Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik

Documenting Data Integration Using Knowledge Graphs

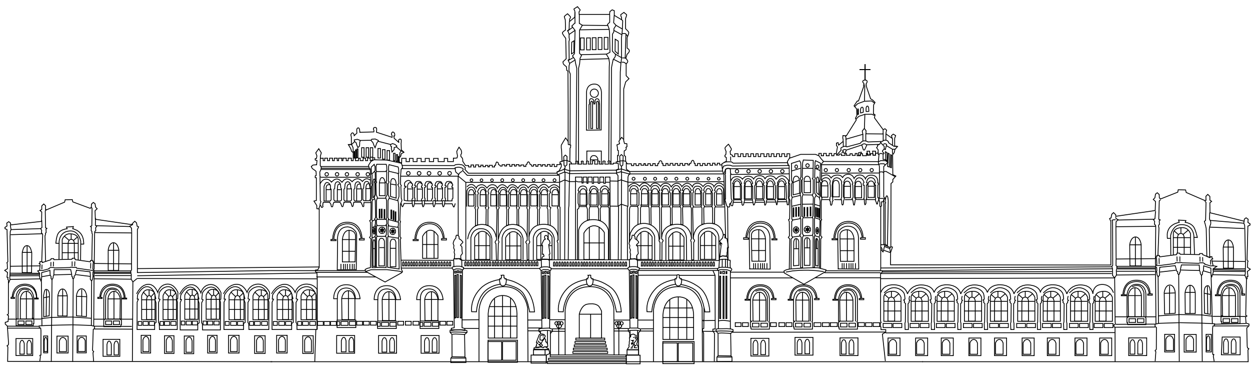
A thesis submitted in fulfillment of the requirements for the degree of
Master of Computer Science

BY

Sepideh Azizi
Matriculation number: 10025839
E-mail: s.azizi@stud.uni-hannover.de

First evaluator: Prof. Dr. Maria-Esther Vidal
Second evaluator: Prof. Dr. Sören Auer
Supervisor: Prof. Dr. Maria-Esther Vidal

June 12, 2023



Declaration of Authorship

I, Sepideh Azizi, declare that this thesis titled, 'Documenting Data Integration Using Knowledge Graphs' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Sepideh Azizi

Signature: ________________________________

Date: ________________________________

Acknowledgements

First of all, I would like to extend my sincere gratitude to Prof. Dr. Maria-Esther Vidal for her invaluable support and guidance throughout my master thesis. Without her encouragement, it would not have been possible to achieve the results presented in this work.

I would also like to express my appreciation to Samaneh Jozashoori for serving as my supervisor and providing help and support during the course of my thesis. I am grateful to the members of the SDM research group at TIB, especially Emetis Niazmand, for welcoming me as a guest member and providing assistance.

I am thankful to my dear friend Amir Khatibdamavandi for his unwavering patience and support during this journey, and for helping me to discover my own abilities.

Finally, I would like to express my love and gratitude to my family, especially my parents and siblings, who have been my constant source of support. Their love and encouragement has been the driving force behind my achievements, and I dedicate this success to them.

Sepideh Azizi

Abstract

With the increasing volume of data on the Web and the proliferation of published knowledge graphs, there is a growing need for improved data management and information extraction. However, heterogeneity issues across the data sources, i.e., various formats and systems, negatively impact efficient access, manage, reuse, and analyze the data. A data integration system (DIS) provides uniform access to heterogeneous data sources and their relationships; it offers a unified and comprehensive view of the data. DISs resort to mapping rules, expressed in declarative languages like RML, to align data from various sources to classes and properties defined in an ontology. This work defines a knowledge graph where data integration systems are represented as factual statements. The aim of this work is to provide the basis for integrated analysis of data collected from heterogeneous data silos. The proposed knowledge graph is also specified as a data integration system, that integrates all data integration systems. The proposed solution includes a unified schema, which defines and explains the relationships between all elements in the data integration system $DIS{=}\langle G, S, M, F\rangle$. The results suggest that factual statements from the proposed knowledge graph, improve the understanding of the features that characterize knowledge graphs declaratively defined like data integration systems.

Keywords: Data integration system, Knowledge Graph, Mapping rules, Ontology, Data source

Contents

List of Figures

List of Tables

Acronyms

CDO	Change Detection Ontology
DCAT	Data Catalog Vocabulary
DIS	Data Integration System
DL	Description Logic
GAV	Global as View
KG	Knowledge Graph
LAV	Local as View
OWL	Web Ontology Language
RDF	Resource Description Framework
RML	RDF Mapping Language
SDM	Semantic Data Modeling
SHACL	Shapes Constraint Language

Chapter 1

Introduction

With the increasing amount of data available, effective data integration systems and knowledge graphs have become paramount.
Knowledge graphs are graph-based models that consist of entities and relationships connected using semantic metadata. They provide a framework for extracting knowledge from raw data, enabling subsequent data integration, sharing, and analysis in a way that is understandable for both humans and machines [4].
Data integration systems combine data from different sources to provide a unified view of the data [11, 26]. The design of such systems is crucial in real-world applications and presents interesting theoretical challenges. This thesis focuses on some of these challenges, particularly concerning the architecture of data integration systems. The systems we are concerned with in this work are characterized by a global schema-based architecture and data sources. This architecture ensures that data from various sources is integrated, transformed into a consistent format, and stored in a centralized repository [20]. Key topics in this area include data sources, the unified schema (ontology), and the mapping rules that establish correspondence between them. Data integration system addresses major challenges in data management such as accessibility, interoperability, data findability, and reuse [7]. Integrating data sources, mappings, and ontologies can significantly improve data management. This work aims to define connections between different elements using RDF rules within a data integration system. This results in a new unified schema that includes all these elements, thus optimizing the mapping and unified schema creation and usage.
Table 1.1 provides the following research questions to assess the unified global schema of data integration systems.

Research Questions	
RQ1	What are the main features that characterize a knowledge graph defined as a data integration system?
RQ2	What are the main challenges and limitations in documenting data integration systems?
RQ3	What are the main data integration approaches?
RQ4	How do data integration system impact tasks of data management and knowledge graph creation?
RQ5	What is the relevant metadata that characterize a data integration system?
RQ6	Which type of concepts should be defined in a data integration system? How are they connected to each other?
RQ7	What is the role of declarative mapping languages in the documentation of data integration systems?

Table 1.1: Research Questions

1.1 Motivating Example

To motivate the work of this thesis, we consider in the current world web the abundance of data sets, multiple ontologies that describe the data sets, and numerous triples maps that connect the data set to the corresponding ontologies. These data sets, ontologies, and mappings come from various sources and concepts. In this thesis, we use two different scenarios as motivating examples.

1.1.1 Scenario 1

The first motivating example is about receiving a new data set, and you need to integrate it into your existing knowledge graph. Consider your work for a research team that creates specific medical treatments for patients. Your team has recently received a new data set from a recent clinic containing information on patients, their diseases, and related information such as names, disease names, disease stages, drugs, and family history. However, this data format differs from what your team typically uses. You need to find a corresponding ontology and map the new data to integrate it into your existing system. By doing this, you can categorize the diseases automatically, analyze the information, and generate tailored medical treatments for each patient based on their specific situations. This will improve the accuracy and efficiency of your services, making it easier for specialists to compare cases and make treatment decisions. In short, mapping the new data set to a corresponding ontology will help your research team provide better results by enabling improved data analysis and medical treatment recommendations.

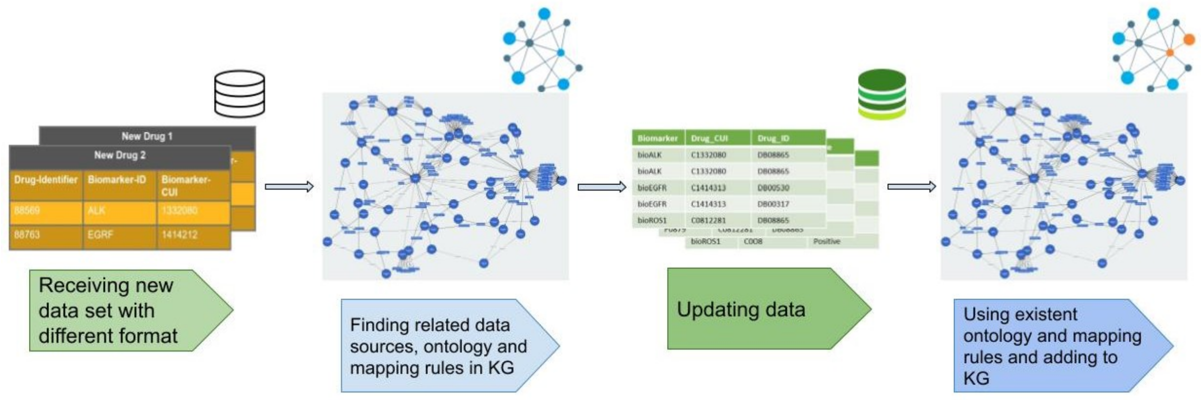


Figure 1.1: Motivating Example. Scenario 1: Receiving new data set and updating the current knowledge graph.

Incorporating a new data source into a data integration system requires identifying a corresponding existing ontology and mapping it to the latest data source. This metadata provides context for the new data and ensures it can be appropriately analyzed and utilized. With a matching ontology and mapping, multiple ontologies and mappings for the same data could accumulate, leading to clarity and inefficiencies. Identifying a matching ontology and mapping is essential to avoid this problem and maintain a clear relationship between the data sources, ontologies, and mappings.

1.1.2 Scenario 2

For the second motivating example in this work, consider the several data integration shown in Figure 1.2. Each data integration consists of ontologies and a set of data sources, with many mapping rules defined to correspond these data sources to the ontology.

Suppose you have a date as a start date for treatment, but it needs to be clarified to which treatment it belongs. Finding a repetitive variable, such as the date, requires checking many triples maps and mapping assertions among all the data sources and mapping rules. The demand for increased accessibility and findability of specific data is crucial in data management, mainly when dealing with massive data sources and many mapping rules with complex relationships. An upper-layer data integration

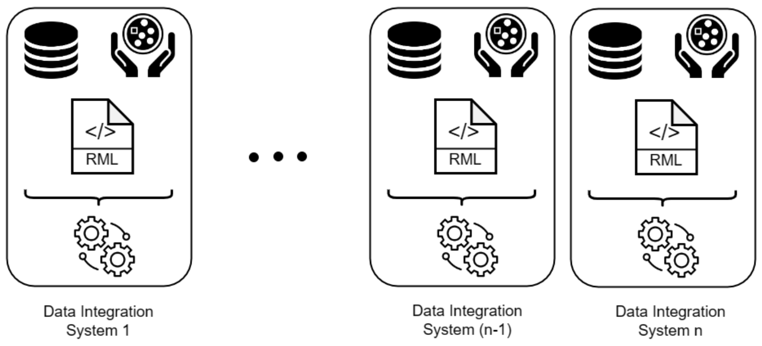

RML
RML
RML
Data Integration System 1
Data Integration System (n-1)
Data Integration System n

Figure 1.2: Motivating Example. Scenario 2: There are many data integration systems to work on them.

system (as shown in Figure 1.3) can be created to address this issue. This system can help locate a specific piece of data, for example, by finding the triples map that has defined the properties of a patient. This ability makes it easier to understand the relationships and explanations of the data.

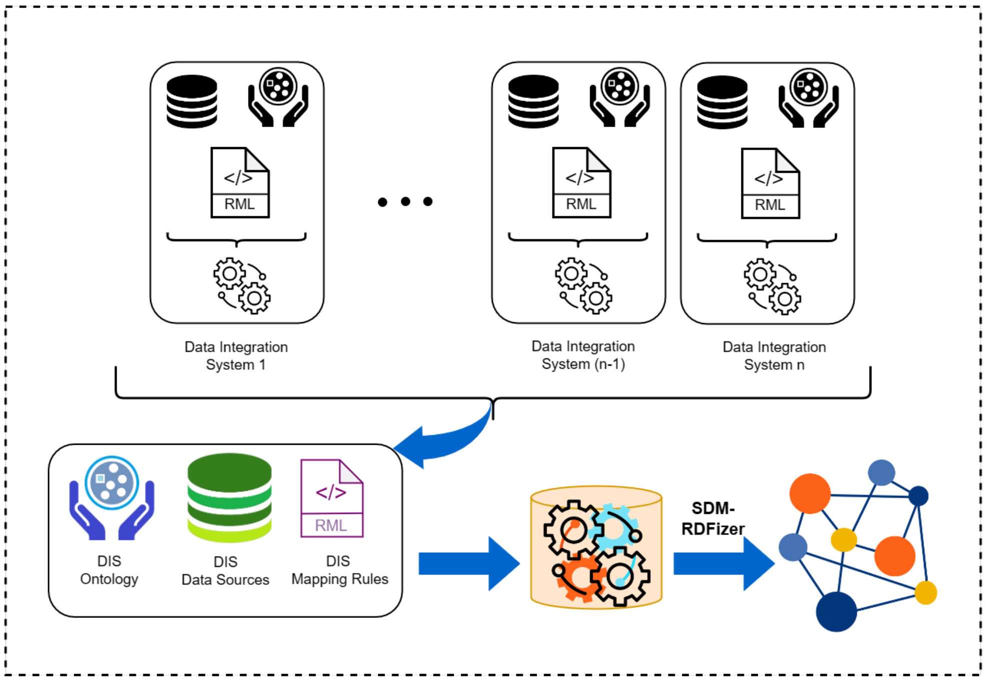

RML
RML
RML
Data Integration System 1
Data Integration System (n-1)
Data Integration System n
RML
DIS Ontology
DIS Data Sources
DIS Mapping Rules
SDM-RDFizer

Figure 1.3: Motivating Example. Scenario 2: Creating knowledge graph based on new data integration system, which helps to track data and reuse resources.

To reach our goal, we use a unified schema that defines the relationships of entities based on the data sources and mapping rules, such as the relationship between a patient and his/her name, treatment, and birth date. We make a set of data sources for the data integration system from ontologies, all data sources, and all mapping rules. The mapping rules of the data integration system correspond to the data sources to the unified schema.
The SDM-RDFizer [1], a mapping rule interpreter for RML, is utilized to create N-triple and semantify data based on the mapping rules (Figure1.4).
With the created knowledge graph, we can easily find specific data, such as the particular triples map that defines the treatment start date for a patient, with just one query on the knowledge graph.

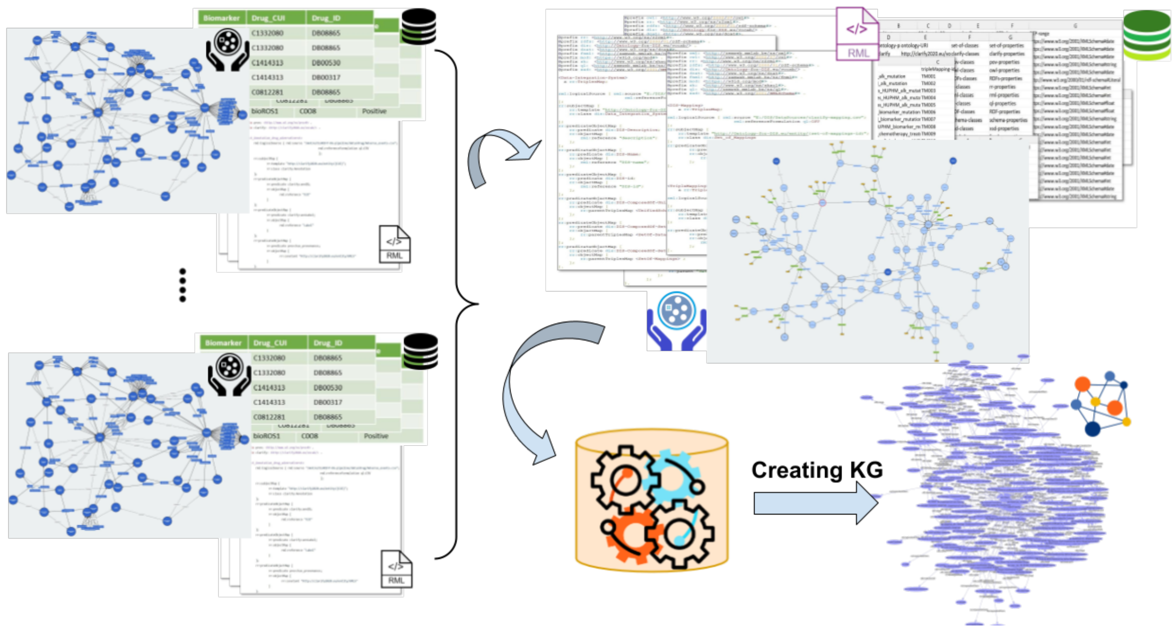


Figure 1.4: Motivating Example. Scenario 2: The knowledge graph of the data integration system is created by using a unified schema and mapping rules to correspond data from different sources to the schema. This is achieved through the use of N-triples generated by SDM-RDFizer, and allows for SPARQL queries to be answered over the resulting knowledge graph.

[1]`https://github.com/SDM-TIB/SDM-RDFizer`

1.2 Contributions

The main contribution of this work is developing a theoretical framework for creating a data integration system using semantic web technologies. This framework is based on the global as view (GAV) approach. It utilizes semantic data modeling to provide three types of designs: **1) Conceptual design**, which involves creating an abstract and high-level representation of the data structure and organization. **2) Logical design** is a data modeling stage that follows the conceptual design and involves refining the conceptual data model for a specific database management system. **3)Physical design** is the final stage in data modeling and consists in translating the logical data model into a design optimized for a specific technology. Additionally, the work includes an **4) Empirical evaluation** of the performance of a knowledge graph based on network analysis using Cytoscape. This evaluation provides valuable insights into the effectiveness of the proposed framework in real-world scenarios.

1.3 Overview of the Document

The thesis is structured into seven chapters. Chapter 1 is the introduction and provides two motivating examples and the problem being addressed in the work. Chapter 2 presents the mathematical notations, symbols, and essential concepts necessary for understanding the subsequent chapters. After understanding the required background, Chapter 3 highlights relevant topics related to this work while referring to relevant papers. The approach, including the formal problem definition and the proposed solution, is discussed in Chapter 4. Chapter 5 offers the reader an implementation of the approach. The implementation's performance is evaluated using benchmarks in Chapter 6. The thesis concludes in Chapter 7.

1.4 Summary of the Chapter

To summarize, this chapter concludes the scope of the thesis. The overall approach is introduced through the use of motivating examples. The challenge of managing data due to its heterogeneity and large volume is discussed, highlighting the need for a solution.

Chapter 2

Background

The data creation and storage rate, regardless of its heterogeneity or homogeneity and format, is rapidly increasing. To make informed decisions, extracting valuable insights from this raw data is crucial. These can include extract and transform processes, which involve removing data from different sources, converting it into a standard format, and loading it into a central repository. Once the data is integrated, it can be cleansed, transformed, and analyzed in a unified way, providing a more accurate and comprehensive view of the data. This highlights the importance of *data integration system* and making a knowledge graph in the semantic web.
A *knowledge graph* can be created using a data integration system comprising various data sources, a unified schema, and mapping rules connecting these sources to ontologies. This allows for the retrieval and tracking of resources by defining relationships and vocabularies between concepts. Data integration systems also play a crucial role in solving the **FAIR** data [2] challenge in data management by making data **F**indable, **A**ccessible, **I**nteroperable, and **R**eusable. In this section, we will discuss fundamental concepts such as data integration systems, Data Catalog Vocabulary (DCAT), mapping rules, and related semantic web technologies like the Resource Description Framework (RDF), RDF Schema (RDFS), the Web Ontology Language (OWL), and the Shapes Constraint Language (SHACL).

2.1 Semantic Web Technologies

By defining an Open Web Platform for application development, the *W3C* standards offer developers an exceptional opportunity to create rich, interactive experiences

[2] `https://www.go-fair.org/fair-principles`

using vast data stores, which can be accessed from any device. This platform's full potential relies on various technologies being developed by the W3C and its partners, such as CSS, SVG, WOFF, various APIs, XML, and the Semantic Web stack [3]. Here there is a brief explanation of the Semantic Web.
Semantic web is promoted and developed by the World Wide Web Consortium (W3C), an international standardization body for the web. The semantic web is a network of data that shows us how the data is stored in databases[4]. According to the inventor of the World Wide Web, Tim Berners-Lee, "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better-enabling computers and people to work in cooperation." you can see the hierarchy of semantic web stack in Figure 2.1.

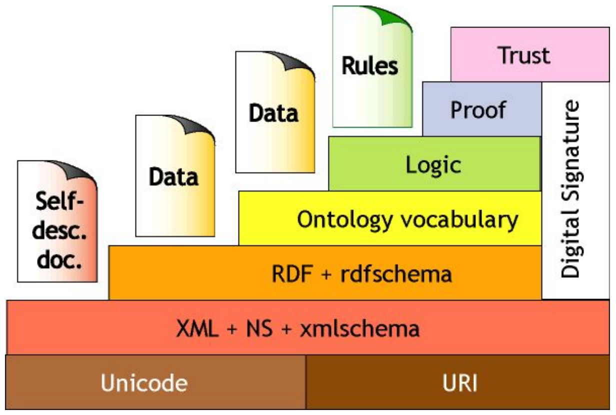


Figure 2.1: The Semantic Web "layer cake" as presented by Tim Berners-Lee. Figure is taken from [10].

The Semantic Web aims to enable machines and humans to use data and knowledge more effectively. *Semantic web technologies* provide the framework for humans to store data on the Web and create vocabularies, define metadata, relationships, and rules for managing data, and connect it to other data on the World of Web[5]. *W3C (World Wide Web Consortium)* has defined some standard technologies such as RDF, OWL, and SPARQL to create a web of linked data that is easily discoverable and machine-readable. We can define the vocabularies, metadata, and rules based on them.

[3]`https://www.w3.org/standards/`
[4]`https://www.w3.org/standards/semanticweb/data`
[5]`https://www.w3.org/standards/semanticweb/`

2.1.1 FAIR Principles

The FAIR principles - **F**indable, **A**ccessible, **I**nteroperable, and **R**eusable - have become a cornerstone of data management and data sharing in the semantic web. The primary goal of these principles is to enhance the value and usefulness of data for researchers, scientists, and the general public by improving its findability, accessibility, interoperability, and reusability [31]. **Findability)** Metadata and data should be easily discoverable by both humans and computers. **Accessibility)** Once data has been located, users should have clear instructions on how to access it, including the necessary authentication and authorization procedures. **Interoperability)** Data often needs to be integrated with other data and be able to interoperate with applications for storage, processing, and analysis. **Reusability)** Metadata and data should be easily replicated or combined in various contexts [29]. In the context of the semantic web, the FAIR principles are particularly important for data integration, as they enable different sources of data to be linked and combined in a meaningful way. This is achieved through the use of common vocabularies, ontologies, and data formats, which help to ensure that data is consistent and standardized across different sources.

To achieve the FAIR principles in the context of the semantic web, there are several best practices that can be followed. First, data should be published in a machine-readable format using open standards, such as RDF (Resource Description Framework) and OWL (Web Ontology Language). This makes it easier for machines to understand and process the data, and enables it to be linked and integrated with other datasets [31, 16].

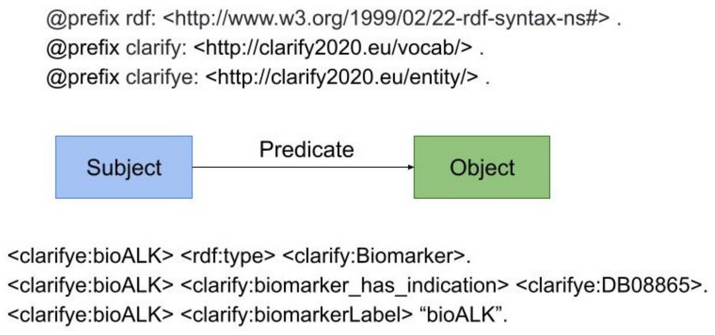
Second, data should be annotated with metadata that describes the content, structure, and provenance of the data. This metadata should be standardized and interoperable, so that it can be easily shared and reused by others [31, 16].

Third, data should be published with persistent identifiers (PIDs), such as DOIs or URIs, that enable the data to be easily located and cited. This helps to ensure that data is findable and accessible to others, and enables researchers to give credit to the data creators [31, 16].

Finally, data should be published with clear terms of use and licensing information that allows others to reuse the data in a responsible and legal way. By adopting these best practices, data managers and publishers can help to make data FAIR and contribute to the development of the semantic web and the broader scientific community [31, 16].

2.1.2 Resource Description Framework: RDF

RDF stands for *Resource Description Framework*. It is a W3C recommendation and an infrastructure for representing the information on the Web and expressing semantics. This framework reuses the structured metadata[22].



```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix clarify: <http://clarify2020.eu/vocab/> .
@prefix clarifye: <http://clarify2020.eu/entity/> .
```

```
<clarifye:bioALK> <rdf:type> <clarify:Biomarker>.
<clarifye:bioALK> <clarify:biomarker_has_indication> <clarifye:DB08865>.
<clarifye:bioALK> <clarify:biomarkerLabel> "bioALK".
```

Figure 2.2: An Example of a RDF triples

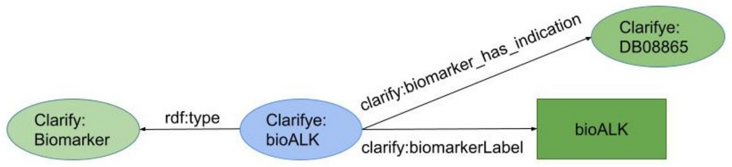


Figure 2.3: An Example of a RDF graph

RDF provides a way to describe resources on the web, such as web pages, books, or people, in a way that can be easily understood and processed by humans and also machines. It uses a simple, graph-based data model that consists of *subject-predicate-object* statements, or triples (s, p, o), similar to simple sentences. Each triple consists of a subject (the resource being described), a predicate (the property or characteristic being described), and an object (the value of the property) [22]. You can see a sample of RDF triple in Figure 2.2.
Graphs are a useful way to represent the structure of a network, and in the RDF framework, graphs are made up of sets of RDF triples. These triples consist of three types of elements: *URIs*, *literals*, and *blank nodes* [21].
The sample RDF graph in Figure 2.3 depicts rectangular nodes as attributes and literals, oval nodes as resources, and arcs as properties representing the relationships between resources and attributes.

2.2 Description Logics: DL

Description Logics (DLs) are a family of formal knowledge representation languages used to represent and reason about the concepts and relationships within a domain of interest. DLs are typically used in the field of artificial intelligence, particularly in areas such as knowledge-based systems, semantic web technologies, and ontology engineering. In DLs, the domain of interest is represented as a set of concepts and relationships between these concepts. Language provides a set of constructors to build complex ideas and relationships from simple ones. These constructors include intersection (AND), union (OR), negation (NOT), role restrictions, and cardinality restrictions. DLs provide a rigorous framework for reasoning about knowledge bases' consistency, satisfiability, and entailment. DL reasoning is usually performed using automated inference procedures that can derive new knowledge from existing knowledge bases. Description Logics usually divide domain knowledge into two parts, namely the **TBox** and the **ABox**. The TBox component represents the terminology or the structure of the domain, analogous to a database schema. On the other hand, the ABox represents knowledge about a specific situation or instance of the domain, similar to a database instance. The combination of the TBox and ABox creates a *knowledge base (KB)* [3]. In a hospital domain, TBox statements could capture knowledge about the structure of the domain, such as the hierarchy of the concepts within the domain. For instance, a TBox statement could specify that an "LCPatient" is a subclass of a "Patient" who has "Lung Cancer". On the other hand, ABox statements from the same hospital domain may include specific instances such as "Alex is a patient", "Lung Cancer is a disease", and "Alex has Lung Cancer". It is important to note that DLs provide formal and logic-based semantics for such statements, a crucial feature of these knowledge representation languages [3]. DLs are widely used in applications such as ontology engineering, natural language processing, and intelligent systems.

2.3 RDFS and OWL: Ontology

Ontologies or vocabularies in the domain of the Semantic Web define concepts and relationships that describe and represent a specific area of concern. Their main role is to facilitate data organization and integration, which results in the creation of integrated data or Linked Data that can be used for reasoning or querying. Ontologies are crucial in enabling automatic knowledge processing, sharing, and reuse among applications. They typically contain a hierarchy of *concepts* or *classes* and their *properties* that relate to a specific area of interest. There are two different

types of properties that are used to represent relationships between classes and their instances, *object property* and *data type property*. An object property represents a relationship between two or more instances of classes. A data type property, on the other hand, is used to represent a relationship between an instance of a class and a literal value of a particular data type. The examples are shown in Figure 2.4.
By utilizing ontologies, computers can comprehend the meaning of data and draw inferences based on that understanding, even if the data is presented in various formats or languages[25].
RDF and OWL are the two languages used to create ontologies. RDF vocabularies provide terms that describe resources, while the W3C *Web Ontology Language (OWL)* is a Semantic Web language designed to represent complex knowledge. OWL comprises RDFS, RDF vocabularies and new terms that describe resources in greater detail. RDF and OWL are the backbone of semantics and OWL makes data smart. In the Table 2.1 you see some examples of RDFS, OWL and DL.

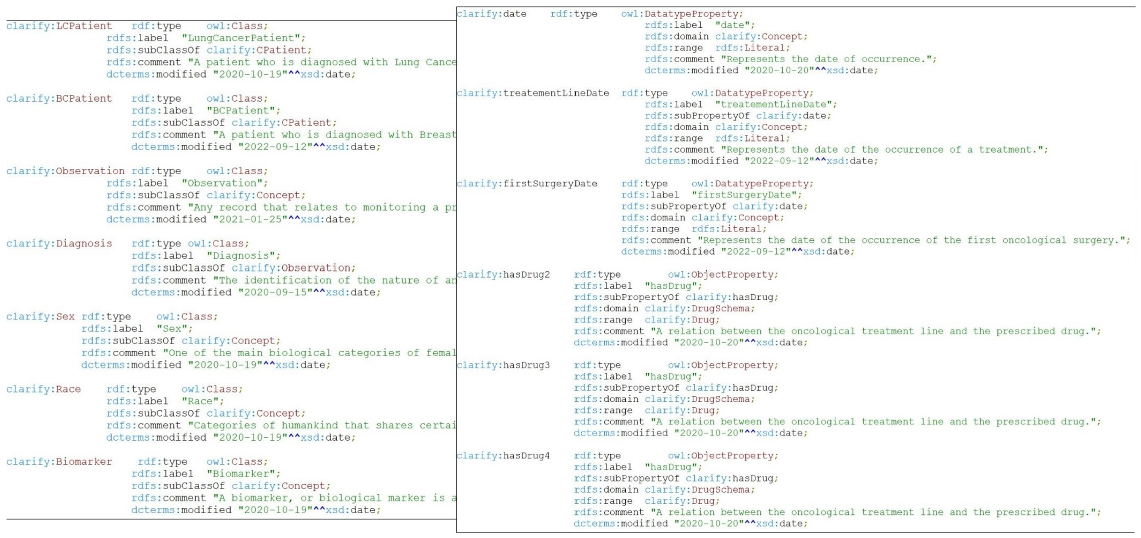
DL Operator	RDFS	OWL
$C1 \subseteq C2$	C1 rdfs:subClassOf C2	
$P1 \subseteq P2$	P1 rdfs:subPropertyOf P2	
$\exists R \subseteq A$	P rdfs:domain A	
$\exists R^{-} \subseteq B$	P rdfs:range B	
$A \equiv B$		A owl:equivalentClass B
$A \cap B \subseteq \bot$		A owl:disjointWith B
A(x)	x rdf:type A	
R(x,y)	x R y	
$x \equiv y$		x owl:sameAs y
$P1 \equiv P2$		P1 owl:equivalentProperty P2

Table 2.1: Some examples of DL, RDFS and OWL. The table is taken from [30]

The Figure 2.5 displays a visualization of a sample ontology.

2.3.1 Axioms

RDFS and OWL provide a set of constructs to define classes, properties, and their relationships. Axioms state the semantics of these constructs from RDFS and OWL; they infer implicit facts in knowledge graphs. For example, in RDFS, axioms establish the meaning of the subclass relationship between classes, specify the domain and



```
clarify:LCPatient   rdf:type    owl:Class;
              rdfs:label  "LungCancerPatient";
              rdfs:subClassOf clarify:CPatient;
              rdfs:comment "A patient who is diagnosed with Lung Cance
              dcterms:modified "2020-10-19"^^xsd:date;

clarify:BCPatient   rdf:type    owl:Class;
                  rdfs:label  "BCPatient";
                  rdfs:subClassOf clarify:CPatient;
                  rdfs:comment "A patient who is diagnosed with Breast
                  dcterms:modified "2022-09-12"^^xsd:date;

clarify:Observation rdf:type    owl:Class;
              rdfs:label  "Observation";
              rdfs:subClassOf clarify:Concept;
              rdfs:comment "Any record that relates to monitoring a pr
              dcterms:modified "2021-01-25"^^xsd:date;

clarify:Diagnosis   rdf:type owl:Class;
                  rdfs:label  "Diagnosis";
                  rdfs:subClassOf clarify:Observation;
                  rdfs:comment "The identification of the nature of an
                  dcterms:modified "2020-09-15"^^xsd:date;

clarify:Sex rdf:type    owl:Class;
          rdfs:label  "Sex";
          rdfs:subClassOf clarify:Concept;
          rdfs:comment "One of the main biological categories of femal
          dcterms:modified "2020-10-19"^^xsd:date;

clarify:Race    rdf:type    owl:Class;
              rdfs:label  "Race";
              rdfs:subClassOf clarify:Concept;
              rdfs:comment "Categories of humankind that shares certai
              dcterms:modified "2020-10-19"^^xsd:date;

clarify:Biomarker    rdf:type    owl:Class;
                  rdfs:label  "Biomarker";
                  rdfs:subClassOf clarify:Concept;
                  rdfs:comment "A biomarker, or biological marker is a
                  dcterms:modified "2020-10-19"^^xsd:date;
```

```
clarify:date    rdf:type    owl:DatatypeProperty;
                    rdfs:label  "date";
                    rdfs:domain clarify:Concept;
                    rdfs:range  rdfs:Literal;
                    rdfs:comment "Represents the date of occurrence.";
                    dcterms:modified "2020-10-20"^^xsd:date;

clarify:treatementLineDate  rdf:type    owl:DatatypeProperty;
                    rdfs:label  "treatementLineDate";
                    rdfs:subPropertyOf clarify:date;
                    rdfs:domain clarify:Concept;
                    rdfs:range  rdfs:Literal;
                    rdfs:comment "Represents the date of the occurrence of a treatment.";
                    dcterms:modified "2022-09-12"^^xsd:date;

clarify:firstSurgeryDate    rdf:type    owl:DatatypeProperty;
                  rdfs:label  "firstSurgeryDate";
                  rdfs:subPropertyOf clarify:date;
                  rdfs:domain clarify:Concept;
                  rdfs:range  rdfs:Literal;
                  rdfs:comment "Represents the date of the occurrence of the first oncological surgery.";
                  dcterms:modified "2022-09-12"^^xsd:date;

clarify:hasDrug2    rdf:type        owl:ObjectProperty;
              rdfs:label  "hasDrug";
              rdfs:subPropertyOf clarify:hasDrug;
              rdfs:domain clarify:DrugSchema;
              rdfs:range  clarify:Drug;
              rdfs:comment "A relation between the oncological treatment line and the prescribed drug.";
              dcterms:modified "2020-10-20"^^xsd:date;

clarify:hasDrug3    rdf:type        owl:ObjectProperty;
              rdfs:label  "hasDrug";
              rdfs:subPropertyOf clarify:hasDrug;
              rdfs:domain clarify:DrugSchema;
              rdfs:range  clarify:Drug;
              rdfs:comment "A relation between the oncological treatment line and the prescribed drug.";
              dcterms:modified "2020-10-20"^^xsd:date;

clarify:hasDrug4    rdf:type        owl:ObjectProperty;
              rdfs:label  "hasDrug";
              rdfs:subPropertyOf clarify:hasDrug;
              rdfs:domain clarify:DrugSchema;
              rdfs:range  clarify:Drug;
              rdfs:comment "A relation between the oncological treatment line and the prescribed drug.";
              dcterms:modified "2020-10-20"^^xsd:date;
```

Figure 2.4: A sample of classes, object properties and data type properties in clarify ontology `https://github.com/SDM-TIB/CLARIFYUnifiedSchema/blob/master/clarify_v8.ttl`

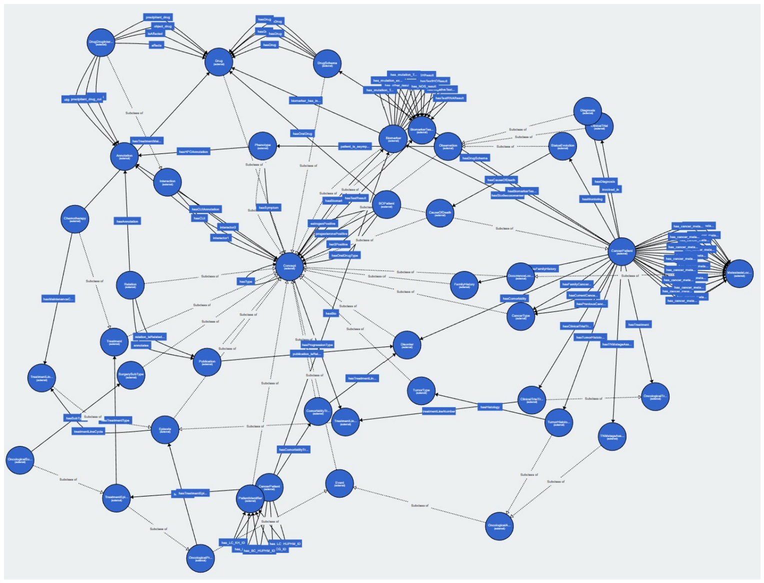


Figure 2.5: The visualization of clarify ontology `https://github.com/SDM-TIB/CLARIFYUnifiedSchema/blob/master/clarify_v8.ttl`

range of properties, and define the cardinality constraints on the properties of classes. Using axioms in the Semantic Web is essential for enabling automated reasoning and inference over knowledge bases. By formally defining the meaning of concepts and relationships, machines can process and integrate information from different sources consistently and accurately [9, 2]. The following sections describe these axioms.

Subclass

This section describes the axioms that define subclasses; the variable e is used to represent either URIs or blank nodes. C, $C1$, $C2$, and $C3$ are variables used to represent RDFS classes [30].

- **Reflexivity**: **IF** (C, rdf:type, **rdfs:Class**) **THEN** (C, **rdfs:subClassOf**, C)
- **Transitivity**:

  IF ($C1$,**rdfs:subClassOf**, $C2$) **AND** ($C2$, **rdfs:subClassOf**, $C3$) **THEN** ($C1$, **rdfs:subClassOf**, $C3$)
- **IF** ($C1$, **rdfs:subClassOf**, $C2$) **AND** (e, rdf:type, $C1$) **THEN** (e, rdf:type, $C2$)

Properties

This section describes the axioms that define subproperties; the variable a is used to represent either URIs or blank nodes, while b is used to represent URIs, blank nodes, or literals. P, $P1$ and $P2$ are used to represent an RDFS property, which can be either an object property or a datatype property. $C1$ and $C2$ are used to represent RDFS classes. Finally, u and v are instances of the classes. [30].

- **IF** (a, P, b) **THEN** (P, rdf:type, **rdfs:Property**)
- **IF** (a, P, b) **AND** (a, rdf:type, $C1$) **THEN** (P, rdfs:domain, $C1$)
- **IF** (a, P, b) **AND** (a, rdf:type, $C2$) **THEN** (P, rdfs:range, $C2$)
- **Inverse Property**: **IF** ($P1$, **owl:inverseOf**, $P2$) **AND** $(u, P1, y)$ **THEN** $(y, P2, u)$
- **Inverse Property**: **IF** ($P1$, **owl:inverseOf**, $P2$) **AND** $(u, P2, y)$ **THEN** $(y, P1, u)$

Cardinalities

Axiom about cardinalities enable to state the different sizes of sets that can be constructed using axioms, and they are represented by cardinal numbers.

- **Max Cardinality Zero**:

  IF (C1, owl:maxCardinality, "0"^^xsd:nonNegativeInteger) **AND** (C1, owl:onProperty, P) **AND** (u, rdf:type, C1) **AND** (u, P, y)

  THEN False

- **Max Cardinality One**:

  IF (C1, owl:maxCardinality, "1"^^xsd:nonNegativeInteger) **AND** (C1, owl:onProperty, P) **AND** (u, rdf:type, C1) **AND** (u, P, y1) **AND** (u, P, y2)

  THEN (y1, owl:sameAs, y2)

2.3.2 Complex Classes

In ontologies, complex classes are defined as other classes or constructs, such as other classes, properties, and logical expressions. These complex classes provide a more expressive way of describing the structure and relationships between resources in a knowledge base.
For example, in the Resource Description Framework Schema (RDFS) and the Web Ontology Language (OWL), complex classes can be defined using constructs such as *intersection, union*, and *complement.*
An intersection class is defined as the set of resources that are instances of all the classes that are part of the intersection.

- ex:$C0$ owl:equivalentClass [owl:**intersectionOf** (ex:$C1$ ex:$C2$ ... ex:Cn)].
- ex:LCPatient owl:equivalentClass [owl:**intersectionOf** (ex:Patient ex:LungCancer)].

A union class is defined as the set of resources that are instances of at least one of the classes that are part of the union.

- ex:$C0$ owl:equivalentClass [owl:**unionOf** ($ex : C1$ ex:$C2$ ... ex:Cn)].
- ex:Patient owl:equivalentClass [owl:**unionOf** (ex:LCPatient ex:BCPatient)].

The Patient class can be also represented by cardinality:

- ex:Patient owl:equivalentClass [owl:intersectionOf (ex:Human [owl:minCardinality "1" ^xsd:nonNegativeInteger ; owl:onProperty ex:hasDisease])]

A complement class is defined as the set of resources that are not instances of the class that is being complemented [30].

- ex:$C0$ owl:equivalentClass [owl:**complementOf** $(ex : C1)$].
- ex:Patient owl:equivalentClass [owl:**complementOf** (ex:healedPerson)].

2.4 Data Integration System: DIS

Information integration and interoperability have been significant challenges in computer information processing since its early days. Initially, attention was focused on a relatively small number of sources. However, over the past few decades, the scope of integration and interoperability has expanded, and much effort has been devoted to addressing these related problems [19]. A data integration system in the Semantic Web enables data integration from multiple sources. The data may be structured or unstructured and may come from a variety of different formats and languages. The purpose of a data integration system is to facilitate the exchange and interoperability of data across multiple systems and to enable reasoning and decision-making based on integrated data. Data integration systems often use ontologies and mappings to align and harmonize data from different sources. A data integration system typically involves the following components: $DIS= <O, S, M>$ where O represents a unified schema consisting of classes, relationships, and properties, S represents a set of data sources, M is a set of mapping rules.
Since the knowledge graph is based on an ontology, we need to define an ontology to describe *data integration system.* The ontology of data integration is mainly created by combining standard ontology vocabularies such as OWL, RDFS for representing the unified schema, DCAT for defining a set of data sources, and RML, and R2RML for defining mapping rules. Figure 2.6 shows that the data integration system ontology imports other ontologies and also defines SHACL constraints [8].

2.5 Data Catalog Vocabulary: DCAT

Data sources are a critical and challenging part of a data integration system. The term *big data* is defined by three main attributes, commonly referred to as the 3Vs: ***v**olume*, ***v**ariety*, and ***v**elocity.* Volume refers to the challenge of managing massive

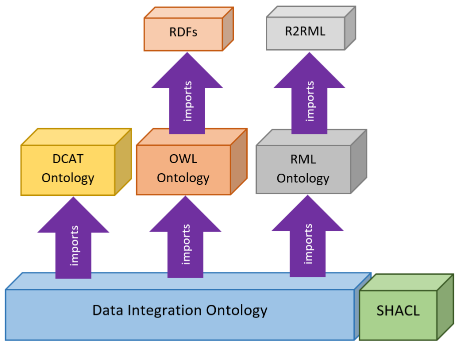


Figure 2.6: Data Integration System Ontology

amounts of data. Velocity refers to the challenge of managing high-speed, continuous data streams. Variety refers to managing diverse data formats and multiple data sources. As a result, addressing the problems associated with the combination of big data is critical for solving many real-world challenges [1].
Data sources can be of different types, formats, and structures and use different terminologies. They can contain errors, duplicates, and missing values. Data sources can change over time, making it challenging to keep the integrated data up-to-date [24]. Linked data datasets constantly evolve, with resources being added or removed [28]. The concept of *freshness* is often discussed in the context of these changes, and it is considered a key factor in determining the quality of data [5].
For describing data sets and developing the Semantic Web by enabling data to be linked, shared and reused, publishers can use the standard model *DCAT (Data Catalog Vocabulary)*. DCAT is a standardized vocabulary that describes datasets and data catalogs in the Semantic Web. It is based on the RDF (Resource Description Framework) data model, which enables data to be linked and shared across the web [6]. You can share data services, distribution, publisher, and many more details about the data sets through *DCAT* on the Web. DCAT improves Interoperability by describing datasets and enhancing the data quality. It allows for creating rich metadata descriptions for datasets, including information about the publisher, license, language, and more.
It provides aggregation of datasets from different sources and makes them available. This increases the findability of data sources. The prefix for the DCAT namespace is *dcat*. The main classes of DCAT version 3 used in this work are *dcat:DatasetSeries, dcat:Dataset* and *dcat:Distribution*. You can see the overview of the DCAT model in

[6]`https://www.w3.org/TR/vocab-dcat-3/`

Figure 2.7.

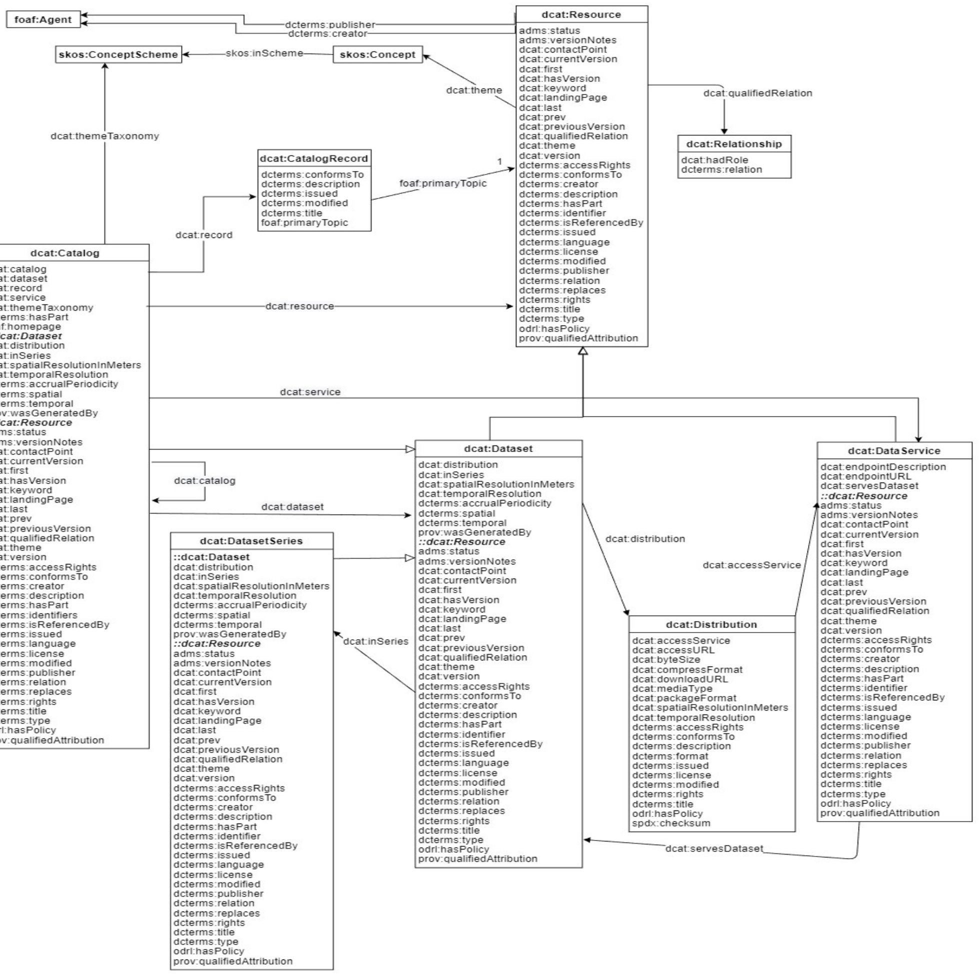


Figure 2.7: Overview of DCAT model,showing the classes of resources that can be members of a Catalog, and the relationships between them. This figure is taken from `https://www.w3.org/TR/vocab-dcat-3/`

2.6 R2RML and RML as Declerative Mapping Languages

A declarative mapping language is a programming language used to define mappings between different data sources and target data structures, such as databases, XML documents, and objects. These languages allow developers to describe mappings concisely and intuitively without writing low-level code, using rules and patterns that automatically generate the required code. A mapping document is a necessary tool for translating data, and mapping languages help establish the relationships between heterogeneous data models in different sources and an RDF version that follows the schema of an ontology. Mapping languages are often used in data integration systems to define the rules for translating non-RDF data into RDF. These data sources can be expressed in various formats, including tabular, JSON, or XML. Mapping languages for KG construction have a common goal of establishing the relationships between data sources and the ontology schema and share inherent characteristics that can be modeled [15]. Figure 2.8 illustrates declarative mapping languages. This work focuses on two specific mapping languages, R2RML [7] and RML[8].

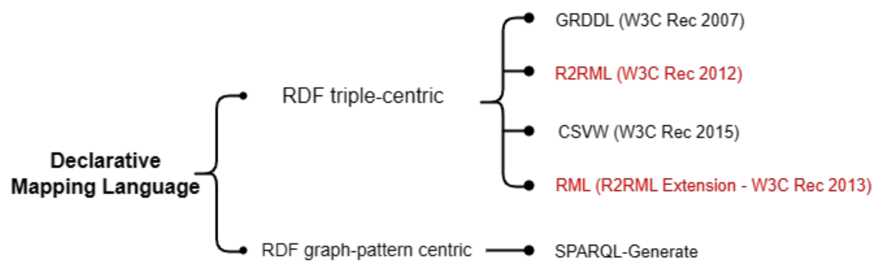


Figure 2.8: Examples of declarative mapping languages. It is taken from [30]

R2RML

R2RML stands for RDB to RDF Mapping Language, a W3C recommendation for mapping relational databases (RDB) to RDF graphs. The idea behind R2RML is to provide a standard way to expose relational data as linked data on the web, which helps integrate and share data across different applications and domains [9]. R2RML works by defining a set of mapping rules (**TriplesMaps**) that specify how to transform the data in a relational database into RDF triples (**LogicalTable**). The mapping rules define the mapping between the database schema and the corresponding RDF graph, including classes, properties, and relationships. A set of triples maps includes a logical table, subject map, and zero or more predicate-object maps[30].

[7]https://www.w3.org/TR/r2rml/
[8]https://rml.io/specs/rml/
[9]https://www.w3.org/TR/r2rml/

- **SubjectMap** *(rr:subjectMap)* identifies the subject of the RDF triples that will be generated, and the *(rr:class)* predicate specifies the subject's type. The *(rr:template)* predicate defines the pattern used to create the subject's identifier.

- **PredicateObjectMap** *(rr:predicateObjectMap)* associates a predicate and object with a subject described using a subject map.

  - **PredicateMap** *(rr:predicate)* defines the predicate of the generated RDF triple.

  - **ObjectMap** *(rr:objectMap)* specifies the object of the triple. The *(rr:template)* predicate defines the pattern for creating the identifier of the object when the predicate corresponds to an object property. Alternatively, when the predicate is a data type property, the *(rr:column)* predicate must be used, followed by the name of the corresponding column. Object maps can also be created using referencing object maps.
    A **referencing object map** indicates that the subject of another triples map corresponds to the value of the object of the defined RDF triple. To define a referencing object map, a join between the logical tables of the two related triple maps may be necessary. The referencing object map includes one *(rr:parentTriplesMap)* property, which must be a triples map and is referred to as the parent triple map of the referencing object map. The referencing object map may also include one or more *(rr:joinCondition)* properties. The values of these properties must be join conditions of the form *(rr:child)* "ChildAttributeName" and *(rr:parent)* "ParentAttribute-Name". These values correspond to the name of the attribute in the triples map that is making the reference and the name of the attribute in the triples map that is referenced, respectively. The join condition is required when the child and parent triples maps are defined over two different tables or SQL queries [30]. The Figure 2.9 is a sample of R2RML.

The Figure 2.10 shows the overview of R2RML.

RML

The *RDF mapping language (RML)* is an extension based on R2RML. RML is a mapping language that expresses the mapping rules from heterogeneous data structures such as CSV, TSV, XML, and JSON data sources and relational databases to

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix schema: <http://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix clarify: <http://clarify2020.eu/vocab/> .

<Biomarker_drug_biomarker>
    rr:logicalTable [ rr:sqlQuery """SELECT DISTINCT biomarker FROM biomarkers """
                    ];
    rr:subjectMap [
        rr:template "http://clarify2020.eu/entity/{biomarker}";
        rr:class clarify:Biomarker
    ];
    rr:predicateObjectMap [
        rr:predicate clarify:biomarkerLabel;
        rr:objectMap [
            rr:column "biomarker"
        ]
    ].
```

Figure 2.9: A sample of R2RML mapping rule.

RDF data models and ontologies [10]. The input of RML is any data source, and the output is an RDF dataset. A triples map sets rules for generating zero or more RDF

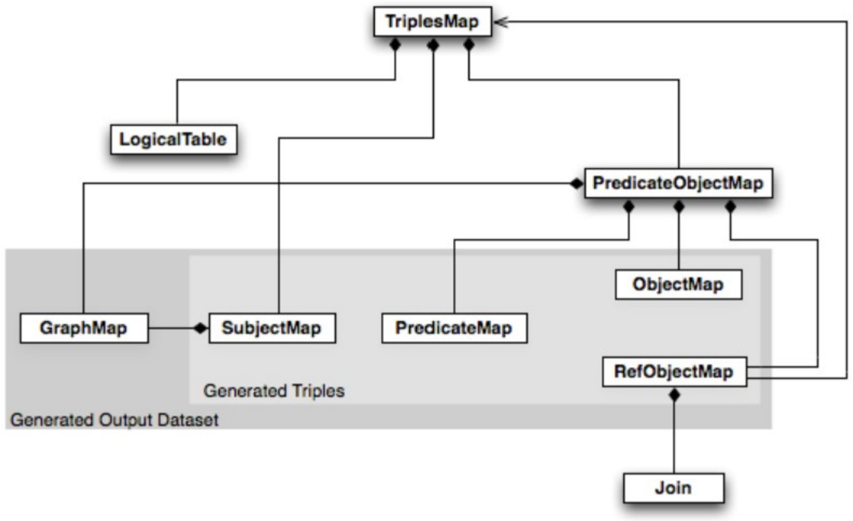


Figure 2.10: An overview of R2RML. Figure taken from `https://www.w3.org/TR/r2rml/`

[10]`https://rml.io/specs/rml/`

triples with the same subject. The triples map consists of a logical source, a subject map, and a mandatory and predicate object map, which is optional in the mapping. The structure of RML is shown in Figure 2.11. The **logical source** *rml:logicalSource*

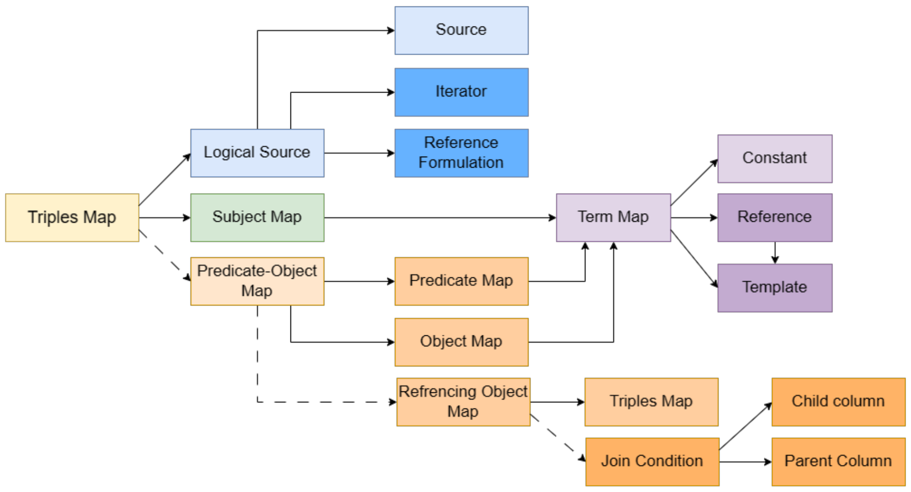
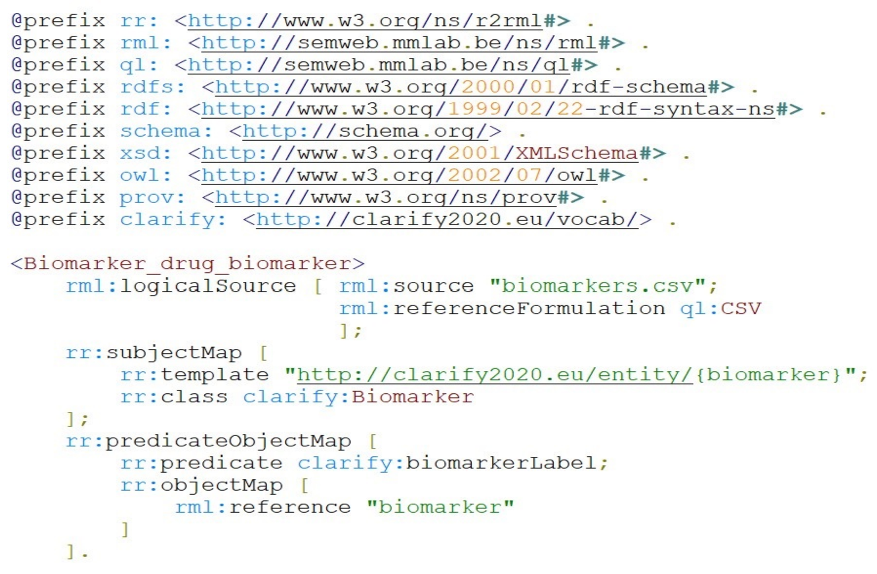


Figure 2.11: An overviwe of RML.

in RML is any data source that is mapped to RDF triples. RML supports any reference to any data input within its logical source. A logical source has exactly one *(rml:source)* specifying where the data source is located, exactly one *(rml:iterator)* defining the iteration loop over data, and null or one valid identifier to specify how to refer to the data as *(rml:referenceFormulation)* used to refer to the elements of the data source. The reference formulation is mandatory in the case of databases and XML and JSON data sources. The reference formulations are not limited but predefined such as ql:CSV, ql:XML, ql:JSONPath, rr:SQL2008 and ql:XPath.
The **subject map** *(rr:subjectMap)* is a term map in R2RML with URI pattern that specifies the rule for generating the subject of the RDF triples map. It can optionally define the type of the subject of the triple. It may consist of one or more class IRIs.

The **predicate Object Map** *(rr:predicateObjectMap)* is a term map in R2RML that consist of *predicate and object maps.* It creates values and predicates. Each predicate object map can have one or more predicate maps (rr:predicate) that specifies how the triple's predicate is generated. An Object Map (rr:objectMap) specifies how the triple's object are generated. In a referencing object map *(rr:parentTriplesMap)* is allowed to use the subjects of another triples map as the object that is generated by a current predicate-object map. When both triples maps are based on different logical sources, this requires a join *(rr:joinCondition)* between the logical sources.



```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix rml: <http://semweb.mmlab.be/ns/rml#> .
@prefix ql: <http://semweb.mmlab.be/ns/ql#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix schema: <http://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix clarify: <http://clarify2020.eu/vocab/> .

<Biomarker_drug_biomarker>
    rml:logicalSource [ rml:source "biomarkers.csv";
                        rml:referenceFormulation ql:CSV
                        ];
    rr:subjectMap [
        rr:template "http://clarify2020.eu/entity/{biomarker}";
        rr:class clarify:Biomarker
    ];
    rr:predicateObjectMap [
        rr:predicate clarify:biomarkerLabel;
        rr:objectMap [
            rml:reference "biomarker"
        ]
    ].
```

Figure 2.12: A sample of RML mapping rule.

Each join condition has exactly one value for the parent property and exactly one value for the child property. No join condition is required if both triples maps use the same logical source [11]. The Table2.2 shows the comparison of the R2RML and RML.

2.7 Mapping Assertions

In the context of the Semantic Web technologies, a mapping assertion is a statement that specifies how a set of data from a particular data source is mapped to RDF triples. Mapping assertions or rules describe how existing data can be represented using the RDF data model. The meaning of data is derived from its relationships to other data [21]. The mapping assertions in mappings are executed to create the instances of the unified schema. Mapping rules are formalized as clauses *body*($\overline{X}$) and *head*($\overline{Y}$)[14].

$$body(\overline{X}) : -head(\overline{Y})$$

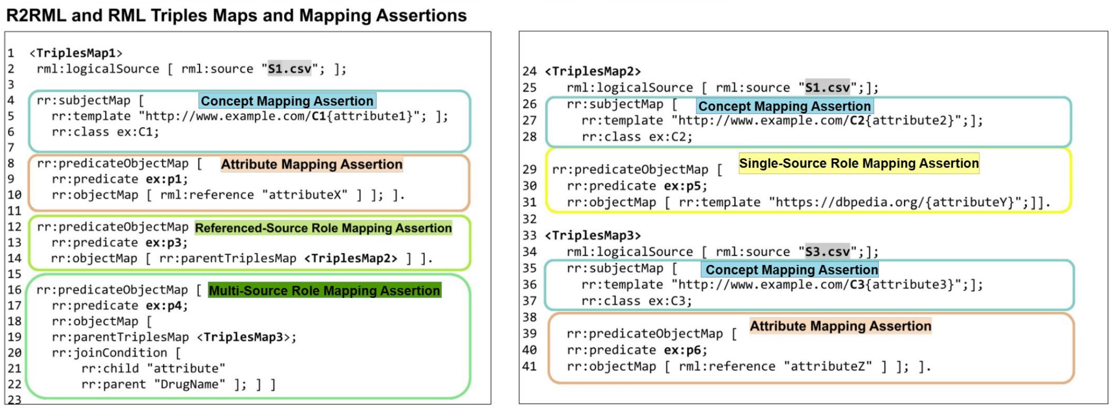
body($\overline{X}$) predicates over data sources and *head*($\overline{Y}$) predicates classes and properties (datatype properties and object properties) in ontology over set of *terms*. The type

[11]`https://rml.io/specs/rml/`

R2RML	RML
Logical Table - only relational database **(rr:logicalTable)**	Logical Source - CSV, XML, JSON, HTML **(rr:logicalSource)**
Table Name **(rr:tableName)**	URI pointing to the source**(rml:source)**
Relational Table Column **(rr:column)**	Reference **(rml:reference)**
SQL query **(rr:sqlQuery)**	Reference Formulation **(rml:referenceFormulation)**
Iteration per row in table	Definition of an iterator over JSON and XML **(rml:iterator)**

Table 2.2: Comparison of R2RML and RML. The table is from [30]

of Mapping Assertion determines the type of Head, which can be a *class*, *object property* or *dataType proprty*. The Body, in this case, is assumed to consist of only one source, without losing generality [13].
There are three type of mapping assertions: *Concept Mapping assertions*, that predicate classes, *Role Mapping Assertions*, that predicate object properties and *Attribute Mapping Assertions*, that predicate datatype properties. In the following, we define the different mapping assertions with the examples in Figure 2.13.


R2RML and RML Triples Maps and Mapping Assertions
Concept Mapping Assertion
Attribute Mapping Assertion
Referenced-Source Role Mapping Assertion
Multi-Source Role Mapping Assertion
Concept Mapping Assertion
Single-Source Role Mapping Assertion
Concept Mapping Assertion
Attribute Mapping Assertion

```
<TriplesMap1>
 rml:logicalSource [ rml:source "S1.csv"; ];

 rr:subjectMap [
   rr:template "http://www.example.com/C1{attribute1}"; ];
   rr:class ex:C1;

 rr:predicateObjectMap [
   rr:predicate ex:p1;
   rr:objectMap [ rml:reference "attributeX" ] ]; ].

 rr:predicateObjectMap
   rr:predicate ex:p3;
   rr:objectMap [ rr:parentTriplesMap <TriplesMap2> ] ].

 rr:predicateObjectMap [
   rr:predicate ex:p4;
   rr:objectMap [
   rr:parentTriplesMap <TriplesMap3>;
   rr:joinCondition [
       rr:child "attribute"
       rr:parent "DrugName" ]; ] ]

```

```
<TriplesMap2>
   rml:logicalSource [ rml:source "S1.csv";];
   rr:subjectMap [
     rr:template "http://www.example.com/C2{attribute2}";];
     rr:class ex:C2;

 rr:predicateObjectMap [
   rr:predicate ex:p5;
   rr:objectMap [ rr:template "https://dbpedia.org/{attributeY}";]].

<TriplesMap3>
   rml:logicalSource [ rml:source "S3.csv";];
   rr:subjectMap [
     rr:template "http://www.example.com/C3{attribute3}";];
     rr:class ex:C3;

   rr:predicateObjectMap [
   rr:predicate ex:p6;
   rr:objectMap [ rml:reference "attributeZ" ] ]; ].
```

Figure 2.13: Mapping Assertions that are expressed in R2RML and RML. Figure taken from [13]

2.7.1 Concept Mapping Assertions

It uses predicate class $C(.)$ to create the instance of a classes C in the ontology O. $S(\overline{X})$ is based on the conjunctive query over the data sources. So the form of the concept mapping assertions is [13]:

$$body(\overline{X}) : -C(f(y))$$

$$S_i(\overline{X}) : -C(f(y))$$

In Figure 2.13 you can see examples of concept mapping assertion in blue color which corresponds to a `rr:subjectMap`. It defines classes $C1$, $C2$ and $C3$ according to attributes *attribute*1 and *attribute*2 in data source $S1$ and *attribute*3 in $S3$. The data sources are represented in `rml:logicalSource`. The function f(.) enables string concatenation, and is represented in RDF syntax using the predicate `rr:template`.

2.7.2 Role Mapping Assertions

A role mapping assertion defines an object property P in the ontology O is terms of the conjunction of predicates presenting data sources:

$$S(\overline{X}) : -P(f_1(y_1), f_2(y_2))$$

There are three types of role mapping assertions: *single-role mapping assertion*, *referenced-source role mapping assertion*, and *multi-source role mapping assertion*. The single-role mapping assertion is used to map a single role. The referenced-source role mapping assertion is connected to another mapping assertion with the same data source. Lastly, the multi-sources role mapping assertion connects mapping assertions with different data sources.

Single-Role Mapping Assertions

Single-role mapping assertion uses the predicate class $P(.,.)$ over data attributes in only one data source. The Figure 2.13 shows the example of single-source role mapping assertion in yellow color. The figure shows that the TriplesMap2 specifies the ex:p5 property as a single-source role assertion. The `rr:predicateObjectMap` rule sets the object value of ex:p5 using the `rr:objectMap`. It uses f_1 and f_2 as functions, which are represented here by `rr:template`. Figure 2.13 shows the definition of a predicate $P(.,.)$, that is shown in R2RML terminology by `rr:predicate`. This predicate is based on *attribute*2 from data source $S1$, which is the value of the instance of class $C2$ processed by function `rr:template` as f_1, and the object value based on *attributeY* in the same data source $S1$ processed by `rr:template` as f_2 [13].

$$S_i(\overline{X}) : -P(f_1(y_1), f_2(y_2))$$

Referenced-Source Role Mapping Assertions

In referenced-source role mapping assertion, the role $P(.,.)$ is used to indicate the object value with respect to a source S_i, which, in turn, defines the subject of a given concept mapping assertion MA. Referenced-source role mapping assertions and single-role mapping assertions both predicate object properties in ontology. The difference is that referenced-source role mapping assertions also predicate object over y_2 and function f_2, which is used in another mapping assertion.

$$S_i(\overline{X}_{i,1}), S_i^{MA}(\overline{X}_{i,2}) : -P(f_1(y_1), f_2(y_2))$$

$$MA : S_i(\overline{X}_{i,2}) : -C_j(f_2(y_2))$$

You can see an example of referenced-source role mapping assertion in light green color in figure 2.13.
According to R2RML terminology, this assertion corresponds to a `rr:RefObjectMap`, and it refers to a mapping assertion MA using the predicate `rr:parentTriplesMap`. Both mapping assertions are defined over the same logical source, S_i. For example, in Figure 2.13, TriplesMap1 defines ex:p3 as the subject of TriplesMap2, both of which are defined over the same logical source, $S1$ [13].

Multi-Source Role Mapping Assertions

The multi-source role mapping assertion defines a property $P(.,.)$ between entities of two classes in the ontology; it differs from the referenced-source role mapping assertion in that it involves two different data sources. The mapping assertion is expressed using the following rule:

$$S_i(\overline{X}_{i,1}), S_j^{MJ}(\overline{X}_{i,2}), \theta(\overline{X}_{i,1}, \overline{X}_{i,2}) : -P(f_1(y_1), f_2(y_2))$$

$$MJ : S_j(\overline{X}_{i,2}) : -C_k(f_2(y_2))$$

The multi-source role mapping assertion enables the definition of a role $P(.,.)$ where the subject and object are defined over different sources, S_i and S_j respectively. Another assertion MJ utilizes source S_j to define the instances of class C_k. Since the sources are different, a join condition $\theta(\overline{X}_{i,1}, \overline{X}_{i,2})$ is necessary to connect mappings across data sources S_i and S_j based on their common data attributes. Figure 2.13 provides an example of a multi-source role mapping assertion in dark green, where role $ex : p4$ predicates the instance of class $C1$ over $attribute1$ in data source $S1$ and the instance of class $C3$ over $attribute3$ from data source $S3$. This assertion in R2RML terminology, is represented by a `rr:RefObjectMap` that includes a `rr:joinCondition`. The join condition θ connects different mappings across data sources S_i and S_j using attributes from both data sources, in this example connects TriplesMap2 and TriplesMap3 based on *attribute* from data source $S1$ and $DrugName$ from data source $S3$ [13].

2.7.3 Attribute Mapping Assertion

An attribute mapping assertion $A(.,.)$ defines the attribute of the class in the ontology over the value of the data attributes in the data source $S(\overline{X})$. The data type property $A(.,.)$ is defined using a function and a literal. This is represented by the clause following rule:

$$S(\overline{X}) : -A(f_1(y_1), y_2)$$

It retrieves the object value of the attribute A using a variable y_2 from the list of variables X. The objectMap inside a *rr:predicateObjectMap* defines the object value as either *rml:reference* or *rr:column* in R2RML terminology. In Figure 2.13, the examples in orange show the attribute mapping assertions, where two attribute mapping assertions specify the attributes $p1$ and $p6$ in TriplesMap1 and TriplesMap3, respectively.

Partitioning of Mapping Assertion

A data integration system can plan the execution of their mapping assertion based on groups of mapping assertions. There are two types of partitions: **Intra-source** and **Inter-source**. An **Intra-source** partition refers to a set of mapping assertions that exclusively utilize a single source, S_k, including concept, attribute, single-source role and referenced-source role mapping assertions. On the other hand, **Inter-source** partitions group mapping assertions that connect two sources, S_i and S_j, through multi-source role mapping assertion. For instance, in the Figures2.13,2.14 depicted, that Partition2 is an **Inter-source** partition that comprises both the multi-source mapping for p_4 and the concept mapping that defines class C_3 [13].

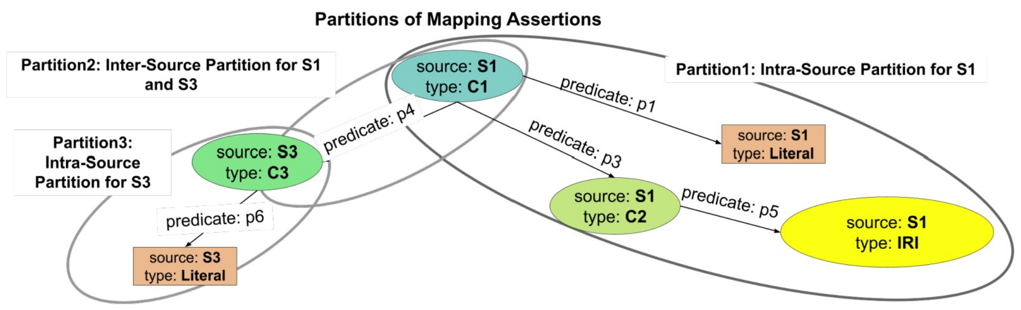


Figure 2.14: Partition of Mapping Assertions. Figure taken from [13]

2.8 SHACL Constraints

OWL is used to build ontologies, to create the metadata and the definition of concepts, and to explain how the concepts are all connected in their specific domain. OWL technology is used to inference data. However, to reuse the data for different requirements, some constraints need to be defined and the data should be valid for different use cases. Constraints support the quality of data after integration and prevent the knowledge graph from accessing bad data. *SHACL (SHAPES and Constraints Language)* is a W3C standard for defining constraints over data. SHACL is based on RDF and is designed for validation. The core elements of SHACL are constraints, target, shape and filter. With SHACL we have different views on the data and actually import different shapes for reusability, the important point in data integration. OWL restrictions are used to infer rules in the open world of the semantic web, but some restrictions are necessary to explicitly indicate limitations on classes and properties. These restrictions serve as validations, providing information on which classes and properties are limited and why.
SHACL is that validator. SHACL (Shapes Constraint Language) is a W3C standard for validating RDF data, specifically designed to be used with linked data. It is a language that enables users to specify constraints on RDF data, such as maximum and minimum string lengths, allowed values, and relationships between resources. SHACL provides a way to define the structure of RDF data, including the types of nodes and properties, their cardinality, and their relationships with other nodes. This helps to ensure that the data in a linked data environment is consistent, accurate, and usable for a range of applications. SHACL can be used in conjunction with other linked data technologies, such as RDF, SPARQL, and OWL, to support the creation of scalable and interoperable linked data systems. The principles constructions of

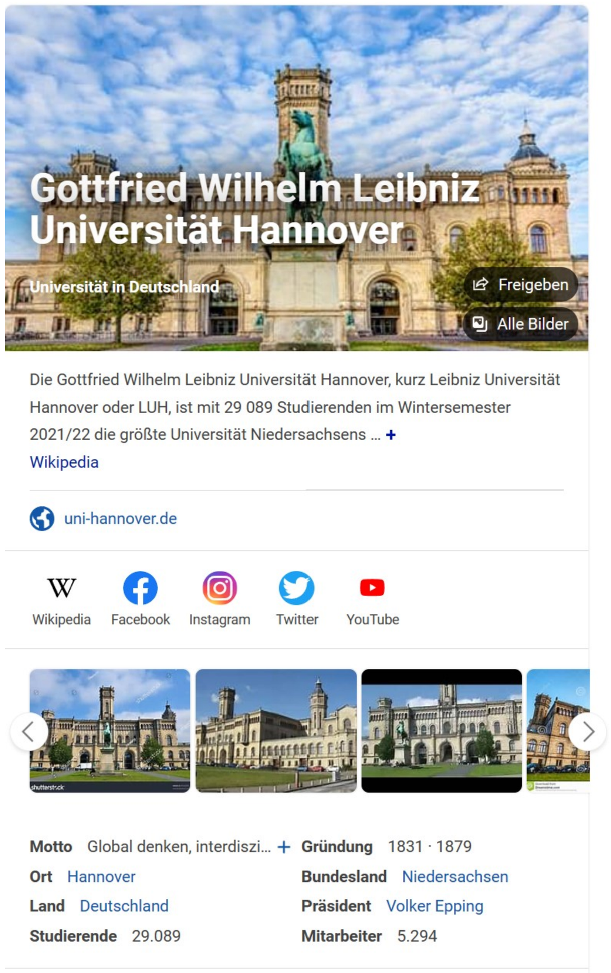
SHACL are described as follows [30]:

- **Shapes:** A shape is a standard way to represent integrity constraints that define the structure of RDF data.
- **Constraints:** A constraint is a rule that limits the values of a node in an RDF graph.
- **Targets:** A target node represent the group of RDF nodes where the constraints are applied.
  - **Classes** are used to define constraints on nodes that belong to a specific RDF class.
  - **Property Classes** are used to define constraints on properties of a specific RDF class.
  - **Node Kinds** define the type of node that the constraint applies to, such as a blank node or a literal.
  - **Datatypes** are used to specify the data type of literal nodes, such as integers or dates.
- **Filters:** they reduce the scope of the constrained nodes. [12].

2.9 Knowledge Graph

Knowledge graph technology can serve as a valuable tool to facilitate the connections between various data points. The term knowledge graph gained popularity following Google's 2012 announcement of its new capabilities for managing data on a global scale. This marked a departure from Google's earlier role as a global index of documents, as search results began to include relevant knowledge cards for almost any concept [21]. Figure 2.15 illustrates a knowledge card for Leibniz University Hanover (LUH).
A knowledge graph (KG) is a structured knowledge representation that captures information about the world and the relationships between various concepts or entities. Typically, it takes the form of a graph-based data model, where graphs are a universal data representation that depicts a networked structure [21].

[12]Connecting Data, People, and Ideas since 2016`https://connected-data.world`



Gottfried Wilhelm Leibniz Universität Hannover

Universität in Deutschland

Die Gottfried Wilhelm Leibniz Universität Hannover, kurz Leibniz Universität Hannover oder LUH, ist mit 29 089 Studierenden im Wintersemester 2021/22 die größte Universität Niedersachsens ... +
Wikipedia

uni-hannover.de

Motto	Global denken, interdiszi... +	**Gründung**	1831 · 1879
Ort	Hannover	**Bundesland**	Niedersachsen
Land	Deutschland	**Präsident**	Volker Epping
Studierende	29.089	**Mitarbeiter**	5.294

Figure 2.15: Example of a Google knowledge card for the leibniz university hanover LUH

A knowledge graph organizes information as a network of nodes and edges. The nodes represent entities such as people, places, or things, while edges represent the relationships between them. Knowledge graphs are declarative, and with a large number of entities and their connections, they create contextual information that can make explicit additional details and metadata [30]. Knowledge graphs can assist computers in comprehending the meaning behind data, which can lead to more precise and personalized results for users.

2.10 Semantic Data Modeling

A type of data model that provides a clear and meaningful definition of the relationships among data entities and the rules for using and processing the data is called *semantic data model (SDM).* It aims to make the meaning of data explicit and machine-readable, making it easier for computers to understand and interpret the data, and allowing for better data integration and interoperability. Semantic data models often use formal ontologies or controlled vocabularies to represent the data and its relationships, making it possible to perform automated reasoning and inferencing over the data. SDM is a conceptual diagram of the data as it relates to the real world. There are several types of semantic data models, including:

- Object-Oriented Data Model: Represents data as objects and classes, and defines the relationships between them using inheritance and encapsulation.
- Entity-Relationship (ER) Model or Extended-Entity-Relationship (EER) Model: Describes the relationships between entities and their attributes.
- RDF (Resource Description Framework) Model: A graph-based model that represents data as triples of subject-predicate-object.
- OWL (Web Ontology Language) Model: An extension of the RDF model that adds rich semantic constructs for expressing complex relationships and constraints between data entities.
- Knowledge graph: A graph model for organizing information about topics and their relationships.

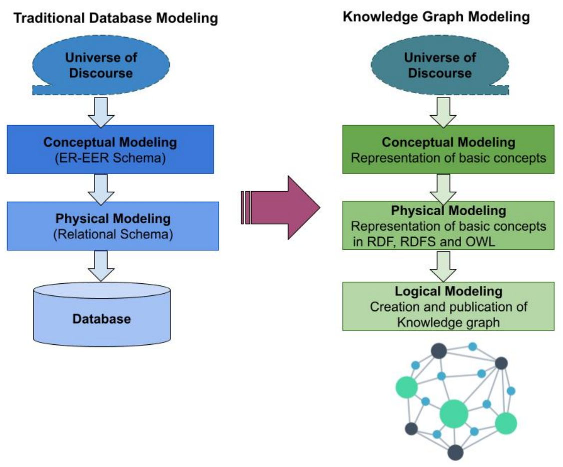

Traditional Database Modeling
Universe of Discourse
Conceptual Modeling (ER-EER Schema)
Physical Modeling (Relational Schema)
Database
Knowledge Graph Modeling
Universe of Discourse
Conceptual Modeling Representation of basic concepts
Physical Modeling Representation of basic concepts in RDF, RDFS and OWL
Logical Modeling Creation and publication of Knowledge graph

Figure 2.16: Semantic data model. Figure taken from [30].

2.11 Summary of the Chapter

This chapter introduced all the concepts required to understand the problem that is explained in this thesis, as well as the proposed solution and the empirical evaluation of the proposed approach.

Chapter 3

Related Work

Data integration has been a highly researched topic in the fields of computer science and information technology; various approaches have been proposed over the years. This chapter presents a brief overview of related work in data integration systems.

3.1 Changes within Data Sources

With the increase of data on the web and the popularity of using knowledge graphs in various fields such as business, politics, and medicine, the dimension of data quality is mainly mentioned in the context of the content of dataset dynamics [6]. Data publishers try to improve the quality of data by continuously updating datasets [27]. With the increasing use of Linked Data, issues like data freshness and tracking to improve data quality are receiving more attention than before. Some efforts have already been made to meet these requirements. These datasets are converted into linked data format using mapping, so the changes in the datasets have an impressive impact on the mapping definition while ensuring the data quality [24].
The approach focuses on capturing information related to changes in the source data used to generate Linked Data datasets, which can result in more up to date and high quality Linked Data, but it doesn't show the effection of changes in whole system or knowledge graph. A knowledge graph for the entire system can be beneficial in extracting and analyzing data, as well as understanding the interrelationships and effects of data and their changes on each other.

3.2 Manage a Storage Repository

The data management dimension that emphasizes FAIR data and facilitates data storage and retrieval plays a crucial role. Enterprises can store and retrieve massive amounts of data with using data lake architectures. While centralizing and storing data in a data lake is beneficial, it does not automatically solve critical data management challenges. Thus, semantic technologies can be utilized. Specifically, ontologies and knowledge graphs can provide essential data lake functions, including cataloging data, tracking provenance, access control, and semantic search [7].
The Layer architecture can be complex to implement and may not be necessary for small data ecosystems, while also depending heavily on the accuracy and completeness of metadata. Introducing new data sources or layers can disrupt existing workflows. An alternative approach could be to create a knowledge graph based on a data integration system that offers less complexity and more flexibility, which could be useful for both large and small data ecosystems and help to improve data management.

3.3 Ontological Approach for Mapping Languages

Knowledge Graphs are created using various techniques and tools, including mapping languages. Due to the wide variety of use cases, data peculiarities, and potential uses, there are numerous declarative mapping languages and associated tools available. Therefore, focusing on an ontological approach for representing the declarative mapping language that does not limit reproducibility and reusability has significant implications for how the mapping created can be extended. The development of an ontology that integrates insights from expert knowledge and a comprehensive comparative analysis of existing mapping languages could prove beneficial [15].
As a result, the need for a model that integrates all related and meaningful data from different parts of a system and has linked data to capture knowledge from them is growing rapidly. The work on the conceptual ontology used to represent the features and connections of the existing declarative mapping language has implications for the creation of the knowledge graph for data integration systems. In this work, we focus on creating an ontology that is used in mappings to build the knowledge graph. We create linked data from the components of the data integration system.

3.4 Summary of the Chapter

This chapter reviews previous research related to the thesis topic and compares it to the current project's work. Specifically, the challenges related to data sources, data integration through creating a data lake and ontological approach for mapping languages are discussed.

Chapter 4

Approach

This thesis tackles the problem of making a data integration system FAIR. Lack of documentation makes understanding the main characteristics of the existing declarative definition of knowledge graphs challenging. Accordingly, in this current section, we formalize the problem and show how integrating data integration systems using ontologies respects the FAIR principles [13]. Then, a solution to create the data integration system is proposed; it includes all the relevant issues which should be documented in data integration systems [20]. Our proposed solution resorts to ontologies. Thus, we present an abstract data integration modeling and design an ontology.

4.1 Problem Statement

The data integration problem is to combine data from different sources into a single one and create a unified view. The challenge in creating a data integration system is to make the repository for the data integration system or, in other words, the *knowledge graph of the data integration system.* Data sources can represent the same concepts differently, so we need an understanding of the data sources.
Providing a shared experience and unifying view of the data leads to the definition of a knowledge graph as an assessment of a data integration system. The knowledge graph is based on ontology, so we must define an ontology to describe the data integration system. Then we can have a more comprehensive understanding of data and improve data quality.

[13]`https://www.go-fair.org/fair-principles`

Challenges of Data Integration

There are various challenges in data integration systems such as *cost*, *duplicates*, *data security*, *maintaining data lineage*, and more. Some of the essential issues are defined below[30].

- **Query** processing queries across different data sources and provides a single view.
- **Number of sources** handling many data sources and providing data integration requirements.
- **Heterogeneity** data are from different sources with different formats and terminologies.
- **Autonomy** tracking the sources and evolution of the data, as the data and its features, such as format, may change.

4.2 Unified Schema for Data Integration System

We add a new semantic layer to a *Data Integration System*, as observed in Figure 2.6. The architecture of data integration system is based on a unified schema and set of sources, where two approaches for data integration can be followed. The first one is *global-as-view*, and the second approach is called *local-as-view*[20]. The both approaches provide unified view and more complete understanding of data and improve data quality, reduce data duplication and leading to increase efficiency. We model the first approach, **global-as-view (GAV)**. To achieve our goal to offer uniform access of data from set of autonomous and heterogeneous multiple sources, in global-as-view approach we define single and unified schema that describes the data structure and how data should be organized and related to each other. With local-as-view (LAV), we provide centralized view from multiple sources, while maintaining the unique qualities of each individual source.

Type of Data Integration System

In this work, we consider the **centralized** and **homogeneous** data integration system, which it means all represented data sources using the same data model. This is in contrast to the distributed-heterogeneous data integration system, which represents data sources with heterogeneity conflicts and is distributed in different locations[30]. Figure 4.1 illustrates the different types of data integration systems. A

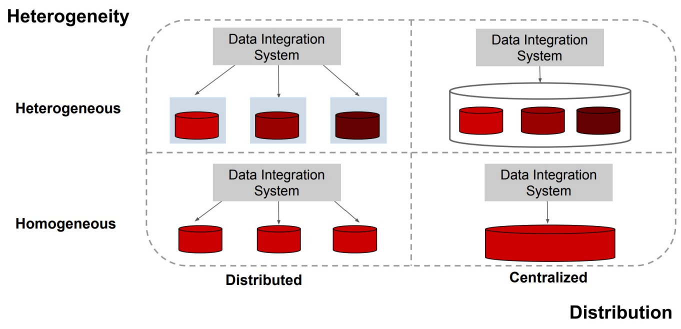


Figure 4.1: Types of data integration system. Figure taken from [30].

data integration system is defined in terms of quadruple $DIS=\langle G, S, M, F\rangle$ [17, 20, 18], where concepts in G are represented following the global-as-view approach GAV:

- **G** is a global schema, which is data structure or model, i.e., ontology in triple form of $U=\langle C, P, A\rangle$. They are referring to vocabulary of ontology, C as classes, P as properties and A is a set of axioms for interpreting the content of vocabulary.
- **S** is a source schema, which is set of data sources and that is used by mapping rules.
- **M** is a set of mapping rules between the G *(global schema)* and the S *(source schema)*; it is formed by set of assertions of the form:

  $$q_S \rightarrow q_G$$
  $$q_G \rightarrow q_S$$

  where q_S and q_G are two queries over the *source schema* and *global schema*[20].
- **F** is a set of functional symbols that represent user-defined and built-in functions. The f as a *Function (F)* can be a simple function or composite one. $f(x_1, ...x_n)$ as a simple function is a *term*, which $(x_1, ..., x_n)$ are variables or constants. $f(x_1, ...x_n)$ as composite function can receive a *function* in any of the arguments, x_i [17].

Global as view

In the *global-as-view* approach for each GAV mapping rules, there is a set of assertions that relate to each element g in G as *global schema* that should be formed in terms of a view q_S over S as *source schema*. The mapping shows how the data to evaluate the unified schema can be retrieved[20].

$$g \rightarrow q_S$$

Knowledge Graphs

A *knowledge graph* is a directed graphical model that provides a unified view from various data sources. In this thesis, knowledge graphs are defined as $KG{=}(O, V, E)$; they are generated from the evaluation of a data integration system DIS[17]. In the knowledge graph, O represents the ontology that consists of classes and properties; V corresponds to the nodes of the classes or instances of classes in ontology; and E are properties in ontology defined as directed labeled edges in the knowledge graph, that are related to nodes in V. Knowledge graphs are valuable for analysis and reporting as they represent knowledge as factual statements; state-of-the-art data management techniques enable querying and reasoning efficiently.

4.3 Conceptual of Design

Conceptual design in semantic data modeling refers to the process of defining the overall structure and organization of the data in a high-level, abstract manner. It involves identifying the key concepts or entities in the domain being modeled, and defining the relationships between them. The result of the conceptual design is a conceptual data model, which represents the data in a way that is independent of any particular implementation or technology. It also provides a basis for later stages of data modeling, such as logical design and physical design, where the data model is refined and optimized for a particular use case or technology. In this work, the enhanced entity-relationship (EER) model is used for conceptual design.

4.3.1 Data Integration System Model

In EER model, the class of *data integration system* is composed of a *source schema, set of mappings, unified schema* and *Integrity Constraints*. Figure 4.2 shows the *DataIntegration_System, UnifiedSchema, SourcesSchema, SetOf_Mappings* and *Schema_ICs* are represented as **classes** in modeling. The relationships between these classes are defined with *composed of* relationships, which are **object properties**. The *id* and *name* as **data type properties** or attributes are assigned to the classes. Each part of the data integration system is further explained in detail in the following sections.

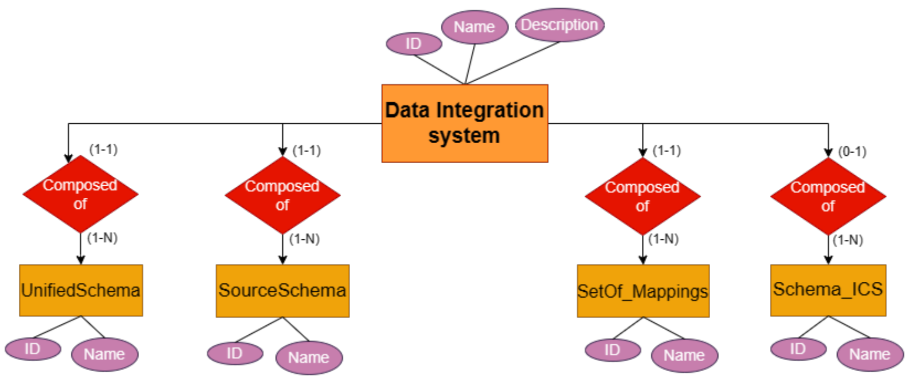


Figure 4.2: Main components of data integration system

Sources Schema

A source schema, which is set of data sources is represented with a class that encompasses all the data sources in the data integration system. Each data source is represented as an instance of this class, and has three attributes: description, id and data source's name. Additionally, each data source is required to have exactly one format and at least one data field with its corresponding data type defined using the *rdfs:dataType* class. As you see in the Figure 4.3 the data field is defined as a weak entity, because it depends on the existence of a data source entity. The classes and relationships described here are depicted in the Figure 4.3 provided.

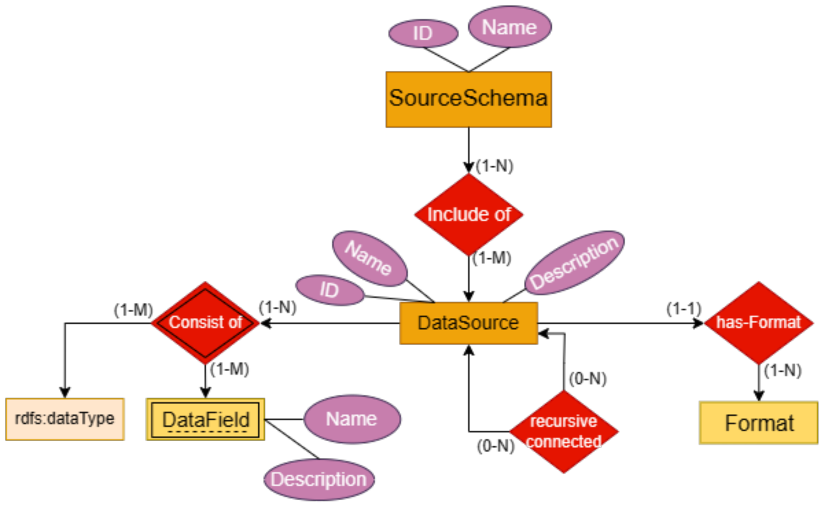


Figure 4.3: Data Sources Components

Unified Schema

A unified schema in a data integration system consists of various ontologies, including OWL, RDF, and RDFS. Each ontology has a namespace, which is a composite attribute. The ontology itself is composed of a set of classes, a set of properties, and a set of axioms. These classes represent entities within the integrated system and may include descriptions to aid in understanding their contents. The set of properties can include either data type properties, which link a class to a data field, or object properties, which connect two classes. As shown in Figure 4.4, the range and domain of every object property are classes. The set of axioms defines logical rules that enable the definition and reasoning of the meaning of terms and concepts within an ontology, as outlined in the background chapter 2.3.1.

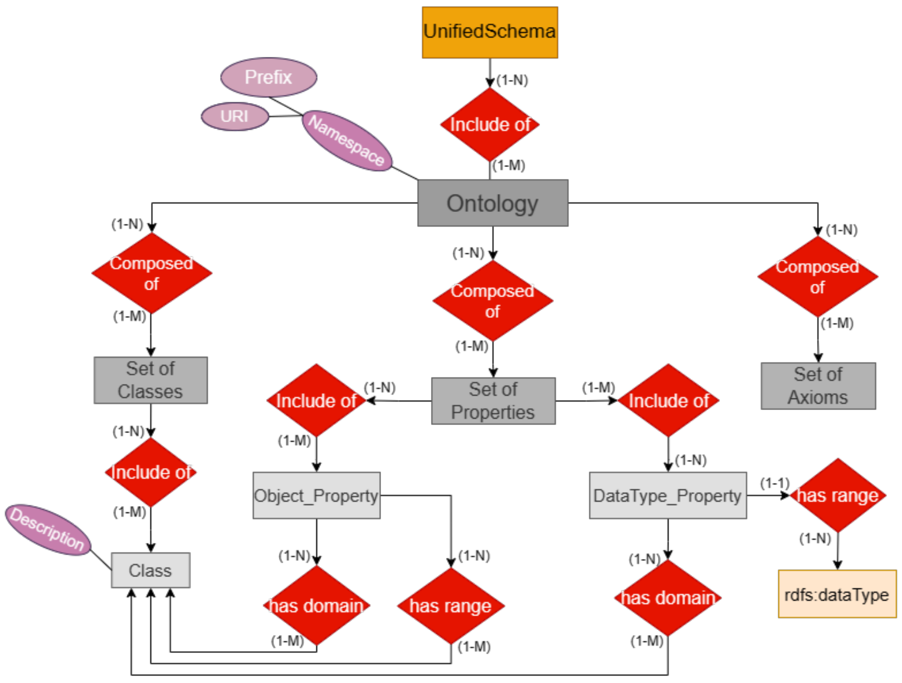


Figure 4.4: Unified Schema Components

Set of Mappings

A set of mappings includes several triple mappings. Each triple mapping comprises from one or more mapping assertions, so mapping assertion is defined as a weak entity in model 4.5. It must have exactly one *concept mapping assertion* and may have zero to several *attribute mapping assertions* and *role mapping assertions*. As depicted in Figure 4.5, each mapping assertion is composed of a *head* and a *body*. The mapping assertions [23] are formalized as such:

$$body(\overline{X}) : -head(\overline{Y})$$

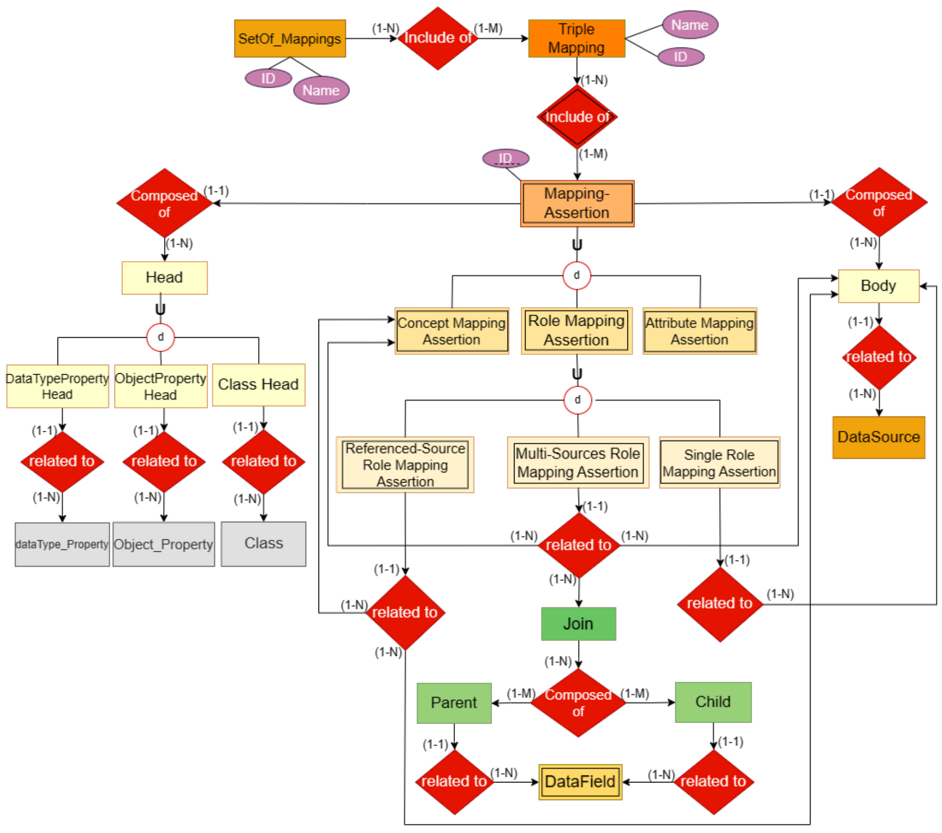


Figure 4.5: Set of Mappings Components

The predicate conjunction $body(\overline{X})$ is defined over a set of *terms*, while the predicate $head(\overline{Y})$ is defined over a separate set of *terms*. In order to define terms within this framework, we utilize an inductive approach consisting of two cases. In the **base case**, we define terms in two ways: **i)** a constant c is considered a term, and **ii)** a variable X can be defined as a term. In the **inductive case**, we introduce a functional symbol h of arity n, and let $t_1, ..., t_n$ be terms. We can then define the expression $h(t_1, ..., t_n)$ as a *term* [17]. You can find in the Figure 4.6 the different term types of an argument as function, variable and constant.

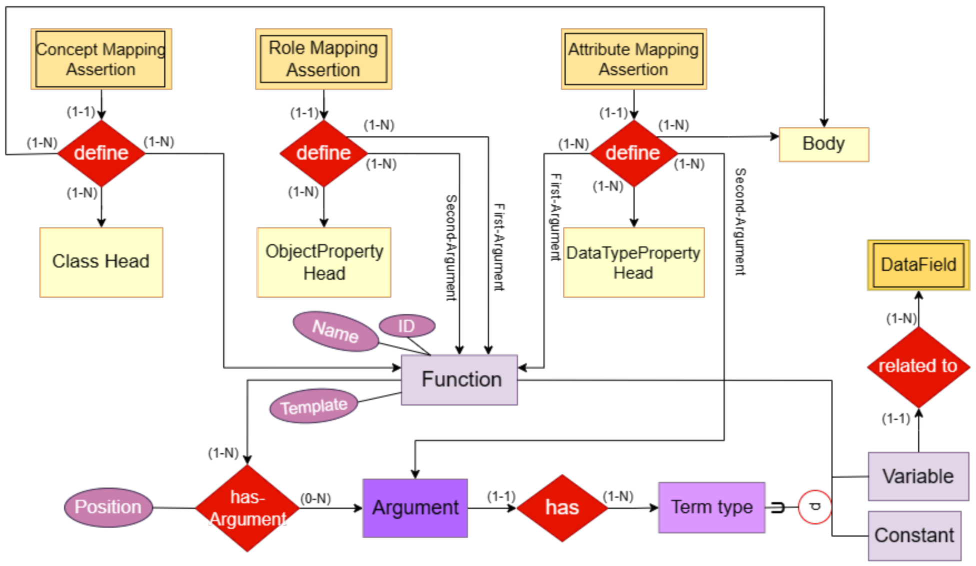


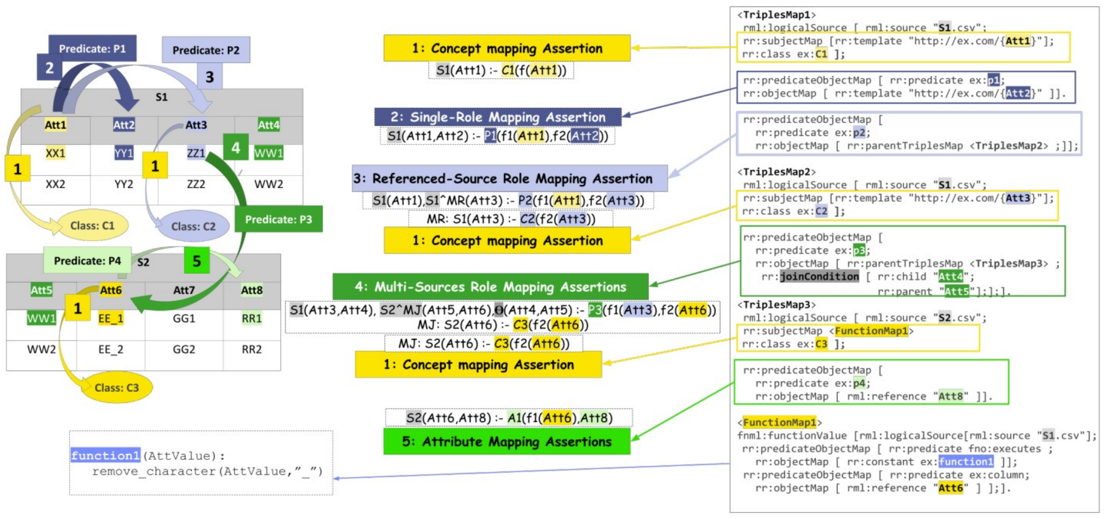
Figure 4.6: Mapping Assertion and Function Components

As mapping assertions are explained in background chapter 2.7, here are their definition based on the represented model.

- **Concept Mapping Assertion:** To define instances of classes C in an ontology, the class predicate $C(.)$ is used over the results of the $f(.)$ function, which receives a term t as input arguments. The input arguments can be a variable, constant, or function, as illustrated in Figure 4.6. The predicate $S(X)$ represents the conjunction of source signatures $S_1(X_1), ..., S_k(X_k)$, where X corresponds to the set of all the variables in the mapping assertion, which is the union of $X_1, ..., X_k$. The predicate $S(X)$ is defined as follows:

$$S_i(\overline{X}) : -C(f(t))$$

  In Figure 4.7, two examples of concept mapping assertions are highlighted in yellow, which define the instances of the classes $C1, C2$, and $C3$ based on the data field values in the data sources $S1$ and $S2$. The data fields $Att1$ and $Att3$ in $S1$, as well as data field $Att6$ in $S2$, are used to define these instances. In Figure 4.7, $f(.)$ corresponds to either a built-in function represented by `rr:template` (which enables the concatenation of strings) or user-defined functions defined


Predicate: P1
Predicate: P2
2
3
S1
Att1
Att2
Att3
Att4
XX1
YY1
ZZ1
WW1
XX2
YY2
ZZ2
WW2
1
1
4
Class: C1
Class: C2
Predicate: P3
Predicate: P4
S2
5
Att5
Att6
Att7
Att8
WW1
EE_1
GG1
RR1
WW2
EE_2
GG2
RR2
1
Class: C3
1: Concept mapping Assertion
S1(Att1) :- C1(f(Att1))
2: Single-Role Mapping Assertion
S1(Att1,Att2) :- P1(f1(Att1),f2(Att2))
3: Referenced-Source Role Mapping Assertion
S1(Att1),S1^MR(Att3) :- P2(f1(Att1),f2(Att3))
MR: S1(Att3) :- C2(f2(Att3))
1: Concept mapping Assertion
4: Multi-Sources Role Mapping Assertions
S1(Att3,Att4), S2^MJ(Att5,Att6),θ(Att4,Att5) :- P3(f1(Att3),f2(Att6))
MJ: S2(Att6) :- C3(f2(Att6))
MJ: S2(Att6) :- C3(f2(Att6))
1: Concept mapping Assertion
S2(Att6,Att8) :- A1(f1(Att6),Att8)
5: Attribute Mapping Assertions

```
function1(AttValue):
   remove_character(AttValue,"_")
```

```
<TriplesMap1>
 rml:logicalSource [ rml:source "S1.csv";
 rr:subjectMap [rr:template "http://ex.com/{Att1}"];
 rr:class ex:C1 ];

 rr:predicateObjectMap [ rr:predicate ex:p1;
 rr:objectMap [ rr:template "http://ex.com/{Att2}" ]].

 rr:predicateObjectMap [
   rr:predicate ex:p2;
   rr:objectMap [ rr:parentTriplesMap <TriplesMap2> ;]];

<TriplesMap2>
 rml:logicalSource [ rml:source "S1.csv";
 rr:subjectMap [rr:template "http://ex.com/{Att3}"];
 rr:class ex:C2 ];

 rr:predicateObjectMap [
   rr:predicate ex:p3;
   rr:objectMap [ rr:parentTriplesMap <TriplesMap3> ;
    rr:joinCondition [ rr:child "Att4";
                       rr:parent "Att5"];];].
<TriplesMap3>
 rml:logicalSource [ rml:source "S2.csv";
 rr:subjectMap <FunctionMap1>
 rr:class ex:C3 ];

 rr:predicateObjectMap [
   rr:predicate ex:p4;
   rr:objectMap [ rml:reference "Att8" ]].

<FunctionMap1>
fnml:functionValue [rml:logicalSource[rml:source "S1.csv"];
 rr:predicateObjectMap [ rr:predicate fno:executes ;
   rr:objectMap [ rr:constant ex:function1 ]];
 rr:predicateObjectMap [ rr:predicate ex:column;
   rr:objectMap [ rml:reference "Att6" ] ];].
```

Figure 4.7: Mapping Assertions Definition. Figure taken from [17]

by `fnml:FunctionTermMap`. In Figure 4.7, `fnml:FunctionTermMap` expresses the definition of data operation functions ($function1$) over $Att6$ [17].

- **Role Mapping Assertions:** The role mapping assertion specifies how properties in ontology are instantiated between instances of two classes in ontology, using a role predicate $P(.)$ over the attributes of data sources. Single-role mapping assertion, referenced-source role mapping assertion and multi-sources role mapping assertion are defined as *subclasses* of a role mapping assertion. These three mapping assertions are role mapping assertion predicate *Object Property*. The single-role mapping assertion is a simple one just connected to the body. Referenced-source role mapping assertion that binds two different mapping assertion with the same data source, connects to concept mapping assertion and body. Multi-sources role mapping assertion binds two different mapping assertion with different data sources; it also relates the body and concept mapping assertion. Moreover, a multi-sources role mapping assertion is connected also to *Join* that expresses the data fields in parent mapping assertion and child mapping assertion .

- **Single-role mapping assertion:** is used to define $P(.,.)$ for data fields of a just one data source, using the functions $f_1(t_1)$ and $f_2(t_2)$, where f_1 and f_2 are part of Function, and t_1 and t_2 are their input arguments. For example, in Figure 4.7, a single role mapping assertion in purple is shown, that defines the role predicate $P1(.,.)$ for instances of the class $C1$, using $Att1$ in the $S1$ data source, processed by the built-in function f_1, and object is based on $Att2$ in the same data source, processed by the function f_2.

$$S_i(\overline{X}) : -P(f_1(t_1), f_2(t_2))$$

- **Referenced-source role mapping assertion:** is similar to the previous assertion, however, in this assertion, an object can be defined using a term t_2 as an input argument for function $f_2(.)$, which are used in another mapping assertion to specify the class instance in the ontology.

  $$S_i(\overline{X}_{i,1}), S_i^{MR}(\overline{X}_{i,2}) : -P(f_1(t_1), f_2(t_2))$$
  $$MR : S_i(\overline{X}_{i,2}) : -C_j(f_2(t_2))$$

  The example of this mapping assertion in Figure 4.7 is shown in violet. The predicate $P2$ in this example returns the instance of class $C1$ based on $Att1$ in $S1$ that is processed by function f_1, and the instance of class $C2$ based on $Att3$ in $S1$ which is used as another concept mapping assertion [17].

- **Multi-sources role mapping assertion:** In contrast to the previous statement, a multi-source role mapping assertion enables the expression of property instances in an ontology between instances of two classes with values over two distinct sources. As the sources S_i and S_j differ, a join condition is necessary.

  $$S_i(\overline{X}_{i,1}), S_j^{MJ}(\overline{X}_{i,2}), \theta(\overline{X}_{i,1}, \overline{X}_{i,2}) : -P(f_1(t_1), f_2(t_2))$$
  $$MJ : S_j(\overline{X}_{i,2}) : -C_z(f_2(t_2))$$

  Figure 4.7 shows an example of a multi-sources role mapping assertion in dark green. Role $P3$ predicates the instance of class $C1$ over $Att3$ in data source $S1$ and the instance of class $C3$ over $Att6$ from data source $S2$. The θ, which is a *join* between common data fields in both data sources, uses $Att4$ from data source $S1$ and $Att5$ from data source $S2$ to bind the mapping assertions together [17].

- **Attribute Mapping Assertions:** defines the properties of a class in ontology by using a predicate $A(.)$ over the values of data fields. These values are expressed in terms of the conjunction of source signatures in $S(X)$.

  $$S_i(\overline{X}) : -A(f(t_1), t_2)$$

  In Figure 4.7 an example of attribute mapping assertion is shown in light green. The predicate $A1$ defines over the instances of the class C3 using data field $Att6$ from $S1$ and $Att8$ as the literal data values in the same data source [17].

Integrity Constraints

Integrity constraints refer to a set of rules or conditions that ensure the accuracy and consistency of data within the system. These constraints are used to validate the data as it is being entered into the system and to prevent it from being entered in a way that would compromise the integrity of the system. These constraints are important for ensuring that the data in the semantic web is reliable and can be used to make informed decisions. As previously discussed 2.8, the validation is based on SHACL (Shapes Constraint Language). SHACL is a language for validating RDF graphs against a set of conditions expressed in RDF graphs. These ”shapes” conditions are stored in a ”shapes graph.” The RDF graphs being validated are known as ”data graphs.” In this sense, SHACL shape graphs can be seen as a description of valid data graphs, as they enforce constraints on the data graphs to ensure they meet specific conditions [14]. The conceptual design stage of the SHACL for data integration system is shown in Figure 4.8.

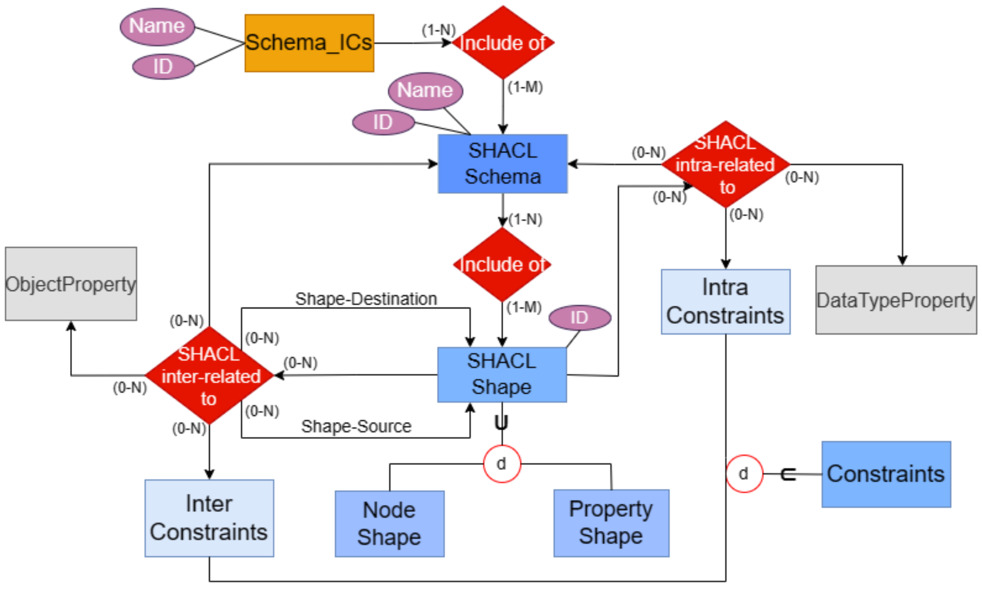


Figure 4.8: Integrity Constraints - SHACL

[14]https://www.w3.org/TR/shacl/

4.4 Ontology Requirements and Competency Questions

We conducted the following steps to identify the requirements of our ontology:

- Review on data management and state-of-the-art approaches for data integration systems;
- Analyze relevant existing ontologies and data models to document the characteristics of datasets

Based on this analysis, we elucidated functional and non-functional requirements, which are reported in Table 4.1 and the competency questions, which are presented in Table 4.2.

Functional Requirements	
Requirements	Description
Structure of a data integration system	Modeling must represent the structural elements of data integration.
Connection among structural elements	Modeling must show the connection among ontology, data source and mapping.
Extensibility	Ontology should be extensible basen on new requirements.
None-Functional Requirements	
Requirements	Description
Findability	Find and track the data sources, mapping rules and defined classes and properties.
Accessibility	Data should be accessible including authentication and authorisation.
Interoperability	To perform analysis, storage, and processing, the data should be integrated with other datasets and made interoperable with applications [15].
Re-useability	Well-describing of data, so the reusability of ontologies, data sources and mappings rules can be possible.

Table 4.1: Ontology requirements

Competency Questions	
Q1	What is the number of data sources available, and how can they be counted?
Q2	What information or data fields does each data source contain?
Q3	What are the data types for the fields in each data source?
Q4	Which mappings have been used for a specific class or property?
Q5	Which properties are defined in ontologies?
Q6	Which classes are defined in ontology?
Q7	Is there specific binding established between two specific classes?
Q8	Is it possible to find specific class among several data integration systems?

Table 4.2: Competency Questions

4.5 Summary of the Chapter

This chapter describe the challenges of documenting Data Integration System. Therefore, a semantic data model was proposed, which can integrate ontologies, data sources and mappings to create the Data Integration System. The aim of this chapter was referring to the lack of data integration system as a problem and explain how can we achieve to a single and more comprehensive view of data.

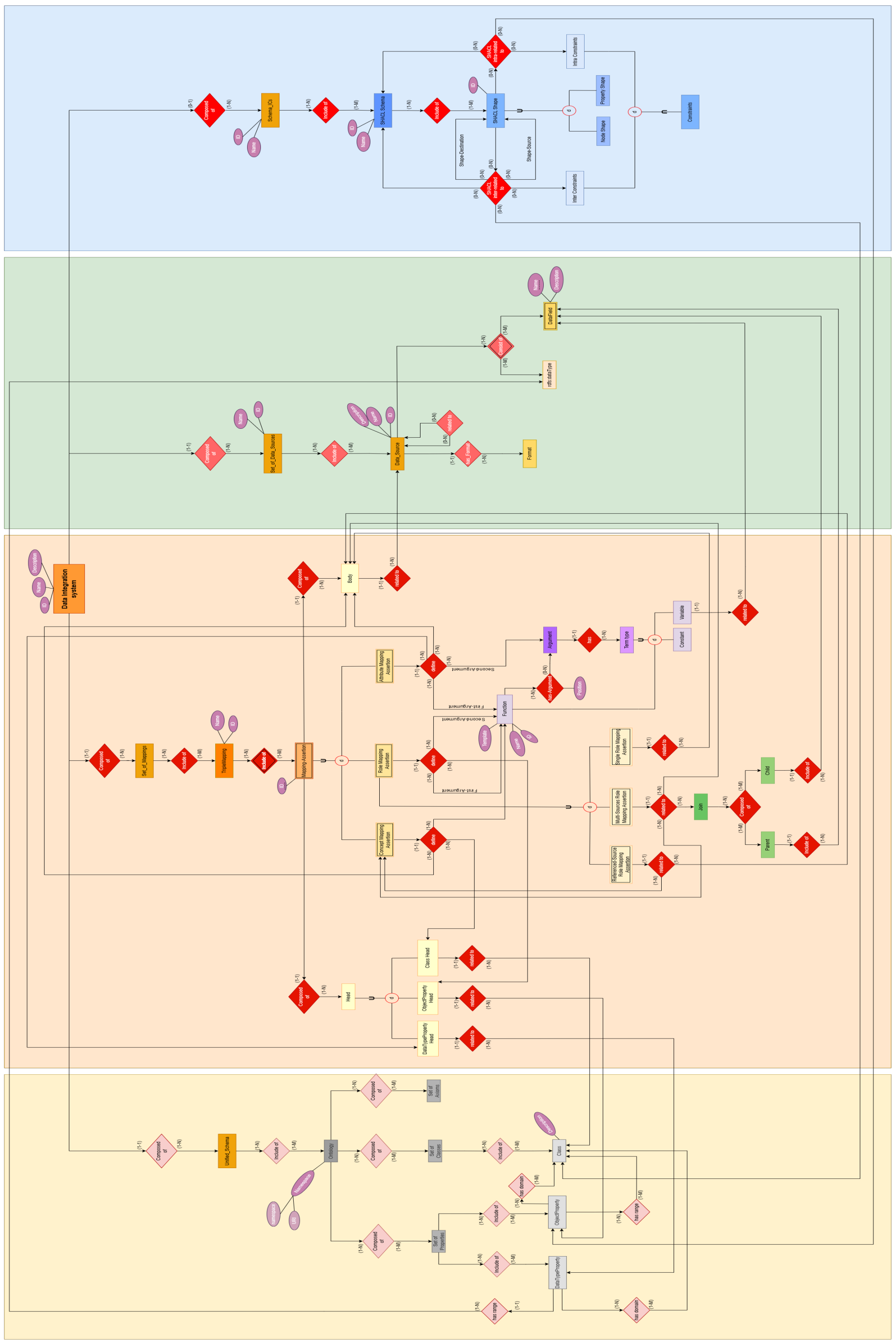


Figure 4.9: Data Integration System

Chapter 5

Implementation

This chapter describes the main aspects of the implementation of our proposed approach. We follow a strategy of semantic data integration, and a pipeline for knowledge graph creation for its implementation. The mapping language RML[16] is utilized to declaratively define knowledge graphs. The pipeline for creating Knowledge graph is divided to three Phases: Conceptual design, and logical design and physical design. Conceptual design is discussed already in The chapter approach 4.3. On the other hand, the Logical design and Physical design are defined in this Chapter.

5.1 Logical of Design

Logical modeling in semantic modeling refers to creating a formal representation of knowledge, concepts, or relationships in the domain under consideration, using symbols, logic, and language [17]. The goal of logical modeling is to create a model that accurately represents the system or process being modeled and can be used to analyze, simulate, or design the system. In logical modeling, the emphasis is on defining the entities, attributes, and relationships between them. The entities are the objects or concepts being modeled, and the attributes are the characteristics or properties of those entities. Relationships describe how the entities are connected or related to each other. Logical models can be represented using formal notations, such as entity-relationship diagrams, UML diagrams, or RDF graphs. These notations provide a standardized way of describing the entities, attributes, and relationships, which makes the model easier to understand and communicate. Logical modeling is essential in the semantic modeling process because it provides a foundation for

[16]`https://rml.io/`
[17]`https://www.ibm.com/topics/data-modeling`

further analysis and design. Logical models can be used to identify inconsistencies, redundancies, or other issues in the system being modeled, which can help improve the accuracy and effectiveness of the model.

5.1.1 WebProtégé

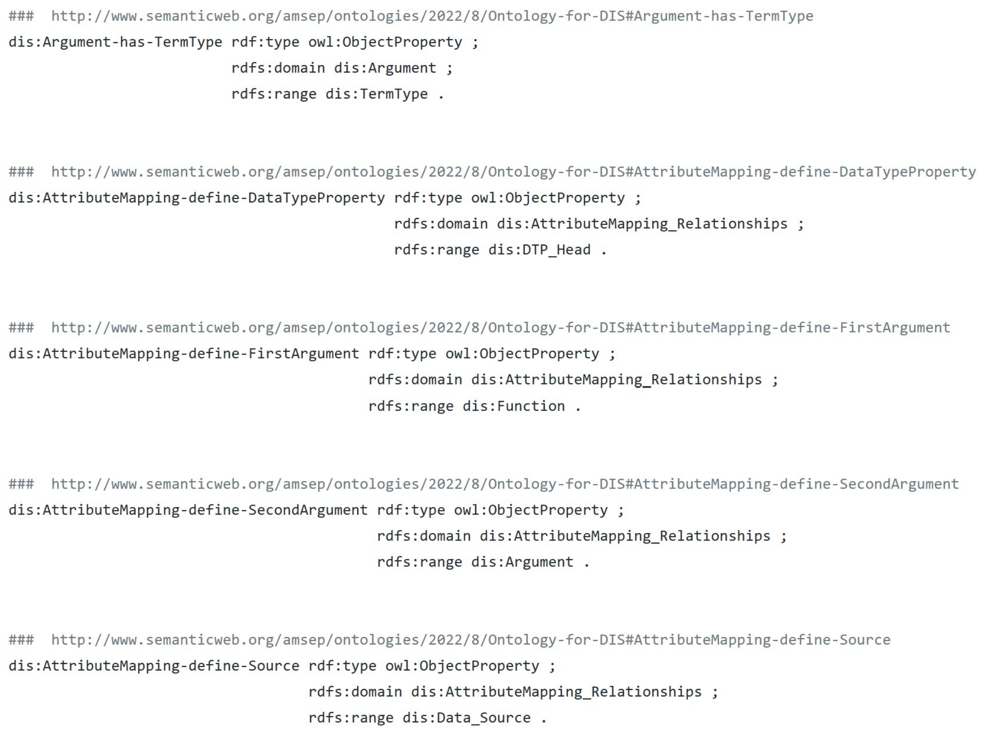
WebProtégé is an open-source, web-based, ontology editor and knowledge management system used to develop and maintain knowledge models, including ontologies, taxonomies, and vocabularies. It is a tool designed for collaborative ontology development and provides a user-friendly interface for creating and editing knowledge models. WebProtégé supports various ontology languages, including OWL, RDF(S), and SKOS [18]. WebProtégé is built on top of the Protégé platform, which is a popular desktop ontology editor. WebProtégé extends the functionality of Protégé by providing a web-based interface that can be accessed from anywhere, without the need for local installation or configuration [19]. To develop a data integration system, it is essential to understand the relationships and specific terminologies involved. As a result, we have created new vocabularies and RDF rules for the ontology of the data integration system based on the represented model in the approach chapter. This thesis defines the concepts using the prefix *dis:* for the *DIS* namespace. The data modeling is followed by defining classes, object properties, and data type properties in WebProtégé and creating the Ontology in OWL. Figure 5.1 shows a sample of created ontology. The repository of the ontology is in GitHub [20]. Visualizing ontologies can make them more accessible, understandable, and usable for various applications. It can also aid in detecting gaps and redundancies within the ontology. Consequently, there are various web applications available for interactive ontology visualization, such as RDF Playground [21] that you can see in Figure 5.2 and WebVOWL [22], which we utilized to visualize our data integration system ontology. Figure 5.3 depicts the data integration system ontology in WebVOWL.

Some classes, which are defined in this work, can be defined as *equivalent class (owl:equivalentClass)* to other ontologies in semantic web. For example the entity dis:Parent is equivalent to class parent in R2RML (rr:parent). You can find all equivalent classes to this work in table 5.1.
For all defined classes in ontology based on their specific properties, we have defined

[18]`https://protegewiki.stanford.edu/wiki/WebProtege`
[19]`https://protege.stanford.edu/products.php`
[20]`https://github.com/tibonto/DIS.git`
[21]`http://rdfplayground.dcc.uchile.cl/`
[22]`http://ontology.tib.eu/DIS/visualization`

DIS Name	External resource	IRI
SourceSchema	dcat:DatasetSeries	`https://www.w3.org/ns/dcat#DatasetSeries`
DataSource	dcat:dataset	`https://www.w3.org/ns/dcat#dataset`
Function	fnml:functionMap	`http://semweb.mmlab.be/ns/fnml/fnml.html#functionMap`
TripleMapping	rr:TriplesMap	`https://www.w3.org/ns/r2rml#TriplesMap`
Join	rr:joinCondition	`https://www.w3.org/ns/r2rml#joinCondition`
Parent	rr:parent	`https://www.w3.org/ns/r2rml#parent`
Child	rr:child	`https://www.w3.org/ns/r2rml#child`
SHACL_Shape	sh:Shape	`https://www.w3.org/ns/shacl#shape`
Constraints	sh:ConstraintComponent	`https://www.w3.org/ns/shacl#ConstraintComponent`
Property_Shape	sh:PropertyShape	`https://www.w3.org/ns/shacl#PropertyShape`
Node_Shape	sh:NodeShape	`https://www.w3.org/ns/shacl#NodeShape`
Ontology	owl:Ontology	`https://www.w3.org/2002/07/owl#Ontology`
Object_Property	owl:ObjectProperty	`https://www.w3.org/2002/07/owl#ObjectProperty`
DataType_Property	owl:DatatypeProperty	`https://www.w3.org/2002/07/owl#DatatypeProperty`
Class	owl:class	`https://www.w3.org/2002/07/owl#Class`
Axioms	owl:Axiom	`https://www.w3.org/2002/07/owl#Axiom`

Table 5.1: Classes shown in Figure 4.9 and their corresponding external resources and IRIs details



```
###  http://www.semanticweb.org/amsep/ontologies/2022/8/Ontology-for-DIS#Argument-has-TermType
dis:Argument-has-TermType rdf:type owl:ObjectProperty ;
                          rdfs:domain dis:Argument ;
                          rdfs:range dis:TermType .


###  http://www.semanticweb.org/amsep/ontologies/2022/8/Ontology-for-DIS#AttributeMapping-define-DataTypeProperty
dis:AttributeMapping-define-DataTypeProperty rdf:type owl:ObjectProperty ;
                                             rdfs:domain dis:AttributeMapping_Relationships ;
                                             rdfs:range dis:DTP_Head .


###  http://www.semanticweb.org/amsep/ontologies/2022/8/Ontology-for-DIS#AttributeMapping-define-FirstArgument
dis:AttributeMapping-define-FirstArgument rdf:type owl:ObjectProperty ;
                                          rdfs:domain dis:AttributeMapping_Relationships ;
                                          rdfs:range dis:Function .


###  http://www.semanticweb.org/amsep/ontologies/2022/8/Ontology-for-DIS#AttributeMapping-define-SecondArgument
dis:AttributeMapping-define-SecondArgument rdf:type owl:ObjectProperty ;
                                           rdfs:domain dis:AttributeMapping_Relationships ;
                                           rdfs:range dis:Argument .


###  http://www.semanticweb.org/amsep/ontologies/2022/8/Ontology-for-DIS#AttributeMapping-define-Source
dis:AttributeMapping-define-Source rdf:type owl:ObjectProperty ;
                                   rdfs:domain dis:AttributeMapping_Relationships ;
                                   rdfs:range dis:Data_Source .
```

Figure 5.1: Created Ontology of data integration system with WebProtégé `https://github.com/tibonto/DIS.git`

the cardinalities. Therefore, the classes are defined as *complex classes* with regarding to cardinalities. Here are some examples in Figure 5.4 and you can find the whole complex classes in ontology in GitHub[23].

5.2 Physical of Design

In semantic modeling, the physical design is a phase where the logical data models are transformed into physical data models. Physical modeling involves designing and implementing the database structure, including tables, columns, keys, relationships, and constraints. During physical modeling, the focus shifts from defining the entities, attributes, and relationships to creating a physical representation of the data

[23]`https://github.com/tibonto/DIS`

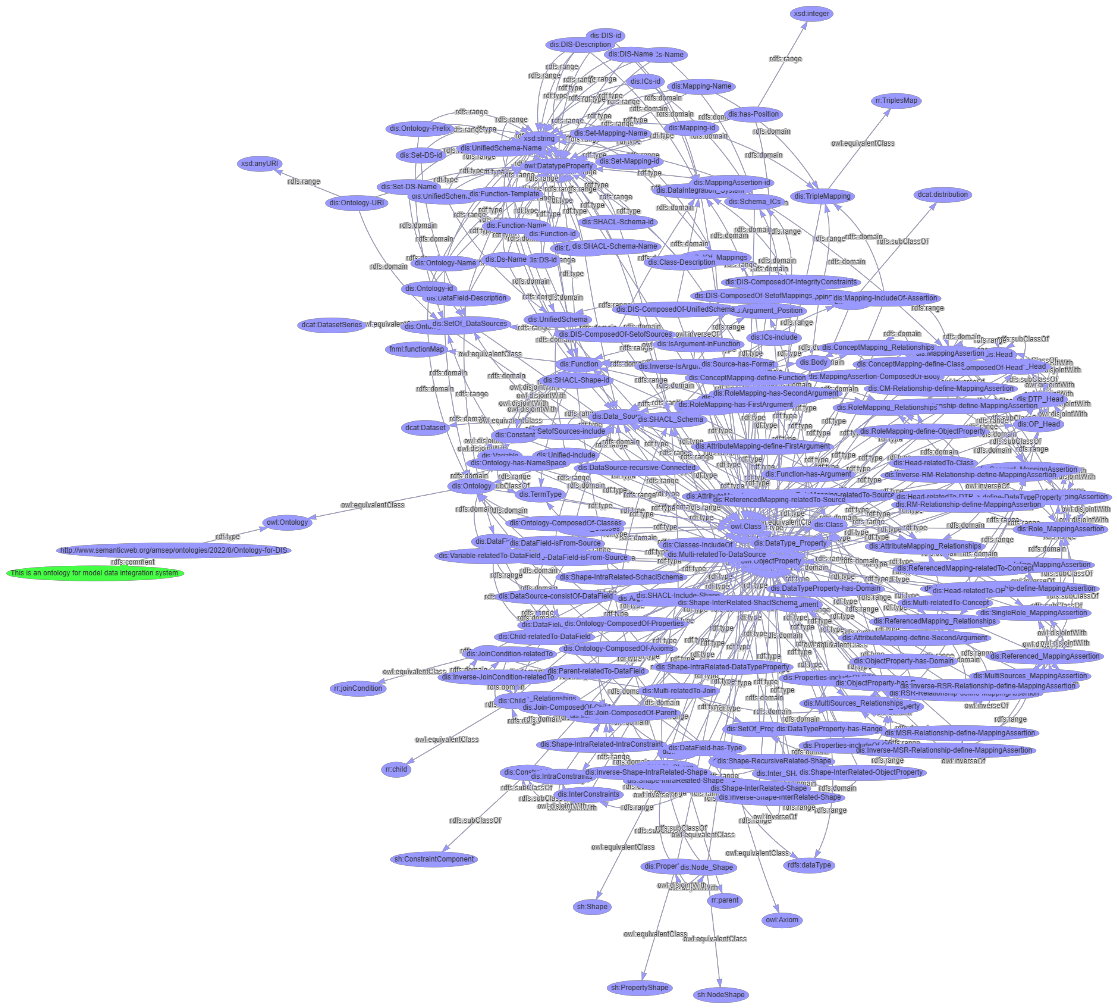


Figure 5.2: Visualization of data integration system ontology with RDFplayGround

model. This includes defining data types, constraints, and other physical aspects of the database. Physical modeling is an essential step in semantic modeling because it transforms the conceptual model into a physical implementation that can be used to store and manage data. It also helps to ensure that the database is efficient, scalable, and optimized for performance.
We use SDM-RDFizer as the RML Engine that creates the RDF knowledge graph. SDM-RDFizer is an interpreter of the RDF Mapping Language (RML) that transforms raw data in various formats into an RDF knowledge graph. It employs advanced algorithms to execute logical operators between RML mappings, enabling it

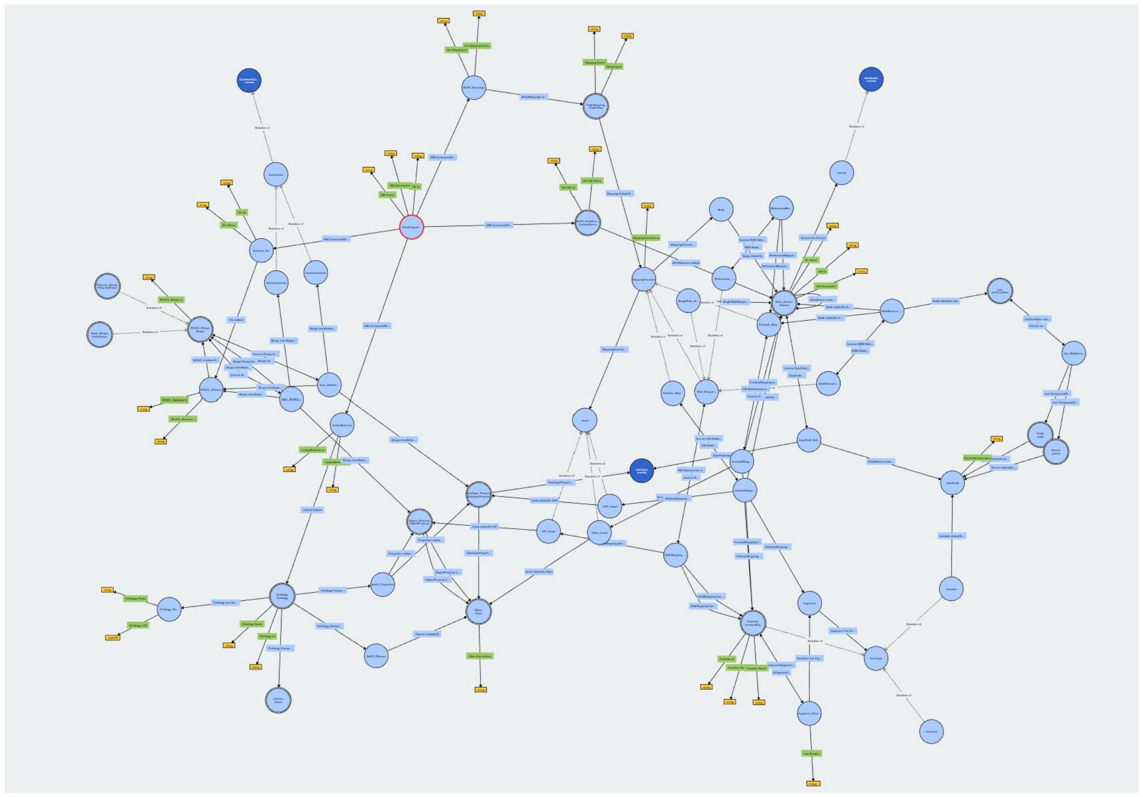
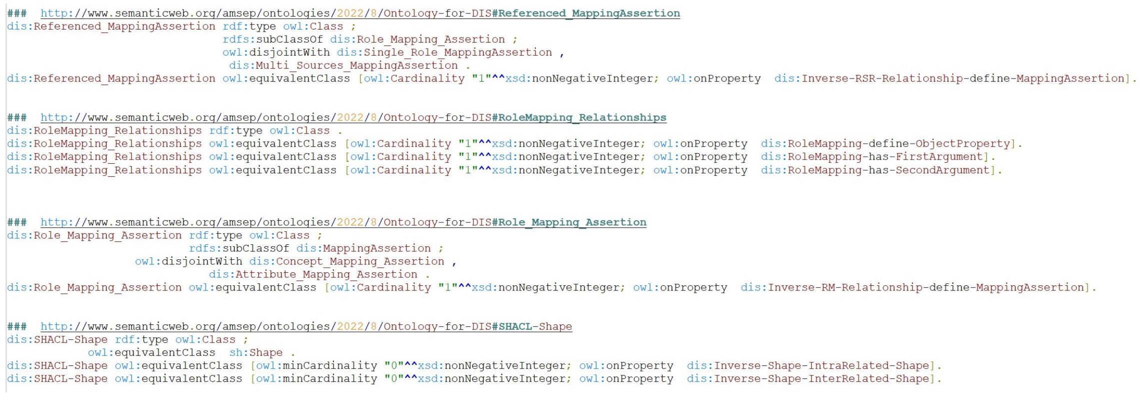


Figure 5.3: Visualization of data integration system ontology with WebVOWL `http://ontology.tib.eu/DIS/visualization`



```
###  http://www.semanticweb.org/amsep/ontologies/2022/8/Ontology-for-DIS#Referenced_MappingAssertion
dis:Referenced_MappingAssertion rdf:type owl:Class ;
                                rdfs:subClassOf dis:Role_Mapping_Assertion ;
                                owl:disjointWith dis:Single_Role_MappingAssertion ,
                                 dis:Multi_Sources_MappingAssertion .
dis:Referenced_MappingAssertion owl:equivalentClass [owl:Cardinality "1"^^xsd:nonNegativeInteger; owl:onProperty  dis:Inverse-RSR-Relationship-define-MappingAssertion].


###  http://www.semanticweb.org/amsep/ontologies/2022/8/Ontology-for-DIS#RoleMapping_Relationships
dis:RoleMapping_Relationships rdf:type owl:Class .
dis:RoleMapping_Relationships owl:equivalentClass [owl:Cardinality "1"^^xsd:nonNegativeInteger; owl:onProperty  dis:RoleMapping-define-ObjectProperty].
dis:RoleMapping_Relationships owl:equivalentClass [owl:Cardinality "1"^^xsd:nonNegativeInteger; owl:onProperty  dis:RoleMapping-has-FirstArgument].
dis:RoleMapping_Relationships owl:equivalentClass [owl:Cardinality "1"^^xsd:nonNegativeInteger; owl:onProperty  dis:RoleMapping-has-SecondArgument].



###  http://www.semanticweb.org/amsep/ontologies/2022/8/Ontology-for-DIS#Role_Mapping_Assertion
dis:Role_Mapping_Assertion rdf:type owl:Class ;
                           rdfs:subClassOf dis:MappingAssertion ;
                   owl:disjointWith dis:Concept_Mapping_Assertion ,
                              dis:Attribute_Mapping_Assertion .
dis:Role_Mapping_Assertion owl:equivalentClass [owl:Cardinality "1"^^xsd:nonNegativeInteger; owl:onProperty  dis:Inverse-RM-Relationship-define-MappingAssertion].


###  http://www.semanticweb.org/amsep/ontologies/2022/8/Ontology-for-DIS#SHACL-Shape
dis:SHACL-Shape rdf:type owl:Class ;
            owl:equivalentClass  sh:Shape .
dis:SHACL-Shape owl:equivalentClass [owl:minCardinality "0"^^xsd:nonNegativeInteger; owl:onProperty  dis:Inverse-Shape-IntraRelated-Shape].
dis:SHACL-Shape owl:equivalentClass [owl:minCardinality "0"^^xsd:nonNegativeInteger; owl:onProperty  dis:Inverse-Shape-InterRelated-Shape].
```

Figure 5.4: Definition of complex classes in ontology of data integration system

to handle complex scenarios with broad and highly-duplicated data [12].

5.2.1 GraphDB

GraphDB is a database management system designed specifically for storing and managing graph data. It is a type of RDF database that represents data as nodes and edges, which are connected to form a graph. Graph databases are particularly useful for managing complex, interconnected data, such as knowledge graphs.
GraphDB supports the Resource Description Framework (RDF) data model and is compatible with the SPARQL query language. It is designed to provide efficient graph processing capabilities, with features such as indexing, querying, and reasoning, that enable fast and scalable processing of graph data [24].

SDM-RDFizer creates an RDF knowledge graph that is then imported into GraphDB to enable query processing by using SPARQL queries. Thus, it is possible to retrieve information on the number of data sources, their data fields, the connections between different classes and many other things. Figure 5.5 illustrates a portion of the knowledge graph for data integration system; it provides an overview of the system's model and their relationships.

[24] `https://www.ontotext.com/products/graphdb/`

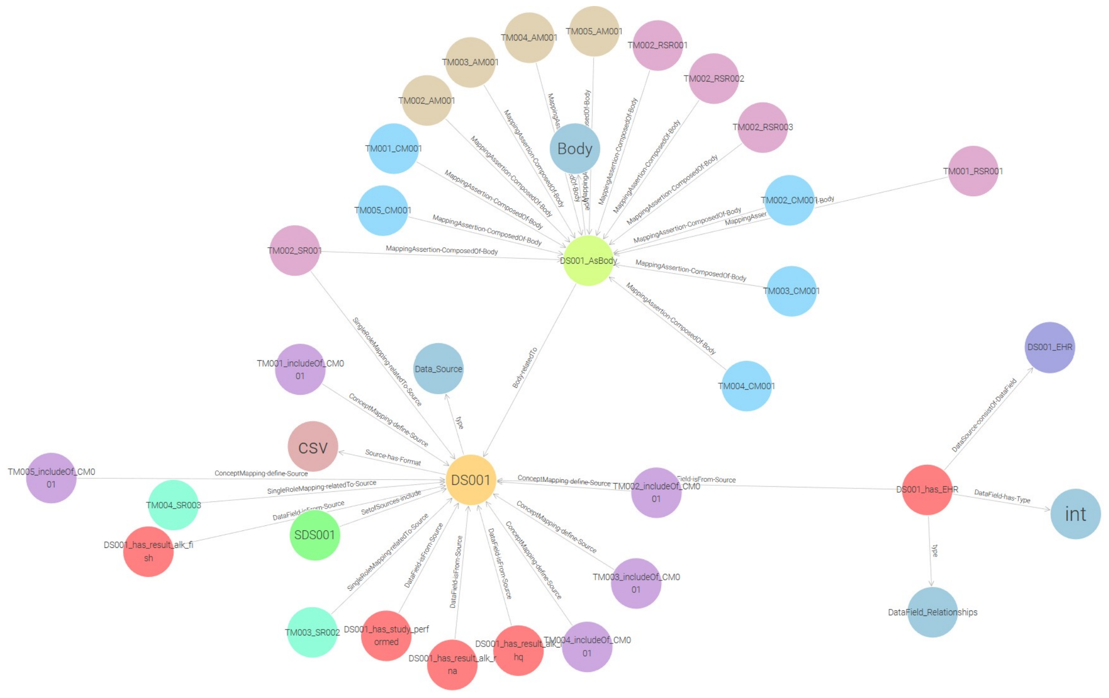


Figure 5.5: Knowledge graph in GraphDB - It shows the location of DS001, all data fields of DS001 and all mapping assertions, which have used DS001 in their bodies as data source.

5.3 Summary of the Chapter

In this chapter, we present the implementation of a data integration system consisting of two key phases: logical design and physical design. The logical design involves creating the ontology in RDF and OWL. In contrast, the physical design builds on the logical design by defining the mapping rules that link the ontology to the data sources and generate the knowledge graph.

Chapter 6

Experimental Evaluation

In this section, the presented approach is evaluated with a combination of proposed modeling with using the knowledge graph concerning the competency questions expressed in Chapter 4 (4.2).
As a proof of concept and to evaluate the effectiveness of our data integration system, we have created a knowledge graph by integrating the data integration system that defines the process of knowledge graph creation in the H2020 funded project [25]. We have represented the CLARIFY data integration system in terms of an ontology, RML mapping rules, and data sources, as shown in Figure 6.1.

6.1 Benchmarks

The knowledge graph that includes data integration system and is created through 6.524 seconds with SDM-RDFizer includes 54 classes, 71 relationships, and 30 attributes, comprising about 5079 RFD triples and a total of 1264 entities. Figure 6.2 provides a visualization of the classes based on the number of instances (entities) for each class, with dis:Class having the most instances in the current version of the KG. Figure 6.3 highlights the top 10 classes and relationships linked to each other and the other classes.

On top of the created KG, we have conducted and evaluation with the aim of assessing the expressiveness of the proposed ontology and method of KG creation. For data integration system performance evaluation, we measured the time it took for the system to analyze the data using ontology. Figure 6.1 reports the results of network analysis, while Table 6.1 shows the specification of the represented ontology

[25]`https://www.clarify2020.eu/`

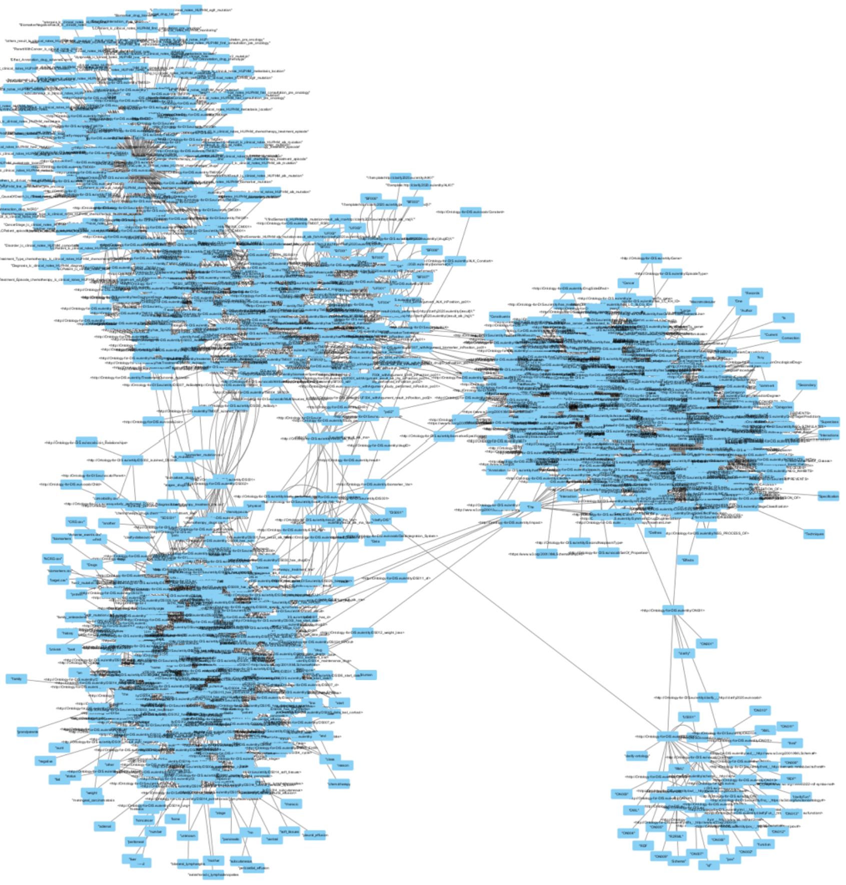


Figure 6.1: Knowledge graph in Cytoscape - It shows the network analysis, such as the number of nodes and edges, network diameter and density, the neighborhood of each node, and many more features.

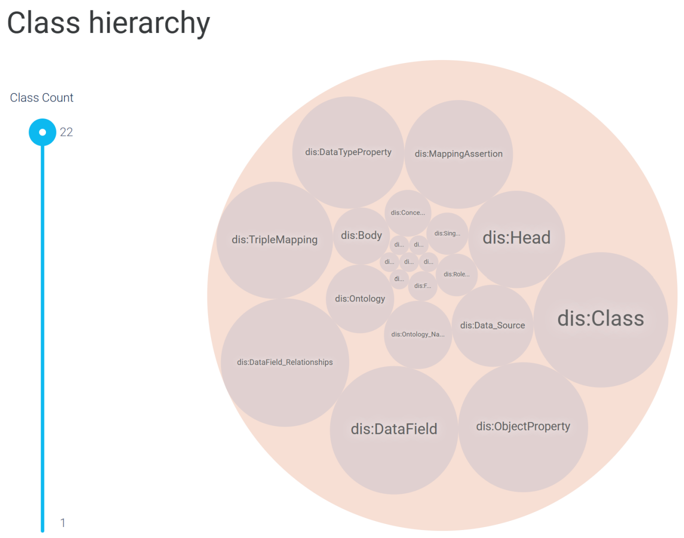
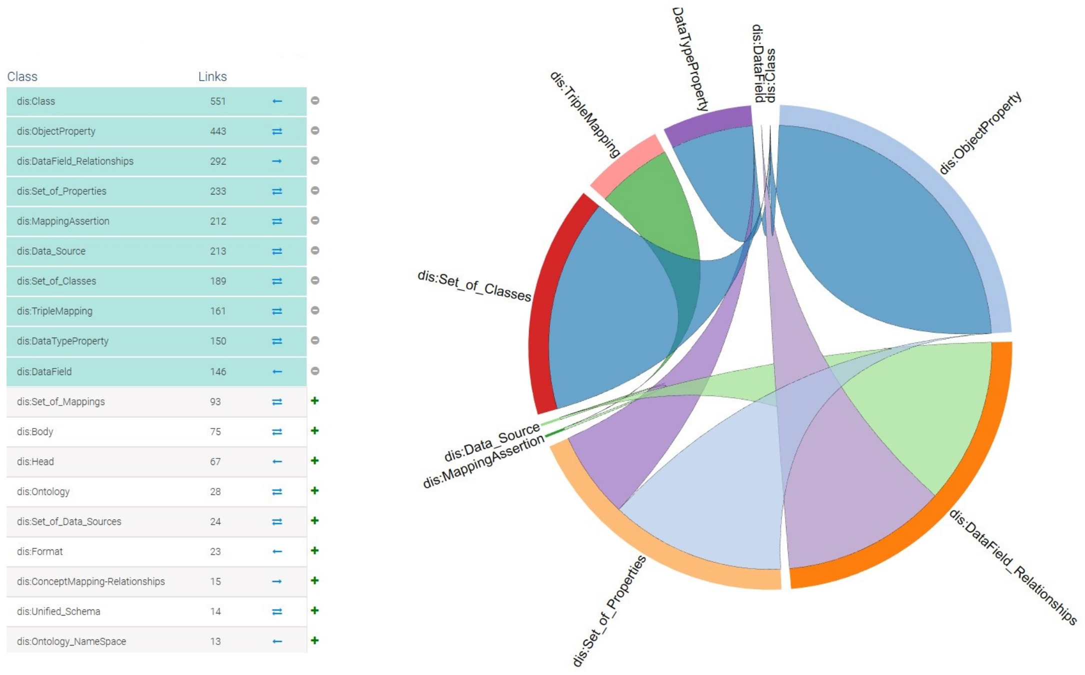


Figure 6.2: Visualization of the classes based on the number of instances (entities) for each class in KG



Class	Links		
dis:Class	551	←	⊖
dis:ObjectProperty	443	⇄	⊖
dis:DataField_Relationships	292	→	⊖
dis:Set_of_Properties	233	⇄	⊖
dis:MappingAssertion	212	⇄	⊖
dis:Data_Source	213	⇄	⊖
dis:Set_of_Classes	189	⇄	⊖
dis:TripleMapping	161	⇄	⊖
dis:DataTypeProperty	150	⇄	⊖
dis:DataField	146	←	⊖
dis:Set_of_Mappings	93	⇄	+
dis:Body	75	⇄	+
dis:Head	67	←	+
dis:Ontology	28	⇄	+
dis:Set_of_Data_Sources	24	⇄	+
dis:Format	23	←	+
dis:ConceptMapping-Relationships	15	→	+
dis:Unified_Schema	14	⇄	+
dis:Ontology_NameSpace	13	←	+

Figure 6.3: Top 10 classes of KG and their relationships linked with each other and other classes

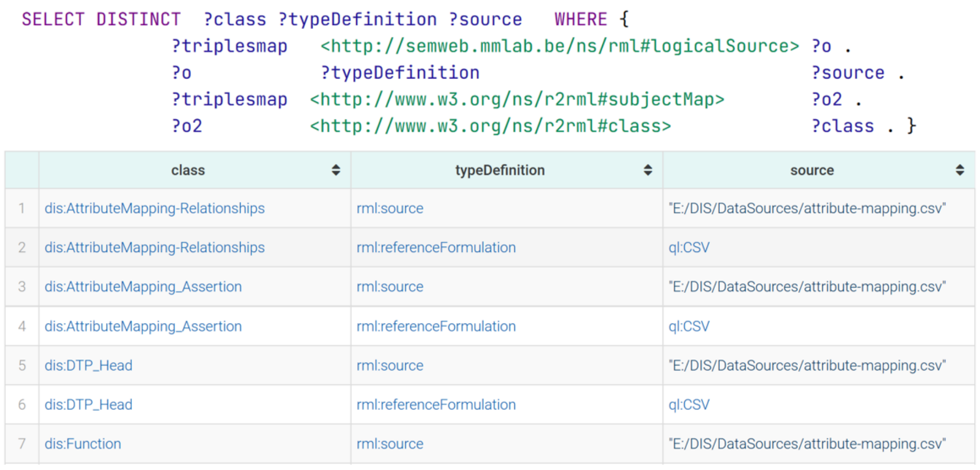
for the data integration system.

Benchmarks	
Number of nodes	1395
Number of edges	3757
Network diameter	7
Network radius	1
Network density	0.002
Analysis time (sec)	11.919

Table 6.1: Benchmarks of data integration system

6.2 SPARQL Queries to Explore RML Mappings of Data Integration System

These SPARQL queries are based on an ontology and RML mapping rules of a data integration system that explain: **i)** the classes defined within the ontology of the data integration system and the data sources, which are used in mapping rules (Figure 6.4, 6.5), **ii)** the number of mapping rules that define each class (Figure 6.6, 6.7), and **iii)** defined datatype properties and object properties within the ontology of the data integration system per class and their object values (Figure 6.8, 6.9).



```
SELECT DISTINCT  ?class ?typeDefinition ?source   WHERE {
              ?triplesmap   <http://semweb.mmlab.be/ns/rml#logicalSource> ?o .
              ?o            ?typeDefinition                               ?source .
              ?triplesmap  <http://www.w3.org/ns/r2rml#subjectMap>        ?o2 .
              ?o2          <http://www.w3.org/ns/r2rml#class>             ?class . }
```

	class	typeDefinition	source
1	dis:AttributeMapping-Relationships	rml:source	"E:/DIS/DataSources/attribute-mapping.csv"
2	dis:AttributeMapping-Relationships	rml:referenceFormulation	ql:CSV
3	dis:AttributeMapping_Assertion	rml:source	"E:/DIS/DataSources/attribute-mapping.csv"
4	dis:AttributeMapping_Assertion	rml:referenceFormulation	ql:CSV
5	dis:DTP_Head	rml:source	"E:/DIS/DataSources/attribute-mapping.csv"
6	dis:DTP_Head	rml:referenceFormulation	ql:CSV
7	dis:Function	rml:source	"E:/DIS/DataSources/attribute-mapping.csv"

Figure 6.4: SPARQL query for classes' definition in terms of mapping assertions

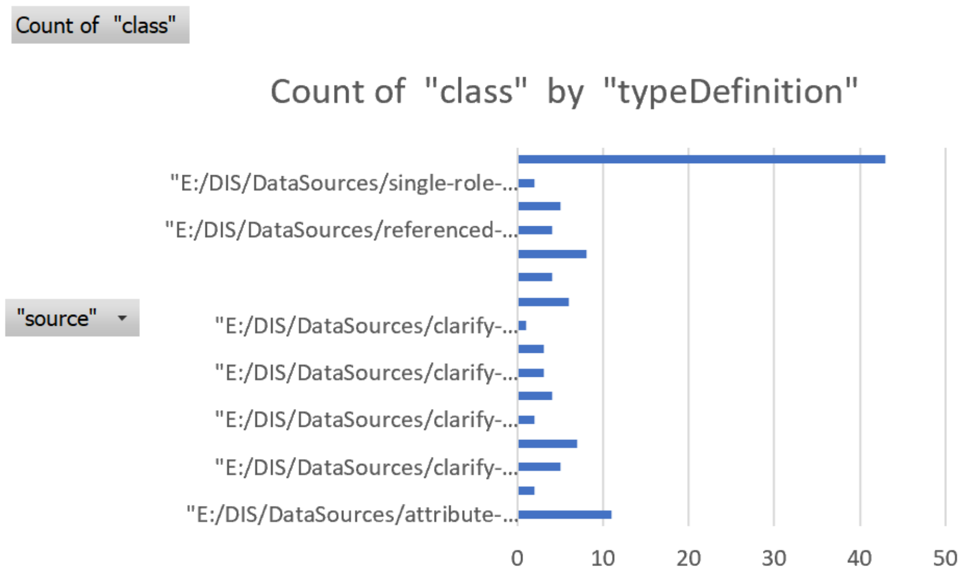
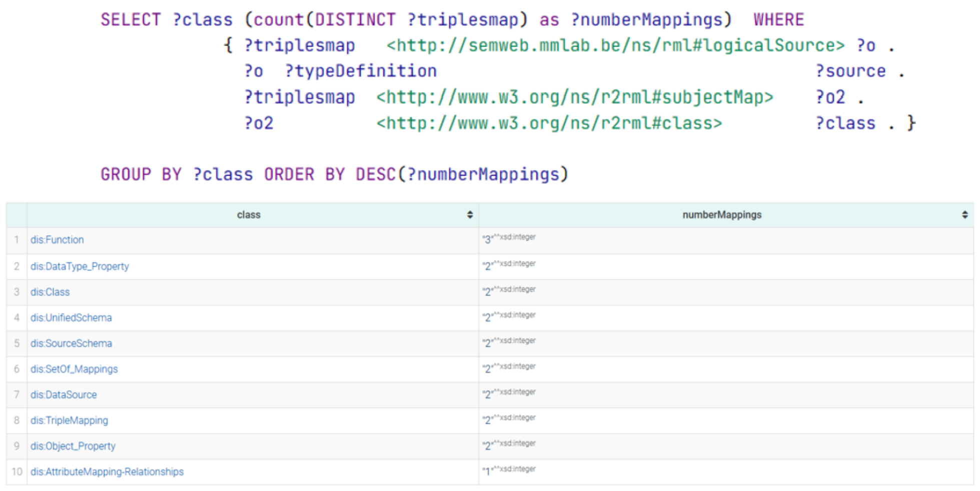


Figure 6.5: This plot shows the number of classes that have used .CSV resourses as rml:source.



```
SELECT ?class (count(DISTINCT ?triplesmap) as ?numberMappings)  WHERE
            { ?triplesmap   <http://semweb.mmlab.be/ns/rml#logicalSource> ?o .
              ?o  ?typeDefinition                                    ?source .
              ?triplesmap  <http://www.w3.org/ns/r2rml#subjectMap>    ?o2 .
              ?o2          <http://www.w3.org/ns/r2rml#class>         ?class . }

GROUP BY ?class ORDER BY DESC(?numberMappings)
```

	class	numberMappings
1	dis:Function	"3"^^xsd:integer
2	dis:DataType_Property	"2"^^xsd:integer
3	dis:Class	"2"^^xsd:integer
4	dis:UnifiedSchema	"2"^^xsd:integer
5	dis:SourceSchema	"2"^^xsd:integer
6	dis:SetOf_Mappings	"2"^^xsd:integer
7	dis:DataSource	"2"^^xsd:integer
8	dis:TripleMapping	"2"^^xsd:integer
9	dis:Object_Property	"2"^^xsd:integer
10	dis:AttributeMapping-Relationships	"1"^^xsd:integer

Figure 6.6: SPARQL query to define number of mapping assertions per class

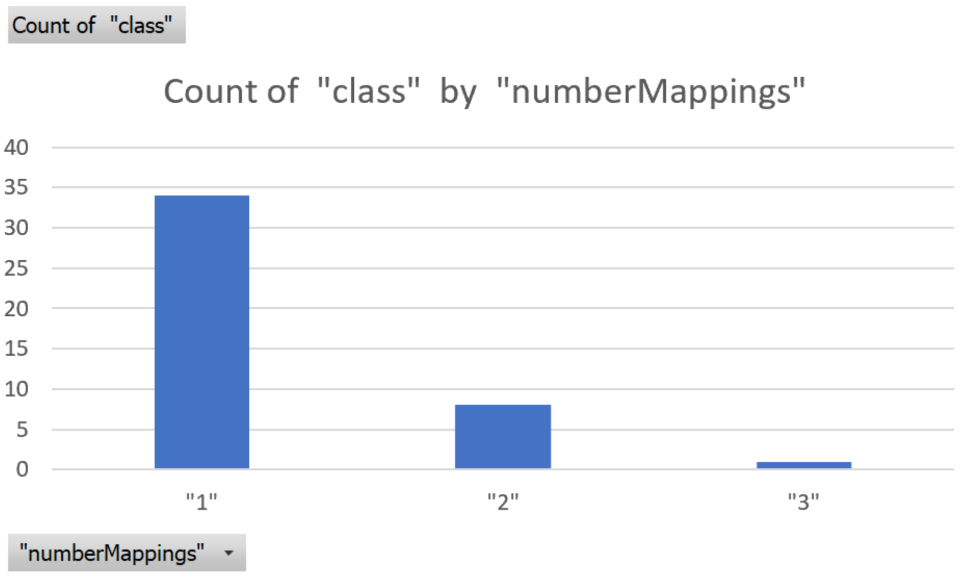
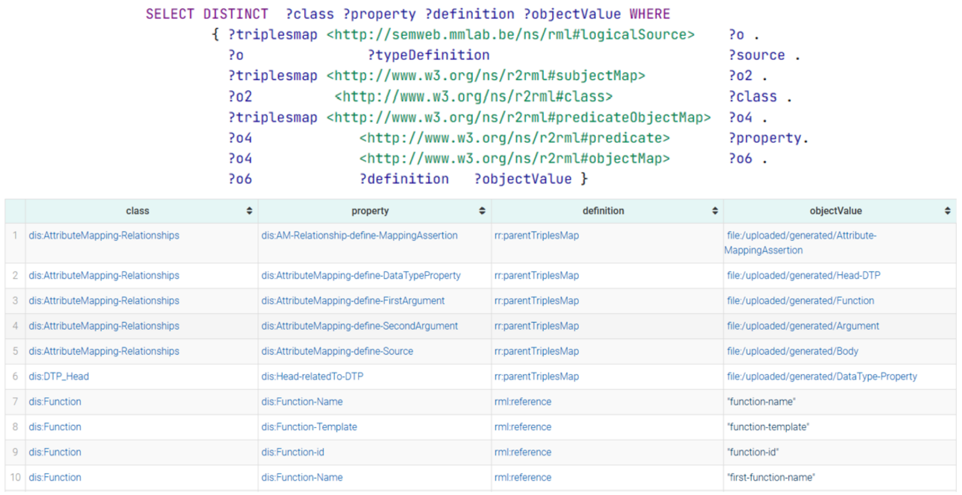


Figure 6.7: This plot shows the number of classes that are used in mapping rules.



```
SELECT DISTINCT  ?class ?property ?definition ?objectValue WHERE
        { ?triplesmap <http://semweb.mmlab.be/ns/rml#logicalSource>    ?o .
          ?o                ?typeDefinition                           ?source .
          ?triplesmap <http://www.w3.org/ns/r2rml#subjectMap>         ?o2 .
          ?o2          <http://www.w3.org/ns/r2rml#class>             ?class .
          ?triplesmap <http://www.w3.org/ns/r2rml#predicateObjectMap> ?o4 .
          ?o4             <http://www.w3.org/ns/r2rml#predicate>      ?property.
          ?o4             <http://www.w3.org/ns/r2rml#objectMap>      ?o6 .
          ?o6             ?definition   ?objectValue }
```

	class	property	definition	objectValue
1	dis:AttributeMapping-Relationships	dis:AM-Relationship-define-MappingAssertion	rr:parentTriplesMap	file:/uploaded/generated/Attribute-MappingAssertion
2	dis:AttributeMapping-Relationships	dis:AttributeMapping-define-DataTypeProperty	rr:parentTriplesMap	file:/uploaded/generated/Head-DTP
3	dis:AttributeMapping-Relationships	dis:AttributeMapping-define-FirstArgument	rr:parentTriplesMap	file:/uploaded/generated/Function
4	dis:AttributeMapping-Relationships	dis:AttributeMapping-define-SecondArgument	rr:parentTriplesMap	file:/uploaded/generated/Argument
5	dis:AttributeMapping-Relationships	dis:AttributeMapping-define-Source	rr:parentTriplesMap	file:/uploaded/generated/Body
6	dis:DTP_Head	dis:Head-relatedTo-DTP	rr:parentTriplesMap	file:/uploaded/generated/DataType-Property
7	dis:Function	dis:Function-Name	rml:reference	"function-name"
8	dis:Function	dis:Function-Template	rml:reference	"function-template"
9	dis:Function	dis:Function-id	rml:reference	"function-id"
10	dis:Function	dis:Function-Name	rml:reference	"first-function-name"

Figure 6.8: SPARQL query to define properties per class

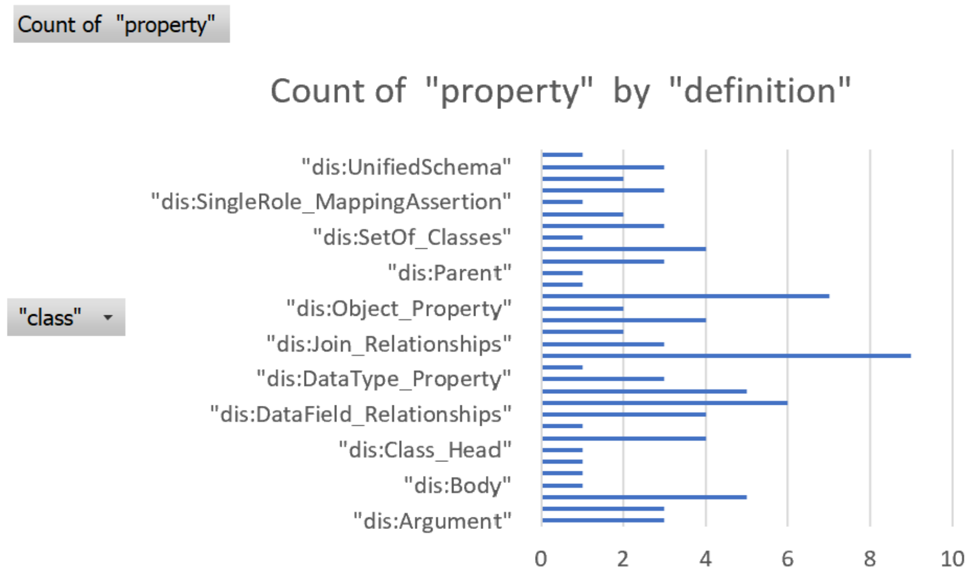
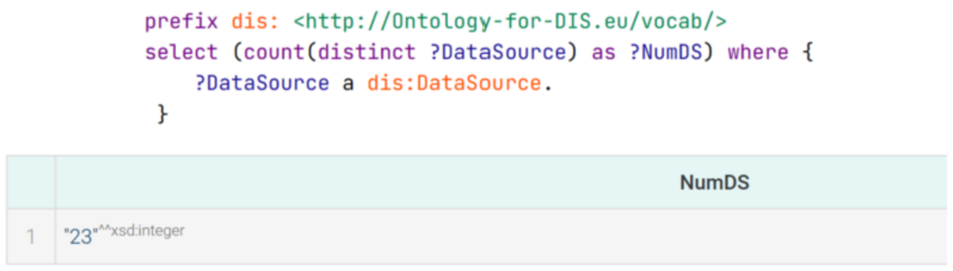


Figure 6.9: This plot shows the number of properties that are defined for each class.

6.3 Satisfaction of Competency Questions

The CLARIFY data integration system comprises 16 different data sources in CSV format, that consist of 33 data sources, 192 classes and 88 Triple maps from CLARIFY. The all details of 9 Triple maps such as mapping assertions and functions are extracted and defined in data integration system's data sources. Based on the created KG, we answer the competency questions; they are specified using SPARQL queries and evaluated on top of the KG.

Q1) What is the number of data sources available, and how can they be counted?

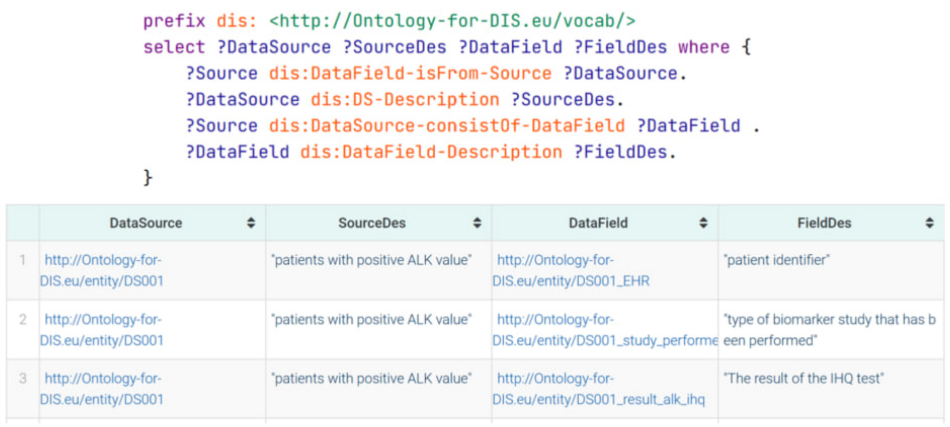


```
prefix dis: <http://Ontology-for-DIS.eu/vocab/>
select (count(distinct ?DataSource) as ?NumDS) where {
    ?DataSource a dis:DataSource.
 }
```

	NumDS
1	"23"^^xsd:integer

Figure 6.10: SPARQL query to define the number of data sources

Q2) What information or data fields do each data source comprise?

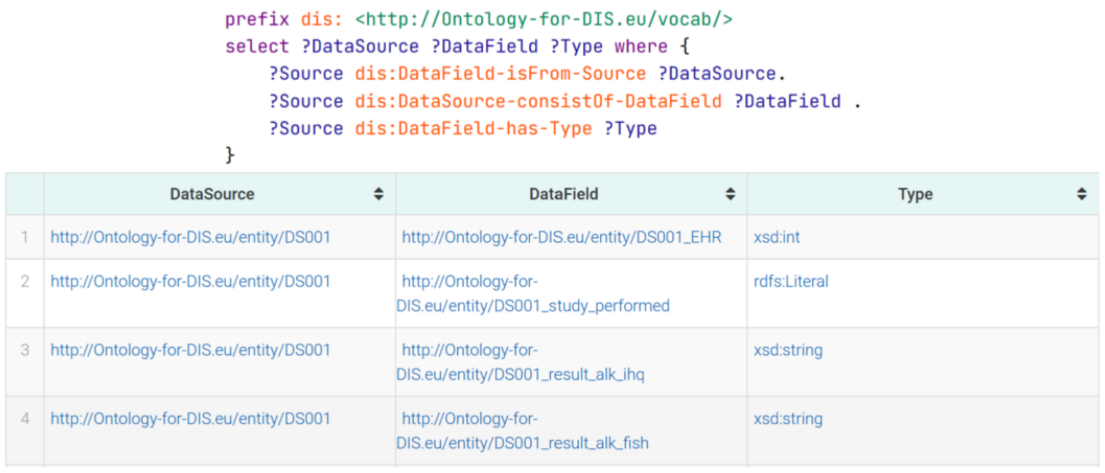


```
prefix dis: <http://Ontology-for-DIS.eu/vocab/>
select ?DataSource ?SourceDes ?DataField ?FieldDes where {
    ?Source dis:DataField-isFrom-Source ?DataSource.
    ?DataSource dis:DS-Description ?SourceDes.
    ?Source dis:DataSource-consistOf-DataField ?DataField .
    ?DataField dis:DataField-Description ?FieldDes.
}
```

	DataSource	SourceDes	DataField	FieldDes
1	http://Ontology-for-DIS.eu/entity/DS001	"patients with positive ALK value"	http://Ontology-for-DIS.eu/entity/DS001_EHR	"patient identifier"
2	http://Ontology-for-DIS.eu/entity/DS001	"patients with positive ALK value"	http://Ontology-for-DIS.eu/entity/DS001_study_performe	"type of biomarker study that has b een performed"
3	http://Ontology-for-DIS.eu/entity/DS001	"patients with positive ALK value"	http://Ontology-for-DIS.eu/entity/DS001_result_alk_ihq	"The result of the IHQ test"

Figure 6.11: SPARQL query to define the description of data source and its data fields

Q3) What are the data types for the fields in each data source?

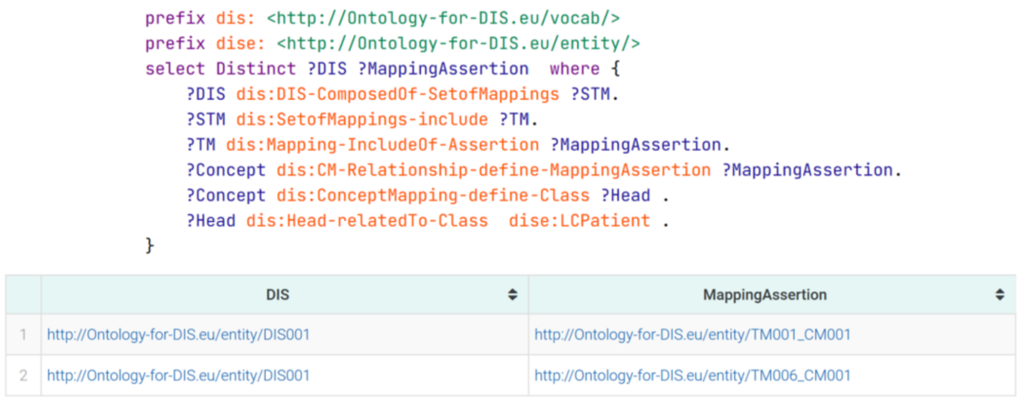


```
prefix dis: <http://Ontology-for-DIS.eu/vocab/>
select ?DataSource ?DataField ?Type where {
    ?Source dis:DataField-isFrom-Source ?DataSource.
    ?Source dis:DataSource-consistOf-DataField ?DataField .
    ?Source dis:DataField-has-Type ?Type
}
```

	DataSource	DataField	Type
1	http://Ontology-for-DIS.eu/entity/DS001	http://Ontology-for-DIS.eu/entity/DS001_EHR	xsd:int
2	http://Ontology-for-DIS.eu/entity/DS001	http://Ontology-for-DIS.eu/entity/DS001_study_performed	rdfs:Literal
3	http://Ontology-for-DIS.eu/entity/DS001	http://Ontology-for-DIS.eu/entity/DS001_result_alk_ihq	xsd:string
4	http://Ontology-for-DIS.eu/entity/DS001	http://Ontology-for-DIS.eu/entity/DS001_result_alk_fish	xsd:string

Figure 6.12: SPARQL query to find the type of the data fields for each data source

Q4) Which mappings have been used for a specific class or property?

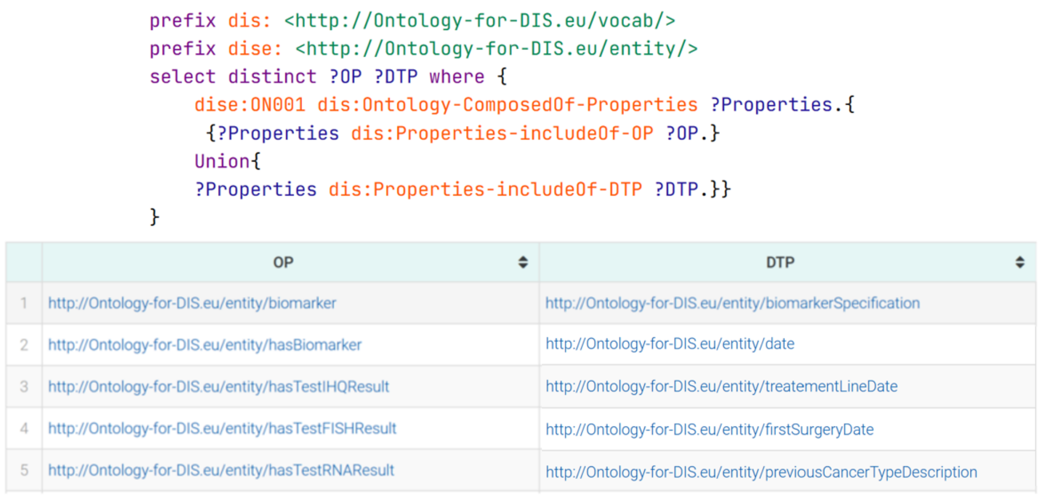


```
prefix dis: <http://Ontology-for-DIS.eu/vocab/>
prefix dise: <http://Ontology-for-DIS.eu/entity/>
select Distinct ?DIS ?MappingAssertion  where {
    ?DIS dis:DIS-ComposedOf-SetofMappings ?STM.
    ?STM dis:SetofMappings-include ?TM.
    ?TM dis:Mapping-IncludeOf-Assertion ?MappingAssertion.
    ?Concept dis:CM-Relationship-define-MappingAssertion ?MappingAssertion.
    ?Concept dis:ConceptMapping-define-Class ?Head .
    ?Head dis:Head-relatedTo-Class  dise:LCPatient .
}
```

	DIS	MappingAssertion
1	http://Ontology-for-DIS.eu/entity/DIS001	http://Ontology-for-DIS.eu/entity/TM001_CM001
2	http://Ontology-for-DIS.eu/entity/DIS001	http://Ontology-for-DIS.eu/entity/TM006_CM001

Figure 6.13: SPARQL query to find the triple mapping and mapping assertion for a LCPatient class

Q5) Which properties are defined in the data integration system ontologies?

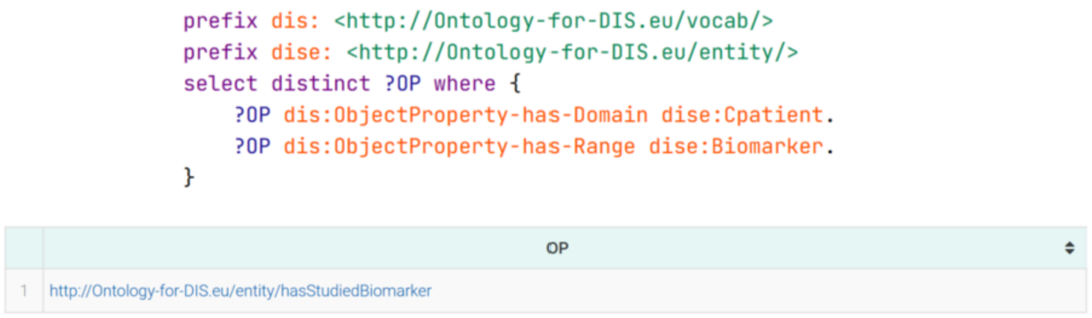


```
prefix dis: <http://Ontology-for-DIS.eu/vocab/>
prefix dise: <http://Ontology-for-DIS.eu/entity/>
select distinct ?OP ?DTP where {
    dise:ON001 dis:Ontology-ComposedOf-Properties ?Properties.{
     {?Properties dis:Properties-includeOf-OP ?OP.}
    Union{
    ?Properties dis:Properties-includeOf-DTP ?DTP.}}
}
```

	OP	DTP
1	http://Ontology-for-DIS.eu/entity/biomarker	http://Ontology-for-DIS.eu/entity/biomarkerSpecification
2	http://Ontology-for-DIS.eu/entity/hasBiomarker	http://Ontology-for-DIS.eu/entity/date
3	http://Ontology-for-DIS.eu/entity/hasTestIHQResult	http://Ontology-for-DIS.eu/entity/treatementLineDate
4	http://Ontology-for-DIS.eu/entity/hasTestFISHResult	http://Ontology-for-DIS.eu/entity/firstSurgeryDate
5	http://Ontology-for-DIS.eu/entity/hasTestRNAResult	http://Ontology-for-DIS.eu/entity/previousCancerTypeDescription

Figure 6.14: SPARQL query to show all defined properties in clarify

Q6) Is there specific binding established between two specific classes?



```
prefix dis: <http://Ontology-for-DIS.eu/vocab/>
prefix dise: <http://Ontology-for-DIS.eu/entity/>
select distinct ?OP where {
    ?OP dis:ObjectProperty-has-Domain dise:Cpatient.
    ?OP dis:ObjectProperty-has-Range dise:Biomarker.
}
```

	OP
1	http://Ontology-for-DIS.eu/entity/hasStudiedBiomarker

Figure 6.15: SPARQL query to find the existence of a object property between the classes Cpatient and Biomarker.

6.4 Summary of the Chapter

This chapter aims to provide insights into testbeds based on benchmarks. Additionally, we present the outcomes of applying our approach to different cases, supported by empirical evidence in the form of graphs and tables. We have processed a part of the clarify data [26] considering the SHACL constraints. Ultimately, we aim to address the competency questions we introduced in the Approach Chapter. Our evaluation results suggest that our data integration system with represented ontology for data integration system with a global view is a promising approach for integrating heterogeneous data from various sources efficiently and accurately. Using ontology for data integration systems can effectively address semantic heterogeneity and improve the integrated data, demonstrating the value of our approach for data integration in various domains.

[26]https://github.com/SDM-TIB/CLARIFY_KG

Chapter 7

Conclusions and Future Work

This thesis addresses the problem of modeling data integration systems using ontologies. Accordingly, an EER-Model is presented to show the entities and their relationships with their cardinalities. Based on the model created, the data integration system ontology, as shown in the Figure 4.9 has complex classes and properties. In this work, we have used the data from clarify version-8 ontology[27] and the clarify mapping rules[28] to create the data sets for the data integration system. The RML mapping language defines mapping rules to establish correspondence between the data sets and the ontology. Based on them, the KG is created and presented format n-triples, which are imported into GraphDB to run the SPARQL queries. The SDM-RDFizer engine [29] was utilized for KG creation.

7.1 Discussions

The evaluation outcomes put into perspective the relevant role of making the definition of data integration respecting the FAIR data principles, i.e., *findability*, *accessability*, *interoperability*, and *reuseability*. Different queries are performed to show the expressiveness of the ontology that models data integration systems. The results are shown in chapter 6.3; the evaluation was guided by the following research questions, presented in chapter 1.1:

RQ1: What are the main features that characterize a knowledge graph defined as a data integration system? A knowledge graph is characterized by several key

[27]`https://github.com/SDM-TIB/CLARIFYUnifiedSchema`
[28]`https://github.com/SDM-TIB/CLARIFY_KG/tree/master/settings/mapping`
[29]`https://github.com/SDM-TIB/SDM-RDFizer`

features, including its representation as a set of nodes and edges and the ontology that define types of entities and relationships that can be represented in a KG.

RQ2: What are the main challenges and limitations in documenting data integration systems? We have identified that large and heterogeneous data sources, security, quality of data, processing query, autonomy.

RQ3: What are the main data integration approaches? Although there are two approaches for defining a data integration system (i.e., Local as view (LAV) and global as view (GAV)), the existing approaches for mapping assertion definition follow the GAV approach.

RQ4: How does the specification of data integration systems impact tasks of data management and knowledge graph creation? The specification of the ontology improves the data Fair(findability, accessibility, interpretability and re-useability)

RQ5: What is the relevant metadata that characterize a data integration system? Data sources, unified schema, mapping rules and, functions. DIS=$\langle G, S, M, F \rangle$

RQ6: Which type of concepts should be defined in a data integration system? How are they connected to each other? For each concept, there are several classes. For example, for data source concept: sources schema, data source, format and data field. The whole relationship is shown in Figure 4.9

RQ7: What is the role of declarative mapping languages in the documentation of data integration systems? Languages for declaratively defining mapping assertions (e.g., RML) enable to state the main correspondences among concepts in unified ontologies and attributes from heterogeneous data sources.

7.2 Limitations

Semantic web-based data integration systems has emerged as promising alternative to specify data integration system. Still, the specification of the main components of a data integration system require a high expertise and large amounts of training data, respectively.
Ontology modeling is a complex process that demands expertise in domain knowledge and ontology languages. Modeling an ontology for a data integration system involves identifying the relationships between different data sources and creating a unified schema that can accommodate all the data. This process can be time-consuming and may require a significant effort to ensure the ontology is accurate

and comprehensive.
Ontology modeling can limit the modeling of evolving data integration systems. Maintaining and updating an ontology for a data integration system can be challenging. As new data sources become available or existing sources change, the ontology must be updated to accommodate these changes. This process requires ongoing effort and resources, which are only sometimes available. Additionally, changes to the ontology may impact the performance of the data integration system, requiring additional testing and optimization.

7.3 Future Work

We suggest researching the following topics to overcome the limitations previously outlined.

- Linking the *Data Integration System (DIS)* to *Change Detection Ontology (CDO)*[24] can lead to better results in data management and data quality. Thus, changes can be detected based on the Change Detection Ontology (CDO). Then, we can track and find valuable data sources close to our new data and update the current data sources instead of creating new ones. This approach improves re-useability, which is one of our objectives.

- Currently, data sources are specified in CSV format and have been created manually; however, this process could be more efficient. Web scraping could be applied for reading and extracting the needed data from all mappings and ontologies in the web-based. Searches could be guided by keywords such as *owl:Class*, *owl:DataTypeProperty*, *owl:ObjectProperty* in ontologies and *rml:logicalSource*, *rr:predicateObjectMap* and *rr:objectMap* in RML mappings and other keys, would be much more efficient and practical to create valuable data sources for a data integration system.

7.4 Summary of the Chapter

This chapter discussed the experimental results and conclusions reached during the execution of this thesis- Furthermore, future works are outlined; they can meaningfully improve the outcomes and usage of this project.

Bibliography

[1] Jeelani Ahmed and Muqeem Ahmed. "Semantic Web Approach of Integrating Big Data- A Review". In: *International Journal of Computer Sciences and Engineering* 6 (Sept. 2018), pp. 529–532. DOI: 10.26438/ijcse/v6i9.529532.

[2] William Ward Armstrong. "Dependency Structures of Data Base Relationships". In: *IFIP Congress*. 1974.

[3] Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler. "Introduction". In: *An Introduction to Description Logic*. Cambridge University Press, 2017, pp. 1–9. DOI: 10.1017/9781139025355.001.

[4] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web : a new form of Web content that is meaningful to computers will unleash a revolution of new possibilities". In: *Scientific American* 284.5 (2001), pp. 34–43. ISSN: 0036-8733. URL: https://www.jstor.org/stable/26059207 (visited on 12/22/2018).

[5] Mokrane Bouzeghoub and Veronika Peralta. "A Framework for Analysis of Data Freshness". In: June 2004, pp. 59–67. DOI: 10.1145/1012453.1012464.

[6] Mokrane Bouzeghoub and Veronika Peralta. "A Framework for Analysis of Data Freshness". In: June 2004, pp. 59–67. DOI: 10.1145/1012453.1012464.

[7] Henrik Dibowski, Stefan Schmid, Yulia Svetashova, Cory Henson, and Tuan Tran. "Using Semantic Technologies to Manage a Data Lake: Data Catalog, Provenance and Access Control". In: Nov. 2020.

[8] Henrik Dibowski, Stefan Schmid, Yulia Svetashova, Cory Henson, and Tuan Tran. "Using Semantic Technologies to Manage a Data Lake: Data Catalog, Provenance and Access Control". In: Nov. 2020.

[9] Marc Ehrig and York Sure. "Ontology Mapping by Axioms (OMA)". In: *Professional Knowledge Management*. Ed. by Klaus-Dieter Althoff, Andreas Dengel, Ralph Bergmann, Markus Nick, and Thomas Roth-Berghofer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 560–569.

[10] Pascal Hitzler, Sebastian Bader, and Artur Garcez. "Ontology Learning as a Use-Case for Neural-Symbolic Integration (position paper)". In: (Jan. 2005).

[11] Richard Hull. "Managing semantic heterogeneity in databases: a theoretical prospective". In: *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 1997.

[12] Enrique Iglesias, Samaneh Jozashoori, David Chaves-Fraga, Diego Collarana, and Maria-Esther Vidal. "SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs". In: CIKM '20. Virtual Event, Ireland: Association for Computing Machinery, 2020, pp. 3039–3046. ISBN: 9781450368599. DOI: 10.1145/3340531.3412881. URL: https://doi.org/10.1145/3340531.3412881.

[13] Enrique Iglesias, Samaneh Jozashoori, and Maria-Esther Vidal. "Scaling Up Knowledge Graph Creation to Large and Heterogeneous Data Sources". In: *CoRR* abs/2201.09694 (2022). arXiv: 2201.09694. URL: https://arxiv.org/abs/2201.09694.

[14] Enrique Iglesias, Samaneh Jozashoori, and Maria-Esther Vidal. "Scaling up knowledge graph creation to large and heterogeneous data sources". In: *Journal of Web Semantics* 75 (2023), p. 100755. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2022.100755. URL: https://www.sciencedirect.com/science/article/pii/S1570826822000397.

[15] Ana Iglesias-Molina, Andrea Cimmino Arriaga, Edna Ruckhaus, David Chaves-Fraga, Raúl García Castro, and Oscar Corcho. "An Ontological Approach for Representing Declarative Mapping Languages". In: *Semantic Web* (Jan. 2022).

[16] Annika Jacobsen, Ricardo de Miranda Azevedo, Navtej Juty, Dominique Batista, Simon Coles, Ronald Cornet, Mélanie Courtot, Merce Crosas, Michel Dumontier, Chris Evelo, Carole Goble, Giancarlo Guizzardi, Karsten Kryger Hansen, Ali Hasnain, Kristina Hettne, Rob Hooft, Melanie Imming, Keith Jeffery, and Erik Schultes. "FAIR Principles: Interpretations and Implementation Considerations". In: (Jan. 2020), pp. 10–29. DOI: 10.1162/dint_r_00024.

[17] Samaneh Jozashoori, Enrique Iglesias, and Maria-Esther Vidal. "Dragoman: Efficiently Evaluating Declarative Mapping Languages over Frameworks for Knowledge Graph Creation". In: *arXiv preprint arXiv:2210.15645* (2022).

[18] Samaneh Jozashoori and Maria-Esther Vidal. "MapSDI: A Scaled-Up Semantic Data Integration Framework for Knowledge Graph Creation". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 58–75. DOI: 10.1007/978-3-030-33246-4_4. URL: https://doi.org/10.1007%2F978-3-030-33246-4_4.

[19] Laks Lakshmanan and Fereidoon Sadri. "Information Integration and the Semantic Web". In: (Jan. 2004).

[20] Maurizio Lenzerini. "Data Integration: A Theoretical Perspective". In: June 2002, pp. 233–246. DOI: 10.1145/543613.543644.

[21] Sean Martin, Benjamin Szekely, and Dean T. Allemang. "The Rise of the Knowledge Graph". In: 2021.

[22] Eric Miller. "An introduction to the resource description framework." In: *D-lib Magazine* (1998).

[23] Manuel Namici and Giuseppe De Giacomo. "Comparing Query Answering in OBDA Tools over W3C-Compliant Specifications". In: *Description Logics* (2018).

[24] Alex Randles1 and Declan O'Sullivan1. "Modelling and Analyzing Changes within LD Source Data". In: 2022.

[25] Muhammad Rizwan Saeed, Charalampos Chelmis, and Viktor K. Prasanna. "Chapter 9 - Automatic Integration and Querying of Semantic Rich Heterogeneous Data: Laying the Foundations for Semantic Web of Things". In: *Managing the Web of Things*. Ed. by Quan Z. Sheng, Yongrui Qin, Lina Yao, and Boualem Benatallah. Boston: Morgan Kaufmann, 2017, pp. 251–273. ISBN: 978-0-12-809764-9. DOI: `https://doi.org/10.1016/B978-0-12-809764-9.00012-3`. URL: `https://www.sciencedirect.com/science/article/pii/B9780128097649000123`.

[26] Jeffrey D. Ullman. "Information integration using logical views". In: *Theoretical Computer Science*. 1997.

[27] Jürgen Umbrich, Michael Hausenblas, Aidan Hogan, Axel Polleres, and S. Decker. "Towards Dataset Dynamics: Change Frequency of Linked Open Data Sources". In: *LDOW*. 2010.

[28] Jürgen Umbrich, Boris Villazon Terrazas, and Michael Hausenblas. "Dataset Dynamics Compendium: A Comparative Study". In: *CEUR Workshop Proceedings* 665 (Jan. 2010).

[29] Maria-Esther Vidal. "Lecture - Responsible Data Management, Modeling Research Data". In: 2020.

[30] Maria-Esther Vidal. "Lecture - Scientific Data Management and Knowledge Graphs". In: Dec. 2022.

[31] Mark Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gaby Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Olavo Bonino da Silva Santos, Philip Bourne, Jildau Bouwman, Anthony Brookes, Tim Clark, Merce Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris Evelo, Richard Finkers, and Barend Mons. "The FAIR Guiding Principles for scientific data management and stewardship". In: *Scientific Data* 3 (Mar. 2016). DOI: `10.1038/sdata.2016.18`.