solutions that work directly on knowledge graphs, while they simultaneously identify the rules and ontology definitions causing them [15]. For instance, there were 2,159 inconsistencies identified in the rules that define how the DBpedia knowledge graph is generated from Wikipedia.

The rules or ontology definitions need to be refined to resolve inconsistencies, but this is not straightforward. The situation aggravates when the set of rules and their relationships grow, or multiple and more complex ontologies are used. For instance, more than 1,300 of the more than 1,200,000 rules that generate DBpedia are involved in at least 1 of the 2,159 identified inconsistencies. Which rules should users inspect first when resolving these inconsistencies, considering that updating a rule can resolve multiple inconsistencies, but it can also create new ones? Furthermore, a number of inconsistencies are best resolved by updating the ontology definitions and not the rules [2]. Should a user inspect the rules or also the ontology definitions, considering that inconsistencies might be caused by both rules and ontology definitions?

We proposed a rule-driven method in previous work to resolve inconsistencies by automatically refining the corresponding generation rules [1]. Inconsistencies are detected by analyzing primarily the rules, but also the knowledge graphs. Predefined refinements are automatically applied to resolve the inconsistencies. However, our rules-driven method assumes that used ontologies align with the user's envisioned semantic model, which is not always the case [2]. More, when a high number of rules are involved in inconsistencies, users have no insights regarding the order with which rules should be inspected. This leads to the following research question to improve the resolution process:

Research Question 3: *How can we score and rank rules and ontology terms for inspection to improve the manual resolution of inconsistencies?*

In this work, we introduce a new method called Resglass that extends our rule-driven method [1] to address this research question. The rules and ontology terms are automatically ranked in order of inspection based on a score that considers the number of inconsistencies a rule or ontology term is involved in. This way, the automatic ranking accelerates the inconsistencies resolution speed, as users do not need manually provide a ranking anymore. Resglass consists of the following steps:

1. (automatically) detect inconsistencies by analyzing generation rules;
2. (automatically) cluster the rules;
3. (automatically) rank rules and ontology terms in order of expected impact upon inspection;
4. (manually) refine rules and ontology terms;
5. (automatically) generate the knowledge graph;
6. (automatically) detect inconsistencies in this graph;
7. (manually) refine rules and ontologies terms.

This research question leads to the hypothesis:

Hypothesis 3: *The automatic inconsistency-driven ranking of Resglass improves, compared to a random ranking, by at least 20% the overlap with experts' manual ranking.*

Our novel contributions include in particular:

- an algorithm to *rank rules* in descending order of number of inconsistencies they are involved in;
- an algorithm to *rank ontology terms* in descending order of number of inconsistencies they are involved in;
- an *implementation of the Resglass* using an existing knowledge graph generation language, RML;
- an *evaluation* that shows how much our automatic ranking improves the overlap to the experts' ranking, compared to a random ranking,

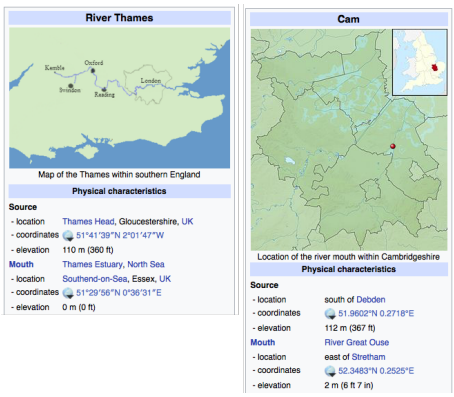


Figure 3.1: River Thames and Cam Wikipedia infobox

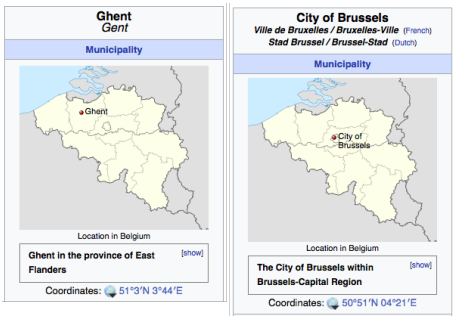


Figure 3.2: Ghent and Brussels Wikipedia infobox

Our evaluation shows that our ranking improves the overlap with experts' ranking by 40% in the case of rules and 20% in the case of ontology terms. Overall, Resglass reduces the effort required during the resolution of inconsistencies in both rules and ontologies, because less manual effort is required from the experts, leading to higher quality knowledge graphs in less time and with less effort.

The remainder of this article is structured as follows: In Section 3.2, we introduce our motivating use case. In Section 3.3, we discuss the state of the art. In Section 3.4, we elaborate on the Resglass and, in Section 3.5, on the implementation. In Section 3.6, we present our evaluation and, in Section 3.7, our conclusions.

3.2 Motivating use case

As motivating use case, we consider DBpedia. The DBpedia knowledge graph is generated from Wikipedia. The rules that define how to annotate the Wikipedia infoboxes

with classes and properties from different ontologies, including the DBpedia, FOAF and GS84 Geo Positioning ontologies are defined in rules expressed in RML. However, inconsistencies appear in this knowledge graph [12] and are caused by both rules and ontology definitions [2].

The two types of Wikipedia infoboxes in Figures 3.1 and 3.2, provide details of two rivers (Thames and Cam) and two cities (Ghent and Brussels). Each infobox presents the relevant information about a single entity based on a predefined infobox template. For example, the infoboxes used for rivers follow the same template[1]. The rules (Listing 3.1) annotate the data with ontology definitions (Listing 3.2) that describe rivers, palaces and cities, and generate the knowledge graph (Listing 3.3). Inconsistencies can then be detected by comparing the rules and ontologies definitions.

1st inconsistency: The triple in line 2 states that `_:b0` is the source position of river Thames and the triple in line 5 states that `_:b0` is of class `dbo:Place` (see Listing 3.3). The same holds for river Cam and its source position `_:b1`. However, the ontology defines that the class `geo:SpatialThing`, and not `dbo:Place`, is in the range of the property `dbo:sourcePosition` (see Listing 3.2).

2nd inconsistency: The triple in line 3 states that `_:b2` is the mouth position of river Thames and the triple in line 8 states that `_:b2` is of class `dbo:Place`. However, the ontology defines that the range of `dbo:mouthPosition` is the class `geo:SpatialThing`.

3rd inconsistency: The triples in line 6 and 16 state `_:b0`'s and `_:b1`'s latitude, respectively. However, the ontology defines that `geo:SpatialThing`, and not `dbo:Place`, is in the domain of `geo:lat`.

The *1st inconsistency* is caused due to the combination of rules 21 and 38 (Listing 3.1). Rule 21 links rivers with their locations via `dbo:sourcePosition` and rule 38 annotates a location with the class `dbo:Place`. This is not consistent with the ontology, as the class `geo:SpatialThing`, and not `dbo:Place`, is in the range of `dbo:sourcePosition`.

The *2nd inconsistency* is caused due to the combination of rules 29 and 52. Rule 29 links rivers with their locations via `dbo:mouthPosition` and rule 52 annotates a location with the class `dbo:Place`. This is not consistent with the ontology, as class `geo:SpatialThing`, and not `dbo:Place`, is in the range of `dbo:mouthPosition`.

The *3rd inconsistency* is caused due to the combination of rules 38 and 43. Rule 38 annotates a location with the class `dbo:Place` and rule 43 the latitude of a location with the property `geo:lat`. This is not consistent with the ontology, as class `geo:SpatialThing`, and not `dbo:Place`, is in the domain of `geo:lat`.

How should these inconsistencies be resolved? On the one hand, which rules should be refined? Is it one of the rules 21, 29, 38, 43, 52, or all of them? How should they be refined: should the class in rule 38 be `geo:SpatialThing` or another class? What are the effects of updating these rules? For instance, this refinement would lead to other inconsistencies, such as caused by rules 38 and 66 if the entity is

[1] `https://en.wikipedia.org/wiki/Template:Infobox_river`

not longer annotated with the class `dbo:Place`. On the other hand, if the ontology can be updated, which definitions should be refined? How should the definitions be refined: remove definitions 7, 9, and 13, or add other definitions to expand the range of `dbo:sourcePosition`, `dbo:mouthPosition`, and the domain of `geo:lat`?

Although we just discussed a small subset of the rules and definitions of DBpedia, we already have a number of different rules and definitions to consider for inspection, and different approaches to resolve the inconsistencies, which might introduce new inconsistencies. This aggravates when we consider the whole of DBpedia where more than 1,300 of the more than 1,200,000 rules are involved in at least one of the 2,159 inconsistencies, and where the DBpedia ontology[2] alone already consists of more than 700 classes and more than 2,800 properties.

Listing 3.1: Subset of example RML rules

```
<#TriplesMap1> rr:subjectMap <#SM1>;
  rr:predicateObjectMap
  <#POM1>, <#POM2>, <#POM3>.

<#SM1> rr:template
  "http://dbpedia.org/resource/{slug}".

<#POM1> rr:predicate rdf:type;
  rr:objectMap <#OM1>.

<#OM1> rr:constant dbo:River.

<#POM2> rr:predicate foaf:name;
  rr:objectMap [rml:reference "name"].

<#POM3> rr:predicateMap <#PM1>;
  rr:objectMap[rr:joinCondition[
    rr:child "slug"; rr:parent "slug";
    rr:parentTriplesMap <#TriplesMap2>]].

<#PM1> rr:constant dbo:sourcePosition.

<#POM3> rr:predicateMap <#PM4>;
  rr:objectMap[rr:joinCondition[
    rr:child "slug"; rr:parent "slug";
    rr:parentTriplesMap <#TriplesMap3>]].

<#PM4> rr:constant
  dbo:mouthPositionPosition.

<#TriplesMap2>
```

[2] `http://dbpedia.org/ontology/`

```
  rr:subjectMap[rr:termType rr:BlankNode].
  rr:predicateObjectMap <#POM4>,<#POM5>.

<#POM4> rr:predicate rdf:type;
  rr:objectMap <#OM2>.

<#OM2> rr:constant dbo:Place.

<#POM5> predicateMap <#PM2>;
  rr:objectMap[rml:reference "latitude"].

<#PM2> rr:constant geo:lat.

<#TriplesMap3>
  rr:subjectMap[rr:termType rr:BlankNode].
  rr:predicateObjectMap <#POM6>,<#POM7>.

<#POM6> rr:predicate rdf:type;
  rr:objectMap <#OM3>.

<#OM3> rr:constant dbo:Place.

<#POM7> predicateMap <#PM3>;
  rr:objectMap[rml:reference "latitude_m"].

<#PM3> rr:constant geo:lat.

<#TriplesMap4>
  rr:predicateObjectMap [
    rr:predicateMap <#PM5>;
    rr:objectMap[rr:joinCondition[
      rr:child "slug"; rr:parent "slug";
      rr:parentTriplesMap <#TriplesMap2>]].

<#PM5> rr:constant dbo:location.
```

Listing 3.2: Ontology that describes people and furniture

```
dbo:River is a class
dbo:City is a class
dbo:River and dbo:City are disjoint
dbo:Place is a class
geo:SpatialThing is a class
dbo:location is a property
dbo:Place is in the range of dbo:location
dbo:sourcePosition is a property
geo:SpatialThing is in the range of dbo:sourcePosition
```

```
dbo:mouthPosition is a property
geo:SpatialThing is in the range of dbo:mouthPosition
geo:lat is a property
geo:SpatialThing is in the domain of geo:lat
geo:lat's object is of the datatype float
```

Listing 3.3: A knowledge graph generated by applying the rules in Listing 3.1 on the data in Figures 3.1 and 3.2

```
dbr:river_thames a dbo:River;
  dbo:sourcePosition _:b0;
  dbo:mouthPosition _:b2.

_:b0 a dbo:Place;
  geo:lat "51°41'39\"N".

_:b2 a dbo:Place;
  geo:lat "51°29'56\"N".

dbr:river_cam a dbo:River;
  dbo:sourcePosition _:b1;
  dbo:mouthPosition _:b3.

_:b1 a dbo:Place;
  geo:lat "52°20'54\"N".

_:b3 a dbo:Place;
  geo:lat "52°20'54\"N".

dbr:ghent a dbo:City;
  dbo:location [
    a dbo:Place;
    geo:lat "51°2'60\"N" ].

dbr:brussels a dbo:City;
  dbo:location [
    a dbo:Place;
    geo:lat "50°50'60\"N" ].
```

3.3 Related work

Knowledge graphs can be generated via rules in different languages (see Section 3.3.1). These graphs can contain inconsistencies, which can be detected by assessing the graphs' quality (see Section 3.3.2). Once these inconsistencies are detected they are resolved to improve the knowledge graph (see Section 3.3.3).

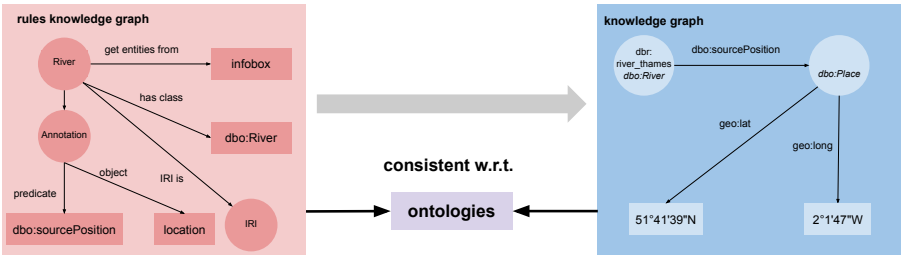


Figure 3.3: A knowledge graph needs to be consistent with regard to the used ontologies. Therefore, rules, which also form a knowledge graph, also have to be consistent with regard to the same used ontologies.

A number of research efforts assume that knowledge graphs are materialized via the Resource Description Framework (RDF) [5, 16]. RDF uses a graph-based model with a set of triples as core structure. Each triple consists of a subject, predicate, and object. A set of triples are called an RDF graph.

3.3.1 Knowledge graph generation

Two approaches prevailed for knowledge graph generation from existing data sources: direct mapping and custom rules. In the former case, the direct mapping defines a simple transformation, providing a basis for defining more complex transformations afterwards [17]. A Direct Mapping of relational data to RDF was recommended by W3C [17]. However, this recommendation only refers to data existing in relational databases and requires defining rules later, e.g., using SPARQL queries [18], to replace the original predefined annotations with custom ones.

In the latter case, knowledge graph generation languages, e.g., the W3C recommended R2RML [6], RML [7], and SPARQL-Generate [8], offer a declarative way to define rules that specify how knowledge graphs are generated from raw data. [R2]RML rules are in RDF, forming themselves a knowledge graph, i.e., the so-called *rules knowledge graph* (see Figure 3.3).

Knowledge graphs are often constructed by consistently applying the terms of certain ontologies, i.e., the graphs respect the restrictions imposed by the definitions in the ontologies. This is also applicable to rules knowledge graphs, i.e., they need to be constructed so that the knowledge graphs, which are generated by executing these rules, respect the restrictions imposed by the used ontologies. However, consistently annotating the existing data sources with ontology terms is not always straightforward. Inconsistencies are introduced when the rules are defined. Indicatively, the Semantic Publishing Challenge required different solutions to generate knowledge graphs from the CEUR-WS proceedings [19]. It was observed that, despite the fact that these solutions were provided by Semantic Web experts, the data was modeled differently and

each solution introduced different inconsistencies.

3.3.2 Knowledge graph quality assessment

There exists a number of methods to assess the quality of knowledge graphs. On the one hand, there are methods applied directly to the knowledge graph, based on e.g., crowdsourcing [20], the comparison of the results of queries [12, 21], inference rules [14, 2], evolution analysis [22], or custom characteristics [23]. These methods have access to the complete knowledge graph and can identify every inconsistency. However, they require the graph to be available, which is not always possible in a time-constrained situation [15].

On the other hand, there are methods applied to the rules that generate knowledge graphs, such as our previous work [1] which is based on the aforementioned comparison of the results of queries originally applied to the knowledge graph [12]. Such methods result in faster execution times, but not all inconsistencies can be identified, as some of them depend on the actual data values in the graph.

Kontokostas et al. [12] introduced a list of common patterns, which are called *constraint types* [24], that can be used to detect inconsistencies in a knowledge graph. These constraint types can also be used to find inconsistencies in rules for knowledge graph generation. As these patterns were designed for the resulting knowledge graph, not all of them can be applied to the rules. More specific, when a pattern refers to specific values, then we can only identify inconsistencies when the rules specify a constant value for these values, i.e., does not refer to actual data in the data source. For example, if we have a constraint on the range of a property's numerical value, and the value in the knowledge graph is based on a number in the existing data source, then we cannot know if the number is within the interval without inspecting the data source. However, we know if a constant value is within the range, as this constant value does not depend on the data source.

3.3.3 Inconsistency resolution

We can identify the following distinct approaches for resolving inconsistencies, which assume RDF as the means to represent knowledge graphs: inconsistencies are resolved by updating (i) the knowledge graph directly, i.e., *triple-level*, (ii) the rules that generate the knowledge graph, i.e., *rule-level*, or (iii) the ontology definitions, i.e., *ontology-level*.

3.3.3.1 Triple-level

Inconsistencies identified in knowledge graphs can be resolved by refining the graphs directly. For RDF, this means adding, removing, or refining certain triples.

Sieve [23] is a framework for quality assessment and fusion of knowledge graphs. The quality assessment task is realized through a flexible module, where users can

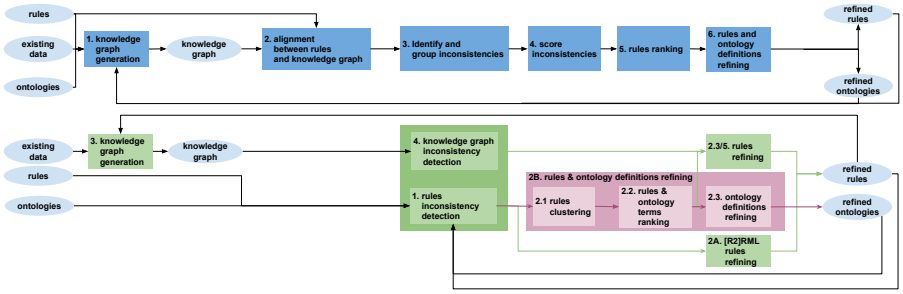


Figure 3.4: The top describes the steps (blue rectangles) followed during the analysis of DBpedia by Paulheim [2]. The bottom describes the steps of our previous work [1] (green rectangles) and Resglass (purple rectangles), where steps 1, 3, 4 and 5 from our previous work are reused by Resglass.

choose which characteristics of the data indicate higher quality, how this quality is quantified, and how it should be stored in the system. The output of this task is a set of scores used during the fusion task. This helps users in determining which data should be removed or transformed.

CLAMS [21] is a system to discover and resolve inconsistencies in knowledge graphs. It defines an inconsistency as a minimal set of triples that cannot coexist. The system identifies all inconsistencies through the execution of a set of queries. The involved triples are ordered based on the number of inconsistencies they participate in. Removing any triple from that set will resolve the inconsistency. Users use the system's graphical user interface (GUI) to update or remove the triples. The GUI allows seeing all inconsistencies that a triple participates in and why it is part of a particular inconsistency. Once a triple is updated or removed the set of inconsistencies and involved triples are updated.

These tools enable resolving inconsistencies in the knowledge graph, but the inconsistencies in the rules remain. Consequently, when regenerating the knowledge graph with the unaltered rules, the same inconsistencies will be present again.

3.3.3.2 Rule-level

Inconsistencies identified in knowledge graphs can be resolved by refining the rules, instead of the generated graphs. Knowledge graph refinement through the use of external methods [3] occurs when the source of knowledge to refine the knowledge graph is not part of the original knowledge graph. In our previous work [1], we proposed such a refinement where the source of the knowledge is the ontologies and the knowledge graph is the rules. Our uniform, iterative, incremental assessment and refinement method for RML rules produces a high-quality knowledge graph, in the form of an RDF dataset (see bottom of Figure 3.4). It is created by applying the assessment process, normally

Table 3.1: Comparison of where the different methods consider refinements, and for what they provide rankings. "(x)" is used when the input is not required at the start.

	Refinements in		**Ranking of**	
	rules	**ontologies**	**rules**	**ontologies**
Dimou et al. [1]		x		
Paulheim [2]	x	x	x	
Resglass	x	x	x	x

Table 3.2: Comparison of the required input of the different methods. "(x)" is used when the input is not required at the start.

	Existing data	**Knowledge graph**	**Rules**	**Ontologies**
Dimou et al. [1]	(x)	(x)	x	x
Paulheim [2]	x	x	x	x
Resglass	(x)	(x)	x	x

applied to the knowledge graph, to the knowledge graph generation rules. This allows discovering inconsistencies, before the knowledge graph is generated. The rules assessment takes significant less time compared to assessing the actual graph which might take a considerable amount of time [15]. Indicatively, assessing the knowledge graph of the English DBpedia takes approximately 16 hours, assessing the knowledge graph of the rules that generate the entire DBpedia takes only 32 seconds. The method consists of the following steps:

1. Inconsistencies are detected via the rules, as it would have been done on the actual dataset.

2. Rules are automatically refined (step 2A at the bottom of Figure 3.4) and reassessed to detect new inconsistencies.

3. The refined version of the rules is used to generate the knowledge graph.

4. The generated knowledge graph is assessed, using the same quality assessment framework, to find remaining inconsistencies.

5. Rules can be refined again to resolve these inconsistencies.

The same constraint types, normally applied to an RDF dataset, are considered for the rules. For example, instead of validating the predicate's domain and range against the triple's subject and object respectively, we validate the rules that define how the subject, predicate, and object are generated. The properties and classes in the rules are identified and their schemas are used to generate test cases, as for the actual dataset.

We adjusted the assessment queries by Kontokostas et al. [12] to apply them to the rules. However, some of the constraint types normally applied to a dataset rely on the final values or refer to the complete dataset and, thus, can only be validated after the rules are executed. For example, when the literal value of a certain property is constrained within a given range. We refer to the original work [12] which details how the violation patterns are aligned to the rules that should be refined.

Although we introduced a rule-driven inconsistency resolution method, the aforementioned two steps focus on [R2]RML [7, 6], while a similar method might also be applied on knowledge graph generation rules defined using a different language. More, it assumes that the used ontologies correctly define the user's envisioned semantic model, which is not always the case [2]. User intervention can be considered to decide if the knowledge graph generation rules or ontologies need to be refined. However, the method does not provide a way to guide users regarding which rules should be inspected first and how these rules and ontologies can be refined.

3.3.3.3 Ontology-level

Paulheim [2] performed a *data-driven analysis* on the DBpedia rules and concluded that refinements might not only be needed on rules, but also on the ontology definitions. An overview of the steps followed during his analysis can be found here and on the top of Figure 3.4:

1. **Identifying and grouping inconsistencies.** Relation assertions and their subject's and object's types are extracted from the knowledge graph. The rules that contribute to these assertions and types are identified. They are used, together with the ontologies, to determine the inconsistencies through reasoning, i.e., which combination of assertions and types are inconsistent with respect to the ontology definitions. All rules are marked that contribute to a specific inconsistency. For each rule, two counters are kept: how often the rule generates an assertion involved in an inconsistency (i_m, were m is a rule) and how often it does not (c_m).

2. **Scoring inconsistencies.** A score ($score(m)$) is calculated for all rules as the harmonic mean of the logarithmic support ($logs(m)$) and confidence ($c(m)$). The logarithmic support is calculated as $logs(m) = \frac{log(i_m+1)}{log(N+1)}$ and the confidence as $c(m) = \frac{i_m}{i_m+c_m}$, were N is total number of statements in the knowledge graph. The final score is calculated as follows $score(m) = \frac{2 \cdot logs(m) \cdot c(m)}{logs(m)+c(m)}$.

3. **Inspect top rules.** The rules are ranked based on the scores and the top rules are manually inspected to determine if the rules or ontology definitions should be refined.

4. **Update rules or ontologies.** The refinements are applied, which results in refined rules and ontology definitions. The process can be repeated to determine

and fix still remaining or newly identified inconsistencies using the refined rules and ontology definitions.

These steps fill a gap in our previous work (see Table 3.1), because it provides a set of top ranked rules that should be manually inspected by an expert, similar to Sieve and CLAMS. However, (i) these steps were followed to provide a *preliminary analysis*, but do not form a systematic method that improves our original work on inconsistencies resolution [1]. For instance, the knowledge graph and its generation rules alignment are case specific, namely it is custom to the DBpedia knowledge graph, thus, it can not be applied beyond the scope of the explored use case. (ii) The alignment among the knowledge graphs and rules that generated them needs to be determined separately in a dedicated module, as the provenance of the knowledge graph needs to be reconstructed, but this might not always be accurate or even possible; (iii) the steps are limited to ranking rules and do not support ontology terms ranking, but only refinements to ontology definitions; and (iv) the complete knowledge graph is needed which increases the execution time, making this method unsuitable for use cases that deal with time constraints [15] (see Table 3.2).

Rashid et al. [22] provide a method on evolving knowledge bases. Their method looked into identifying completeness issues using knowledge based evolution analysis, and consistency issues based on integrity constraints. However, their method did not look into addressing the results of quality assessment and, thus, do not provide any suggestions with regard to the inconsistencies resolution.

3.4 Resglass

We propose a new method called Resglass to resolve inconsistencies in knowledge graphs that occur due to the (i) rules that define how the graphs are generated, or (ii) ontology terms that annotate the data. To achieve this, we extend our previous work on assessing rules knowledge graph [1], by ranking the rules, as proposed in the data-driven analysis [2] (see Section 3.3.3.2), and by ranking the ontology terms (see Table 3.1).

Resglass (i) detects inconsistencies by analyzing the rules, instead of the knowledge graph; (ii) clusters the rules involved in an inconsistency; (iii) ranks the rules and ontology terms in the order that they should be inspected by experts; (iv) refines the rules and ontologies based on the refinements given by the experts; (v) generates the knowledge graph; (vi) detects inconsistencies in the knowledge graph; and (vii) refines the rules and ontologies based on these inconsistencies. An overview of Resglass can be found here and at the bottom of Figure 3.4:

1 **Rules inconsistency detection.** We validate the rules. The outcome is a set of inconsistencies and the (combination of) rules and ontology terms involved in these inconsistencies.

2 **Rules and ontology definitions refinement.**

   2.1 **Rules clustering.** We cluster the rules per entity, because rules concerning a single entity impact each other, and, thus, their refinements too.

   2.2 **Rules and ontology terms ranking.** We calculate a score for each rules cluster and ontology term. We use this score to provide a ranking for the rules and ontology terms.

   2.3 **Rules and ontology definitions refinement.** We inspect the top rules and ontology definitions that correspond with the terms manually and apply necessary refinements.

3 **Knowledge graph generation.** We use the refined rules and ontology definitions to generate the knowledge graph.

4 **Knowledge graph inconsistency detection.** We validate the knowledge graph. The outcome of this step is a set of inconsistencies and the (combination of) rules and ontology terms involved in these inconsistencies.

5 **Rules and ontology definitions refinement.** We further refine the rules and ontology definitions to resolve the newly discovered inconsistencies.

In the remainder of this section, we provide a detailed explanation of the aforementioned steps.

3.4.1 Rules inconsistency detection

We validate the rules to determine which inconsistencies are present with respect to used ontologies (see step 1 at the bottom of Figure 3.4). This step is analogous to the first step of our previous work and consists of three substeps:

1 Instantiated constraints are generated by aligning the constraint types with the axioms from the ontologies. This is done assuming that axioms can be interpreted as constraints [12] (see Section 3.3.2). This substep does not depend on how the constraints are described, other constraints can also be used, such as SHACL constraints [25].

2 Rules that could make an instantiated constraint fail are grouped, based on the types of rules that are involved in each constraint.

3 The groups are analyzed to assess if they respect the related constraint. If this is not the case, an inconsistency is found. For each inconsistency that is present, we call the group of rules that cause it the *involved rules*, and the related ontology terms *involved ontology terms*.

Example Consider the axiom on line 13 in the ontology (see Listing 3.2). The corresponding instantiated constraint is as follows: for every rule that annotates an entity with the property `geo:lat`, there should exist a rule that annotates that same entity with the class `geo:SpatialThing`. We determine all subsets of rules that could lead to the failure of this constraint. In our example we have two subsets: rules 38 and 43, and 52 and 57 (see Listing 3.1). The constraint fails because the entity is annotated only with `dbo:Place` and not `geo:SpatialThing`.

3.4.2 Rules and ontology definitions refinement

This step uses the knowledge about the inconsistencies to refine the rules and ontology definitions via three steps: rules clustering, rules and ontology terms ranking, and rules and ontology definitions refinement. These steps are different from our previous work where we use an automatic approach that applies a set of predefined refinements to the rules (see Section 3.3.3.2, steps 2A and 2B at the bottom of Figure 3.4).

3.4.2.1 Rules clustering

The involved rules are clustered based on their contribution to generate a knowledge graph from the different records in the same source, e.g., rows in a table or infoboxes in Wikipedia. First, the specific type of the record to which a rule contributes is identified, e.g., Wikipedia infoboxes that follow the river template are a single type of record. Next, the rules are grouped per type of record, e.g., all rules that contribute to the infoboxes of the river template are in the same group. Clustering has been applied in other research domain, such as information exploration [26]. For example, Web search results are clustered, based on a set of similarities, to allow users to quickly browse through the returned documents, because they are related to each other [27, 28]. We applied the same approach to rules where the similarity is determined by the type of record. More specific, rules that contribute to the same type of record are related to each other, i.e., other rules might affect the refinement that needs to be applied to a rule. Therefore, we cluster the rules based on the type of record, so that they can be inspected together.

Example Rules 21, 29, 38, 43, 52, and 57 are involved in inconsistencies (see Listing 3.1). Rules 38 and 43 are related to the source locations. Rules 52 and 57 are related to the mouth locations. Rules 21 and 29 are related to the river entities. This results in three clusters: one with rules 38 and 43, one with rules 52, and 57 and one with rules 21 and 29.

3.4.2.2 Rules and ontology term ranking

Once the rules are clustered, we rank both the rules and ontology terms to determine the order in which they should be inspected. Thus, we need to define a score to allow

such ranking, similar to the score used in the data-driven analysis of DBpedia (see Section 3.3.3.3).

$$score_c(c) = \frac{|I_c|}{|I|} \tag{3.1}$$

$$score_t(t) = \frac{|I_t|}{|I|} \tag{3.2}$$

The scores are calculated based on the inconsistencies in which the rules clusters and ontology terms are involved. R is the set of all rules. C is the set of all rules clusters. T is the set of all ontology terms. I is the set of all inconsistencies. I_r is the set of inconsistencies in which rule r is involved and $r \in R$. I_c is the set of inconsistencies in which cluster c is involved, i.e., $I_c = \{i | \exists r : (r \in c \wedge \ i \in I_r)\}$. I_t is the set of inconsistencies in which ontology term t is involved.

The score of a rules cluster c is defined as $score_c(c)$ (see Equation (3.1)). It is calculated as the number of inconsistencies a cluster is involved in over the total number of inconsistencies. The score of an ontology term t is defined as $score_t(t)$ (see Equation (3.2)). It is calculated as the number of inconsistencies an ontology term is involved in over the total number of inconsistencies. Both scores increase if the number of inconsistencies the rules clusters and ontology terms are involved increases. As a result, they will be ranked higher and inspected earlier by experts. If two scores are equal, then determining which cluster or term is ranked higher happens arbitrarily.

Example We calculate the scores for the three clusters and the five ontology terms. For a cluster, we count the unique inconsistencies in which its rules are involved. The first cluster contains the rules 38 and 43, and the second cluster rules 52 and 57. Both clusters are involved in one inconsistency, resulting in a $score_c$ of 0.25 for both, as there are 4 inconsistencies in total. The third cluster contains the rules 21 and 29. Therefore, the number of inconsistencies is 2, resulting in a $score_c$ of 0.50. This means that the third cluster will be ranked before the other two. Thus, experts will first inspect rules 21 and 29, because they are together involved in the most inconsistencies and are related to the same entity. The five ontology terms are `dbo:Place`, `dbo:sourcePosition`, `dbo:mouthPosition`, `geo:lat`, and `geo:SpatialThing` (see lines 4, 5, 8, 10, and 12 in Listing 3.2). `dbo:sourcePosition` and `dbo:mouthPosition` are only involved in one inconsistency. Therefore, their corresponding $score_t$ is 0.25. `geo:lat` is involved in two inconsistencies. Therefore, its corresponding $score_t$ is 0.50. `geo:SpatialThing` and `dbo:Place` are involved in all inconsistencies. Therefore, their corresponding $score_t$ is 1. This means that `dbo:Place` and `geo:SpatialThing` will be ranked first, followed by `geo:lat`. Note that clusters and terms can be ranked in different ways when they have the same score.

3.4.2.3 Rules and ontology definitions refinement

Once the rules are ranked, we select the top rules clusters and ontology terms for inspection and determine which refinements should be applied, if any. The inspection is done manually, because Resglass is designed to help experts in determining the desired refinements. Afterwards the rules and ontology definitions can be validated again to detect remaining or newly introduced inconsistencies, which restarts Resglass.

Example Assuming that we only inspect the top rules cluster and ontology term. We start with the rules, followed by the terms. We inspect rules 21 and 29 and observe that the link between a river and its location of the source and mouth are correctly annotated using `dbo:sourcePosition` and `dbo:mouthPosition`. Thus, we leave the rules unchanged. We inspect `dbo:Place` and its corresponding definitions 4 and 7 in the ontology. They state that place is a class and in the range of `dbo:location`. This is still correct and, thus, we leave the definition unchanged. We inspect also `geo:SpatialThing` and its corresponding definitions 5, 9, and 11, 13 in the ontology. We see that `dbo:SpatialThing` is in the domain of `dbo:sourcePosition` and `dbo:mouthPosition`, but the locations are annotated with the class `dbo:Place`. One way to resolve this issue is to remove the rules that annotes a location with the class `dbo:Place` add a new rule that annotates it with `geo:SpatialThing`.

Once the refinements are applied, we restart Resglass. We notice that a new inconsistency was introduced, because we did remove the class `dbo:Place` for a location, while this is class in the range of `dbo:location`. This can be resolved by applying another iteration of Resglass' steps, resulting in, e.g., the annotation of entity with both classes `dbo:Place` and `geo:SpatialThing` or creation of a ontology definition that states that `geo:SpatialThing` is a subclass of `dbo:Place`.

3.4.3 Knowledge graph generation

The knowledge graph is generated by applying the semantic annotations to existing data sources via rules (see step 3 at the bottom of Figure 3.4). This graph does not contain the inconsistencies resolved in the previous steps, because the refined rules and ontology definitions are used.

Listing 3.4: Knowledge graph generated by applying the refined rules on the data in Figures 3.1 and 3.2

```
dbr:river_thames a dbo:River;
  dbo:sourcePosition _:b0;
  dbo:mouthPosition _:b2.

_:b0 a dbo:Place, geo:SpatialThing;
  geo:lat "51°41'39\"N".

_:b2 a dbo:Place, geo:SpatialThing;
```

```
  geo:lat "51°29'56\"N".

dbr:river_cam a dbo:River;
  dbo:sourcePosition _:b1;
  dbo:mouthPosition _:b3.

_:b1 a dbo:Place, geo:SpatialThing;
  geo:lat "52°20'54\"N".

_:b3 a dbo:Place, geo:SpatialThing;
  geo:lat "52°20'54\"N".

dbr:ghent a dbo:City;
  dbo:location [
    a dbo:Place;
    geo:lat "51°2'60\"N" ].

dbr:brussels a dbo:City;
  dbo:location [
    a dbo:Place;
    geo:lat "50°50'60\"N" ].
```

Example We generate the knowledge graph based on the refined rules and ontology definitions (see Listing 3.4). The triples stating that locations are also spatial things are added, in accordance with the refinements applied to the rules done in the previous step.

3.4.4 Knowledge graph inconsistency detection

The knowledge graph is validated to determine inconsistencies (see step 4 at the bottom of Figure 3.4). This is analogous to our previous work and can be done via a number of methods, such as the comparison of the results of queries and inference rules (see Section 3.3). Inconsistencies that could not be detected using the rules can be detected in this step.

Example Triples 6, 9, 16, 19, 24, and 29 in Listing 3.4 state the latitude of the locations and cause six inconsistencies. The ontology definitions require latitude to be given as a float (see line 14 in Listing 3.2), but this is not the case, leading to a inconsistency every time `geo:lat` is used.

3.4.5 Rules and ontology definitions refinement

Inconsistencies detected in the previous steps might be resolved by refining the rules and ontology definitions (see step 5B at the bottom Figure 3.4). This is different from

the last step of our previous work, where only the rules are refined (see step 5A in Figure 3.4).

Example Rules can be added to transform the latitudes of locations for use with `geo:lat` instead of the original, unchanged values.

3.5 Implementation

In this work, we use the RDF Mapping Language (RML) [7] as the underlying knowledge graph generation language to apply Resglass (see Section 3.5.1), because it is (i) an extension of R2RML [6], the only W3C recommended knowledge graph generation language (see Section 3.3.1); (ii) used in our previous work [1] (see Section 3.3.3.2); and (iii) used during the data-driven analysis of DBpedia [2] (see Section 3.3.3.3). We describe the implementation of the Resglass' steps (see Section 3.5.2). Note that we only discuss the first four steps, because the others are analogue to the method of our previous work, and not the ontology definitions, because these are independent of the rules and, thus, the language. The complete implementation is available at `https://github.com/RMLio/rule-driven-resolution`.

3.5.1 RDF Mapping Language (RML)

RML is a declarative language to define how RDF graphs are generated from existing data sources through a set of rules. RML, as opposed to R2RML [6], does not only support relational databases, but also data in CSV, JSON, and XML format, as well as files, Web APIs, and so on. RML is extensible, i.e., to data sources in other formats. We describe here the details of the language that are relevant for this work. For the full specification, we refer to `http://rml.io/spec.html`. A subset of the corresponding RML rules for our motivating use case is given in Listing 3.1.

For every entity there is a corresponding Triples Map (`rr:TriplesMap`): `<#TriplesMap1>` for the rivers and `<#TriplesMap2>` for the source locations. The Term Maps define how the subjects, predicates, and objects of the triples are generated: Subject Map, Predicate Object Map, Predicate Map, and Object Map. The Subject Map (`rr:SubjectMap`) defines how IRIs are generated for the RDF triples' subjects via a template (`rr:template`). The Predicate Object Maps define how the triples' predicates and objects are generated; each one requires at least one Predicate and Object Map. In our example, for `<#TriplesMap1>` we have three Predicate Object Maps: class, link to the source location, and link to the mouth location. For the Predicate Object Map that defines the class, we use the `rdf:type` as the predicate (`rr:predicate`). Note that the class in this example does not depend on the data. Therefore, it is constant (`rr:constant` in the Object Map). For the Predicate Object Maps that annotate an entity with an attribute, such as the latitude of a location, we use values from the data (`rml:reference` in the Object Map) instead of a constant.

3.5.2 Resglass

In this section, we discuss our implementation of the different steps of Resglass, including accompanying examples. For this we rely on RML as our knowledge graph generation language.

1. Rules inconsistency detection The inconsistencies in RML rules are detected via a rule-based reasoning system [14]. For each constraint type the corresponding inference rules are created. The RML rules, ontologies, and inference rules, which assess the constraints, serve as the reasoning system's input. The output is inconsistencies with references to the involved RML rules, ontology terms and constraint types.

We rely on a rule-based reasoning system [14], instead of an approach where the results of queries are compared (see Section 3.3.2), which we used in our previous work, for: (i) supporting [R2]RML shortcuts via a custom entailment regime, (ii) finding implicit inconsistencies, and (iii) determining the root cause. [R2]RML defines many shortcuts to make it easier for humans to write the rules. This means that different rule sets can generate the same RDF dataset [6]. Thus, every shortcut needs to be defined separately per constraint type in a queries execution approach.

By enabling a custom entailment regime via rule-based reasoning, we can (i) define [R2]RML shortcuts as custom concrete entailment regimes that can be reused for different constraint types; (ii) detect implicit inconsistencies if needed by including another entailment regime [24]; and (iii) precisely determine the root causes of individual inconsistencies using the formal proof, even when including custom entailment regimes, due to the formal logical framework of this reasoning system. When using an approach where the results of queries are compared, we need a different system to reason over custom [R2]RML entailment regimes. The connection with the original rules set is lost and the original root cause cannot be found. However, accurately and correctly identifying root causes across different rule sets important is for Resglass.

Example Consider the axiom on line 9 in the ontology (Listing 3.2). The corresponding instantiated constraint is: for every combination of Term Maps (Predicate Object Map, Predicate Map, and Object Map) that annotates an entity with `dbo:sourcePosition`, the object should refer to an entity with the class `geo:SpatialThing`. Next, we determine all groups of Term Maps that could lead to the failure of this constraint. In our example we have one group consisting of `<#PM1>` and `<#OM2>`. The constraint fails as the location entity is annotated with the class `dbo:Place`.

2. RML rules clustering The RML rules are clustered by determining the Triples Map to which the rules, i.e., Term Maps, correspond. This occurs because every entity is represented by a Triples Map in the rules and every Term Map is related to at least one Triples Map. If the rule is a Predicate Object Map, we determine

the corresponding Triples Map via the `rr:predicateObjectMap` that defines the relationship between the former and latter. If the rule is a Subject Map, we determine the corresponding Triples Map via the `rr:subjectMap`. If the rule is a Predicate or Object Map, we determine the Triples Map by first determining the corresponding Predicate Object Map via the `rr:predicateMap` and `rr:objectMap`, respectively. The corresponding Triples Map of the identified Predicate Object Map is determined as described earlier. If the rule is a Triples Map, the corresponding Triples Map is itself.

Example If we determine the clusters of the Term Maps `<#PM1>`, `<#PM2>`, `<#PM3>`, `<#PM4>`, `<#OM2>`, and `<#OM3>`, then we have three clusters. The first corresponds with the Triples Map `<#TriplesMap1>`, and contains `<#PM1>` and `<#PM4>`. The second corresponds with the `<#TriplesMap2>`, and contains `<#OM2>` and `<#PM2>`. The third corresponds with the `<#TriplesMap3>`, and contains `<#OM3>` and `<#PM3>`.

3. RML rules ranking We calculate the score of every rules cluster and ontology term to be able to rank them. We iterate over each cluster, identified by the Triples Map that represents an entity. We iterate over every Terms Map that is in the cluster and count the inconsistencies in which it is involved. Note that for a single cluster we count an inconsistency only once, even if two Term Maps are involved in the same inconsistency. The score equals this count over the total number of inconsistencies (see Equation (3.1)). The ontology terms ranking does not depend on the used language. Thus, it is done as described in Section 3.4.2.2.

Example The cluster of `<#TriplesMap1>` is involved in two inconsistencies, and `<#TriplesMap2>` and `<#TriplesMap3>` both in one. Thus, the scores of these cluster are 0.50, 0.25, and 0.25, respectively, analogues to our example in Section 3.4.2.2.

4. RML rules refinement Once the ranking is done, experts inspect the top rules clusters, identified by the Triples Maps, and apply the necessary refinements to the RML rules. Note that the refinement of the ontology definitions does not depend on the used language. Thus, it is done as described in Section 3.4.2.3.

Example Let's assume that we only inspect the cluster of `<#TriplesMap2>`. We inspect the Predicate Maps `<#OM2>` and `<#PM2>`. One possibility is to replace `<#OM2>` with a new Object Map that annotates a location with the class `geo:SpatialThing`. Once the refinements are applied, we restart Resglass to determine whether all inconsistencies are resolved or if new inconsistencies were introduced.

5. Knowledge graph generation The knowledge graph is generated by applying the semantic annotations to the existing data sources via the RML rules. This graph does not contain the inconsistencies resolved in the previous steps, because the refined RML rules and ontology definitions are used (see Listing 3.4).

6. Knowledge graph inconsistency detection The knowledge graph, which is an RDF graph, is validated to determine inconsistencies. Inconsistencies that could not be detected using rules can be detected in this step. More, this step is independent of the used language, i.e., RML, because only the knowledge graph is used and not the rules.

Example Triples 6, 9, 16, 19, 24, and 29 in Listing 3.4 state the latitudes and cause 6 inconsistencies. The ontology definitions require the latitude to be float (see line 14 in Listing 3.2), but this is not the case, leading to an inconsistency.

7. RML rules refinement Inconsistencies detected in the previous steps might be resolved by refining the RML rules and ontology definitions.

Example RML rules can be added to transform the latitude of the locations instead of the original, unchanged values. This is done by using a function that returns the float version of a latitude. We use the Function ontology [29] to declarative describe this transformation in the RML rules [30]. For the latitude in `<#TriplesMap2>`, we replace rule 41 with `rr:objectMap <#parseLatitude>`, and we add the following rules:

```
<#parseLatitude>
  fnml:functionValue [
    rr:predicateObjectMap [
      rr:predicate fno:executes;
      rr:objectMap [
        rr:constant ex:parseLatitude]];

    rr:predicateObjectMap [
      rr:predicate ex:inputString;
      rr:objectMap [
        rr:reference "latitude"]]].
```

The data of "latitude" is used as input for the function `ex:parseLatitude`[3], which returns the float version of a string. For a detailed description about functions, we refer to `https://w3id.org/function/`.

[3] `ex` is the prefix for the namespace `http://example.com/`

3.6 Evaluation

We conducted a comparison to validate our hypothesis (see Section 3.1). We compare the ranking of the rules and ontology terms provided by experts to the automated ranking provided by Resglass.

In Section 3.6.1, we elaborate on the evaluation method, namely the procedure and participating experts. In Section 3.6.2, we discuss the results.

3.6.1 Method

In this section, we discuss the procedure followed during our evaluation, together with the experts that participated.

Procedure Experts on creation of ontologies and knowledge graph generation languages were directly contacted by the authors. Those who agreed, partook in the following experiment: we selected 25 Wikipedia infobox templates that are annotated with RML rules to generate triples that are part of the DBpedia knowledge graph. The templates selection was random but we opted for the ones whose rules and ontology terms were involved in at least one inconsistency.

Two lists were presented to the experts: (i) a list with URLs of the Triples Maps that correspond with the templates, and (ii) a list with URLs of the ontology terms that are involved in at least one inconsistency. We provided a file for each template containing the inconsistencies in which the specific template is involved. The file includes the type of inconsistency and rules involved, together with the number of rules that might be affected when the involved rule is updated.

The experts had two tasks: to rank the Triples Maps and ontology terms in the order that they would inspect them. The ranking was done by assigning a score to every item in the two lists. The score is a number between 0 and 1 and multiple items can have the same scores. When two items have the same score, both items are equally important to be inspected. The full, detailed instructions and files given to the experts can be found at `https://doi.org/10.6084/m9.figshare.7410479.v2`.

The random rankings were automatically generated by assigning an integer, representing the rank, to each Triples Map and ontology term. The integers were randomly generated within the range of 1 and the total number of Triples Maps or ontology terms. We generated 100 rankings for the Triples Maps and 100 for the ontology terms. The corresponding code can be found at `https://github.com/RMLio/rule-driven-resolution/tree/cluster/resglass-random`.

Experts Three experts partook in the experiment, their age range was 28 to 32. All were highly educated: two had a PhD and one a master's degree. They were experts in knowledge graph generation rules and had experience in the use and definition of ontologies.

Table 3.3: Comparison of the experts', Resglass, and random rankings showing that Resglass has 80% overlap with the experts' and that it improves random rankings by a least 20%.

Expert	**RBO (%)**			
	Rules		**Ontology terms**	
	Resglass	random	Resglass	random
1	94	34	78	56
2	59	45	81	57
3	90	37	78	56
Average	81	39	79	56

3.6.2 Results

We compare the two rankings of each expert with (i) our automatic rankings produced with Resglass and (ii) random rankings. The comparison is performed using the Ranked-Bias Overlap (RBO) [31]. It is a measure to compare two lists and returns a value between 0 and 1. The measure allows ties of items and puts more weight on the top-ranked items in a list, which are not simultaneously supported by measures like Kendall's tau [32] and Spearman's rho. Ties are possible in case two or more rules, i.e., Triples Maps, or ontology terms are equally important to inspect. More weight is put on the top-ranked items. This aligns with cases where experts fix a subset of the inconsistencies, and then analyze the refined rules and ontology definitions again to find the remaining, and possibly, new inconsistencies.

The average overlap is 81% for the rules and 79% for the ontology terms. The former means that there is a large overlap between the experts' manual ranking and our automatic ranking when more attention is put to the top ranked rules then the lower ranked rules. The latter means that again there is a large overlap between the experts' ranking and our automatic ranking, but now regarding ontology terms.

Furthermore, our automatic ranking improves the overlap with experts' manual ranking with 41% for rules and 23% for ontology terms, compared to a random ranking. The RBO values for each expert's rules and ontology term rankings can be found in Table 3.3.

3.7 Conclusion

Knowledge graphs are often generated by applying generation rules that annotate data in existing data sources with ontology terms. However, the knowledge graphs might suffer from inconsistencies, which can be introduced by the combination of rules and ontologies. These inconsistencies can be resolved by either refining the rules or ontologies. In this article, we introduce a rule-driven method called Resglass that ranks the rules, via clustering, and ontology terms involved in an inconsistency. The top

rules and ontology terms are inspected by experts and the necessary refinements are applied to resolve the inconsistencies.

Refinements can be applied to both rules and ontology terms to resolve the inconsistencies. Nevertheless, experts need to carefully determine whether the former or latter needs to be refined. For example, is it desired to refine an ontology that models the data of a specific use case? Is it desired to keep to the ontology as it is and refine the rules? Or should another ontology be used instead of the current one? Resglass, including the rankings, provides valuable insights to answer these questions, such as the entities that need to most attention when applying refinements, and the specific ontology terms and definitions that are involved in a lot of inconsistencies and, thus, might be problematic.

Our evaluation with experts shows that our ranking has 80% overlap with the experts' rankings for both the rules and the ontology elements. Furthermore, our ranking improves the overlap with experts' with 41% for rules and 23% for ontology terms, compared to a random ranking. Thus, this evaluation provides evidence towards the acceptance of Hypothesis 3 "The automatic inconsistency-driven ranking of Resglass improves, compared to a random ranking, by at least 20% the overlap with experts' manual ranking."

Note that in Step 3 of Resglass the complete knowledge graph is generated, but this does not mean that Resglass cannot be applied during use cases where virtual knowledge graphs are generated instead. These graphs are the result of translating SPARQL queries into another query supported by the underlying data source and evaluated on the original dataset [33]. Instead of generating the complete knowledge graph in Step 3, only the graph that is the result of a SPARQL query is generated. Inconsistencies can still be detected in Step 4, with exception of inconsistencies that can only be detected when the complete graph is available, because in most cases the result of the query is only a subset of the complete graph.

The ranking proposed of Resglass can be used not only by experts to explore manual refinements, but can also be used to drive (semi-)automatic solution. For example, this can be done by building on our previous work were we proposed an automatic solution that resolve the inconsistencies by applying a predefined set of refinements to the rules. The ranking, which is use case-specific, can be used to make a more informed decision regarding which refinements should be applied, instead of solely relying on a predefined set, which is created independent of the use case.

Acknowledgements

The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (AIO), the Research Foundation – Flanders (FWO), and the European Union. Special thanks to Tom Seymoens and Io Taxidou for their valuable advice.

References

[1] Anastasia Dimou, Dimitris Kontokostas, Markus Freudenberg, Ruben Verborgh, Jens Lehmann, Erik Mannens, Sebastian Hellmann, and Rik Van de Walle. "Assessing and Refining Mappings to RDF to Improve Dataset Quality". In: *Proceedings of the 14th International Semantic Web Conference*. Vol. 9367. Lecture Notes in Computer Science. 2015, pp. 133–149. DOI: 10.1007/978-3-319-25010-6_8. URL: http://dx.doi.org/10.1007/978-3-319-25010-6_8.

[2] Heiko Paulheim. "Data-Driven Joint Debugging of the DBpedia Mappings and Ontology". In: *The Semantic Web*. Springer, 2017, pp. 404–418. DOI: 10.1007/978-3-319-58068-5_25. URL: https://doi.org/10.1007/978-3-319-58068-5_25.

[3] Heiko Paulheim. "Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods". In: *Semantic Web* 8.3 (2017), pp. 489–508. DOI: 10.3233/SW-160218. URL: https://doi.org/10.3233/SW-160218.

[4] Christian Bizer, Tom Heath, and Tim Berners-Lee. "Linked Data: The Story So Far". In: *Semantic Services, Interoperability and Web Applications: Emerging Concepts* (2009), pp. 205–227. DOI: 10.4018/978-1-60960-593-3.ch008. URL: https://doi.org/10.4018/978-1-60960-593-3.ch008.

[5] World Wide Web Consortium. *RDF 1.1 Concepts and Abstract Syntax*. Tech. rep. 2014. URL: https://www.w3.org/TR/rdf11-concepts/.

[6] Souripriya Das, Seema Sundara, and Richard Cyganiak. *R2RML: RDB to RDF Mapping Language*. Working Group Recommendation. W3C, Sept. 2012. URL: http://www.w3.org/TR/r2rml/.

[7] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data". In: *Workshop on Linked Data on the Web*. 2014. URL: http://events.linkeddata.org/ldow2014/papers/ldow2014_paper_01.pdf.

[8] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. "A SPARQL Extension for Generating RDF from Heterogeneous Formats". In: *Proceedings of the 14th Extended Semantic Web Conference*. Springer. 2017, pp. 35–50. DOI: 10.1007/978-3-319-58068-5_3. URL: https://doi.org/10.1007/978-3-319-58068-5_3.

[9] Nicola Guarino, Stati Uniti, and Pierdaniele Giaretta. "Ontologies and knowledge bases: towards a terminological clarification". In: *Towards Very Large Knowledge Bases*. IOS Press, 1995.

[10] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. *OWL 2 Web Ontology Language*. Tech. rep. W3C, 2012. URL: https://www.w3.org/TR/owl2-syntax/.

[11] Dean Allemang and James Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. 2nd ed. Morgan Kaufmann Publishers Inc., 2011. URL: https://www.elsevier.com/books/semantic-web-for-the-working-ontologist/allemang/978-0-12-385965-5.

[12] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. "Test-driven Evaluation of Linked Data Quality". In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW '14. Seoul, Korea: ACM, 2014, pp. 747–758. DOI: 10.1145/2566486.2568002. URL: http://doi.acm.org/10.1145/2566486.2568002.

[13] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. "DBpedia–a Large-scale, Multilingual Knowledge Base Extracted from Wikipedia". In: *Semantic Web* 6.2 (2015), pp. 167–195. DOI: 10.3233/SW-140134. URL: https:/doi.org/10.3233/SW-140134.

[14] Dörthe Arndt, Ben De Meester, Anastasia Dimou, Ruben Verborgh, and Erik Mannens. "Using Rule-Based Reasoning for RDF Validation". In: *Proceedings of the International Joint Conference on Rules and Reasoning*. Vol. 10364. Lecture Notes in Computer Science. 2017, pp. 22–36. DOI: 10.1007/978-3-319-61252-2_3. URL: https://doi.org/10.1007/978-3-319-61252-2_3.

[15] Anastasia Dimou, Dimitris Kontokostas, Markus Freudenberg, Ruben Verborgh, Jens Lehmann, Erik Mannens, and Sebastian Hellmann. "DBpedia Mappings Quality Assessment". In: *Proceedings of the 15th International Semantic Web Conference: Posters and Demos*. Vol. 1690. CEUR Workshop Proceedings. Oct. 2016. URL: http://ceur-ws.org/Vol-1690/paper97.pdf.

[16] Michael Färber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. "Linked Data Quality of DBpedia, Freebase, OpenCyc, Wikidata, and YAGO". In: *Semantic Web* (2016), pp. 1–53. DOI: 10.3233/SW-170275. URL: https://doi.org/10.3233/SW-170275.

[17] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, and Juan Sequeda. *A Direct Mapping of Relational Data to RDF*. Working Group Recommendation. W3C, 2012. URL: http://www.w3.org/TR/rdb-direct-mapping/.

[18] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF*. Tech. rep. URL: http://www.w3.org/TR/rdf-sparql-query/.

[19] Anastasia Dimou, Sahar Vahdati, Angelo Di Iorio, Christoph Lange, Ruben Verborgh, and Erik Mannens. "Challenges as enablers for high quality Linked Data: insights from the Semantic Publishing Challenge". In: *PeerJ Computer Science* 3 (Jan. 2017), e105. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.105. URL: https://doi.org/10.7717/peerj-cs.105.

[20] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer, and Jens Lehmann. "Crowdsourcing Linked Data Quality Assessment". In: *Proceedings of the 12th International Semantic Web Conference*. Berlin, Heidelberg, 2013, pp. 260–276. DOI: 10.1007/978-3-642-41338-4_17. URL: https://doi.org/10.1007/978-3-642-41338-4_17.

[21] Mina Farid, Alexandra Roatis, Ihab F. Ilyas, Hella-Franziska Hoffmann, and Xu Chu. "CLAMS: Bringing Quality to Data Lakes". In: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD '16. San Francisco, California, USA: ACM, 2016, pp. 2089–2092. DOI: 10.1145/2882903.2899391. URL: http://doi.acm.org/10.1145/2882903.2899391.

[22] Mohammad Rashid, Giuseppe Rizzo, Marco Torchiano, Nandana Mihindukulasooriya, Oscar Corchó, and Raúl GarcíaCastro. "Completeness and consistency analysis for evolving knowledge bases". In: *Journal of Web Semantics* (2018). DOI: 10.1016/j.websem.2018.11.004. URL: https://doi.org/10.1016/j.websem.2018.11.004.

[23] Pablo N. Mendes, Hannes Mühleisen, and Christian Bizer. "Sieve: Linked Data Quality Assessment and Fusion". In: *Proceedings of the 2012 Joint EDBT/ICDT Workshops*. EDBT-ICDT '12. Berlin, Germany: ACM, 2012, pp. 116–123. DOI: 10.1145/2320765.2320803. URL: http://doi.acm.org/10.1145/2320765.2320803.

[24] Thomas Bosch, Andreas Nolle, Erman Acar, and Kai Eckert. "RDF Validation Requirements – Evaluation and Logical Underpinning". In: *arXiv preprint arXiv:1501.03933* (2015). URL: http://arxiv.org/abs/1501.03933.

[25] Holger Knublauch and Dimitris Kontokostas. *Shapes Constraint Language (SHACL)*. W3C Recommendation. W3C, 2017. URL: https://www.w3.org/TR/shacl/.

[26] Marti A Hearst. "Clustering versus faceted categories for information exploration". In: *Communications of the ACM* 49.4 (2006), pp. 59–61. DOI: 10.1145/1121949.1121983. URL: https://doi.org/10.1145/1121949.1121983.

[27] Hua-Jun Zeng, Qi-Cai He, Zheng Chen, Wei-Ying Ma, and Jinwen Ma. "Learning to cluster web search results". In: *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2004, pp. 210–217. DOI: 10.1145/1008992.1009030. URL: https://doi.org/10.1145/1008992.1009030.

[28] Mika Käki and Anne Aula. "Findex: improving search result use through automatic filtering categories". In: *Interacting with Computers* 17.2 (2005), pp. 187–206. DOI: 10.1016/j.intcom.2005.01.001. URL: https://doi.org/10.1016/j.intcom.2005.01.001.

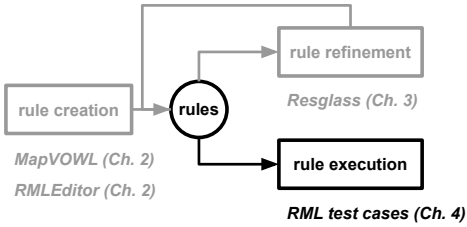
[29] Ben De Meester, Anastasia Dimou, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "An Ontology to Semantically Declare and Describe Functions". In: *The Semantic Web; ESWC 2016 Satellite Events*. Vol. 9989. Lecture Notes in Computer Science. Springer, Oct. 2016, pp. 46–49. DOI: 10.1007/978-3-319-47602-5_10. URL: https://doi.org/10.1007/978-3-319-47602-5_10.

[30] Ben De Meester, Wouter Maroy, Anastasia Dimou, Ruben Verborgh, and Erik Mannens. "Declarative Data Transformations for Linked Data Generation: The Case of DBpedia". In: *Proceedings of the 14th Extended Semantic Web Conference*. Vol. 10250. Lecture Notes in Computer Science. Springer, May 2017, pp. 33–48. DOI: 10.1007/978-3-319-58451-5_3. URL: https://doi.org/10.1007/978-3-319-58451-5_3.

[31] William Webber, Alistair Moffat, and Justin Zobel. "A similarity measure for indefinite rankings". In: *ACM Transactions on Information Systems* 28.4 (2010), p. 20. DOI: 10.1145/1852102.1852106. URL: https://doi.org/10.1145/1852102.1852106.

[32] Maurice G Kendall. "Rank Correlation Methods". In: (1955). DOI: 10.1111/j.2044-8317.1956.tb00172.x. URL: https://doi.org/10.1111/j.2044-8317.1956.tb00172.x.

[33] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. "Linking data to ontologies". In: *Data Semantics X* (2008), pp. 133–173. DOI: 10.1007/978-3-540-77688-8_5. URL: https://doi.org/10.1007/978-3-540-77688-8_5.



Chapter 4

RML conformance test cases for rule execution

The first knowledge graph generation rule language (R2RML [1]) only supports relational databases. Other extensions and adaptations were applied to account for other types of data sources (e.g., RML [2], xR2RML [3], and XSPARQL [4]), together with processors that execute them (e.g., the RMLMapper[1], CARML[2], GeoTriples [5], and Ontario[3]). However, unlike for R2RML, there are no test cases available to determine the conformance to the specifications of these extensions and adaptations. As a result, processors are either not tested or only tested with custom test cases, which do not necessarily assess every aspect of the specification. Consequently, no test cases are available that allows comparing the different processors that generate knowledge graphs from heterogeneous data sources based on the conformance to their specifications. This way, it is hard for users to determine the most suitable tool for a certain use case.

This chapter contributes to rule execution, where Chapters 2 and 3 contribute to rule creation, and describes an initial set of conformance test cases for RML, based on the R2RML test cases. We use RML, because it supports heterogeneous data sources and is based on R2RML for which already test cases are available. This is done towards designing test cases that are independent of RML and are applicable to all languages that generate knowledge graphs from heterogeneous data sources, which addresses Research Question 4 "What are the characteristics of test cases for processors that generate knowledge graphs from heterogeneous data sources independent of languages' specifications?"

[1] `https://github.com/RMLio/rmlmapper-java`
[2] `https://github.com/carml/carml`
[3] `https://github.com/WDAqua/Ontario`

*

Pieter Heyvaert, David Chaves-Fraga, Freddy Piyatna, Oscar Corcho, Erik Mannens, Ruben Verborgh, and Anastasia Dimou

Published as "Conformance Test Cases for the RDF Mapping Language (RML)" in *Knowledge Graphs and Semantic Web (KGSWC 2019)*. Springer, 2019. pp. 162 – 173.

Abstract

Knowledge graphs are often generated using rules that apply semantic annotations to data sources. Software tools then execute these rules and generate or virtualize the corresponding RDF-based knowledge graph. RML is an extension of the W3C-recommended R2RML language, extending support from relational databases to other data sources, such as data in CSV, XML, and JSON format. As part of the R2RML standardization process, a set of test cases was created to assess tool conformance the specification. In this work, we generated an initial set of reusable test cases to assess RML conformance. These test cases are based on R2RML test cases and can be used by any tool, regardless of the programming language. We tested the conformance of two RML processors: the RMLMapper and CARML. The results show that the RMLMapper passes all CSV, XML, and JSON test cases, and most test cases for relational databases. CARML passes most CSV, XML, and JSON test cases. Developers can determine the degree of conformance of their tools, and users can determine based on conformance results the most suitable tool for their use cases.

4.1 Introduction

Knowledge graphs are often generated based on rules that apply semantic annotations to raw or semi-structured data. For example, the DBpedia knowledge graph is generated by applying classes and predicates of the DBpedia ontology to Wikipedia [6]. Software tools execute these rules and generate corresponding RDF triples and quads [7], which materialize knowledge graphs. In the past, custom scripts prevailed, but lately, rule-driven tools emerged. Such tools distinguish the *rules* that define how RDF terms and triples are generated from the *tool* that executes those rules. R2RML [1] is the W3C-recommended language to define such rules for generating knowledge graphs from data in relational databases (RDBs). An R2RML processor is a tool that, given a set of R2RML rules and a relational database, generates an RDF dataset. Examples of R2RML processors include Ultrawrap [8], Morph-RDB [9], Ontop [10], and XSPARQL [11]. A subset of them was included in the RDB2RDF Implementation Report [12] which lists their *conformance* to the R2RML specification. Conformance is

assessed based on whether the correct knowledge graph is generated for a set of rules and certain relational database.

Given that R2RML is focused on relational databases only, extensions and adaptations were applied to account for other types of data sources. These include RML [2], XSPARQL [11], and xR2RML [3]. RML provides an extension of R2RML to support heterogeneous data sources, including different *formats* such as CSV, XML, JSON, and *access interfaces*, such as files and Web APIs. Various RML processors emerged, such as the RMLMapper[4], CARML[5], GeoTriples [5], and Ontario[6]. Unlike R2RML, there are no test cases available to determine the conformance to the RML specification. As a result, processors are either not tested or only tested with custom test cases, which do not necessarily assess every aspect of the specification. Consequently, no implementation report is available that allows comparing the different processors that generate knowledge graphs from heterogeneous data sources based on the conformance to the specification. This way, it is hard to determine the most suitable processor for a certain use case.

In this work, we introduce an initial set of RML test cases, which contains 297 test cases based on the 62 existing R2RML test cases. Instead of only considering relational databases as data sources, we also consider data in CSV, XML, and JSON format. Furthermore, we tested the conformance of the RMLMapper and CARML: every test case was executed by both processors and we noted whether the generated knowledge graph matches the expected one. The corresponding implementation report is available at `http://rml.io/implementation-report`. This helps determining which processor is the most suitable for a certain use case. For example, do users want a processor that supports the complete specification, or do they prefer a processor that does not support certain aspects of the specification, but executes the rules faster?

The test cases results shows that the RMLMapper (v4.3.2) passes all test cases regarding CSV, XML, and JSON format, and most test cases for RDBs, but fails the test cases for automatic datatyping of literals. CARML (v0.2.3) passes most test cases regarding CSV, XML, and JSON format, except of the test cases that deal, for example, with multiple RDF terms generation. Users can now determine how conformant the different processors are to the RML specification and use this conformance to determine the most suitable processor for their use cases.

The remainder of the paper is structured as follows. In Section 4.2, we discuss related work. In Section 4.3, we discuss the test cases. In Section 4.4, we elaborate on the test cases execution and results. In Section 4.5, we conclude the paper.

[4] `https://github.com/RMLio/rmlmapper-java`
[5] `https://github.com/carml/carml`
[6] `https://github.com/WDAqua/Ontario`

4.2 Related work

In this section, we describe the related work that is relevant to the paper. First, we explain the most important knowledge graph generation language specifications, including R2RML and RML, and processors that execute those rules. Second, we discuss the differences between R2RML and RML. Finally, we describe the R2RML test cases, how they are defined and implemented and their corresponding implementation report with results of a few processors.

4.2.1 Knowledge graph generation languages and tools

R2RML [1] is the W3C recommended language for describing rules to generate RDF from data in RDBs. Currently, many tools support this specification. These tools follow either an Extract-Transform-Load (ETL) process, where a knowledge graph is materialized, e.g., DB2Triples[7] and R2RMLParser[8], or they provide virtual RDF views, focusing more on formalizing the translation from SPARQL to SQL and optimizing the resulted SQL query, e.g., Morph-RDB[9] and Ontop[10].

We describe in more detail pioneering tools for executing R2RML rules: DB2Triples is a tool for extracting data from relational databases, semantically annotating the data extracts according to R2RML rules and generating knowledge graphs. The R2RMLParser [13] deals in principle with incremental knowledge graph generation. Each time a knowledge graph is generated, not all data is used, but only the one that changed (so-called incremental transformation). Morph-RDB [9] and Ontop [10] adapt the algorithm defined by Chebotko, Lu, and Fotouhi [14] on SPARQL-to-SQL translation, using the information provided by the R2RML rules. Both apply several semantic optimizations (e.g., self join elimination) that generate efficient SQL queries to speed up the evaluation time.

RML [2] is defined as an extension of R2RML to specify rules for generating knowledge graphs from data in different formats, such as CSV, JSON, XML, and different access interfaces, e.g., open data connectivity and Web APIs [15]. Different other languages build upon RML for generating knowledge graphs from heterogeneous data sources, e.g., xR2RML [3] or RMLC [16].

A set of processors that support the RML specification are proposed. The RMLMapper is a Java library and command line interface that executes RML rules to generate RDF. Following the same approach, CARML executes RML rules, but also includes its own extensions, such as MultiTermMap (to deal with arrays) and XML namespace (to improve XPath expressions). GeoTriples [5] is a processor that generates and executes RML rules for generating RDF from geospatial data from different sources. The processor supports data stored in raw files (shapefiles, CSV, KML, GML,

[7] `https://github.com/antidot/db2triples`
[8] `https://github.com/nkons/r2rml-parser`
[9] `https://github.com/oeg-upm/morph-rdb`
[10] `https://github.com/ontop/ontop`

Table 4.1: Summary of the main differences between R2RML and RML

	R2RML	**RML**
input reference	Logical Table	Logical Source
data source language	SQL (implicit)	Reference Formulation (explicit)
value reference	column	logical reference (valid expression following Reference Formulation)
iteration	per row (implicit)	per record (explicit – valid expression following Reference Formulation)

and so on), but also geospatial RDBs such as PostGIS[11] and MonetDB[12]. The generated RDF is based on well-known geospatial vocabularies, such as GeoSPARQL [17] and stSPARQL [18]. Ontario [19] is a federated query processor that uses RML rules to transform heterogeneous data sources during the query processing. Basically, the processor performs the generation using RML during the query processing step and executes federated SPARQL queries over the resulted RDF graphs. These processors are evaluated using ad-hoc examples or feasibility approaches, but a thorough representation of their capabilities is not provided. For that reason, we notice that RML test cases are needed to assess the capabilities of the different processors.

4.2.2 R2RML and RML differences

RML is an extension of R2RML and, thus, follows the core concepts of R2RML's specification, such as Triples Maps, Term Maps, Subject Maps, and so on. However, there is a difference on the reference to the data to support heterogeneous data sources with respect to their format, e.g. CSV, XML, JSON, and access interface, e,g. files or Web APIs (see Table 4.1).

Logical Source A Logical Source extends R2RML's Logical Table and describes the input data source used to generate the RDF. The Logical Table is only able to describe relational databases, whereas the Logical Source defines different heterogeneous data sources, including relational databases.

[11] PostGIS, `https://postgis.net/`
[12] MonetDB, `https://www.monetdb.org/`

Reference Formulation As RML is designed to support heterogeneous data sources, data sources in different formats needs to be supported. One refers to data in a specific format according to the grammar of a certain formulation, which might be path and query languages or custom grammars. For example, one can refer to data in an XML file via XPath and in a relational database via SQL. To this end, the *Reference Formulation* was introduced indicating the formulation used to refer to data in a certain data source.

Iterator In R2RML it is specified that processors iterate over each row to generate RDF. However, as RML is designed to support heterogeneous data sources, the iteration pattern cannot always be implicitly assumed. For example, iterating over a specific set of objects is done by selecting them via a JSONPath expression. To this end, the *Iterator* was introduced which determines the iteration pattern over the data source and specifies the extract of data used to generate RDF during each iteration. The iterator is not required to be specified if there is no need to iterate over the input data.

Logical Reference When referring to values in a table or view of a relational database, R2RML relies on column names. However, as RML is designed to support heterogeneous data sources, rules may also refer to elements and objects, such as in the case of XML and JSON. Consequently, references to values should be valid with respect to the used reference formulation. For example, a reference to an attribute of a JSON object should be a valid JSONPath expression. To this end, (i) the `rml:reference` is introduced to replace `rr:column`, (ii) when a template is used, via `rr:template`, the values between the curly brackets should have an expression that is valid with respect to the used reference formulation, and (iii) `rr:parent` and `rr:child` of a Join Condition should also have an expression that is valid with respect to the used reference formulation.

4.2.3 W3C recommendations and their test cases

In the context of Semantic Web, several specifications were recommended by W3C, such as SPARQL [20], RDF [7], SHACL [21], Direct Mapping of relational data to RDF (DM) [22], and R2RML [1]. Each of these specifications has several related tools that support them. A set of test cases was defined for each one of them (SPARQL test cases[13], RDF 1.1 test cases[14], SHACL test cases[15], and R2RML and Direct Mapping test cases[16], respectively) that provides useful information to choose the tool that fits

[13] `https://www.w3.org/2001/sw/DataAccess/tests/r2`
[14] `http://www.w3.org/TR/rdf11-testcases/`
[15] `http://w3c.github.io/data-shapes/data-shapes-test-suite/`
[16] `https://www.w3.org/TR/2012/NOTE-rdb2rdf-test-cases-20120814/`

better to certain needs. It is also a relevant step in the standardisation process of an technology or specification. We describe the R2RML test cases in more detail.

Determining the conformance of tools executing R2RML rules in the process of RDF generation is a step to provide objective information about the features of each tool. For this reason, the R2RML test cases [23] were proposed. It provides a set of 63 test cases. Each test case is identified by a set of features, such as the SQL statements to load the database, title, purpose, specification reference, review status, expected result, and corresponding R2RML rules. All the test cases are semantically described using the RDB2RDF-test[17] and Test Metadata Vocabulary[18]. Several R2RML processors were assessed for their conformance with the R2RML specification by running the test-cases. The results are available in the R2RML implementation-report [12]. The results are also annotated semantically using the Evaluation and Report Language (EARL) 1.0 Schema[19].

4.3 RML test cases

In this section, we propose test cases to determine the conformance of RML processors to the RML specification. The proposed test cases are based on the R2RML test cases, but they take into account different heterogeneous data sources and the corresponding differences in RML (see Section 4.2.2). Our preliminary set of test cases includes (i) adjusted R2RML test cases for relational databases (including MySQL[20], PostgreSQL[21], and SQL Server[22]) and (ii) new test cases for files in the CSV, XML (with XPath as the reference formulation), and JSON format (with JSONPath as the reference formulation). The test cases are described at `http://rml.io/test-cases/` and the corresponding files are available at `https://github.com/rmlio/rml-test-cases`. In Section 4.3.1, we describe the data model that is used to represent the test cases. In Section 4.3.2, we elaborate on the different files making up a test case. In Section 4.3.3, we discuss the differences between the R2RML and RML test cases.

4.3.1 Data model

We describe the test cases semantically to increase their reusability and sharability. To this end, we created a semantic data model[23], with as main entity the test case (see Figure 4.1). For each test case, the following details are described: unique identifier, title, description, relevant aspect of the RML specification, data sources (optional), expected knowledge graph or error, and RML rules.

[17] `http://purl.org/NET/rdb2rdf-test#`
[18] `https://www.w3.org/TR/2005/NOTE-test-metadata-20050914/`
[19] `https://www.w3.org/TR/EARL10/`
[20] `https://www.mysql.com/`
[21] `https://www.postgresql.org/`
[22] `https://www.microsoft.com/en-us/sql-server/`
[23] `http://rml.io/test-cases/#datamodel`

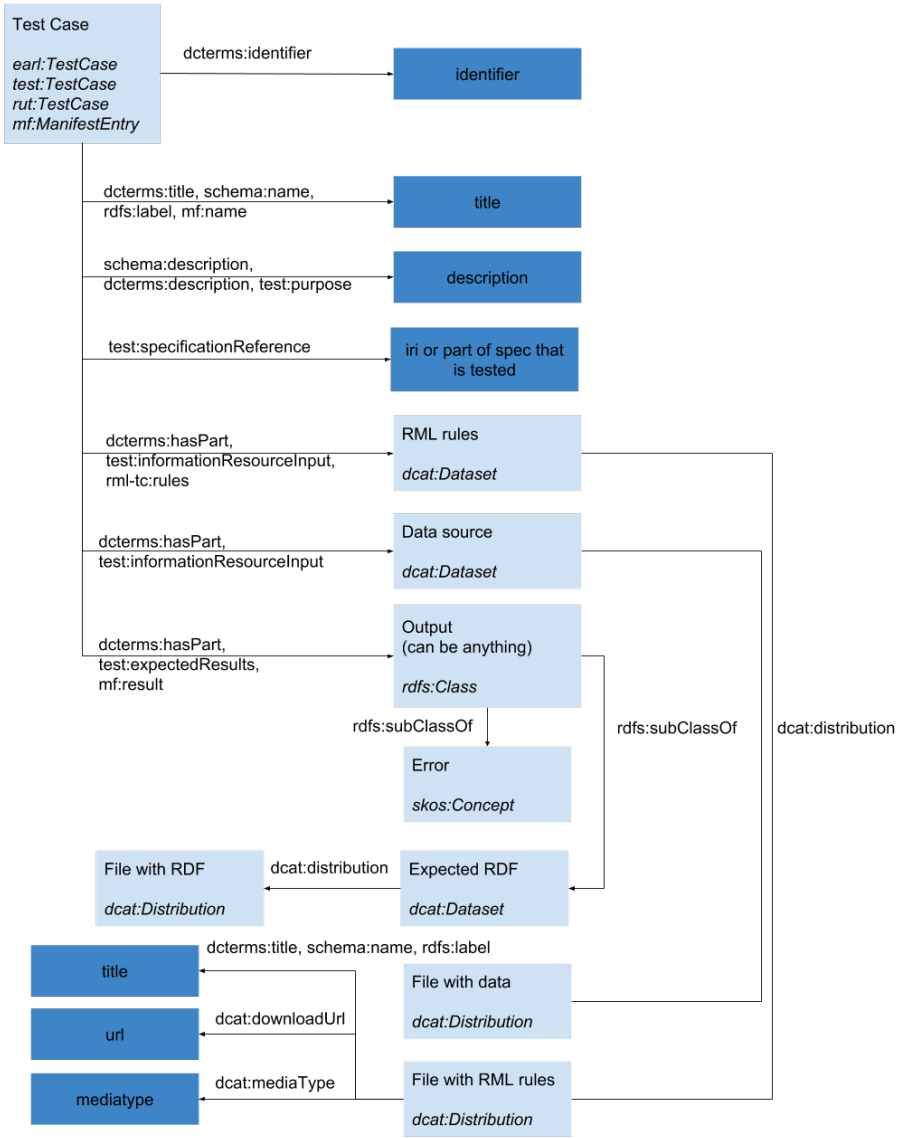


Figure 4.1: Data model of the RML test cases

To provide the corresponding semantic descriptions, the model uses mostly

the Evaluation and Report Language (EARL) 1.0 Schema[24], the Test case manifest vocabulary[25], the Test Metadata vocabulary[26], and the Data Catalog Vocabulary[27]. A test case is annotated with the classes `earl:TestCase`, `test:TestCase`, and `mf:ManifestEntry`. The identifier, title, description, and the specific aspect of the RML specification that is being tested are added as datatype properties. The files that are provided as input to the tools are linked to the test cases via `test:informationResourceInput` and `dcterms:hasPart`. The file with the RML rules is also linked via `rml-tc:rules`[28]. The objects of these properties are of the class `dcat:Dataset`, which in turn link to a `dcat:Distribution` that includes a link to a file. The expected output, whether that is a knowledge graph or an error, is linked via `test:expectedResults`, `mf:result`, and `dcterms:hasPart`. In the case of a knowledge graph, the object of these properties is a `dcat:Dataset`, linked to a `dcat:Distribution`, to describe the file containing the graph. In the case of an error, we link to the expected error.

4.3.2 Test case files

Each test case consists of a set of files that contain the input data sources, the RML rules, and the expected RDF output. In practice, the files are organized as follows: all files for a single test case are contained in a single folder. There are three types of files for each test case:

- 0 or more **data source files** for CSV (with extension `csv`), XML (with extension `xml`), and JSON (with extension `json`), or 1 file with SQL statements to create the necessary tables for relational databases (called `resource.sql`);
- 1 **file with the RML rules** (in Turtle format, called `mapping.ttl`); and
- 0 or 1 **file with the expected RDF** (in N-Quads format, called `output.nq`).

Distinct test cases assess different behaviours of the processors. Certain test cases assess the behaviour of the tools when (i) the required data sources are not available, and others when (ii) an error occurs and no output is generated. In the former, no data sources files or SQL statements are provided. In the latter, no file with the expected RDF is provided. The test cases are independent of how the processors materialize the knowledge graph: a data dump, as done by the RMLMapper, or on the fly, as done by Ontario [24].

[24] `https://www.w3.org/TR/EARL10/`, with prefix `earl`
[25] `http://www.w3.org/2001/sw/DataAccess/tests/test-manifest#`, with prefix `mf`
[26] `https://www.w3.org/2006/03/test-description#`, with prefix `test`
[27] `https://www.w3.org/TR/vocab-dcat/`, with prefix `dcat`
[28] `http://rml.io/ns/test-cases`, with prefix `rml-tc`

4.3.3 Differences with R2RML test cases

For most R2RML test cases, we created an RML variant for CSV, XML, JSON, MySQL, PostgreSQL, and SQL Server, leading to 6 RML test cases per R2RML test case. For R2RML test cases that focus on specific features of SQL queries, we only created 3 RML test cases, i.e., for MySQL, PostgreSQL, and SQL Server.

For test cases with CSV, XML, and JSON files as data sources, we created the corresponding files with the data based on the tables of the relational databases. For CSV, we used the table created by the SQL statements of the R2RML test case and stored it as a CSV file. For XML, the name of the table was used for the root of the XML document and every row of the table was used to create an XML element. Within this element, elements were created for each column and their values are the values of the corresponding columns in the table. For JSON, we followed a similar approach as XML. The file contains a JSON object at the root with the name of the table as the only attribute. This attribute has as value an array, where each element of the array corresponds with a row in the table. For each row, attributes were created for each column and their values are the values of the corresponding columns in the table.

Listings 4.1 to 4.4 contain the RML rules for MySQL, CSV, JSON, and XML that are based on the R2RML test case with id "R2RMLTC0002a"[29]. They test if the subjects consisting of two data fractions are generated correctly. The differences between the different RML rules are on the Logical Sources: they are dependent on the data format. For example, for the reference formulations of CSV, JSON, and XML are different: specific to the data format. For MySQL `rml:source` refers to the description of the MySQL database, while for CSV, JSON, and XML it refers to a local file. The remainder of the rules remain the same, because they are not affected by the data format.

Listing 4.1: RML rules to test if subjects consisting of two data fractions are generated correctly when using MySQL (RMLTC0002a-MySQL)

```
<TriplesMap1> a rr:TriplesMap;

  rml:logicalSource [
    rml:source <#DB_source>;
    rr:sqlVersion rr:SQL2008;
    rr:tableName "student";
  ];

  rr:subjectMap [
    rr:template "http://example.com/{ID}/{Name}";
    rr:class foaf:Person
  ];

  rr:predicateObjectMap [
```

[29] `https://www.w3.org/TR/rdb2rdf-test-cases/#R2RMLTC0002a`

```
    rr:predicate ex:id ;
    rr:objectMap [ rml:reference "ID" ]
  ];

  rr:predicateObjectMap [
    rr:predicate foaf:name ;
    rr:objectMap [ rml:reference "Name" ]
  ].

<#DB_source> a d2rq:Database;
  d2rq:jdbcDSN "CONNECTIONDSN";
  d2rq:jdbcDriver "com.mysql.cj.jdbc.Driver";
  d2rq:username "root";
  d2rq:password "" .
```

Listing 4.2: RML rules to test if subjects consisting of two data fractions are generated correctly when using CSV (RMLTC0002a-CSV)

```
<TriplesMap1> a rr:TriplesMap;

  rml:logicalSource [
    rml:source "student.csv";
    rml:referenceFormulation ql:CSV
  ];

  rr:subjectMap [
    rr:template "http://example.com/{ID}/{Name}";
    rr:class foaf:Person
  ];

  rr:predicateObjectMap [
    rr:predicate ex:id ;
    rr:objectMap [ rml:reference "ID" ]
  ];

  rr:predicateObjectMap [
    rr:predicate foaf:name ;
    rr:objectMap [ rml:reference "Name" ]
  ].
```

Listing 4.3: RML rules to test if subjects consisting of two data fractions are generated correctly when using JSON (RMLTC0002a-JSON)

```
<TriplesMap1> a rr:TriplesMap;

  rml:logicalSource [
    rml:source "student.json";
```

```
    rml:referenceFormulation ql:JSONPath;
    rml:iterator "$.students[*]"
  ];

  rr:subjectMap [
    rr:template "http://example.com/{ID}/{Name}";
    rr:class foaf:Person
  ];

  rr:predicateObjectMap [
    rr:predicate ex:id ;
    rr:objectMap [ rml:reference "ID" ]
  ];

  rr:predicateObjectMap [
    rr:predicate foaf:name ;
    rr:objectMap [ rml:reference "Name" ]
  ].
```

Listing 4.4: RML rules to test if subjects consisting of two data fractions are generated correctly when using XML (RMLTC0002a-XML)

```
<TriplesMap1> a rr:TriplesMap;

  rml:logicalSource [
    rml:source "student.xml";
    rml:referenceFormulation ql:XPath;
    rml:iterator "/students/student"
  ];

  rr:subjectMap [
    rr:template "http://example.com/{ID}/{Name}";
    rr:class foaf:Person
  ];

  rr:predicateObjectMap [
    rr:predicate ex:id ;
    rr:objectMap [ rml:reference "ID" ]
  ];

  rr:predicateObjectMap [
    rr:predicate foaf:name ;
    rr:objectMap [ rml:reference "Name" ]
  ].
```

Data errors. 2 of the R2RML test cases expect a data error to happen, e.g., when the subject IRI of an entity cannot be generated. In this case, an error is thrown and no knowledge graph is generated. With RML for entities where no subject IRI can be generated there is also no output generated, but, in contrast to R2RML, for the other entities the corresponding output is still generated. Therefore, for the corresponding RML test cases the processors can still throw an error, but the generation of the knowledge graph must not be halted.

Inverse expressions. 3 of the R2RML test cases are designed to test the use of inverse expressions[30]. However, inverse expressions are only used to optimize the knowledge graph generation and no differences are observed in the generated knowledge graph. Thus, whether inverse expressions are used by a processor or not cannot be verified by such test cases. Thus, we do not include them for RML.

SQL-specific features. 18 of the R2RML test cases focus on specific features of SQL queries, e.g., a duplicate column name in a SELECT query. As there are no corresponding RML test cases for CSV files, XML files with XPath, and JSON files with JSONPath, we only provide 54 corresponding test cases for MySQL, PostgreSQL, and SQL Server.

Null values. 1 of the R2RML test cases tests null values in the rows. However, a corresponding RML test case cannot be provided for the CSV and XML format, because both formats do not support null values.

Spaces in columns. 1 of the R2RML test cases is designed to test the behaviour when dealing with spaces in the columns of the SQL tables. However, a corresponding RML test case cannot be provided for the XML format, because it does not allow spaces in names.

In total, we have 297 test cases: 39 for CSV, 38 for XML, 40 for JSON, and 180 for relational databases. Of these 297, 255 test cases expect an knowledge graph to be generated, while 36 expect an error that halts the generation.

4.4 Test case execution and results

In this section, we describe the execution of the test cases and their results for two RML processors: the RMLMapper (v4.3.2) and CARML (v0.2.3). The implementation report can be found at `http://rml.io/implentation-report`.

We ran the RML test cases over the RML processors and annotated the obtained results using the EARL Schema. Three types of results are possible per test case:

[30] `https://www.w3.org/TR/r2rml/#inverse`

Table 4.2: The RMLMapper (v4.3.2) passes the majority of test cases, with the exception of 20 cases for relational databases.

	CSV	XML	JSON	My-SQL	Post-gres	SQL Server	Total
passed	39	38	40	53	54	53	**277**
failed	0	0	0	7	6	7	**20**
inapplicable	0	0	0	0	0	0	**0**

Table 4.3: CARML (v0.2.3) passes almost 3 out of 4 test cases, but does not support relational databases.

	CSV	XML	JSON	My-SQL	Post-gres	SQL Server	Total
passed	29	28	27	0	0	0	**84**
failed	10	10	13	0	0	0	**33**
inapplicable	0	0	0	60	60	60	**180**

"passed", "failed", and "inapplicable". **Passed** (`earl:passed`) is used either when the actual output matches the expected output when no error is expected, or when the tool throws an error when an error is expected. **Failed** (`earl:failed`) is used either when the actual output does not match the expected output if no error is expected, the processor returns an error trying to execute a test or the tool does not throw error if an error is expected. **Inapplicable** (`earl:inapplicable`) is used when the tool clearly states that specific features are not supported.

In Table 4.2 we show the results for the RMLMapper. It passes all CSV, JSON and XML test cases, but fails in 20 test cases for the RDBs. The failures are related to the automatic datatyping of literal for RDBs specified by R2RML[31]. RMLMapper should pass the failed test cases in next versions of the processor.

In Table 4.3 we show the results for the CARML. It partially passes the CSV, JSON and XML test cases, but it does not provide support for any of the RDBs test cases. The failures are related to multiple Predicate Maps and Named Graphs. The developers of the tool declare that CARML will support these features in next versions of the processor. However, at the moment of writing, we do not have any information about whether CARML will provide support for RDBs or not.

Finally, we can declare that testing a RML processor with the defined cases and analysing the obtained results offers a general view of the current status of it. These results also give useful information to the tool developers on knowing where they should put their effort to improve the conformance of the processor.

[31] `https://www.w3.org/TR/r2rml/#dfn-natural-rdf-literal`

4.5 Conclusion

With the introduction of an initial set of RML test cases (i) developers can determine how conformant their RML processors are to the RML specification, and (ii) users can use the test cases results to select the most appropriate processor for a specific use case. Before, users were only able to rely on the custom test cases, if any, which not necessarily assess every aspect of the specification. Now, users can rely on well-defined set of test cases that (a) clearly define what the input and expected output is, and (b) are reusable across different processors written in different programming languages.

The results of the test cases execution with the RMLMapper and CARML show that the CSV, XML, and JSON formats are almost fully supported, but RDBs cause difficulties or are not supported at all. The RMLMapper passes more test cases than CARML and therefore, the former is better when considering the conformance to the RML specification.

Our set of test cases is based on the R2RML test cases, and therefore, it covers a big part of the RML specification, as it is based on R2RML. However, as the R2RML test cases focus on relational databases, they do not take into account the specifics of hierarchical data formats, such as nested structures in JSON and XML files, which can be used with RML. Therefore, further research should be directed towards creating new test cases that tackle these specifics taking into account the differences between the different hierarchical formats and their corresponding reference formulations.

Acknowledgements

The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (AIO), the Research Foundation – Flanders (FWO), and the European Union. The work presented in this paper is partially supported by the Spanish Ministerio de Economía, Industria y Competitividad and EU FEDER funds under the DATOS 4.0: RETOS Y SOLUCIONES - UPM Spanish national project (TIN2016-78011-C4-4-R) and by an FPI grant (BES-2017-082511).

References

[1] Souripriya Das, Seema Sundara, and Richard Cyganiak. *R2RML: RDB to RDF Mapping Language*. Working Group Recommendation. W3C, Sept. 2012. URL: `http://www.w3.org/TR/r2rml/`.

[2] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data". In: *Workshop on Linked Data on the Web*. 2014. URL: `http://events.linkeddata.org/ldow2014/papers/ldow2014_paper_01.pdf`.

[3] Franck Michel, Loïc Djimenou, Catherine Faron-Zucker, and Johan Montagnat. "Translation of Relational and Non-relational Databases into RDF with xR2RML". In: *11th International Confenrence onWeb Information Systems and Technologies (WEBIST '15)*. 2015, pp. 443–454. URL: https://hal.archives-ouvertes.fr/hal-01207828/document.

[4] Stefan Bischof, Stefan Decker, Thomas Krennwallner, Nuno Lopes, and Axel Polleres. "Mapping between RDF and XML with XSPARQL". In: *Journal on Data Semantics* 1.3 (2012), pp. 147–185. ISSN: 1861-2032. DOI: 10.1007/s13740-012-0008-7. URL: https://doi.org/10.1007/s13740-012-0008-7.

[5] Kostis Kyzirakos, Dimitrianos Savva, Ioannis Vlachopoulos, Alexandros Vasileiou, Nikolaos Karalis, Manolis Koubarakis, and Stefan Manegold. "GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings". In: *Journal of Web Semantics* 52 (2018), pp. 16–32. DOI: 10.1016/j.websem.2018.08.003. URL: https://doi.org/10.1016/j.websem.2018.08.003.

[6] Jens Lehmann et al. "DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia". In: *Semantic Web* 6 (2015), pp. 167–195. DOI: 10.3233/SW-140134. URL: https://doi.org/10.3233/SW-140134.

[7] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. Recommendation. World Wide Web Consortium (W3C), Feb. 2014. URL: http://www.w3.org/TR/rdf11-concepts/.

[8] Juan F. Sequeda and Daniel P. Miranker. "Ultrawrap: SPARQL execution on relational data". In: *Web Semantics: Science, Services and Agents on the WWW* (2013). ISSN: 1570-8268. DOI: 10.1016/j.websem.2013.08.002. URL: https://doi.org/10.1016/j.websem.2013.08.002.

[9] Freddy Priyatna, Oscar Corcho, and Juan Sequeda. "Formalisation and Experiences of R2RML-based SPARQL to SQL Query Translation Using Morph". In: *23rd International Conference on WWW*. 2014. DOI: 10.1145/2566486.2567981. URL: http://doi.acm.org/10.1145/2566486.2567981.

[10] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. "Ontop: Answering SPARQL Queries over Relational Databases". In: *Semantic Web* (2017). DOI: 10.3233/SW-160217. URL: https://doi.org/10.3233/SW-160217.

[11] Stefan Bischof, Stefan Decker, Thomas Krennwallner, Nuno Lopes, and Axel Polleres. "Mapping between RDF and XML with XSPARQL". In: *Journal on Data Semantics* (2012). ISSN: 1861-2040. DOI: 10.1007/s13740-012-0008-7. URL: http://dx.doi.org/10.1007/s13740-012-0008-7.

[12] Boris Villazón-Terrazas and Michael Hausenblas. *RDB2RDF Implementation Report*. W3C Note. W3C, 2012. URL: https://www.w3.org/TR/rdb2rdf-implementations/.

[13] Nikolaos Konstantinou, Dimitrios-Emmanuel Spanos, Nikos Houssos, and Nikolaos Mitrou. "Exposing scholarly information as Linked Open Data: RDFizing DSpace contents". In: *The Electronic Library* 32.6 (2014), pp. 834–851. DOI: 10.1108/EL-12-2012-0156. URL: https://doi.org/10.1108/EL-12-2012-0156.

[14] Artem Chebotko, Shiyong Lu, and Farshad Fotouhi. "Semantics preserving SPARQL-to-SQL translation". In: *Data & Knowledge Engineering* 68.10 (2009), pp. 973–1000. DOI: 10.1016/j.datak.2009.04.001. URL: https://doi.org/10.1016/j.datak.2009.04.001.

[15] Anastasia Dimou, Ruben Verborgh, Miel Vander Sande, Erik Mannens, and Rik Van de Walle. "Machine-interpretable Dataset and Service Descriptions for Heterogeneous Data Access and Retrieval". In: *Proceedings of the 11th International Conference on Semantic Systems*. SEMANTICS '15. ACM, 2015. DOI: 10.1145/2814864.2814873. URL: http://doi.acm.org/10.1145/2814864.2814873.

[16] David Chaves-Fraga, Freddy Priyatna, Idafen Perez-Santana, and Oscar Corcho. "Virtual Statistics Knowledge Graph Generation from CSV files". In: *Emerging Topics in Semantic Technologies: ISWC 2018 Satellite Events*. Vol. 36. Studies on the Semantic Web. IOS Press, 2018, pp. 235–244. DOI: 10.3233/978-1-61499-894-5-235. URL: https://doi.org/10.3233/978-1-61499-894-5-235.

[17] Robert Battle and Dave Kolas. "Geosparql: enabling a geospatial semantic web". In: *Semantic Web* 3.4 (2011), pp. 355–370. URL: http://www.semantic-web-journal.net/sites/default/files/swj176_1.pdf.

[18] Manolis Koubarakis and Kostis Kyzirakos. "Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL". In: *Extended Semantic Web Conference*. Springer. 2010, pp. 425–439. DOI: 10.1007/978-3-642-13486-9_29. URL: https://doi.org/10.1007/978-3-642-13486-9_29.

[19] Maria-Esther Vidal, Kemele M Endris, Samaneh Jozashoori, Farah Karim, and Guillermo Palma. "Semantic Data Integration of Big Biomedical Data for Supporting Personalised Medicine". In: *Current Trends in Semantic Web Technologies: Theory and Practice*. Springer, 2019, pp. 25–56. DOI: h10.1007/978-3-030-06149-4_2. URL: https://doi.org/10.1007/978-3-030-06149-4_2.

[20] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language*. Recommendation. World Wide Web Consortium (W3C), Mar. 2013. URL: https://www.w3.org/TR/sparql11-query/.

[21] Holger Knublauch and Dimitris Kontokostas. *Shapes Constraint Language (SHACL)*. W3C Recommendation. W3C, 2017. URL: https://www.w3.org/TR/shacl/.

[22] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, and Juan Sequeda. *A Direct Mapping of Relational Data to RDF*. Working Group Recommendation. W3C, 2012. URL: http://www.w3.org/TR/rdb-direct-mapping/.

[23] Boris Villazón-Terrazas and Michael Hausenblas. *R2RML and Direct Mapping Test Cases*. W3C Note. http://www.w3.org/TR/rdb2rdf-test-cases/. W3C, 2012.

[24] Anastasia Dimou, Pieter Heyvaert, Ben De Meester, and Ruben Verborgh. "What Factors Influence the Design of a Linked Data Generation Algorithm?" In: *Proceedings of the 11th Workshop on Linked Data on the Web*. Apr. 2018. URL: http://events.linkeddata.org/ldow2018/papers/LDOW2018_paper_12.pdf.

Chapter 5

Conclusion

In this chapter, we review our research questions and hypotheses and how our contributions address the research challenges (see Table 5.1). Furthermore, we discuss remaining challenges and future directions.

Table 5.1: Alignment between our contributions, the challenges, research questions, hypotheses, and whether the contribution is part of the rule creation, refinement, or execution.

Contribution	**Research challenge**	**Research question**	**Hypothesis**	**Rule creation, refinement, or execution?**
MapVOWL	1, 2	1	1	creation
RMLEditor	1, 2	2	2	creation
Resglass	2	3	3	refinement
RML test cases	3	4		execution

5.1 Impact of contributions

We discuss each research question and corresponding hypothesis together with the contribution that tackles them, followed by how the contribution impacts the challenges. The first research question is

Research Question 1: *How can we design visualizations that improve the cognitive effectiveness of visual representations of knowledge graph generation rules?*

with corresponding hypothesis

Hypothesis 1: *MapVOWL improves the cognitive effectiveness of the generation rule visual representation to generate knowledge graphs compared to using RML directly.*

The evaluation conducted for MapVOWL provided evidence to accept Hypothesis 1: the use of MapVOWL is preferred over RML for representing knowledge graph generation rules. Furthermore, users would also prefer to use MapVOWL to create and edit rules (see Chapter 2). The second research question is

Research Question 2: *How can we visualize the components of a knowledge graph generation process to improve its cognitive effectiveness?*

with corresponding hypothesis

Hypothesis 2: *The cognitive effectiveness provided by the RMLEditor's GUI improves the user's performance during the knowledge graph generation process compared to RMLx.*

The evaluation conducted for the RMLEditor provided evidence to accept Hypothesis 2: the RMLEditor is preferred over RMLx for creating rules. This is due to the use of MapVOWL, which is independent of the underlying language, as pointed out by the participants of the evaluation (see Chapter 2). By tackling Research Questions 1 and 2, MapVOWL and the RMLEditor contribute to tackling the following challenges

Challenge 1: *Improvement of users' understanding of the rule creation's components.*

Challenge 2: *Avoidance and removal of inconsistencies introduced by concepts, relationships, and semantic model.*

MapVOWL improves the users' understanding of the rules without being required to understand the syntax and grammar of the used language, because users only need to understand the visual notation. This notation is translated to language-specific rules by, for example, the RMLEditor. Furthermore, it improves the users' understanding of how the data sources are related to the used concepts and relationships, and how the semantic model affects the generated knowledge graphs. The RMLEditor further aids users by dealing with the visualization-related scalability issues, the visualization of heterogeneous data sources, and the integration of transformations of the data sources in the visualizations of the rules.

The third research question is

Research Question 3: *How can we score and rank rules and ontology terms for inspection to improve the manual resolution of inconsistencies?*

with corresponding hypothesis

Hypothesis 3: *The automatic inconsistency-driven ranking of Resglass improves, compared to a random ranking, by at least 20% the overlap with experts' manual ranking.*

The evaluation conducted for our ranking, which is part of Resglass, provided evidence to accept Hypothesis 3: our ranking has 80% overlap with experts' rankings for both rules and ontology terms. Furthermore, our ranking improves the overlap with expert's with 41% for rules and 23% for ontology terms, compared to a random ranking (see Chapter 3). By tackling Research Question 3, Resglass contributes to tackling Challenge 2: it assists users with the removal of inconsistencies introduced by ontologies and semantic model. However, it does not assist in avoiding them.

The fourth research question is

Research Question 4: *What are the characteristics of test cases for processors that generate knowledge graphs from heterogeneous data sources independent of languages' specifications?*

With the introduction of an initial set of RML test cases (i) developers can determine how conformant their RML processors are to the RML specification, and (ii) users can use the test cases results to select the most appropriate processor for a specific use case (see Chapter 4). By working towards Research Question 4, our RML test cases provide useful insights for the following challenge

Challenge 3: *Selection of the most suitable processor for the use case at hand.*

These test cases help towards determining the characteristics of test cases that are independent of the RML language, because they already list a number of differences between test cases of languages that only work with relational databases and languages that support multiple, heterogeneous data sources.

5.2 Remaining challenges and future directions

MapVOWL is preferred over RML to visualize knowledge graph generation rules. However, future research is needed to investigate how to visualize common data structures, such as lists and bags. Furthermore, the representation of these rules should not necessarily only happen via visualizations (see Chapter 2): there are users who desire a text-based approach. To this end, initial steps have been taken by introducing YARRRML, a human-readable text-based representation for knowledge graph generation rules [1]. Future research efforts can compare YARRRML to existing languages, such as RML and SPARQL-Generate [2], through user evaluations that are similar to the one for MapVOWL. This allows a better understanding of the advantages and disadvantages of representations, together with what needs to be improved. Furthermore, research with respect to how collections, such as lists and bags, are represented is also of interest, because although they do not have a dedicated notation in RDF it is not straightforward to define how they are generated using the existing representations.

The RMLEditor is preferred over RMLx. However, future research is needed to avoid or clarify specific terminology and combine MapVOWL with other representations, such as YARRRML, to support users who do not (only) rely on visualizations.

Furthermore, additional features can be added to the RMLEditor to improve the overall rule creation process, such as importing an ontology for which the corresponding rules are created based on the classes, properties, and datatypes in the ontology. Future research efforts can compare the RMLEditor to other rule editors through user evaluations that are similar to the one with RMLx. This allows a better understanding of how the RMLEditor compares to other editors that are different from RMLx, i.e., apply a different approach or support different features.

Resglass, including its ranking of rules and ontology terms, improves the manual resolution of inconsistencies in knowledge graphs. Future research can investigate how to further assist users in both detecting and resolving inconsistencies, such as, for example, suggesting possible resolutions based on the used and other ontologies, and rules of other use cases. Furthermore, Ressglass' ranking can also be used to drive a (semi-)automatic solution. For example, this can be done by building on our previous work [3] where we proposed an automatic solution that resolves the inconsistencies by applying a predefined set of refinements to the rules. The ranking, which is use case-specific, can be used to make a more informed decision regarding which refinements should be applied, instead of solely relying on a predefined set, which is created independent of the use case. Besides the removal of inconsistencies, future research is needed to avoid inconsistencies as early as possible in the rule creation process. This can be done by extending the RMLEditor to guide users when selecting relevant data, ontologies, and their terms.

The initial set of RML test cases benefits both developers and users of RML processors, and highlights differences with test cases for relational database-specific languages. However, future research is needed to further investigate the characteristics of test cases that can easily be applied to other languages, that combine different data formats, and that tackle the use cases where data streams are used to generate knowledge graphs instead of static files or databases. This might and hopefully will trigger the discussion on what should be part of a knowledge graph generation language and what not, which will be reflected on what is tested by the test cases. Past research efforts showed that supporting multiple, heterogeneous data sources is a desired feature. However, to what extend is for example pre-processing of the data sources required? Pre-processing can go from fixing capitalization in sentences to deduplication of entities using additional data sources. Do we include the former and the latter? Or only one of them? Thus, which features are in the scope of a knowledge graph generation language and which are not? Furthermore, investigating other metrics, besides conformance, further improves the selection of the most suitable processor. Such metrics could include but are not limited to speed, memory consumption, usability, and extensibility.

References

[1] Pieter Heyvaert, Ben De Meester, Anastasia Dimou, and Ruben Verborgh. "Declarative Rules for Linked Data Generation at your Fingertips!" In: *Proceedings of the 15th ESWC: Posters and Demos*. Vol. 11155. Lecture Notes in Computer Science. Springer, June 2018, pp. 213–217. DOI: 10.1007/978-3-319-98192-5_40. URL: https://doi.org/10.1007/978-3-319-98192-5_40.

[2] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. "A SPARQL Extension for Generating RDF from Heterogeneous Formats". In: *Proceedings of the 14th Extended Semantic Web Conference*. Springer. 2017, pp. 35–50. DOI: 10.1007/978-3-319-58068-5_3. URL: https://doi.org/10.1007/978-3-319-58068-5_3.

[3] Anastasia Dimou, Dimitris Kontokostas, Markus Freudenberg, Ruben Verborgh, Jens Lehmann, Erik Mannens, Sebastian Hellmann, and Rik Van de Walle. "Assessing and Refining Mappings to RDF to Improve Dataset Quality". In: *Proceedings of the 14th International Semantic Web Conference*. Vol. 9367. Lecture Notes in Computer Science. 2015, pp. 133–149. DOI: 10.1007/978-3-319-25010-6_8. URL: http://dx.doi.org/10.1007/978-3-319-25010-6_8.

Afterword

Jah is never too late.
Jah is always on time.

---

28 years have passed. Wauw. 28 years. If you would have asked the 15-16 year old version of me what I wanted to become, he would have said a fighter pilot in the army. Yeah, might not be what you would have expected. But not that strange either, considering that it's in the top 15 kids' dream jobs (based on the first hit on Google. A totally scientific, reliable reference I know.) How things have changed. The reason why that never happened is because I had to bury that option the moment they told me that you can't become a pilot if you don't have a 20/20 vision. And I was already wearing glasses. However, because it was not a real real dream, I quickly managed to get over it and move on with my teenage life (woohoo video games). I graduated from high school and decided to go to Ghent University to become a Software Engineer (I liked computers), where I graduated in 2014.

Discipline and hard work, instilled in me by my parents at a young age. That is the reason why I graduated from high school and university. You don't need anything else. Some would argue that talent has something to with it. Bullshit. Talent might give you a headstart, but the outcome at the finish line is determined by your own hard work.

After starting university in 2009, I joined one of the local dance schools in Ghent, B.O.S.S. Dance Complex in October the following year. Little did I know back then the impact it would have on my life: I call myself a dancer before anything else. I began taking one class a week. Hip Hop Lyrical, every Friday evening from 20:00 till 21:00. Every semester I took one or more extra classes. Hip Hop Basic. Hip Hop Open Level. Ragga, which lead me to discover Dancehall. Modern. Jazz. Krump. I even attempted Ballet a few times. Yes, a sight to admire I can guarantee you that.

A new world had been opened. A world that many considered was the opposite of the scientific universe that I was exploring at the university. Some claimed that I was walking two paths that should not be combined. Or could not be combined. Getting a master degree at a university and dancing 20 to 25 hours a week at the same time. They said "why would you wanna do both. University is more important after all. And

it's pretty impossible to combine in any case." But to say it in the words of the late Nelson Mandela "It always seems impossible until it's done."

I've never met adversity in my life, in the common meaning of the word. I have never had to worry about money. I never had to worry about going to school. I never had to deal with my parents being divorced. I never had to worry about having food for the next meal. I never had to worry about not having presents for Christmas or my birthday. Because of my parents: they worked hard so they could and can provide my brother and me with the best. I never had to deal with life-threatening diseases, in the strict sense. I never had to worry about my grades. Consider this bragging if you want, but I've put in sufficient effort at school and my grades reflected that. I could immediately start my PhD at Ghent University after graduation and didn't have to worry about a job. Looking at it that way: I never met adversity in my life.

So when the moment came that through a series of injuries I couldn't dance anymore, the phrases "but how can you feel sad? Ok you can't dance anymore, but you don't have to worry about money. You have a job. Some people have it worse," were often fired at me at moments when I was "sad". Or "in a bad mood", which I apparently was not entitled to have. Apparently I shouldn't have let it influence my interaction with others, because it's not their fault after all. Easier said than done, by people who haven't just lost the most important thing in their lives and who clearly understand everything that is going on in your mind, and heart. I will ask you this: have you ever gone through a period of time when you didn't have a reason to wake up anymore; when you went to sleep wishing that there was no next day?

I've walked through the valley of the shadow of death and it ain't pretty. It ain't pretty. And I honestly hope that nobody reading this has to ever deal with that. Because there were days, weeks, months that I believed that I would never return from that place. The valley of the shadow of death. But if it learned me one thing then it's at least this: the only reason I can say that there were only shadows in the valley is my love for my parents. The love that I now have because of the love they showed me for the past 28 years. I don't show it enough through my words or actions, but mom, dad, yes I love you both, with all my heart. Until the shadows are gone and long after that.

A little bit of advice. Call it a small rant, if you want. When talking to someone at his or her darkest moment, don't ever say that "you understand", you probably don't. You don't. Unless you have walked the path they have walked, you can only have a limited understanding of how they feel. You are downplaying their feelings as if it is just some trivial thing that everybody experiences and thus they should not make a fuss about it. Don't think so highly of yourself, that you understand everything that is happening around you. It's ok if you don't. Accepting that you don't, can be more liberating than you think. On that note: don't ever say "everything will be alright", you don't know that. You don't know that. Can you predict the future? Can you? It all might be with a good intention, but remember that the road to hell is paved with good intentions. Having shattered dreams is worse than having no dreams at all.

But not all was grim, as I can say that I'm not longer residing at the valley of the shadow of death. I passed through it. Although it changed me forever. After looking

for different things, hobbies to replace my love for dancing, as part of my quest to find my escape route out of that valley, I came to the conclusion that there is nothing that will ever replace dancing. So I made the decision that I will dance again or die trying. And I meant that in the literal sense, no metaphor, no reading between the lines here. Of course, as much I would have loved that thinking those words would suddenly fix everything, it didn't. My spirits had a hard time with the first doctor saying he couldn't do anything about my injuries. I cried a lot. My spirits had a hard time with the second doctor saying he couldn't do anything for me. I cried some more. At random moments. During a sad song in the car. My spirits had a hard time with the third doctor saying he couldn't do anything for me. I cried. When my friends were talking about their future, about their plans, while I had nothing left to care about. It was not until the fourth doctor who suggested surgery that I saw the gates opening again to that world I entered in 2010. A surgeon from the Ghent University Hospital, not unimportant to mention. Dr. Wim Bongaerts, you might never read this: for you this surgery was a routine one, as you said yourself the moment you gave me the required hospital papers. For me it changed my life. It saved my life: I'm dancing again. I will never be able to repay you for that.

Besides sparking the fire in my feet again, this turning point in my life also signaled the start of the most fun, interesting, exciting part of my PhD, although both are unrelated. Paths that sometimes appear so different, reinforce each other in ways that are hard to explain. The same goes for people, friends, colleagues, dancers, and fellow competitors.

*

Here we are now. The year 2019. Submitted my PhD book to Ghent University, which I have been part of for the last 10 years now. My university based in Ghent, which I call my city when people from abroad ask me where I'm from, although I don't live here.

I know that often the thank you-part is mostly about the people who contributed to the PhD and related activities. Mine will be about life. It's only at a limited number of moments that we can really separate everything: school vs hobbies, family vs friends, life vs work, life vs love, studying vs learning, irrelevant vs irreplaceable. Life is not a pool of water drops that you carefully select. Life is an ocean of vastes amounts of water that contribute to who you are: buckets of experience, subconscious underwater rivers, and currents that we try to ride or avoid.

A super big thank you to all medical personnel that helped and continue to help me: my GP Dr. Isabelle De Leenheer, my podiatrist Michaël De Geyter, my surgeon Dr. Wim Bongaerts, my physiotherapists Kris De Witte, and Jasmien Van Wambeke.

To my favorite fellow graduates of Ghent University: Gertjan, Simon, Jens,

Emanuel, Yves, Nils, Timothy, and Nicola. The moments shared during those five years of classes. The projects we completed together. The times we went out for the night. Cliche I know, but I could not have wished for a better group of friends to experience my student life. Full of jokes and silliness (*cough* playing Worms on the iPad during class *cough*), but also seriousness when needed. Thank you all for everything.

To my colleagues who accompanied me during my second adventure at Ghent University: Anastasia, Ben, Sven, Ruben V., Joachim, Miel, Dörthe, Gerald, Martin, Julián, Pieter C., Ruben T., Erik, Pieter B., Femke O., Alexander, Tom, Dieter D. P., Dieter D. W., and Laurens. You all contributed in your own, unique way. But of course there are a few people that I want to thank specifically. To Erik: you said that I would be thankful that you convinced me to write my first journal, although I was not really in favor of that. Well, you can now say "told you so" haha. But credit where credit is due! To Ruben V.: as one of my PhD advisors I can thank you not enough for the help and guidance from the beginning to finish. To Sven: although you are not working on the same topics as me and Ben, you are still part of our team! And you joining has only made the team more complete and of course funnier! In case this makes you blush, don't blush for too long because I'm not sure if that is how you should spend your time. To Ben: I'm so happy that we both did our PhD at the same time. Sharing experiences (Piña coladas at conferences woohoo!) and annoyances and frustrations together (*insert example yourself*). Based on how we get along, I truly believe that you would perfectly fit in my group of university friends: you like computers and beer and so do they. Thanks for everything so far and for everything that is still to come, whether it are pirates, parrots, unicorns or other mythical and less mythical creatures and ideas! To Anastasia: where should I start? I has been quite the journey hasn't it? As my closest PhD advisor, I'm pretty sure that you agree that I'm not always the easiest person to work with at certain times. And from time to time I also had the uncontrollable feeling to strangle you, but look at us now: I'm submitting my PhD and you are watching your first PhD student (sorry Ben) submitting his PhD. Although it has my name on the cover, I will always see it as something we did together. Thanks for everything! *Insert a big hug*

To my Greeks Vicky and Sotiris: παιδιαااااα! Of course I have to include you in this too. It would be crazy if I didn't do that! We met at a conference in 2015 and you invited me to visit you when I was in Greece, and that, like, actually happened! And since then we keep visiting, talking, calling eachother. Whether you are in Thessaloniki or London, who cares?! All the laughs we shared. All the food. Bougatsa! Yes!! OMG frappeeeeeee! I cannot thank you and your family enough for the hospitality you showed me. To Michalis: always be nice to your sister, but not too nice, she doesn't deserve it haha! To κυρια Matina: thank you for welcoming me in your home, where a guest never starves! You already know this, but it has to be said again: you can be really proud of your daughter. Moving alone to a different country, getting a job, living all alone. She did that! To κυριε Anestēs: I'm not gonna tell you too how proud you should be of Vicky or we will never hear the end of it (poor Sotiris)! Also a big thank you for welcoming me. It was always a pleasure to visit. I know you are not

a man of many words, so when you said "you are like a son to us" the moment I saw you for the last time, in London, it touched me more than I would have ever thought. You are missed. To Vicky and Sotiris, again: Εεε καλε! We have to call again soon!! Σας αγαπω!

To my Kobo Bruce: when I joined your dance school, I had no clue who you were, what you already achieved in the past, or what you would achieve in the future. You became my teacher when I started taking one of your Hip Hop classes. You became my mentor and coach when you gave me the male leading role in school's show of 2013. Now, you are all of this and more: you are also my friend, together in the same crew and movement Kobo Power. Of all the people mentioned in this afterword, you are and always be the one who understands best the dance journey I embarked on. You experienced in the past what I'm experiencing now. You sometimes had to deal with setbacks and disappointments that I have to cope with now too. We both know how it feels to win. We both know how it feels to lose. It's a long and sometimes lonely road, so I keep grinding, keep pushing, keep working, keep training, keep labbing, and keep battling. I will not quit. You know already. Thank you for everything you did so far. I will never forget, unlike others, and will never hold back in telling people that you are one of the main reasons that I'm that dancer that I'm now today. Bless.

To The Nine: we have all been friends since high school. I've met all of you at different times: some when I was 13 years old, some when I was 17 years old. With some of you I have been in the same class for one year or two years, with some never. Nevertheless, during those times bonds were being made that would last years. A couple of months ago, when I was telling that we were going celebrating the first bachelor party of one of you, people were surprised that after all these years after finishing high school we still keep in touch. It's not merely keeping in touch. We get together regularly, celebrate new year's eve together, birth days together, weddings together, random evenings at someone's place together. You guys are my best friends. Originally, the idea was to address all of you individually as I did with the others, but it felt kinda weird doing that. All the things of which I have the fondest memories were not merely shared with only one of you, but with at least three, four, five of you guys. Addressing each of you separately would break these memories into incoherent chunks incorrectly matched to each of you as individuals. Don't get me wrong. You all have your unique personalities, skills, and characteristics that make you who are and that guaranteed you a spot in The Nine. But to me, when I think about one of you will always think about all of you and when I think about all of you I will always think about each of you. I hope from the depths of my heart that you all achieve in life what you desire, whatever, wherever, and whenever that may be. Guys, I love you.

To my parents: as I said earlier, I don't show my gratitude enough, either through my words or my actions. So I find it more than fitting to compensate for that here and now, knowing very well that it will never be enough. Thank you for raising me the way you did. The fruits of this PhD are also for you to reap. The trophies that I lifted and will lift to the sky are a symbol of not only a proof of victory for me, but a proof of your caring and support. Thank you for my youth. Thank you for my education.

Thank you for a roof above my head. Thank you for our house. Thank you for our home. Thank you for the food in my mouth. Thank you for the discipline you required from me. Thank you for your desire for me to be the best, better than the other, better than myself. Thank you for giving me a mother. Thank you for giving me a father. Thank you for you. Mom, dad, I love you both, with all my heart. Until the shadows are gone and long after that.

Pieter Heyvaert
June 2019