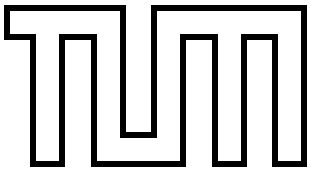


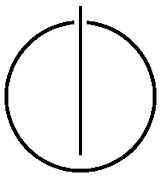
FAKULTÄT FÜR INFORMATIK

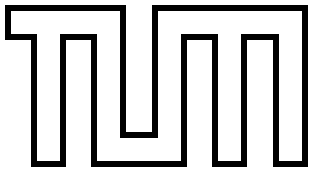
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Masterarbeit in Informatik

Analysis and design of a semantic modeling language to describe public data sources

Branislav Vidojevic





FAKULTÄT FÜR INFORMATIK

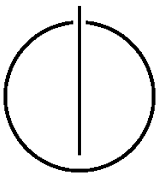
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Masterarbeit in Informatik

Analysis and design of a semantic modeling language to describe public data sources

Analyse und Design einer semantischen Modellierungssprache zur Beschreibung öffentlicher Datensätze

Author:	Branislav Vidojevic
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	M.Sc. Patrick Holl
Date:	March 15, 2019



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, March 15, 2019 Branislav Vidojevic

Acknowledgments

First and foremost, I am genuinely grateful to M.Sc. Patrick Holl, my advisor at the Technical University of Munich, for providing me the opportunity to work on my thesis under his supervision. Also, I would like to thank him for his guidance, discussions, mentoring, and continuous support throughout the writing of this thesis.

Next, I would like to thank Prof. Dr. Florian Matthes, my supervisor at the Technical University of Munich, for his valuable research guidelines and quick responses to my queries. His supervision was of great help in completing this thesis.

Finally, I want to thank the chair for Software Engineering for Business Information Systems (SEBIS) for organizing Advanced Seminar and providing other students and me with appreciated feedback and research guidance.

Finally, I want to thank my family and friends who supported me continuously.

Abstract

Ever since Tim Berners-Lee introduced the Semantic Web term, the research community, and afterward also the companies, have changed the way how they perceive data on the Internet. As the Internet expands rapidly, a need arose to organize and interconnect existing data with the new data that is coming into the Internet. Semantic Web has a significant influence in this area as it comes in the form of an add-on onto the World Wide Web and provides a set of standards and protocols for data sharing and reusing across the Internet. Since utilizing the benefits of Semantic Web could not be as fast as harvesting the benefits of data sharing, data integration, and remote data access, solutions that use the power of the Semantic Web are not as widely in use as other solutions. What will be given in this Master Thesis is an overview of state-of-the-art solutions in the area of data integration with the emphasis on the handling of semantic metadata. Based on that analysis, we give a recommendation on how to semantically annotate data that resides in multiple disparate open data sources. Afterward, we describe the process of implementation of a library, which can help in solving problems as such. In the end, we utilize the developed solution in the existing platform for data integration where we describe benefits and downfalls of solutions that use the Semantic Web.

Keywords

Semantic Web, Data Integration, Data Engineering, Linked Data, Open Data, Web of Data, JSON, JSON-LD, REST, MIDAS

Contents

Outline of the Thesis

CHAPTER 1: INTRODUCTION

This chapter introduces the topic of the thesis. It presents the thesis motivation, the objectives, the research questions it addresses and research method that will be applied.

CHAPTER 2: FUNDAMENTALS

In this chapter, we provide a short introduction to Web of Data, in particular, Open Data, Linked Data, RESTful services, JSON, JSON for Linked Data and Metadata. All concepts are important so we can fully understand how the Web of Data functions today and what drives Web to expand and transform continuously. We will show where data resides and how it is transferred on the Internet.

CHAPTER 3: RELATED WORK

In this Chapter, we give an overview of Data Integration platforms that offer functionalities for managing metadata, in addition to data management and integration. Also, we describe several platforms of interest in details. We also describe initiatives that are founded with a purpose to tackle the rising amount of structured and unstructured data that needs to be exchanged between large entities. Finally, we list key points that are important for a platform that manages metadata.

CHAPTER 4: MIDAS PLATFORM

This Chapter covers the design architecture of the Midas platform with a description of each module contained in the platform. Also, we give an overview of all features that Midas platform provides. In the end, we will give a short description of design decisions that are taken into account for the development of user interface for Midas platform.

CHAPTER 5: IMPLEMENTATION

This chapter presents the technology stack used in the development of the new Midas platform module - Midas Metadata module. Also, we give an overview of all the features that the new module provides. We also describe the deployment process and how module functionalities are documented.

CHAPTER 6: EVALUATION

In this Chapter, we will propose a simple method for measuring the fitness of an entity from Schema.org vocabulary to describe an unlabeled property of a dataset. Also, we

will apply that simple method on 22 Open Data datasets that are supported in the Midas platform and give comments on gathered results.

CHAPTER 7: CONCLUSION

In this last Chapter, we provide a summary of this thesis with a retrospective onto defined research questions. Finally, we give potential directions for future work.

1. Introduction

This chapter introduces the topic of the thesis. It presents the thesis motivation, the objectives, the research questions it addresses and research method that will be applied.

1.1. Motivation

Since the start of the ubiquitous use of the internet, we are collecting digital data to enable business strategies [1]. In addition to that, a vast amount of data is generated and published online, in many different formats, structure, quality and accessibility [2]. Many devices are continually producing and distributing data, e.g., sensors in mobile phones, cars, smart houses and many more. Increasing data volume does not lead to a simultaneous increase in understanding of those data, so the exploration of new methods and processes is necessary [3].

The phenomenon of processing a vast amount of data nowadays is referred to as Big Data. Big Data is a term describing the storage and analysis of large and or complex data sets using a series of techniques including, but not limited to NoSQL, MapReduce, and machine learning [4]. For some time, many had a misconception that Big Data is a black box where processing of the enormous amount of data takes place and where the result of processing is data insights [5]. In practice, data scientists are spending a significant amount of time to prepare their data for analysis and exploration.

In the process of data preparation, data scientists have to load data from various data sources, transform data in a way that data is suitable for processing by specific tools and methods [6]. Getting to know the data may be an extensive process and sometimes requires a domain expert for data interpretation.

Today, we can find many free datasets, in open format and available for reuse, commonly known as Open Data datasets [7, 8]. Even though they are publicly accessible, it takes time to explore datasets and to check the quality of the data. Some data could be rated five stars using Tim Berners-Lee scale for data quality assessment [9] but most of the time, data vary in quality, while the recommendation is to have at least data rated three stars [10].

To minimize the time needed for getting to know the data and data preprocessing, we will propose a solution that will leverage the Semantic Web. Therefore, in this thesis, we study the possibility of using vocabulary Schema.org to annotate Open Data datasets that can be used to enrich ”local” data. Even though there are many benefits of using semantic

annotations, a substantial amount of work remains to be done to improve the current state of research in the area of supporting semantic web services [11].

Semantically annotated data is more straightforward to query and manage. It enables matching of semantically equivalent entities from different datasets thus making it possible to join disparate datasets. Also, semantically annotated data is more straightforward to query, because machines can understand semantic annotations. Besides that, it makes it possible to reuse queries, because queries can reference attributes not only by attribute names but also by their semantic meaning.

1.2. Research Questions

This thesis will give answers to the following research questions (RQ):

- **RQ 1 - How are state-of-the-art solutions handling metadata?**

  Many businesses today are data-driven. Sometimes using only data is not enough, so enrichment of data with metadata often takes place. Metadata can be used to improve internal processes or to provide users with more information. We want to find out how some data enrichment and data handling platforms are using metadata. We are interested in why they use metadata, how they store it, what are benefits and downfalls.

- **RQ 2 - How to add metadata to existing columnar data?**

  Having the data without metadata can make it challenging to add metadata at a later stage. Making the system that can add metadata to data without disrupting the existing system seems like a great challenge.

- **RQ 3 - Where to store metadata?**

  Once when we know why we use metadata, which metadata is useful and how to improve processes, we have to think of where to store metadata and how to manage it.

- **RQ 4 - How to leverage public vocabularies in metadata management?**

  The usual practice for semantic annotation of data on the Internet is to use some public vocabulary to describe data. For that purpose, the creation of many vocabularies took place, where some are domain specific, and others are formalizing the knowledge about general terms. Some public vocabularies, like Schema.org, are supported and used by companies whose work affects millions of people.

1.3. Research Method

Design Science Research seeks to find answers for questions of a specific problem domain by deriving knowledge from the inception and evaluation of an innovative artifact that provides a possible solution [12, 13]. In Information Systems research, typical artifacts are models, methods, and prototypical systems [13]. This thesis will be based on the Design Science Research Methodology (DSRM) of Peffers et al. [12].

Its nominal process sequence consists of six process steps [12]:

- **Problem identification and motivation**

  We have explained our motivation in Section 1.1. While using many data integration tools, we often had doubts about what some specific attribute of a dataset represents. Usually, the solution to that problem was mitigated by asking someone who already knows the datasets or by extensive online search. At that time, we were curious to see if this problem already has a solution.

- **Definition of the objectives for a solution**

  Section 1.2 contains research question we want to tackle in this thesis. We set our objectives in relation to the problem definition and our knowledge of what is possible and feasible, based on the analysis of a variety of industry data integration tools. By analyzing well-established platforms, we can see the benefits and downsides of different approaches that are used in those platforms.

- **Design and development**

  Enriched with the knowledge about industry platforms and their solutions, and with fundamental knowledge about related research areas, we can design and develop a solution that represents an artifact in the form of a computer software library.

- **Demonstration**

  The functionalities of the proposed solution are explained in this thesis. We provide examples where it is applicable, with appropriate comments. Also, the solution created in this thesis is used for making of a live demo showcase.

- **Evaluation**

  In this thesis, we evaluate the proposed solution by annotating 22 different Open Data datasets. To be able to evaluate our research findings, we propose a scoring method for measuring if a dataset is properly semantically annotated.

- **Communication**

  In the end, we communicate our research findings so others can benefit from it. Our research findings are documented in this thesis and the presentation, both publicly

available on the official website of the SEBIS chair. Besides that, the programming code of the solution created as a result of this thesis is well documented, and it is possible to extend it further.

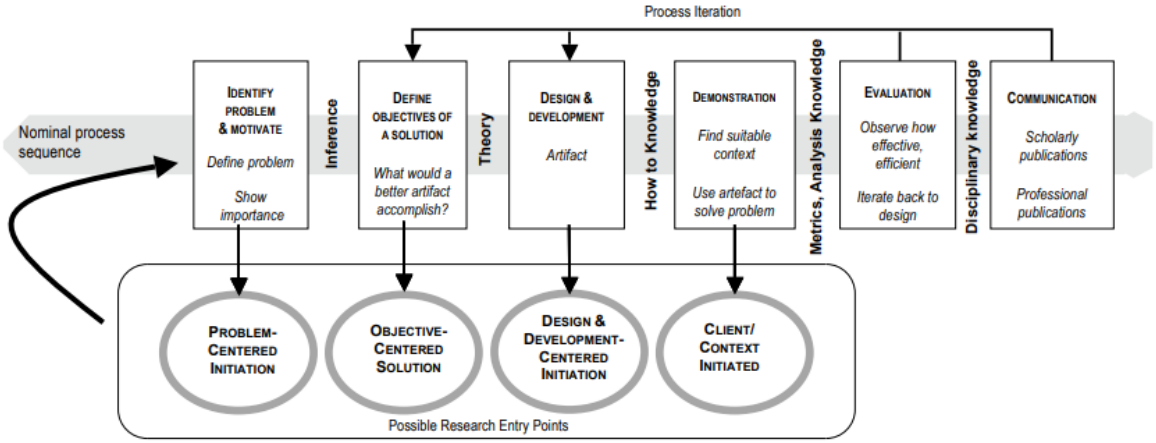


Figure 1.1.: Design Science Research Methodology Process Model taken from [12]

The methodology can be approached from different research points, as we can see in Figure 1.1:

- **Problem-centered Initiation** - We define problems in current Midas platform solution that we want to solve;
- **Objective-centered Initiation** - We propose an artifact (solution) than can help in overcoming defined problems;
- **Design and development-centered Initiation** - Through iterative and incremental development, we provide working prototype that can be used as a solution;
- **Client-context Initiation** - By utilizing created solution, we solve previously defined problems.

2. Fundamentals

In this chapter, we provide a short introduction to Web of Data, in particular, Open Data, Linked Data, RESTful services, JSON, JSON for Linked Data and Metadata. All concepts are essential, so we can fully understand how the Web of Data functions today and what drives Web to expand and transform continuously. We will show where data resides and how it is transferred on the Internet.

2.1. Open Data

Open data is the idea that some data should be freely available to everyone to use and republish as they wish, without restrictions from copyright, patents or other mechanisms of control [7]. It is a long-lasting mission of many government and non-government institutions to open data by default.

With the growth of the World Wide Web, Open Data initiative also got traction. As more content is put on the Internet, a need for data sharing was recognized, and with it, Open Data initiative was identified as an essential aspect of the World Wide Web.

There are several important aspects that we have to consider when we talk about Open Data. Sir Tim Berners-Lee, the inventor of the World Wide Web, introduced a 5-star deployment scheme for Open Data [14, 9]:

- **1 star** - Make your data available on the Web (whatever format) under an open license;
- **2 star** - Make your data available as structured data (e.g. Excel instead of image scan of a table);
- **3 star** - Make your data available in a non-proprietary open format (e.g. CSV or JSON instead of Excel)
- **4 star** - Use Unique Resource Identifiers (URIs) to denote things, so that people can point at your data;
- **5 star** - Link your data to other data to provide context.

From the scale above we can see that it is relatively easy to make data open. The challenge is actually to open the data to be always available, under an open license, in the

right open source format, structured using URIs with provided context. Today, we have different regulatory bodies that can certify the quality of data and most of them are using the 5-star scale as a basis for open data quality assessment. One of those is Open Data Certificate tool[1] developed by Open Data Institute. This tool can create a certificate that can help others to understand the quality of the data [14].

There are many benefits to making data open. One of the most significant movements in the Open Data initiative is the opening of government data. Every country collects and processes data from which many stakeholders may benefit - from citizens, startups, corporations to state institutions. In many countries, Open Data is identified as a great opportunity which can drive the economy further in the right direction. One of those countries is Germany. On July 13, 2017, Germany's first Open Data law came into effect, finally enabling free access to government data [15]. It was a result of the 4-year long effort of many stakeholders. As a law, it can be continuously discussed, adjusted and improved based on the evolving nature of Open Data and the needs of the Open Data consumers.

Open data has no value in itself; it only becomes valuable when used [16]. Data should be open by default, but in the way that it can be used to give a value to the user. Also, by 2020, it is forecasted that European Union will have just under 100,000 Open Data jobs, which represents a 32% growth over five years [2].

2.2. RESTful services

REST stands for Representational State Transfer, a term coined by Roy Fielding in 2000. REST is a software architectural style that defines a set of best practices and constraints to be used for creating Web services. Web services that conform to the REST architectural style termed RESTful Web services provide interoperability between computer systems on the Internet [17].

REST service can be called via the Internet by initiating a standardized HTTP method request with or without payload. It will provide a response in the desired or available format. HTTP methods that can be used in REST context are[17] [3]:

- **GET** - The GET method means retrieve whatever information (in the form of an entity) is identified by the Request Unique Resource Identifier (URI);
- **HEAD** - The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response;
- **POST** - Usually used to create a new resource on the remote server. The new resource is included as a part of the request - request payload;

[1] `https://certificates.theodi.org/en/about/badgelevels` (31.01.2019.)

[2] `https://www.europeandataportal.eu/en/highlights/creating-value-through-open-data` (02.02.2019)

[3] `https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html` (15.01.2019)

- **PUT** - Usually used to update existing resource on the remote server by creating a new version of it;
- **PATCH** - Used to partially update remote resource;
- **DELETE** - Used to delete remote resource. It should not have a request payload;
- **OPTIONS** - This method allows the client to determine the options or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.

By leveraging existing HTTP methods, it makes REST services more accessible to consume and adapt to the current Internet protocols. The list above represents only a recommendation and desired behavior by following the REST architectural style. This style helps developers/consumers to interact with remote resources easily.

Several constraints should be taken into account when developing software and using REST services[4]:

- **Uniform interface** - A resource in the system should have only one logical URI, and that should provide a way to fetch related or additional data;
- **Client-server** - This means that client application and server application MUST be able to evolve separately without any dependency on each other. A client should know only resource URIs;
- **Stateless** - Server will not store anything about latest HTTP request client made. It will treat every request as new;
- **Cacheable** - Caching shall be applied to resources when applicable, and then these resources MUST declare themselves cacheable. Caching can be implemented on the server or client side;
- **Layered system** - REST architecture allows usage of a layered system architecture where, e.g., organizations deploy the APIs on server A, and store data on server B and authenticate requests in Server C.

All constraints serve as a guideline to make an extensible and maintainable system that will be easy to use.

[4] `https://restfulapi.net/rest-architectural-constraints/` (13.01.2019.)

2.3. JSON

JSON stands for JavaScript Object Notation (JSON) and it represents an open-standard file format[5]. JSON format is actually a text format and it is completely platform independent.

JSON structure makes it perfect for data interchange. It is a tree-like structure of key-value pairs. JSON is built on two structures:

- **Collection of name/value pairs** - In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array;
- **List of values** - In most languages, this is realized as an array, vector, list, or sequence.

A value in JSON attribute can be [6]:

- **Object** - It is wrapped in curly brackets; key and a value are separated by a colon, as you can see in the Figure 2.1;

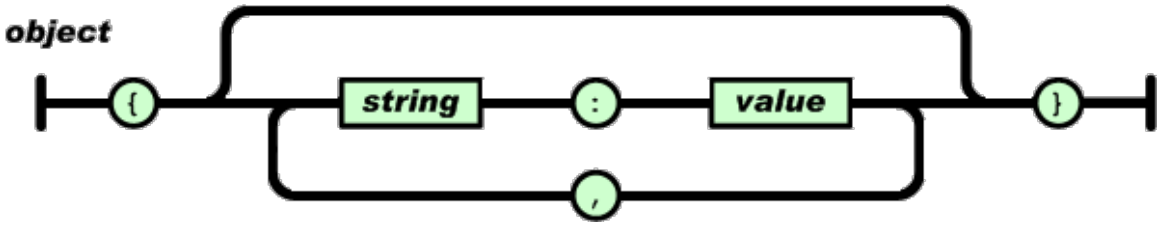


Figure 2.1.: JSON object structure - taken from `https://www.json.org/` (02.02.2019)

- **Array** - It is wrapped in regular brackets. It represents an array of JSON objects separated by semicolon, as you can see in the Figure 2.2;

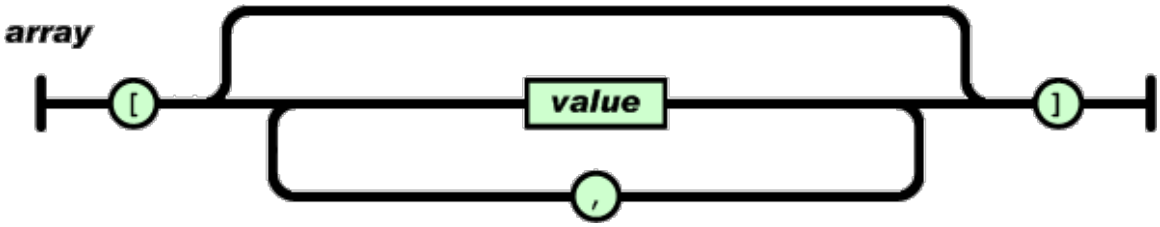


Figure 2.2.: JSON Array structure - taken from `https://www.json.org/` (02.02.2019)

- **Value** - A JSON attribute can also take a primitive value: string, number, boolean or null.

[5] `https://www.json.org/` (02.02.2019.)
[6] See 5

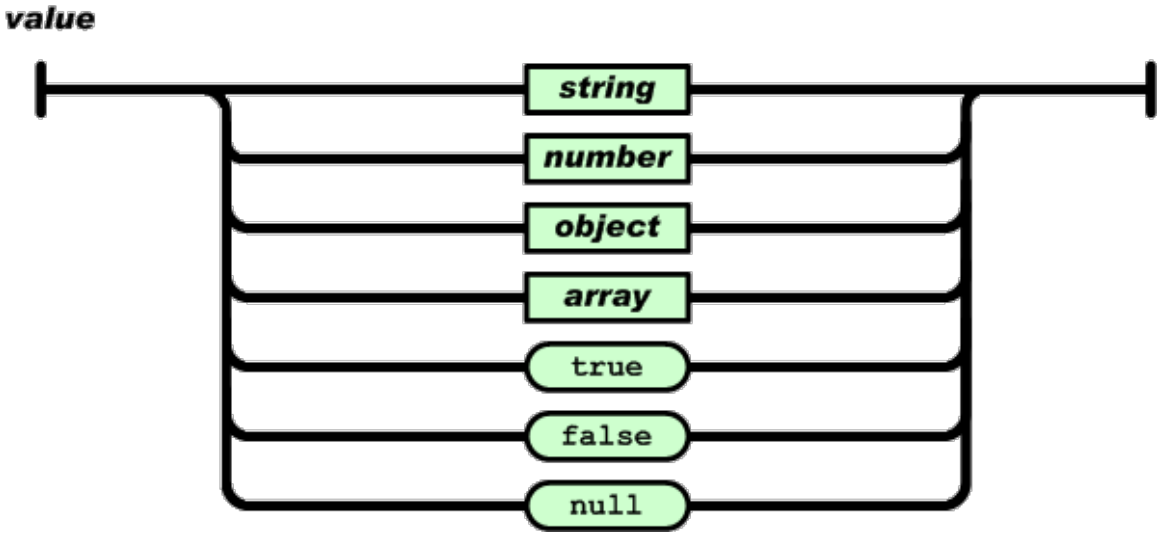


Figure 2.3.: JSON value options - taken from `https://www.json.org/` (02.02.2019)

We can say that today, JSON is a standard data format for data interchange. Almost all, if not all, programming languages support working with JSON natively or through library support.

There are many advantages of working with JSON compared to other data format that is often used as Extensible Markup Language (XML) or Comma Separated Values (CSV). Compared to XML, JSON has a smaller footprint - there are no closing tags, which makes JSON almost as twice as lighter, and it also makes it more readable as there are no repeatable opening and closing tags. CSV only supports data in a columnar format which makes it difficult to work with more complicated data structures - e.g., nested data structures, which JSON supports.

2.4. Linked Data

The concept of Linked Data is based on the idea of linking publicly available data ”silos” on the Internet utilizing semantic methods and networks [3]. Semantic Web technology has established a framework for creating a ”web of data” where the nodes correspond to resources of interest in a domain and the edges correspond to logical statements that link these resources using binary relations of interest in the domain [18].

Linked Data as a concept emerged when interned stated to increase at a considerable pace. Many scientists noticed that many data on the Internet are repetitive and that there should be a way to somehow link one resource to another. It refers to a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs [18].

> "Linked Open Data (LOD) is Linked Data which is released under an open licence, which does not impede its reuse for free. - Tim Berners-Lee [9]"

If Linked Data is open to everyone, then it represents Linked Open Data. One of the most significant efforts to create open linked datasets is the DBpedia project [19]. DBpedia is made of structured information extracted from Wikipedia. The English version of the DBpedia knowledge base currently describes 4.58 million things [7]. Also, there is a Linked Open Data cloud that consists of datasets published in Linked Data format. There is an excellent visualization of Linked Open Data cloud that can also be used to explore datasets by domains[8].

The reasoning behind creating Linked Data is not only to reuse existing resources but also to enable users that are exploring Linked Data to quickly "jump" from one resource to another, only by following the hyperlinks. The advantage of the Linked Open Data approach relies on the possibilities of retrieving and subsequently analyzing data that have been related through associative links [3]. Another power of Linked Data principles is that it aligns well with REST architectural style (e.g., one URI point to one and only one resource), but in practice, those two are not that well-connected [20].

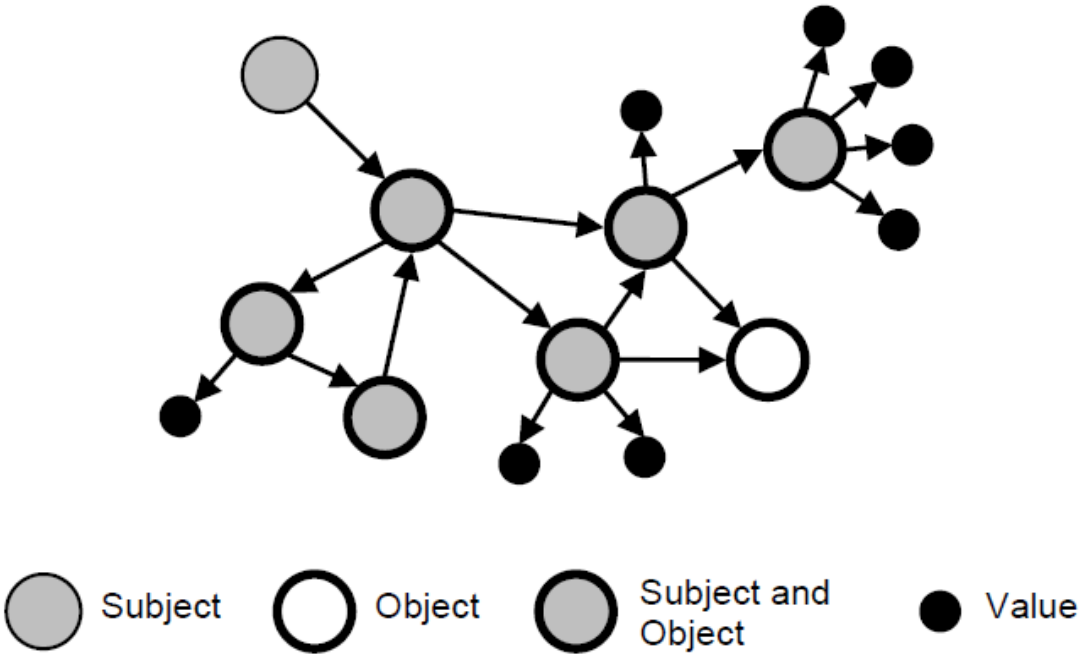


Figure 2.4.: Linked Data Graph [20]

In Figure 2.4 we can see that in the Linked Data graph we have *Nodes* and *Edges*. Nodes

[7] `https://wiki.dbpedia.org/about/facts-figures` (03.02.2019.)
[8] `https://lod-cloud.net` (05.02.2019.)

represent *subjects* or *objects* and Edges represent *properties*. A subject is a node with at least one outgoing edge and object is a node with at least one incoming edge. Graph structure perfectly suits the structure of Linked Data as we should be able to explore data going from one Subject/Object to another thus exploring Linked Data.

A critical property of Linked Data format is that all data contained in Linked Data is well-structured and annotated in a way that Actors (machine systems) when processing data from Linked Data can interpret data semantically based on connections and properties - all of which means that machines can understand data without the need for human interference. Before, computers were passing data around without knowing what data represents. Machine before could not know what, e.g., ”name” means, but now with proper annotation in Linked Data machines can understand that ”name” is an attribute of a Person and that data is textual.

2.5. JSON for Linked Data (JSON-LD)

JSON for Linked Dat (JSON-LD) is a lightweight data format for Linked Data representation[9]. It is based on the JSON data format that we have explained in Section 2.3. That makes it an ideal data format for rich REST web services and data interchange. JSON-LD is an attempt to create a simple method not only to express Linked Data in JSON but also to add semantics to existing JSON documents [20].

JSON-LD is designed around an idea of enriching the existing data by providing the ”context” to it. Adding context means that data can be described semantically with provided relationships with other entities. Many developers are familiar with JSON format, but not so many developers are familiar with the Semantic Web technologies. Because of that, massive efforts have been put into JSON-LD so that developers do not have to be knowledgeable about other semantic Web technologies[20].

To be able to transform JSON into JSON-LD, at least two additional attributes have to be used [20]:

- **@context** - It is used to provide a common understanding of unknown terms. It helps in connecting the JSON attribute with the definition (semantics) from some vocabulary.
- **@id** - It serves as an identifier of a Node (Entity). For nodes to be truly linked, the concept of a unique identifier is essential - dereferencing the @id of a Node, should result in the entity representation of that Node. Attribute @id is in Internationalized Resource Identifier (IRI) format, which is a complement to the Uniform Resource Identifier (URI)[10].

[9] `https://json-old.org/` (06.02.2019.)
[10] `https://tools.ietf.org/html/rfc3987` (07.02.2019.)

```
{
    "@context": "https://schema.org/Person.jsonld",
    "@id": "http://dbpedia.org/resource/John_Lennon",
    "name": "John Lennon",
    "born": "1940-10-09",
    "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}
```

Listing 2.1: Example of JSON-LD document - modified and taken and from `https://json-ld.org/` (07.02.2019.)

In the Listing 2.1 we can see that this JSON object looks like a regular JSON object, except the two new attributes, *@id* and *@context*. Looking at the @context, we know that the definition of all attributes will be looked up in the definition of a Person that is contained in Schema.org vocabulary. Also, we can see that dereferencing the value of the @id attribute gives all data about Jonh Lennon entity that is contained in DBpedia. Even with the usage of Linked Data attributes, this JSON-LD object can be treated as a regular JSON object.

JSON-LD can represent Linked Data in several forms:

- **Compact** - Information about the attributes is contained in the context of the JSON-LD document. This improves readability, but increases complexity for processing.

```
{
    "@context": "https://schema.org/Person.jsonld",
    "@id": "http://dbpedia.org/resource/John_Lennon",
    "givenName": "John",
    "lastName": "Lennon",
    "born": "1940-10-09"
}
```

Listing 2.2: Compact form of JSON-LD

- **Expanded** - Information about the attribute is contained in the attribute name. It makes JSON-LD be in a uniform structure which can help to process the JSON-LD document, but it decreases readability.

```
[{
    "https://schema.org/givenName":[
        {"@value": "John"}
    ],
    "https://schema.org/familyName":[
        {"@value": "Lennon"}
    ],
    "https://schema.org/birthDate":[
        {"@value": "1940-10-09"}
    ]
}]
```

Listing 2.3: Expandend form of JSON-LD

Those forms are giving great flexibility of JSON-LD so a user can pick the one that suits their use case the best. Many libraries can be used for processing JSON-LD, and most of them support compaction and expansion of JSON-LD documents.

2.6. Schema.org

Schema.org is a community activity with a mission to create, maintain, and promote schemas for structured data on the Internet, on web pages, in email messages, and beyond [21].

Schema.org became an essential part of the Internet when big companies like Google, Yahoo, Bing, and Yandex decided to create and maintain a standard set of schemas for structured data in the Internet [22]. Those well defined and structured schemas together make a vocabulary, which is used to describe data semantically. It is extensively used for Search Engine Optimization (SEO). If data in web pages are annotated adequately with information from Schema.org, it will help search engine to understand better (semantically) what is the page about. If it is correctly done, it can lead to a better position in search engine results and also to the more comfortable showing of structured information from the website in the search engine.

```
{
"@context": {
    "dct": "http://purl.org/dc/terms/",
    "owl": "http://www.w3.org/2002/07/owl#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "rdfa": "http://www.w3.org/ns/rdfa#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "schema": "http://schema.org/",
    "xsd": "http://www.w3.org/2001/XMLSchema#"
},
    "@id": "schema:givenName",
    "@type": "rdf:Property",
    "rdfs:comment": "Given name. In the U.S., the first name of a Person. This
    can be used along with familyName instead of the name property.",
    "rdfs:label": "givenName",
    "schema:domainIncludes": {
        "@id": "schema:Person"
    },
    "schema:rangeIncludes": {
        "@id": "schema:Text"
    },
    "schema:sameAs": {
        "@id": "https://schema.org/givenName"
    }
}
```

Listing 2.4: Schema of givenName property from Schema.org

The introduction of Schema.org was a significant step forward for Semantic Web as it allows a broad range of data, ranging from events and recipes to products and people, to be annotated with a shared vocabulary which is understood by all major search engines[20]. In the Listing 2.4 we can see scheme for a property **givenName** from Schema.org. We can see that it is in the JSON-LD format and that it contains several contexts. Based on several properties of the **givenName** schema definition we can see:

- **@id** - givenName is defined in the context of schema (http://schema.org/);
- **@type** - It shows that, by rdf context, givenName entity represents a property of some other entity;
- **rdsf:comment** - Descriptive information about the givenName entity;
- **rdfs:label** - Short and easy to read name of the entity;
- **schema:domainIncludes** - It shows that this property can be found in entities of the type **Person**;
- **schema:rangeIncludes** - It shows that the value of this property should be in the Text format.

The number of schemas in Schema.org is continually growing. Schema.org as a project is open-source, and everyone can contribute. The project is hosted in Github[11] and new schemas can be proposed/submitted by creating a pull request. A new schema can be added to the Schema.org release only if it is a general schema that represents some commonly used terms and definitions. The whole project is extensible, and developers can extract existing schema and extend it to suit their needs.

2.7. Metadata

Metadata is data that describes other data. Meta is a prefix that in most information technology usages means "an underlying definition or description" [23]. Metadata was used even before the phenomenon called the Internet. Organizations who were managing large amounts of data had to keep registers of data about data for easier finding and processing, like in libraries. With the development of Internet technologies and increased volume of data stored online, metadata handling had to be implemented in the online world.

There are various ways to associate metadata with resources [24]:

- **Embedded metadata** - Resides within the markup of the resource. Digital documents often have this type of metadata [25]. It can help with struggling to track rapidly expanding collections of digital media assets such as photos and video clips. One example is sorting images based on size, type, date taken or some other metadata.

[11] `https://github.com/schemaorg/schemaorg` (12.02.2019)

- **Associated metadata** - Maintained in files tightly coupled to the resources they describe. It gives a possibility to edit metadata without changing the actual resource. It also increases the complexity of metadata management because some changes may occur in both data and metadata. It is like having a document filled with information when each of the pictures on the computer is taken. That document is easy to update, but in case of deletion of a picture, information about that picture should be deleted from the document.

- **Third-Party metadata** - Maintained in a separate repository by an organization that may or may not have direct control over or access to the content of the resource. When some pictures are stored on a computer, some computer program may generate metadata for easier search or sorting of those pictures. Users of the computer cannot access to those metadata, but that metadata exists and enables that program to provide better functionalities. It is a usual practice that some vendors store and maintain metadata in proprietary file formats and repositories [26].

Depending on the nature of data, infrastructure that handles data and intended usage, one, or even all three methods, can be used to associate metadata with data resources.

Different types of metadata can be used [27]:

- **Descriptive metadata** - For finding and/or understanding a resource. It can include elements such as title, abstract, author, keywords, description etc.

- **Administrative metadata**

  Technical metadata - For decoding and rendering files;

  Preservation metadata - For long-term management of files;

  Rights metadata - For attaching intellectual property rights to content.

- **Structural metadata** - Relationship of parts of resources to one another;

- **Markup languages** - Integrates metadata and flags for other structural or semantic features within content.

Metadata Type	Example Properties	Primary Uses
Descriptive metadata	Title Author Subject Genre Publication Date	Discovery Display Interoperability
Technical metadata	File type File size Creation date/time Compression scheme	Interoperability Digital object management Preservation
Preservation metadata	Checksum Preservation event	Interoperability Digital object management Preservation
Rights metadata	Copyright status License terms Rights holder	Interoperability Digital object management
Structural metadata	Sequence Place in hierarchy Data format	Navigation
Markup languages	Paragraph Heading List Name Date	Navigation Interoperability

Table 2.1.: Example of properties for different types of metadata and their primary usage - modified and taken from [27]

Table 2.1 shows example properties and primary uses for various metadata types. List of properties is extensive, and for the sake of brevity, this Figure shows only a couple of example properties. Most of the time, we have to combine those types of metadata so we can adequately provide meta information about the dataset and processes that are using that dataset. It is essential to make sure that there is a common understanding between all interested parties of how, when and where metadata is used. There are standards like ISO/IEC 11179[12] that gives the specification of how to implement metadata registry and apply best practices for metadata exchange with various stakeholders.

[12] `http://metadata-standards.org/11179/` (27.02.2019.)

2.7.1. Semantic Metadata

Richly interlinked, machine-understandable data constitutes the basis for the Semantic Web. Annotating web documents is one of the major techniques for creating metadata on the Web [28].

As we have seen in the previous section, there are various types of metadata. We use semantic metadata - a combination of technical and structural metadata that uses definitions from some digital vocabulary in order to describe the semantic nature of the entity that it is attached to.

The term *Self-annotating Web* was introduced by Cimano et al. [29]. The idea of the self-annotating Web is that it uses globally available Web data and structures to semantically annotate, or at least facilitate annotation of, local resources. Adapting that concept in our case means that we will use entities from some public vocabulary (globally available Web data and structures) to semantically annotate our local datasets or annotated Open Data datasets (local resources in combination with public data resources).

There are various challenges in the semantical annotation of data. Egyedi et al. [30] describe the process of annotation of data with semantically rich metadata as the mixture of Web-based acquisition forms and a population of spreadsheets. They also have pointed out that annotation often requires manual effort and that is based on the process of trials and errors.

Schema mappings are widely used in all data management applications that involve data sharing or data transformation; in particular, schema mappings are essential building blocks in information integration, data exchange, metadata management, and peer-to-peer data management systems [31]. In our case, we leverage existing, well-established schema mappings instead of creating new schemas that are not publicly available.

Nagarajan Meenakshi describes the process of metadata enhancement [32]: In the process of identifying entities in the document, it is possible to find values for attributes or relationships that were not previously present in the knowledge base. Enhancing the existing metadata could be as simple as entering values for attributes, in which case they could be automated; or as complex as modifying the underlying schema, in which case some user involvement might be required. In Chapter 5 we will describe a semi-automated process for vocabulary entity discovery.

3. Related Work

In this Chapter, we will give an overview of Data Integration platforms that manage metadata in some form. Also, we will describe in details several platforms of interest. In Section 3.2 we will describe initiatives that are founded with a purpose to tackle the rising amount of structured and unstructured data that needs to be exchanged between large entities. Finally, we will list key points that will help us in the development of a solution based on the analysis documented in this Chapter.

3.1. State-of-the-art Solutions in the area of Data Integration

Many platforms provide ETL solutions in many different processes. We will focus on platforms that provide solutions in the area of data integration, data enrichment and metadata management. By analyzing those solutions, we will be able to design and implement a module for metadata management that leverages Semantic Web.

In table 3.1 we can see seven different platforms with high level overview of their basic characteristics:

- **Year of foundation** - Metric that shows how long the platform is on the market;
- **Business model** - Gives insights into platform monetization and features delivery;
- **User Interface** - It gives an overview in which direction platform will develop and what is their focus; e.g., Platforms with versatile CLI commands and under-developed Web application can lead us to conclude that platform will be used maybe as a part of the pipeline and not as a standalone application with rich user interface features;
- **Export and Import capabilities** - It shows versatility of the solution; platforms that support more import and export formats are covering wider audience while highly specialized platforms provide only several specific data formats;
- **Extensibility** - Platforms that are intended to be used as end-to-end solutions or to provide a great number of features that are usually extensible. Sometimes basic solutions are offered for free, while extensions with special features are sold and used for gaining profit;

- **Deployment** - Cloud-based or on-premise solution; It can help in estimating cost calculation, effort and for utilizing specific platform. It also plays a role in data governance planing.
- **Metadata handling** - Meta information plays important roles in data integration. It can help in providing specific custom processes, integrating with other platforms and exploring data conveniently.

Platform	Founded	Business Model	User Interface	Export/Import	Extensibility	Deployment	Metadata
Wolfram Data Framework	2009	Proprietary; Offers free web applications; Pro version available	Web and mobile applications; programming libraries	Multiple data formats supported; Most functionalities supported for WDF format	Usually offered as a service	Cloud; Possible to have Wolfram engine on-premise	Generated; arbitrary metadata also supported
RapidMiner	2006	Proprietary; Based on open core solution	Rapid Miner studio desktop application	Multiple data formats/data sources supported	RapidMiner extensions marketplace	Cloud and on-premise	Linked Open Data extension
Google BigQuery	2010	Offered as a paid service; Free trials	Web application; CLI	Cloud Storage, Excel, Various data connectors	Offered as a service; Extensible with other Google cloud solutions	Cloud	Generated; provides mechanism for adding dataset labels
Dremio	2017	Open Source; Enterprise edition for large organizations	Web application	Various data connectors	Provides many integrations (Amazon S3, ADLS, RDBMS, NoSQL, Hadoop, and more)	On-premise; deployable to cloud	Technical and descriptive metadata on dataset and node level
Deeper [33]	2017	Open Source; Used in lectures at SFU	Web application	CSV	DBLP, Yelp, Aminer / Not out of the box	On-premise	Entity resolution based on Schema matching
IBM InfoSphere	2008	Proprietary; Server, Enterprise, MVS Edition	Desktop applications; Web application	DBMS, big data sources, messaging queues, ERP and other packaged applications, industry formats and mainframe systems	Many IBM and third party modules	Cloud and on-premise	IBM InfoSphere Metadata Workbench; Business, Technical, Operational
Talend	2005	Free Open Source;User-based subscription	Web application	Any database, Excel or CSV, Parquet, AVRO files etc.	Cloud, RDBMS, SaaS etc.	Cloud and on-premise	Talend Data Catalog

Table 3.1.: Overview of Data integration/enrichment platforms

3.1.1. Wolfram Data Platform

Wolfram Research company was founded in 1987 by Stephen Wolfram. Since then, many tools and solutions were built, and they have millions of active users - in schools, academia, and industry. Their first and longstanding flagship product is Mathematica (first released in 1988). It is very popular, and they have an active community around it. In 2009 they introduced Wolfram Alpha subsidiary which shaped Wolfram Language as we know it today. Wolfram Data Framework is a platform invented by engineers from Wolfram Al-

pha company, and it is based on Wolfram Mathematica, Wolfram language and Wolfram Knowledgebase.

Platform Description

The primary purpose of the Wolfram Data Framework is to help users to take data and make it meaningful. They want to provide a standardized representation of real-world construct and data which is computable.

Wolfram Data Framework exposes ontology which is defined in Wolfram Alpha. Wolfram Alpha can compute answers using algorithms, knowledge bases and AI technology. They support a variety of topics mainly divided into four groups:

- Mathematics (Algebra, Calculus, Geometry, etc.)
- Science and Technology (Physics, Chemistry, Transportation, etc.)
- Society and Culture (People, Arts, Dates, History, etc.)
- Everyday Life (Health, Finance, Hobbies, etc.)

Wolfram Alpha is a knowledge inference engine that computes answers to queries from a structured knowledge base about the world, rather than providing a list of documents [34].

A user can even input arbitrary data, and Wolfram Data Framework will try to process it. It will also suggest what kind of data the input might be. The main strength of Wolfram Data Framework lies in the newly developed format WDF - a human-readable plain text format that can be rendered in JSON, XML, and other data formats. WDF is not just a language for representing real-world data but also defines canonical forms based on knowledge on millions of entities.

Figure 3.1.: WDF Workflow

As we can see in figure 3.1, WDF workflow is divided into three stages. In the first stage, unstructured data is converted into WDF format so it can be used across Wolfram ecosystem. In the second stage, they describe the process of storing the data as WDF in a file, database or even Wolfram Cloud. In the last step, WDF data can be computed directly into Wolfram Language which would enable exploration of the data using all the features that Wolfram Language provides.

Wolfram Data Framework is considered to be ideal for:

- **Multidomain organizations** - Different parts of an organization understand data the same way;
- **Long-term data storage** - Enables storing of the data without the need for additional metadata management; everything needed is already included in WDF, which is also human-readable data format;
- **Data repositories** - WDF internally manages metadata and it also connects data to their Knowledgebase for providing interactive data exploration;
- **Research publishing** - For data-backed publications, where data can be immediately used and computed;
- **Smart input fields** - Semantically smart input fields in which WDF can automatically recognize many input types;
- **Internet of Things** - WDF provides a connection between real-world data and abstract computations.

Implementation Details

Wolfram Language is considered to be a programming language but more than that. In addition to built-in operation and constructs, it has substantial built-in computational capabilities and knowledge. It is interpreted line by line where the result of interpretation of the line can be immediately seen. Wolfram Language provides many built-in functions which makes use of third-party libraries unnecessary in contrast to other languages like Java or Python. In addition to formulas, constructs, and data, Wolfram language can process other entities like files and pictures - everything in the Wolfram Language is a symbolic expression. That means for example that a file can be used as an input argument in a function.

Wolfram Knowledgebase is one of the most extensive knowledge bases ever made. It provides trillions of elements, and it is continuously growing. Most of the data contained in the Knowledgebase is curated from primary sources, but they also use data derived from many websites, data streams, and experts knowledge. They use many automated and manual methods to check the quality of the data. Data is updated regularly depending on the data nature - some data are updated by consuming data streams, while others are updated daily, weekly, monthly, yearly or when data become available.

Wolfram Data Framework relies on capabilities of the Wolfram Language and Wolfram Knowledgebase. It gives an overview of the functions used over the data where knowledge about the data is stored directly in the WDF - explicitly written or left to Wolfram Data Framework to automatically process the data. The primary purpose of Wolfram Data Framework is to convert unstructured data into correct canonical WDF format which is

later expressed in Wolfram Language and reuses knowledge from Wolfram Knowledgebase to give an expressive and rich overview of the data.

One of the design decisions was to provide a way for both machines and humans to understand the data. It makes use of additional documents to describe data completely obsolete. Another design decision was to enable users to attach arbitrary metadata thus allowing additional information about the data that are being processed. In addition to representing WDF as a Wolfram Language, it is possible to convert WDF into JSON or XML representation so WDF can be treated and parsed by various data-parsing tools.

Metadata Handling

There is a way to import a file and to convert it into a WDF using *SemanticImport* function. The most straightforward function call is $SemanticImport[" fileName.csv"]$ where $fileName.csv$ represents a file in comma-separated values (CSV) format. Wolfram Data Framework will automatically try to process each column and interpret it as an *Entity*. As a result, the user can extract automatically attached properties of those entities and further explore the data.

For example, we have a comma-separated values file *visits.csv* which contains date and city. It has only two rows.

1 Jan 2018	Munich
25 Feb 2018	New York

Table 3.2.: CSV Example data

By executing $visits = SemanticImport["visits.csv"]$ we will convert our CSV example data showed in table 3.2 into WDF. As canonical entity types support dates and cities, Wolfram Data Framework will automatically interpret them as entity type *Date* and entity type *City*. Because of this feature commands like $visits[2, "City"]["Population"]$ can be used to extract population details from large Wolfram Knowledgebase.

That is an elementary example, but it illustrates the power of such a simple statement. There is a lot of different options which can be used in *SemanticImport* function. A user can explicitly assign *EntityType* to an arbitrary number of columns, skip lines, add missing values, set a specific delimiter, change the orientation of data from column to row and more.

Another function worth mentioning is *Interpreter*. In essence, *SemanticImport* uses *Interpreter* function to convert data into specific entity type. It can also be used to test if the that entity is of the specific entity type. This function supports a large variety of entity types from primitive data types to complex ones like City, Country, Sound and many more.

MetaInformation is a function that enables adding arbitrary metadata to any object. Metadata that is added is in the form of key-value pairs, where the key should be a String literal

and value can be any WDF expression. That is an essential feature as it allows adding custom information about the data and enables storing additional knowledge in the data.

Extensive documentation can be found on the official website for Wolfram Data Framework [35].

3.1.2. RapidMiner

RapidMiner builds a software platform for data science teams that unites data preparation, machine learning, and predictive model deployment. Formerly knows as YALE (Yet Another Learning Environment) was developed in 2001 by Ralf Klinkenberg, Ingo Mierswa, and Simon Fischer at the Artificial Intelligence Unit of the Technical University of Dortmund. In 2007, the name of the software was changed from YALE to RapidMiner.

RapidMiner provides data mining and machine learning procedures in addition to data loading and transformation (ETL - extract, transform, load), data preprocessing and visualization, predictive analytics and statistical modeling, evaluation and deployment.

Platform Description

RapidMiner Studio is a visual workflow designer that enables the entire analytics team to easily use data science and machine learning solutions. By using visual workflow, a need for writing code is brought down to a minimum. It also enables connecting multiple data sources by using different extensions for specific vendors. Extensions can be downloaded via their Marketplace. It is possible to run data preparation and ETL processes directly in the database (for vendors they are supporting - MySQL, PostgreSQL, and Google BigQuery). All data can be explored through many visualizations which can help in identifying data quality flaws. Flaws like missing values or outliers can be mitigated using RapidMiner Studio built-in features. Processes created in RapidMiner Studio are expressive, and they can be used to communicate machine learning models. It also supports validation of machine learning models so only proper models can reach production. RapidMiner Studio is offered for free usage with a limited number of features.

RapidMiner platform offers other products in addition to RapidMiner Studio:

- RapidMiner Server - For sharing and re-using predictive models, automating processes and deploying models into production;
- RapidMiner Turbo Prep - For ETL processing via web based user interface;
- RapidMiner Auto Model - For creating predictive models using automated machine learning and data science best practices;
- RapidMiner Radoop - It exposes Hadoop and Spark functionalities for code free machine learning;

The core of the RapidMiner platform is open-source, and contribution is welcomed. It is also possible to develop a custom extension which can be used inside RapidMiner Studio.

Implementation Details

RapidMiner is written in the Java programming language, and it provides a graphical user interface to design and execute analytical workflows. Those workflows are called *Processes* and they consist of multiple *Operators* which are connected through *Ports*. Each operator performs a single task within the process, and its output represents an input of the next one. Also, each operator can be modified by changing its *Parameters*. Those *Operators* are used e.g. for data extraction, transformation, processing etc. Alternatively, the engine can be called from other programs or used as an API (Application Programming Interface). Individual functions can be called using the CLI (Command Line Interface).

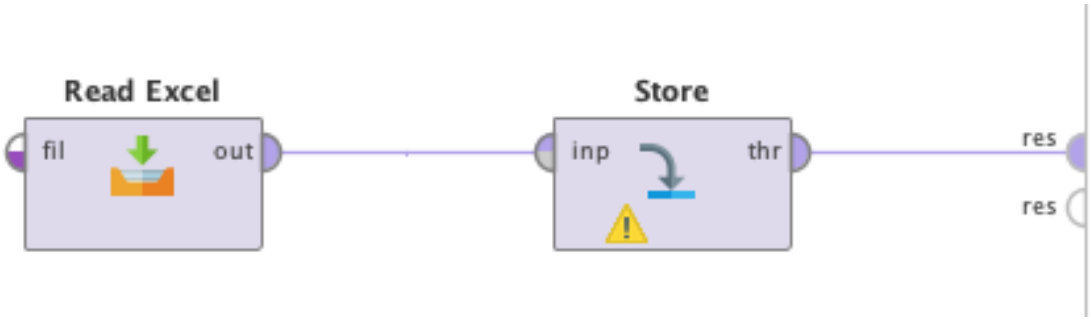


Figure 3.2.: Simple RapidMiner Process - taken from official documentation

In figure 3.2 we can see a straightforward process with two operators - Read Excel and Store. It shows reading an excel file content, storing it and displaying the result of the process. That is how a process looks like in RapidMiner Studio. Most processes can be made by using only wizards and visual workflow designer - with no programming.

Metadata Handling

The RapidMiner Linked Open Data Extension is an extension which was developed for RapidMiner platform. It allows using data from Linked Open Data both as an input for data mining as well as for enriching existing datasets with background knowledge. The RapidMiner Linked Open Data Extension is based on the earlier FeGeLOD (Feature Generation from Linked Open Data) framework (which is discontinued). It was developed at the University of Mannheim in 2014. The same year it won a first prize award at Semantic Web Challenge at the International Semantic Web Conference held in Italy.

While many domain-specific applications use Linked Open Data, general-purpose applications rarely go beyond displaying the small data and provide little means of deriving additional knowledge from the data [36].

This extension provides additional *Operators* which are useful for Linked Data handling - data import, data linking, feature generation, schema matching, and feature subset selection [36]. They are divided into three main categories:

- Data importers that load data from Linked Open Data into RapidMiner for further processing
- Linkers that create links from a given dataset to a dataset in Linked Open Data
- Generators that gather data from Linked Open Data and add it as attributes in the dataset at hand. Some data type value is guessed so, e.g., integer values would be stored as integers after importing.

The code base for this extension is open source and published on GitHub [1]. Unfortunately, the latest supported version of RapidMiner Studio is version 5. It also can be used with version 6, but all version afterward are not supported (at the moment of writing the latest version was 9.1). Also, the code is not well documented, and it is not maintained for more than two years. There is an old user manual[2] which provides all information needed to use this extension.

3.1.3. Google BigQuery

Google BigQuery is a fast, scalable, cost-effective, and managed cloud data warehouse for analytics, with built-in machine learning. It is Google's serverless enterprise data warehouse that can be paid per usage (usage-based pricing). Because there is no infrastructure to manage, the focus is on analyzing data to find meaningful insights using SQL queries without the need for a database administrator [3].

Platform Description

Google handles Big Data to provide services like Search, YouTube, Gmail, and Google Docs. Google is a company that has hundreds of millions of customers and providing real-time analytics on such scale is a challenging task. There are many features that BigQuery can provide [4] [37]:

- **Serverless** - There are no servers to manage, everything is in the cloud and automatically provisioned if there is a need for more computational and storage resources;

[1] `https://github.com/petarR/RapidMiner-LOD-extension` (27.12.2018.)
[2] `https://dws.informatik.uni-mannheim.de/fileadmin/lehrstuehle/ki/research/RapidMinerLODExtension/RapidMinerLODExtensionManual.pdf` (27.12.2018.)
[3] `https://cloud.google.com/bigquery/` (28.02.2019.)
[4] Same as 3

- **Petabyte scale** - BigQuery can provide enough resources for users so they can make queries over Petabytes of data and get results in a matter of seconds;
- **Real-time analytics** - There is a possibility to analyze new data in real-time when data is imported by using high-speed streaming insertion API;
- **Flexible pricing models** - BigQuery can be used with on-demand (usage-based) pricing or flat-rate pricing;
- **Automatic high availability** - Replication of data is offered for free which means users can expect high availability even in case of failures in some data centers;
- **Data encryption and security** - BigQuery can be set up with fine-grained access controls by using Identity and Access Management. Also, all data is encrypted - in data silos and in transit;
- **Standard SQL** - BigQuery supports a standard SQL dialect which is ANSI:2011 compliant. That would make easier migration for systems that are already using SQL. Also, in practice, it has been shown that business users are able to use SQL and that they are using it in their regular job [38];
- **Data locality** - Users can control in which region they want to store their data, e.g. to be able to comply with specific regulations;
- **Federated query and logical data warehousing** - BigQuery can abstract disparate data sources like Cloud Storage, Cloud Bigtable or spreadsheets in Google Drive. That means that users do not have to take care of source and format of their data and that they can query all of them at the same time;
- **Foundation for artificial intelligence (AI)** - BigQuery ML provides a possibility to apply machine learning methods on top of the data that is used in BigQuery. Also, data can be transformed and analyzed in BigQuery and made ready for analysis by leveraging machine learning methods;
- **Storage and compute separation** - Storage does not mean compute. Users can only pay for storage, and they will pay for computing their data only when they are using those data in computational purposes;
- **Foundation for business intelligence** - BigQuery enables data integration, transformation, analysis, visualization, and reporting by using tools from Google or third-party solutions;
- **Automatic backup and easy restore** - BigQuery automatically replicates data and keeps a seven-day history of changes;

- **Flexible data ingestion** - BigQuery supports importing data from different third-party data sources like Informatica or Talend;
- **Geospatial datatypes and functions** - BigQuery supports arbitrary points, lines, polygons, and multi-polygons in WKT and GeoJSON format, which can simplify geospatial analyses;
- **Data governance** - Fine-grained Cloud Identity and Access Management control data access. That means that data will not be available to non-authorized users;
- **Data transfer services** - The BigQuery Data Transfer Service automatically transfers data from external data sources, like Google Marketing Platform, Google Ads, and YouTube, to BigQuery on a scheduled and fully managed basis;
- **Programmatic interaction** - BigQuery provides a REST API for easy programmatic access and application integration. Also, business users can use Google Apps Script to access BigQuery from Google Sheets;
- **Big Data ecosystem integration** - BigQuery provides integration with the Apache Big Data ecosystem, allowing existing Hadoop/Spark and Beam workloads to read or write data directly from BigQuery;
- **Rich monitoring and logging with Stackdriver** - BigQuery provides rich monitoring, logging, and alerting through Stackdriver Audit Logs;
- **Cost control** - Costs can be limited per user basis, daily limit, amount limit and many more.

In Figure 3.3 we can see that BigQuery can be used as an Analytics engine that supports streaming and batch pipeline. Results from BigQuery can be exported to many already well-established platforms like Qlik or Tableu, Google Sheets or some database.

Implementation Details

Google had a big problem with data analysis because they have an enormous amount of data. They were using MapReduce to analyze big data, but even then they could not query data interactively without waiting for the query to finish. That is why they have developed an internal tool called Dremel which enables running extremely fast SQL queries on large datasets [38].

Dremel is based on two core technologies which give Dremel unprecedented performance [37]:

1. **Columnar Storage** - Data is stored in a columnar storage fashion which makes possible to achieve a very high compression ratio and scan throughput. It is mainly used

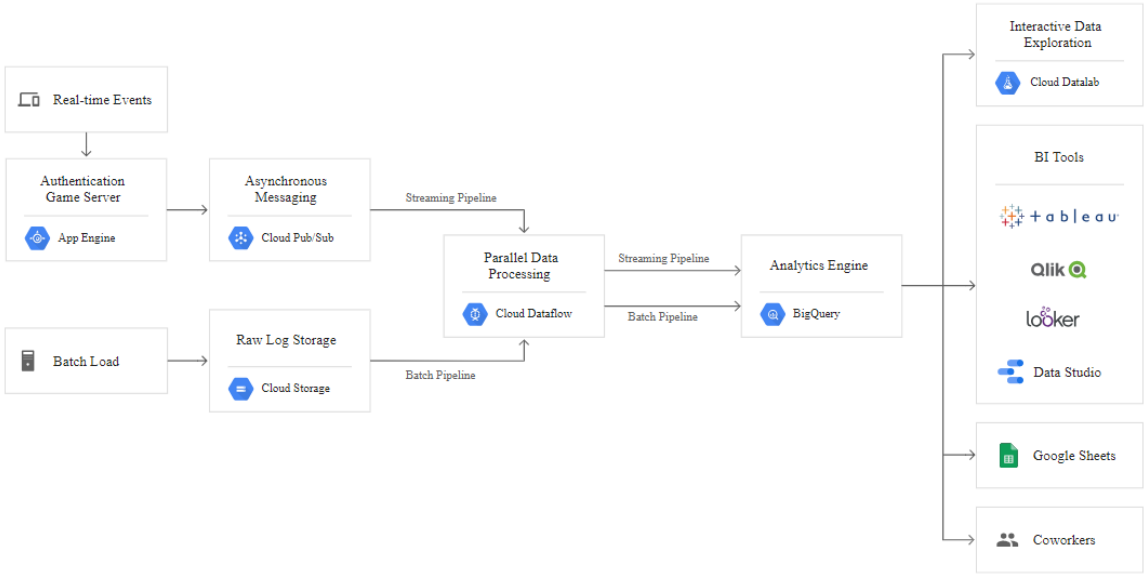


Figure 3.3.: Data warehousing solution architecture that can be utilized using BigQuery - taken from `https://cloud.google.com/bigquery/` (28.02.2019.)

in a read-only type of usage because update operations are expensive when using columnar storage.

2. **Tree Architecture** - Used for dispatching queries and aggregating results across high number of distributed machines in a mater of seconds.

BigQuery provides the core set of features available in Dremel to third-party developers. It does so via a REST API, command line interface, Web UI, access control, data schema management and the integration with Google Cloud Storage [37].

BigQuery and Dremel share the same underlying architecture and performance characteristics, which means that users can fully utilize the power of Dremel by using BigQuery to take advantage of Google's computational infrastructure [37].

Metadata Handling

By using BigQuery, users are able to store following information about their dataset [5]:

- Description
- Default expiration time for new tables
- Default partition expiration for new partitioned tables
- Access controls

[5] `https://cloud.google.com/bigquery/docs/updating-datasets` (28.02.2019.)

- Labels

Big Query uses key-value pairs to represent labels for data resources. When a resource is created in BigQuery, labels are optional. There are several limitations to using labels [6]:

- Each resource can have multiple labels, up to a maximum of 64;
- Each label must be a key-value pair;
- Keys have a minimum length of 1 character and a maximum length of 63 characters, and cannot be empty;
- Values can be empty, and have a maximum length of 63 characters;
- Keys and values can contain only lowercase letters, numeric characters, underscores, and dashes. All characters must use UTF-8 encoding, and international characters are allowed;
- The key portion of a label must be unique. However, that label can be reused on multiple resources;
- Keys must start with a lowercase letter or international character.

As labels are usually used per resource, they can be used to add custom properties to a resource. Later those labels can be used for querying resources, grouping expenses, checking computational resources on specific resources, and more.

3.2. State-of-the-art Initiatives in the Open Data community

Open Data as a concept emerged and gained traction as a community effort. The community often has members that can be considered to be influencers; innovative members, members who push the community forward, and the ones who are showing an excellent example to the rest of the community members. In this Section, we describe two initiatives, the Open Data Initiative and Data Transfer Project, backed by some of the biggest and most innovative companies on the market.

3.2.1. Open Data Initiative

Open Data Initiative is an initiative announced by chief executive officers of three large companies - Microsoft, SAP, and Adobe. It addresses common problems in customer experience management, and it creates new opportunities to extract more value from the data and to provide better customer experience in a shorter time.

[6] `https://cloud.google.com/bigquery/docs/adding-using-labels` (28.02.2019.)

In today's world, the main asset of a business is the data they possess. The user interacts with a company through many different services and channels, and they leave their data print in the process. All those data are stored in separate silos that most of the time are not connected. It is challenging for a company to get a holistic view of the customer which has his data stored in disconnected data silos. All that makes integration with other business solutions quite troublesome because it makes data governance complex and challenging.

This initiative represents the first step to provide a standard for how to store customers data and how to interact with it. Also, it will provide a framework to seamlessly connect other solutions over the existing data and provide better services for customers. Microsoft, SAP, and Adobe as three large global companies can provide an architecture and a framework for other players in the industry to follow this initiative so everyone can benefit from it.

Platform Description

Open Data Initiative is based on three guiding principles[39]:

- Every organization owns and maintains complete, direct control of all their data;
- Customers can enable AI-driven business processes to derive insights and intelligence from unified behavioral and operational data;
- A broad partner ecosystem should be able to easily leverage an open and extensible data model to extend the solution.

In the Figure 3.4 we can see that in the core of the idea is that all stakeholders share the same view of the data models thus enabling businesses not to worry about disconnected data silos. In addition to Microsoft, SAP, and Adobe, other participants will be welcomed to participate. The collaboration will be easier for all participants because they will share a common understanding of the data.

This initiative will also give more control to users - they will be able to control who and when can use their data, all from one place because users will also be able to see all their data as a whole.

Implementation Details

There are no concrete implementation details as the whole initiative is in the conceptual phase. One of the most significant obstacles will be how to define data models that will be appropriate for all participants that are already using their specific models. Governance of the data is also one of the essential topics that need to be addressed.

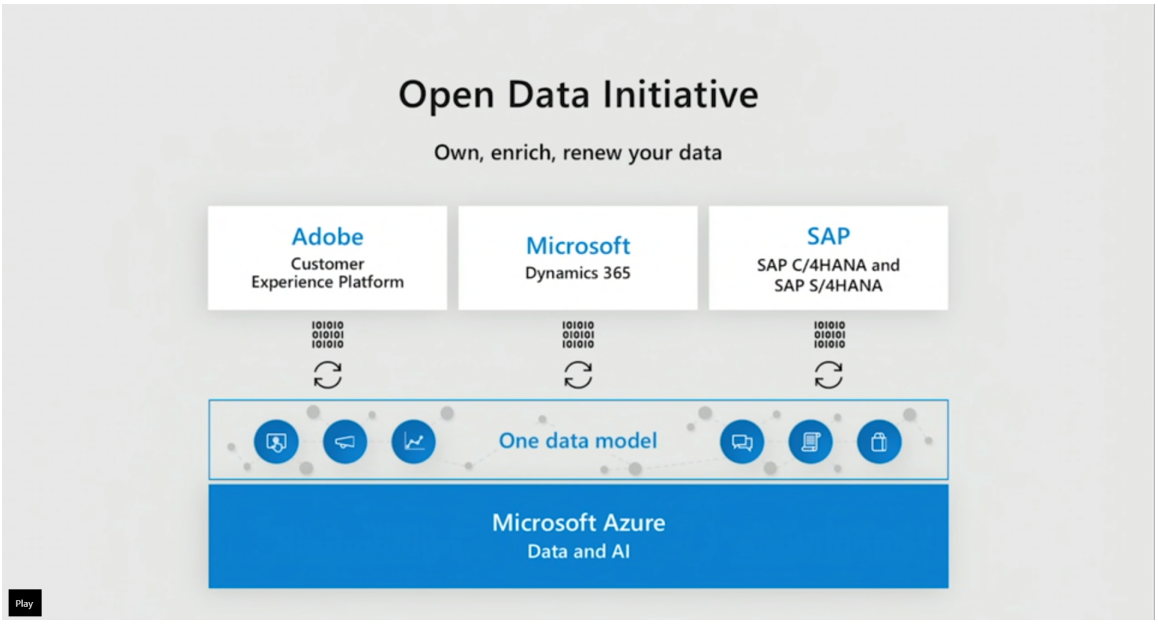


Figure 3.4.: Open Data Initiative concept [39]

3.2.2. Data Transfer Project

The Data Transfer Project (DTP) extends data portability beyond a user's ability to download a copy of their data from their service provider, to provide the user the ability to initiate a direct transfer of their data into and out of any participating provider [40].

The Data Transfer Project is an open source initiative formed in 2017 by Google, Facebook, Microsoft, and Twitter to enable data transfer from service to service thus allowing faster data portability while eliminating the need for users to download data they want to transfer directly.

Platform Description

https://www.overleaf.com/project/5b96c5b0b433cc55a87bb922 Whole Data Transfer Project is based on the following principles [40]:

- Build for users - Data portability tools should be easy to find, intuitive to use, and readily available for anyone;
- Privacy and security - Providers on each side of the portability transaction should have strong privacy and security measures, and the user should be notified about those measures;
- Reciprocity - A user's decision to move data to another service should not result in any loss of transparency or control over that data;

- Focus on user's data - Portability should not extend to data that may negatively impact the privacy of other users, or data collected to improve service, including data generated to improve system performance or train models that may be commercially sensitive or proprietary;
- Respect everyone - Data portability tools should focus only on providing data that is directly tied to the person requesting the transfer.

An ecosystem of *Adapters* powers The DTP (Data Transfer Project). They convert a range of proprietary formats into a small number of canonical forms (Data Models) useful for transferring data. That allows data transfer between any two *Providers* using the provider's existing authorization mechanism, and allows each provider to maintain control over the security of their service [40]. The main advantage of this platform lies in power to eliminate the need for users to download their data and then re-upload into a different ecosystem. That also opens new opportunities if providers collaborate - users will be able to move between different platforms easily which can give better exposure to new customers for all participants.

Implementation Details

As we already mentioned, DTP is open source and hosted on Google's GitHub account [7]. It is mostly developed using JAVA programming language and at the moment of writing engineers actively developed it from Google, Microsoft, and others.

The system comprises three main components [40]:

- *Data Models* are the canonical formats that establish a common understanding of how to transfer data;
- *Adapters* provide a method for converting each provider's proprietary data and authentication formats into a form that is usable by the system;
- *Task Management Library* provides the plumbing to power the system.

Data Models are crucial for this project to be successful. All *Adapters* will serve to convert proprietary (provider specific) data models from/to *Data Models*. The effort in the development of those *Adapters* should not be too high. Participants should see the benefits of creating their data models to be aligned with DTP data models. In the long run, collaboration and common understanding of data models should be better, so the effort put into new collaborations can be lower.

in Figure 3.5 arrows represents collaboration between two participants. They have to cooperate tightly, so they share the knowledge about data models both parties are using.

[7] `https://github.com/google/data-transfer-project`

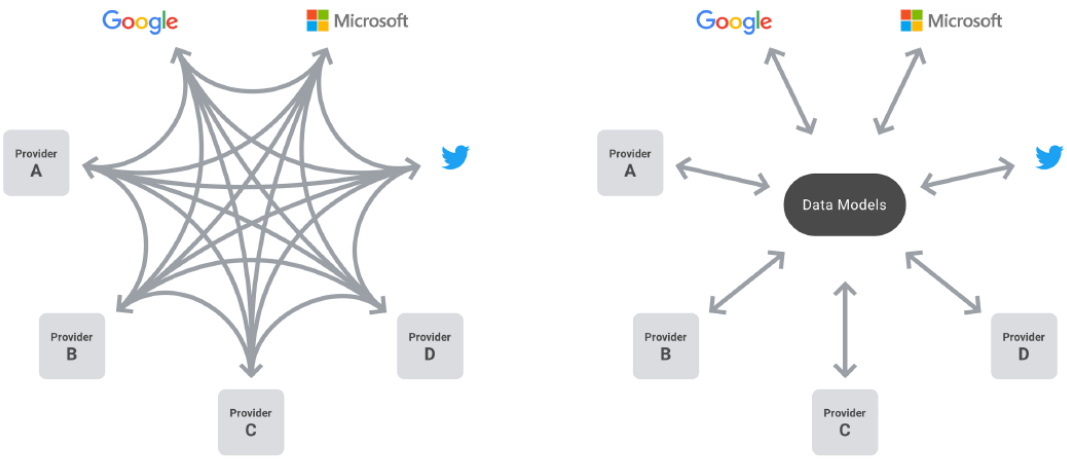


Figure 3.5.: Portability without and with utilizing DTP [40]

We can see that the graph is much simpler and there is a significant decrease in the number of specific connections. If all participants use the same data model as a reference, collaboration is seemingly easier as participants do not have to disclose internal data models. That is how a common understanding of data models can make collaboration easier for all participants.

DTP will be mostly developed using other open-source tools and existing technologies so other stakeholders can easily contribute and improve. Currently, all DTP defined all models in JSON format. Preferred authentication mechanism to be used is OAuth.

DTP project welcomes and supports all contributors. Besides that, the participation of big players like Google, Microsoft, Facebook, and Twitter encourages others to invest their time into this project.

3.3. Key Points

After analyzing the state-of-the-art data enrichment platform and open data initiatives, we can derive several key points that can help in developing the Midas platform:

- **User oriented vs. Business oriented** - Solutions like Talend, IBM InfoSphere, Dremio and, to some extent, BigQuery is more Business oriented than User-oriented. Most of those services are not free, and it requires some time to start exploring their functionalities. Solutions like Wolfram Data Framework, RapidMiner, and Deeper, are more oriented to users and it is easier to set them up and start using them out of the box.
- **Modularity** - Platforms like Talend, IBM InfoSphere and RapidMiner have a separate module for more specific metadata management. Wolfram Data Framework

uses highly coupled metadata management which enables users to discover enriched datasets automatically. BigQuery and Dremio have metadata attached directly to their resources and provide limited functionalities.

- **Interface** - Most of the solutions provide Web applications, Desktop applications and CLI tools for communicating with the system. Depending on the granularity of user access rights, also the complexity of the system varies.
- **Data formats** - Some platform like Wolfram Data Framework and RapidMiner are using platform-specific data formats in addition to standard open formats. In that way, they can offer rich user experience, but the learning curve for using those systems is steeper because of that.
- **Data connectors** - Most of the enterprise solutions like Google BigQuery, Dremio, IBM InfoSphere, and Talend are providing data connectors for integration with third-party solutions. For some platforms like Deeper, or in some cases RapidMiner, it requires some custom development to make it possible to interact with data from outside systems.
- **Semantic Metadata** - Of all solutions, only RapidMiner was providing a solution which was utilizing Semantic Web. Unfortunate is that latest version of RapidMiner does not support Linked Open Data extension. Other solutions like Deeper could be adapted to utilize Semantic Web. In most cases, enterprise solutions could leverage the Semantic Web but with great development effort.
- Not a single solution provides Open Data catalogs for exploration out-of-the-box.

Based on all that, we can decide on some design decision that could help in positioning the Midas platform:

- Midas metadata module should be pluggable into Midas platform without interfering with regular workflows;
- Store metadata in open format like JSON;
- Midas metadata module should provide well-organized REST API interface for metadata management;
- Metadata format should be extensible;
- Midas Metadata module should provide support for storing metadata on dataset and attribute level for better user experience;
- Midas Metadata module should be able to leverage Semantic Web and well-established vocabularies like Schema.org;

- Metadata format should provide a possibility to store arbitrary metadata to prevent limiting specific user needs.

All those key points can help in positioning Midas platform and Midas Metadata module as a modular solution that leverage Semantic Web while providing industry standard interface for integration with third-party systems using open format technologies that prevent vendor lock-in and steep learning curve and offer natural on-boarding process.

4. Midas Platform

Data scientists are continually searching for more data to improve their machine learning models. There are numerous sources of data which differ in quality, structure, availability, price. That is why an increasing number of them uses publicly available data such as open data for their analysis. Data which is open and free to use is considered to be the Open Data. Integrating this data into the organizations' analytics environment, however, often requires the involvement of skilled data engineers. By Tim Berners-Lee's scale for rating the quality of Open Data, we know that the quality of open data varies tremendously, where most of the available data can be considered lower to medium quality.

Midas is a distributed system that helps scientists to integrate open data interactively. It is a generic solution that can suit the needs of many data scientists and organizations. In the Midas, we have a novel approach to integrating open data to existing internal datasets. By using Midas platform, data scientists can join external Open Data datasets or REST APIs (open and closed) with internal datasets. It is based on the SQL execution engine similar to Google's Dremel but extended with special notations for linked open datasets and REST APIs. Midas platform can execute all the necessary network requests and to process the response of the request in columnar format. Because of that, the platform can use hash-based request hashing for increased performance when data scientists are executing queries interactively by reducing the number of network requests.

4.1. Architecture

Midas platform is built in a distributed manner. Because of that, it is possible to scale the platform based on usage. Also, it increases reliability as unavailable components can be easily replaced. It is worth mentioning that the Midas platform is implemented in a modular fashion to it is relatively easy to extend the functionalities of the platform.

This platform consists of the following modules:

1. Client application - Web application implemented in React.js javascript framework. It is used as an interface for data scientists to manage their data.

2. Server application - Application implemented in Node.js server-side programming language. Business logic is implemented in that layer. In addition to business logic, communication with cache and data layer as well as with query execution engine is implemented.

3. Arango.DB data storage - NoSQL database engine in which metadata and executed queries are stored.

4. MySQL data storage - MySQL database which contains internal data. Other types of data storage can be used (e.g., MongoDB)

5. Redis in-memory cache - It is used for hash-based query cashing, so h number of network requests to external data sources can be minimized. It increases performance and user experience for end users.

6. Apache Drill query engine - Schema-free SQL Query Engine for Hadoop, NoSQL and Cloud Storage. It is used to execute the same queries on different types of data storage. Also, it can scale to support a large number of queries.

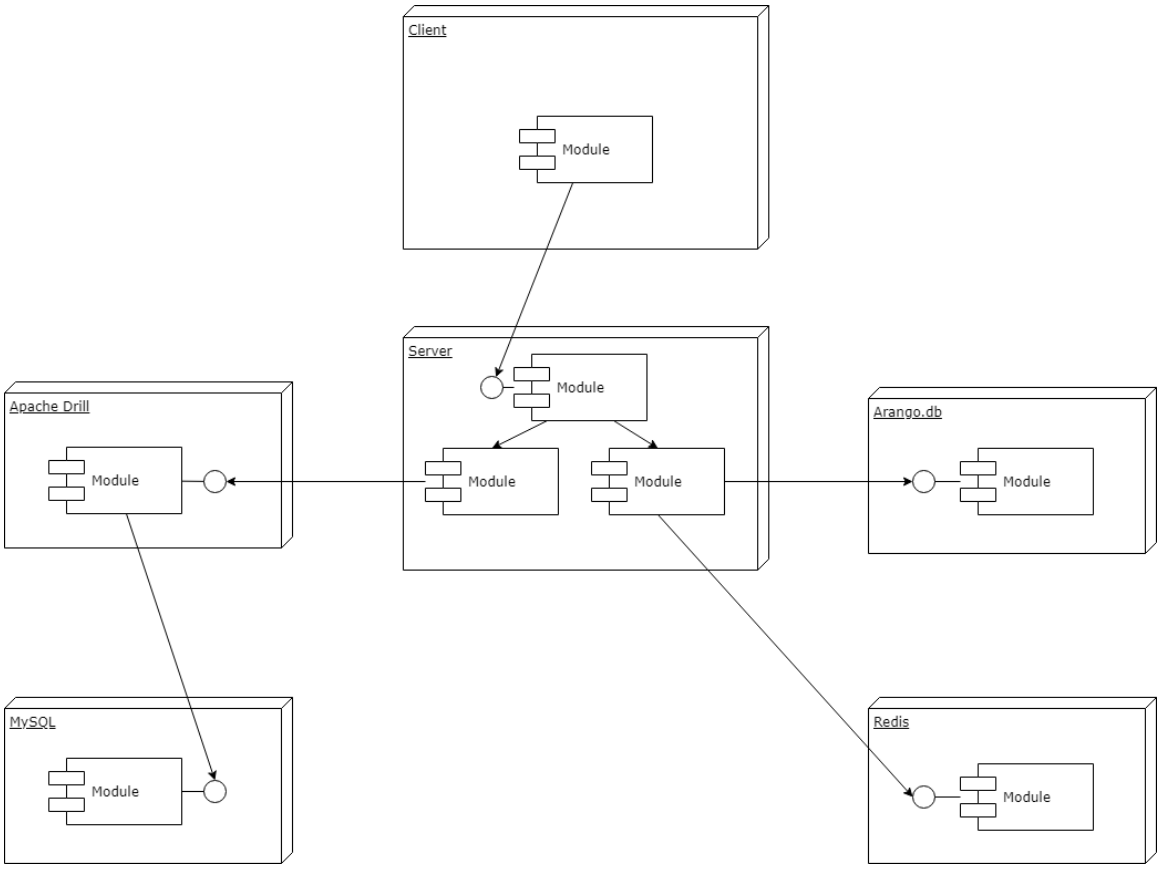


Figure 4.1.: Midas architecture

4.2. Functional Overview

Midas platform is a generic tool for data enrichment. It offers a wide range of functionalities that can help data annalists and other interested parties to enrich their data with open data, to query newly created datasets, to explore open data in addition to existing data.

First of all, Midas platform can be connected to multiple different data sources. Even though those sources can be different (e.g., MySQL, Mongo.Db, etc.), Midas platform gives one single overview of those data in a columnar form. By providing data from various sources in a unified interface, users can explore their data faster. It is worth mentioning that configuration of connection to every data source can be configured directly in the Midas platform.

Data in the Midas platform can be explored by executing SQL-like queries. The result of the query is immediately available. That gives users a possibility to change their query, run it and see the results, all in a single application and a single view. What is also important to mention is that all the queries are cached. By executing queries, multiple users can make a large number of network requests. Also, queries can be run over large datasets with they can provide results that can have a significant impact of data transfer over the network. For all those reasons, all queries are cached, so real-time performance is not affected by a large number of query executions. As we already mentioned, Redis is used as a hash-based cache engine.

In Midas web application, there is an overview of open data sources that are connected with the Midas platform. Each dataset is provided with a description of the data and a description of the source. Also, users can see an example of open data dataset for every open data source that is provided. In that way, users can see the data even before extracting the data itself from remote open data dataset.

All queries can be saved for later use. The main benefit of saving queries is that the original dataset is never changed. Midas creates a virtual dataset that can be saved or exported for future analysis. This gives freedom to users to explore their data, to preprocess their data without worrying about modifying their original dataset. If a saved query is opened at a later time, it will execute itself again, and the user will be presented with the same result as at a time when the query was initially created.

Currently, Midas platform supports exporting data in Tableau format - a data format which can be imported into Tableau software for further analysis.

4.3. User Interface Analysis

Midas client web application is implemented in React.js framework. React.js is used for development of Single Page Applications (SPA). That means that once web application is loaded in the browser, users can use the app as if they are using a program on their computer - navigation through the web application is not triggering page refreshes - all

data is fetched dynamically by using Ajax requests for consuming REST APIs.

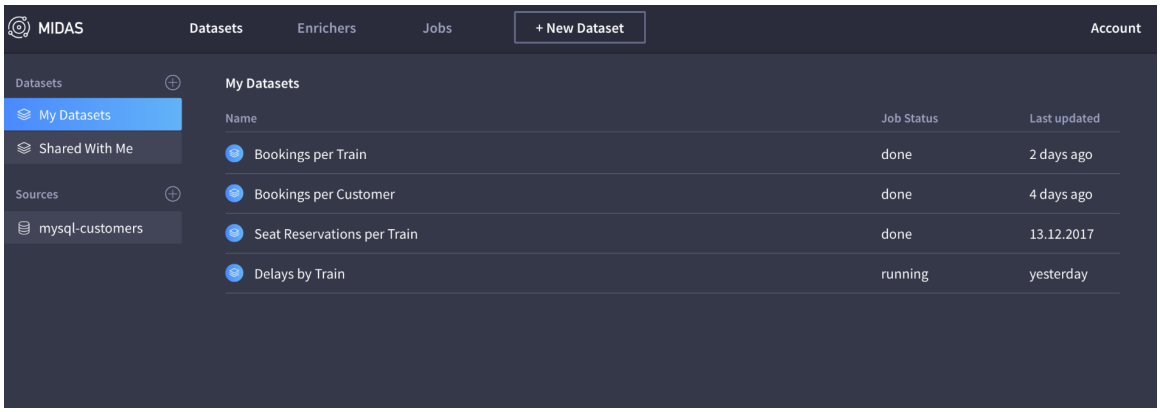


Figure 4.2.: Midas homepage design

In Figure 4.2 we can see that all the essential features are already shown on the homepage. Navigation is simple with nested navigation. Datasets and data sources are grouped in left sidebar while main navigation is attached to the top of the page. And as we have already mentioned, all actions in Midas web application is done without page reloading.

As Midas platform is in the development phase, user interface and user experience will be improved through continuous feedback and development.

5. Implementation

The Midas platform added one module to its portfolio - Midas metadata. The sole purpose of this module is to take care of all metadata and to accommodate the needs of users so that they can understand data easily and faster.

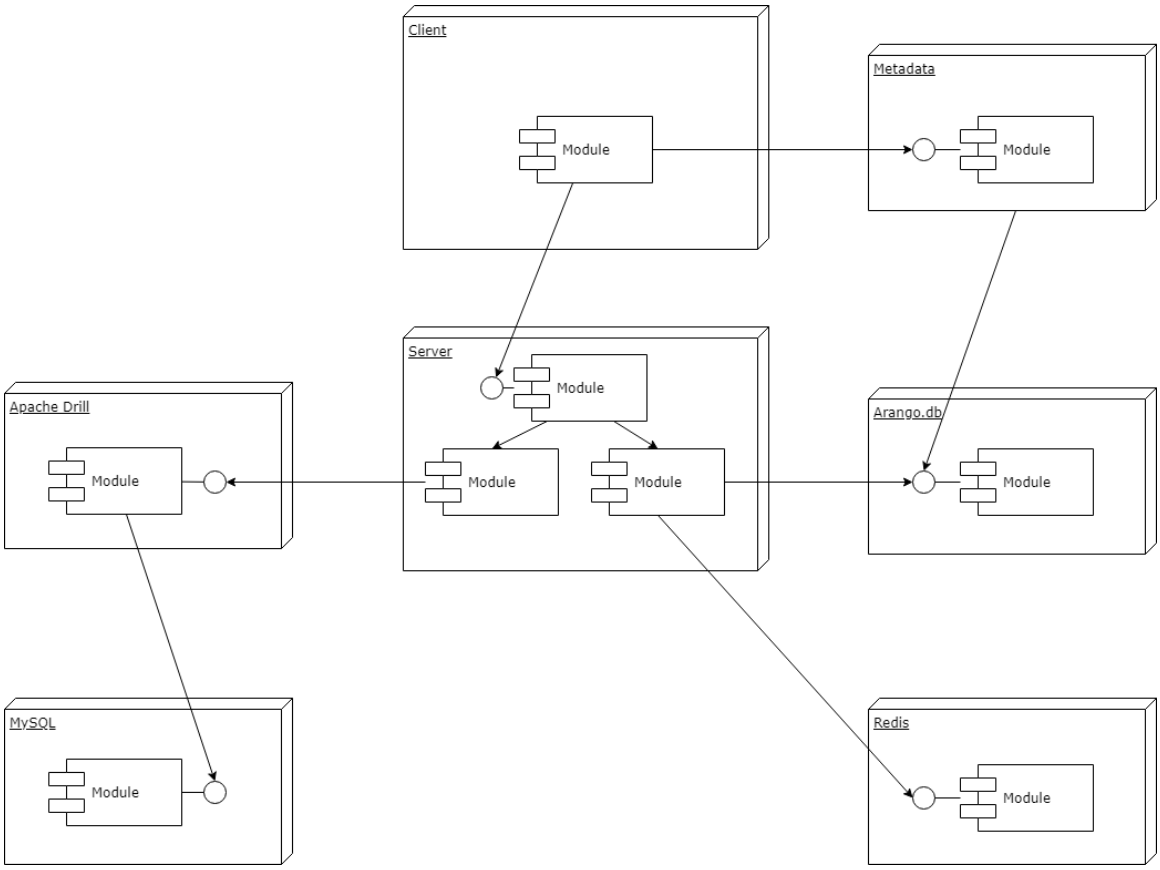


Figure 5.1.: Midas architecture with Midas metadata module

In the Figure 5.1, compared to the Figure 4.1, we can see a completely new module named Metadata. The client application communicates directly with the Metadata module, while the Metadata module communicates directly with Arango.db instance.

There are several design decisions that, in the context of metadata management, led to leaving current Server implementation out:

- There is an intention to re-implement existing server implemented in Node.js to be

implemented in Java technologies - typed programming language where better integration modules exist and with better support;

- To decrease coupling and to increase modularity; whole metadata management was left to reside in a single module;
- In the future, there may be other modules that can be interested in metadata in the Midas platform, which this architecture enables other stakeholders to communicate with the Midas Metadata module directly.

5.1. Technology Stack

Midas Metadata module was implemented using Java technologies. It is based on RESTfull architecture - all functionalities are exposed via RESTfull services - services that can be called using well-established interfaces and network protocols, as described in Chapter 2. It is possible to make full, industry standard, applications using only a small number of dependencies. By keeping the number of dependencies short, maintainability of the whole module increases significantly. Following frameworks and libraries are used in the development of the Midas Metadata module:

- Spring framework[1] - an end-to-end solution for servlet API based applications on Java Virtual Machine. Spring MVC framework provides all needed functionalities to implement RESTfull based services that are maintainable and implemented by following best industry practices;
- Arango.DB Java connector[2] - used for executing Arango.db queries on a remote Arango.db server by using wrapped methods implemented in Java;
- Docker[3] - a tool designed to make it easier to create, deploy, and run applications by using containers. It automates Building and running of the module inside of a container, so it is easier to manage the module and its lifecycle;
- Maven[4] - a build automation tool used primarily for Java projects and it addresses two aspects of building software: how software is built, and it describes software project dependencies;
- Swagger[5] - a tool used for documenting API on-the-fly. Most APIs today use Swagger to create interactive documentation automatically.

List of all dependencies and respective versions is given in Appendix B.

[1] `https://spring.io/` (05.02.2019)
[2] `https://github.com/arangodb/arangodb-java-driver` (05.02.2019)
[3] `https://www.docker.com/` (05.02.2019)
[4] `https://maven.apache.org/what-is-maven.html` (05.02.2019)
[5] `https://swagger.io/` (05.02.2019)

5.2. Metadata Model

In the inception phase, we have to consider the needs of users of the Midas platform, extensibility, maintainability, and also all key points defined in Section 3.3. The core of the Midas metadata module should be generic in a way that it can suit a variety of different metadata. Besides that, it should also provide intuitive and easy to understand interfaces for easier adoption and usage.

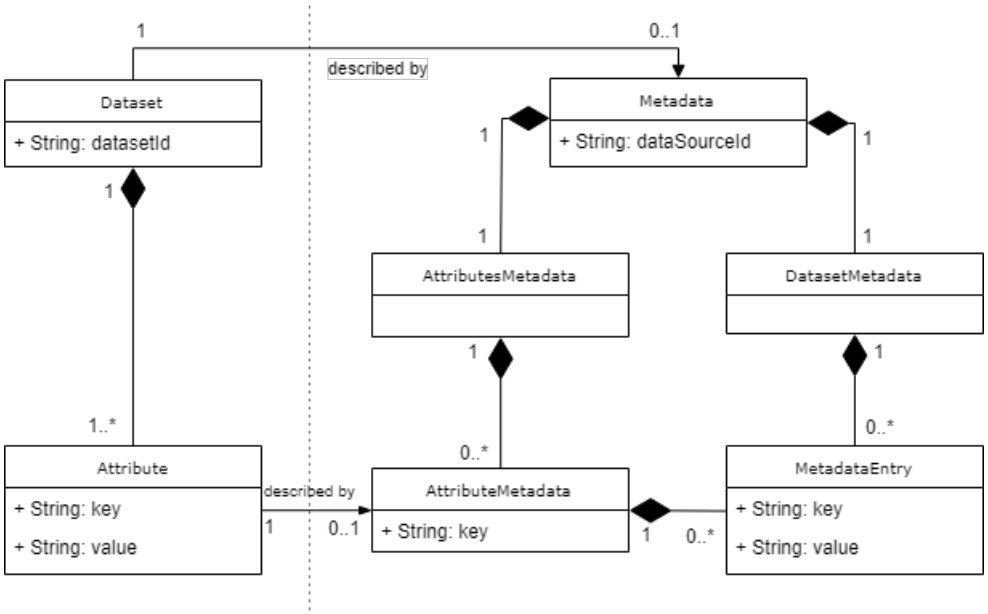


Figure 5.2.: UML diagram of relationship between Dataset and Metadata

UML diagram from Figure 5.2 represents a data model of **Metadata**, which is the core entity from Midas metadata module. To be able to understand design decisions, firstly we have to take a look into the **Dataset** data model. As data in the Midas platform is columnar, **Dataset** is composed of a list of attributes. Besides, **Dataset** also has a *datasetId* which uniquely defines that **Dataset**.

Each **Dataset** can be described with maximum one **Metadata** instance. That is why **Metadata** has an attribute *dataSourceId* which should have the same value as the attribute *datasetId* of the **Dataset**. Also, **Metadata** has two additional properties, **AttributesMetatada** and **DatasetMetadata**. They will hold metadata information for **Dataset attributes** and the **Dataset** itself, respectively. **AttributesMetadata** contains list of **AttributeMetadata**, which contains property *key*. That property has the same value as the *key* property of **Attribute** from the **Dataset**. Because of that, it is possible to easily find metadata for specific **Attribute**. Each **AttributeMetadata** may contain list of **MetadataEntry**-s. **MetadataEntry** serves as a placeholder for one metadata pair - *key* represents metadata property name, while *value* represents metadata property value. In the similar manner, **DatasetMetadata**

contains list of **MetadataEntry**-s that describe the **Dataset** better.

Dotted vertical line in Figure 5.2 divides **Dataset** and **Metadata**. Metadata is managed in in the Midas metadata module. Also, Midas metadata module does not take care of data stored in Datasets, nor it needs that data to provide its functionalities.

5.3. Feature Overview

In this Section, we give an overview of the most important features that we have implemented in the Midas metadata module. Also, we describe the deployment process of the module and how to explore provided services.

5.3.1. From columnar data representation to tree structure (JSON)

As we described in Chapter 4, all data in the Midas platform is shown in tabular format. That helps users to have a view over different data sources. But sometimes that tabular format is challenging to read as some columns are logically related, and some are not.

```
{
    "name": "John",
    lastName": "Doe",
    "address": "Strasse 12",
    "postalCode": "12345",
    "city": "Munich"
}
```

Listing 5.1: Columnar data example

In the Listing 5.1 we can see a small example of columnar data which we can call flat data - all data is represented as key-value where key is an attribute name and value represents attribute value. In addition we can see that some attributes are semantically closer; we can see two grups e.g. *name* and *lastName* of a **person**, and *address*, *postalCode* and *city* of some **location**.

```
{
    "person": {
        "name": "John",
        "lastName": "Doe"
    },
    "location": {
        "address": "Strasse 12",
        "postalCode": "12345",
        "city": "Munich"
    }
}
```

Listing 5.2: JSON data example

In the Listing 5.2 we can see that it is now easier to see attributes that are members of the same logical group. For grouping attributes from flat data into logical groups, JSON format is a natural choice. As JSON is a tree-like structure, we only need a piece of information which node is the parent node.

To provide the information of the parent node of an attribute, we can use metadata attribute **parent**. Some rules should be followed:

- If parent metadata has no value, the attribute is a first level attribute;
- If parent value is another existing node, convert that parent node into a JSON object and put its value as a first attribute (name of that attribute will be previous attribute name + value string, e.g., *address* will become *address-value*);
- If parent value is non-existing in the tree, create a new node.

```
{
    "data": {
        "name": "John",
        "lastName": "Doe",
        "address": "Strasse 12",
        postalCode: "12345",
        city: "Munich"
    },
    "metadata": {
        "dataSourceId": "testSourceId",
        "datasetMetadata":{
        },
        "attributesMetadata": {
            "name": {
                "parent": "person"
            },
            "lastName": {
                "parent": "person"
            },
            "address": {
                "parent": "location"
            },
            "postalCode": {
                "parent": "location"
            },
            "city": {
                "parent": "location"
            }
        }
    }
}
```

Listing 5.3: An example of data to pass to the REST service

In the Listing 5.3 we can see an example of data payload for calling the REST service that is implemented in Midas metadata application. The whole payload is divided into two sections:

1. data - contains flat data that we want to represent in tree-like JSON format
2. metadata - contains a reference if existing metadata (not mandatory) and actual metadata to be used in the process.

Parent attribute in metadata is not mandatory; if it does not exist, an attribute will be considered to be a first level attribute in the new JSON.

The newly implemented service can be called by sending a POST request to /flat2json endpoint. As a response, you will get data in JSON format as in the Listing 5.2.

5.3.2. Search interface for Schema.org entities

One of the drawbacks using Schema.org entities to enrich the metadata is that, at the moment of writing of this thesis, no service was implemented to search for entities that are described in Schema.org.

We have decided to implement a custom implementation of a service that will go through Schema.org and search for the desired entity. Whole Schema.org is described and saved as a single file in JSON-LD format. For each release, Schema.org is saved and stored in open source repository in GitHub[6]. At the moment of writing of this thesis, the latest released version was 3.4.

As we already know from Section 2, JSON-LD format is a valid JSON. Because of that fact, we have downloaded the latest release of Schema.org, and we have created a collection in our Arango.db server and we have put that JSON in our new collection.

We have provided a REST endpoint where users can pass a query parameter by which we will search for entities. Users are also able to specify the number of top results that they would like to get as a response. All results are ranked by a score property which represents a numerical value of similarity between an entity name and query parameter.

```
1 http://${hostname}/entities?query=name&limit=2
```

Listing 5.4: An example URL used for searching for top 2 entities by query "name"

We can see in the Listing 5.4 that endpoint */entities* accepts two query parameters, **query** and **limit**. By tweaking these two parameters, we can provide the desired number of search results. That tweaking mimics familiar behavior of search input fields (e.g., typing the query in an input field and providing several top results on every query change). Going through endless web pages of Schema.org can be tiresome and time-consuming. That is why this search feature represents an important part of the Midas metadata module that can significantly increase the usability of the system.

[6] https://github.com/schemaorg/schemaorg/tree/master/data/releases

```
[
    {
        "referent": {
            "properties": {
                "http://www.w3.org/2002/07/owl#equivalentProperty": {
                    "@id": "http://purl.org/dc/terms/title"
                },
                "rdfs:label": "name",
                "@type": "rdf:Property",
            "http://schema.org/rangeIncludes": {
                    "@id": "http://schema.org/Text"
                },
                "rdfs:comment": "The name of the item.",
                "rdfs:subPropertyOf": {
                    "@id": "rdfs:label"
                },
                "http://schema.org/domainIncludes": {
                    "@id": "http://schema.org/Thing"
                },
                "@id": "http://schema.org/name"
            }
        },
        "string": "name",
        "score": 100,
        "index": 370
    },
    {
        "referent": {
            "properties": {
                "rdfs:label": "alternateName",
                "@type": "rdf:Property",
                "http://schema.org/rangeIncludes": {
                    "@id": "http://schema.org/Text"
                },
                "rdfs:comment": "An alias for the item.",
                "http://schema.org/domainIncludes": {
                    "@id": "http://schema.org/Thing"
                },
                "@id": "http://schema.org/alternateName"
            }
        },
        "string": "alternateName",
        "score": 90,
        "index": 278
    }
]
```


Listing 5.5: An example response for query from Listing 5.4

We can see in the Listing 5.5 that a response from the service comes in a format of a

sorted JSON array. Attributes of the individual elements in the response are:

- **string** - contains entity name;
- **score** - numerical value of confidence that an entity is an entity that we are searching for;
- **referent.properties** - represents the actual Schema.org entity extracted from Schema.org stored in our Arango.db collection. It is in an unaltered form in case the information is needed in a JSON-LD format that is defined by Schema.org.

Core functionality is provided by using FuzzyWuzzy[7] java library which is a Java version of popular and well-maintained python library for String similarity calculation. It compares query string against entity name and calculates the score which represents a numerical value of confidence that two strings are similar. The algorithm uses Levenshtein distance [8] to calculate similarity between strings.

5.3.3. Create, Read, Update and Delete Metadata Entities

To be able to manage metadata that is stored in Arango properly.db we have created full CRUD services for Metadata entities. When we use term *Metadata entity* we are thinking of whole metadata for one dataset.

```
{
    "dataSourceId": "dataset_id",
    "datasetMetadata":{
        "meta_1": "meta_1_value",
        "meta_2": "meta_2_value",
        "meta_3": "meta_3_value"
    },
    "attributesMetadata": {
       "attribute_1": {
            "meta_1": "meta_1_value",
            "meta_2": "meta_2_value",
            "meta_3": "meta_3_value"
        },
        "attribute_2": {
            "meta_1": "meta_1_value",
            "meta_2": "meta_2_value",
            "meta_3": "meta_3_value"
        }
    }
}
```

Listing 5.6: Metadata model template

[7] https://github.com/xdrop/fuzzywuzzy (15.01.2019)
[8] https://www.cuelogic.com/blog/the-levenshtein-algorithm (15.01.2019)

In the Listing 5.6 we can see a model of Metadata entity. It is descriptive and extensive enough to support many use cases. Only one attribute is mandatory - **dataSourceId**. That attribute is uniquely used to identify a dataset for which this metadata is stored. Every other first level attribute has the same name as a column in a dataset referenced by **dataSourceId**. Each attribute can have from zero to many attributes which represent actual metadata for that specific column. In addition to metadata on attribute level, it is possible to add metadata on the dataset level. Some metadata are applicable for all attributes, and some metadata are only concerning the dataset, e.g., creation date, owner, etc.

All CRUD operations are supported only on Metadata entity level. Updating Metadata on an attribute level would increase the number of network requests, the complexity of Arango.db queries and in the end maintainability of those services. The payload for transferring metadata by network request is today considered as a small payload. Metadata model similar to the one that we have shown in Listing 5.6 but with 20 attributes and each with three metadata properties takes around 3KB in an uncompressed JSON file.

```
1 http://${hostname}/metadata/{dataSourceId}
```

Listing 5.7: An endpoint used for CRUD operations for Metadata entities

Supported operations are:

- Create - */metadata*

  Request Type: POST

  Payload: Metadata entity

  Response: Newly created Metadata entity

- Read - */metadata/{dataSourceId}*

  Request Type: GET

  Response: Metadata entity with provided dataSourceId

- Update - */metadata/{dataSourceId}*

  Request Type: PUT

  Payload: Metadata entity

  Response: Newly updated Metadata entity

- Delete - */metadata/{dataSourceId}*

  Request Type: DELETE

  Response: Old deleted Metadata entity

By using this structure of requests, we are following well-known intuitive RESTfull principles. This structure also enables us to extend service offerings in case of future development easily.

5.3.4. Metadata enrichment

In addition to CRUD operations on Metadata entity, it is possible to enrich Metadata entity by calling one specific service.

```
http://${hostname}/metadata/enrich
```

Listing 5.8: An endpoint used for enrichment of Metadata entity

Service for Metadata enrichment expects a POST request with Metadata entity as a payload.

Enrichment of a Metadata entity works in a manner that it checks if an attribute has a metadata property **type.label** which should contain the name of the entity in Schema.org specification. Based on that label, service finds the appropriate entity in Schema.org, and it converts each attribute of that entity and attaches it as metadata to the original attribute.

```
{
    "dataSourceId": "testSourceId",
    "datasetMetadata":{
    },
    "attributesMetadata": {
        "name": {
            "type.label": "givenName"
        }
    }
}
```

Listing 5.9: Example payload for enrichment

```
{
    "dataSourceId": "testSourceId",
    "datasetMetadata":{
    },
    "attributesMetadata": {
        "name": {
            "parent": null,
            "type.label": "givenName",
            "rdfs:label": "givenName",
            "@type": "rdf:Property",
            "http://schema.org/rangeIncludes": "{@id=http://schema.org/Text}",
            "rdfs:comment": "Given name. In the U.S., the first name of a Person. This can be used along with familyName instead of the name property.",
            "http://schema.org/domainIncludes": "{@id=http://schema.org/Person}",
            "@id": "http://schema.org/givenName"
        }
    }
}
```

Listing 5.10: Example response for enrichment of the payload from Listing 5.9

In addition to this service that returns enriched Metadata entity without saving it, it is possible to do this enrichment on entity creation mentioned in Section 5.3.3. To trigger this enrichment, you only have to provide a query parameter *enrich=true*. The result will be saved in Arango.db for later usage.

```
http://${hostname}/metadata?enrich=true
```

Listing 5.11: Metadata enrichment on creation

5.4. Deployment

The configuration of the application is stored in a separate file under name **config.properties**.

```
get.entity=FOR d IN schema_entities FOR p IN d.@graph FOR t IN p.@graph FILTER t.`rdfs:label` == @label RETURN t
get.entities=FOR d IN schema_entities FOR p IN d.@graph FOR t IN p.@graph RETURN t
get.metadata=FOR m IN metadata FILTER m.dataSourceId == @dataSourceId RETURN m
get.metadata.key=FOR m IN metadata FILTER m.dataSourceId == @dataSourceId RETURN m._key
delete.metadata=FOR m IN metadata FILTER m.dataSourceId == @dataSourceId REMOVE m IN metadata LET removed=OLD RETURN removed

//Arango.DB configuration
host=127.0.0.1
port=32768
password=123456789
collection.metadata.name=metadata
```

Listing 5.12: Midas Metadata configuration

What we can configure are queries that we are using for CRUD operations on Metadata entities. Important part is the configuration of Arango.Db connection parameters. We can configure *host*, *port* and *password*. Property *collection.metadata.name* represents name of the collection where all metadata will be stored.

This application was developed using Maven as a build tool. Maven enables us to pack our project as a Web Application Archive (WAR file) which is deployable on Java Servlet Container servers like Tomcat. WAR is equivalent to Java archives (JAR), but it is used for applications that are meant to be used on the web.

To generate the WAR file, you have to navigate to the root of the project and to execute **mvn package**. WAR file will be stored in folder *target* under name **midas-metadata.war**.

To make it easier for someone to start this project after generating the WAR file, we have provided a Dockerfile script which will deploy generated Application to a Tomcat server.

```
FROM tomcat:9.0.16-jre8
COPY /target/midas-metadata.war /usr/local/tomcat/webapps/
```

Listing 5.13: Simple Dockerfile for Midas metadata application

In the Listing 5.13 we can see that we can quickly start Tomcat web-server and deploy our Midas metadata application. It uses Docker image tomcat:9.0.16-jre8 and copies the Midas metadata WAR file into its folder *webapps*. That means that when we run the image created using this docker file, our application will automatically start.

```
1 docker build -t midas_metadata .
2 docker run -d -p 8080:8080 --name midas_metadata midas_metadata
```

Listing 5.14: Docker commands for Midas metadata module deployment

5.5. Documentation

All REST services were documented using a tool named Swagger, which helps to maintain up-to-date documentation of provided REST services while enforcing best practices for writing those services. If the REST services development is done by following the best practices, Swagger can generate documentation automatically.

Documentation can be provided as an interactive web application in which users can test provided services. It also includes sample input for requests and sample output for service responses. In Figure 5.3 we can see nodes in the web page that are generated automatically. By opening a specific node, it makes it possible to explore the provided service.

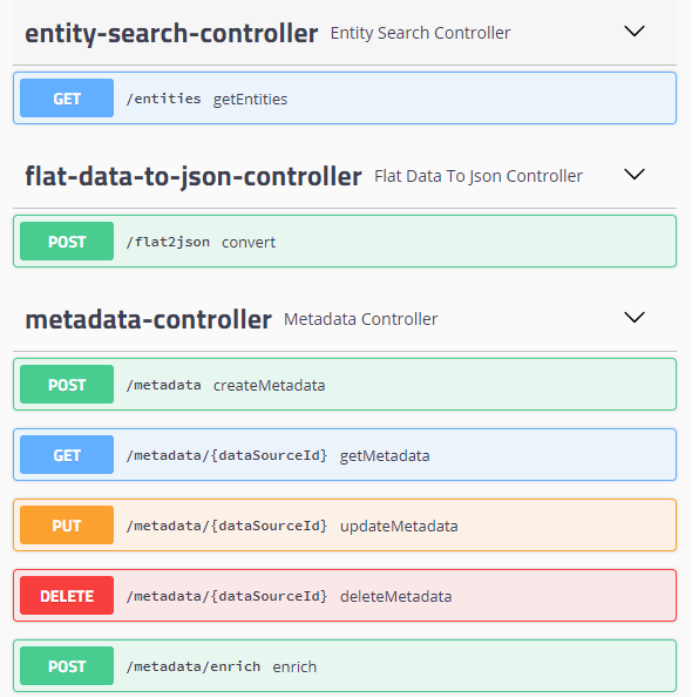


Figure 5.3.: REST Api documentation generated by Swagger in web application form

6. Evaluation

First, in Section 6.1, we are going to propose an approach for measuring the fitness of a semantic label for an unlabeled attribute of the dataset. In Section 6.2, we are going to apply a method proposed in Section 6.1 on Open Data datasets offered by the Midas platform and discuss the results that we have obtained.

6.1. The 4-level scoring method

As it is difficult to find a perfect semantic match from Schema.org, we are proposing a simple method for rating the fitness of semantic annotation to annotate an unlabeled entity.

Name	Last Name	Address	Postcode	Group Name	Result	Result Percentile
John	Doe	Wonderland 10b	11000	B3	45	20
Alice	Ranger	Moonstreet 21	99123	C1	55	3

Table 6.1.: Exam results

In Table 6.1 we can see an example table with exam results for two students. Domain expert who created that data can add metadata so that other colleagues can better understand what dataset represents.

Tim Berners-Lee's 5-star open data rating system [9] is an inspiration for defining this new scoring method. We are proposing a scoring mechanism with a four-level scale that rates if an entity from Schema.org is a good fit to annotate an unlabeled attribute of a dataset:

- **0** - There was no entity defined in Schema.org vocabulary that could describe an attribute. (e.g., a percentage of any kind);
- **1** - Entity could be described only by its data type (e.g. Number or Text);
- **2** - There is an entity that semantically describes an unlabeled entity, but it does not describe it unambiguously (e.g., Branch Code can is a property of both City and Country entities);
- **3** -There is an entity in Schema.org vocabulary that fully describes an unlabeled attribute in the dataset.

Attribute	Score	Schema.org Entity URL
Name	3	`https://schema.org/givenName`
Last Name	3	`https://schema.org/familyName`
Address	3	`https://schema.org/address`
Postcode	3	`https://schema.org/postalCode`
Group Name	2	`https://schema.org/name`
Result	1	`https://schema.org/Number`
Result Percentile	0	-

Table 6.2.: Semantic annotations from Schema.org and score values for data from Table 6.1

In Table 6.2 we show scores based on the proposed scoring method and entity mappings from Schema.org vocabulary:

- **Name** - It represents the first name of the student. There is an entity in Schema.org vocabulary that is used to describe the first name of a person - *givenName*; thus it achieves the maximum score of three.
- **Last Name** - It represents the last name of the student. There is an entity in Schema.org vocabulary that is used to describe the last name of a person - *familyName*; thus it achieves the maximum score of three.
- **Address** - It represents the address of the student. There is an entity in Schema.org vocabulary that is used to describe address - *address*; thus it achieves the maximum score of three.
- **Postcode** - It represents the postal code of the students residence. There is an entity in Schema.org vocabulary that is used to describe the postal code - *postalCode*; thus it achieves the maximum score of three.
- **Group Name** - It represents the name of the group in which student belongs. There is no entity in Schema.org vocabulary that is used to describe the group name fully. That is why we have to search for a broader semantic definition that can semantically describe Group Name - *name*; thus it achieves the score of two.
- **Result** - It represents the result that the student has achieved in the exam. As there are no entities in Schema.org vocabulary to describe Exam result, or result, the only piece of information we can attach is that Result attribute represents a *number*; thus it achieves the score of one.
- **Result Percentile** - It represents the percentile in which the student's exam results fit in comparison to all exam results. As there is no entity in Schema.org vocabulary to describe percentiles, we could not find any entity to describe Result Percentile property semantically; thus it achieves the score of zero.

By using the proposed method, we can understand better if the dataset is annotated correctly or not. A binary comparison between cases of *there is an entity in Schema.org* and *there is no entity in Schema.org*, cannot provide us with information if annotations are the appropriate fit for the labeled entity.

We could have also annotated only attributes that have rating *three* by the proposed scoring method, but that approach would have several downsides. First of all, that would leave us with a high number of unlabeled attributes, which would not make sense if someone wants to get to know the dataset by reading about those attributes. Also, we would miss the opportunity to provide at least a piece of information about the attribute that can define that attribute better (e.g., *deaths* - it is valuable to know if that attribute represents a *Number* of deaths or if that attribute is an *Enumeration* that could represent the type of death).

As all attributes have a score, we can use them to calculate a dataset score. It is a number on a scale from zero to one hundred that represent how well semantic annotations describe attributes of that dataset. We calculate dataset score by using formula from Figure 6.1.

$$x' = \frac{x - min(x)}{max(x) - min(x)} * 100$$

Figure 6.1.: min-max normalization [41] on a scale from 0 to 100

We calculate a maximum possible score by multiplying the number of attributes with number three, which is the best score that represents a good fit. So the maximum possible score contains two information - the number of attributes and the maximum achievable score by labeling those attributes. In the same manner, by multiplying the number of attributes with zero, which is the minimum possible attribute score, we can calculate the minimum score. Also, we multiply the normalized score with one hundred to represent normalized score on a scale from zero to one hundred.

Based on data that we have in Table 6.2, we have:

$$x = 3 + 3 + 3 + 3 + 2 + 1 + 0 = 15$$

$$max(x) = 7 * 3 = 21$$

$$min(x) = 7 * 0 = 0$$

$$x' = \frac{15 - 0}{21 - 0} * 100 = 71$$

Figure 6.2.: Dataset score calculation based on the data provided in Table 6.2

Dataset score calculated like that can show the quality of annotations of its attributes - a higher score means that dataset is more correctly (semantically) annotated.

6.2. Results

Currently, Midas platform is offering 22 different Open Data datasets for exploration. All datasets had no semantic annotations, which made those datasets difficult to explore if the user is not familiar with the data. We have manually mapped all attributes from those 22 datasets to some entity from Schema.org with a purpose of describing those attributes better.

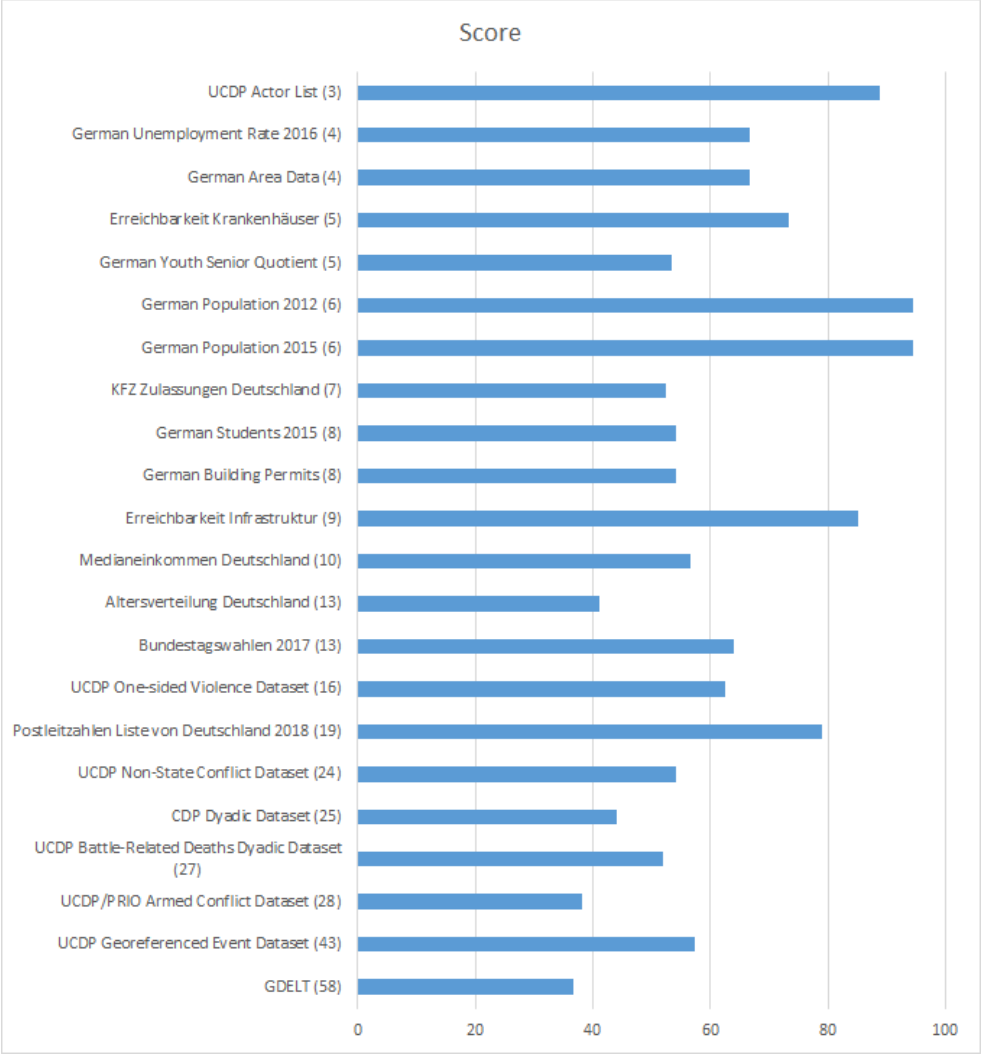


Figure 6.3.: Normalized score per dataset

In Figure 6.3 we can see the scores for all of the 22 Open Data datasets from Midas platform. Datesets are sorted by number of attributes. Figure 6.3 shows on y-axis names of datasets and their number of attributes. On x-axis it shows score range from zero to one hundred. We can see the tendency that datasets with a higher number of attributes do not have such a good score. Usually, datasets with a large number of attributes combine

several closely connected entities into one single dataset. That makes it difficult to find proper entities for describing the dataset because Schema.org represents general vocabulary with a small number of specific definitions that usually do not have a large number of attributes. Average dataset score is 62, while median dataset score is 57.

Results shows that full semantic annotation of a dataset is hard to achieve using single vocabulary. Usage of industry specific vocabularies could increase overall dataset score.

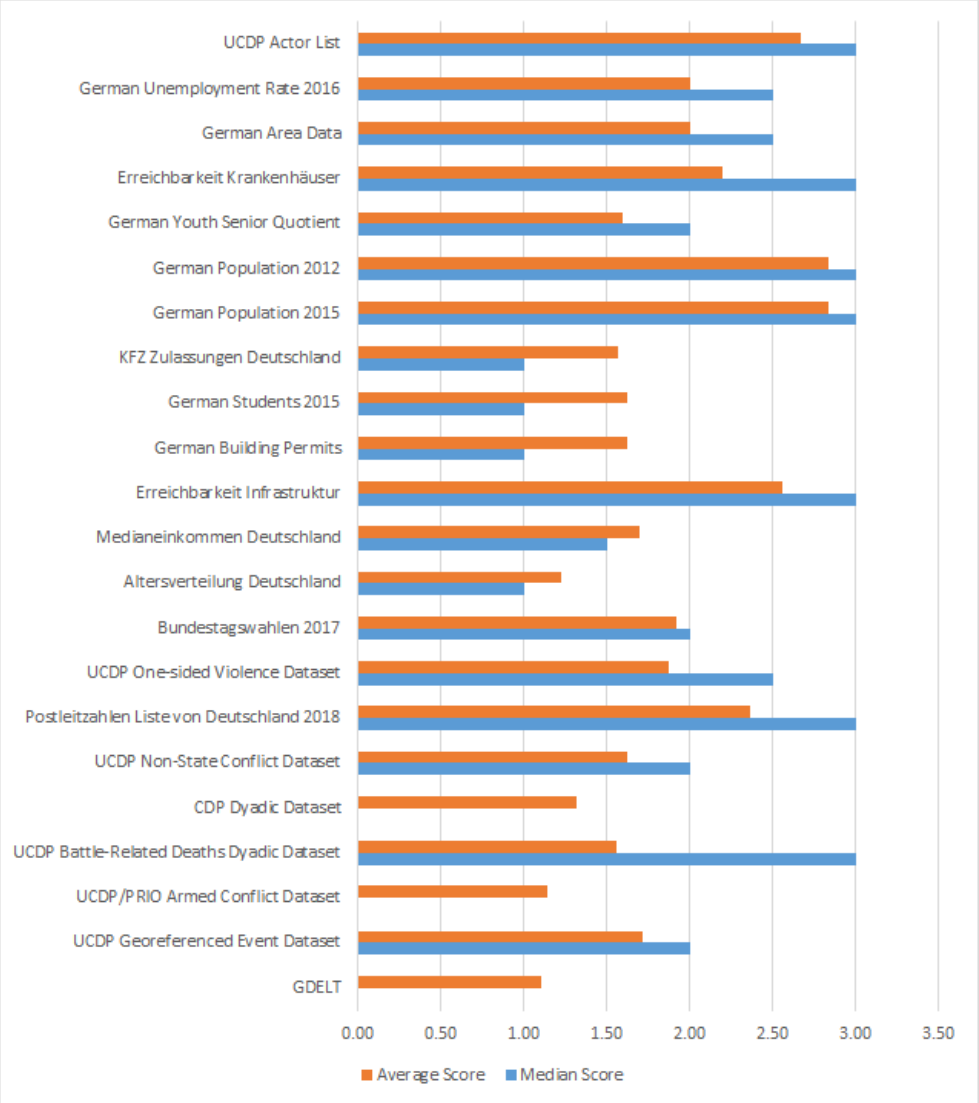


Figure 6.4.: Comparison between Average attribute score and Median attribute score per dataset

In Figure 6.4 we can see an average score per dataset. We can see that most datasets have an average score of around 1.5, which does not say a lot if we only have scores of 0 and scores of 3. That is why it makes sense to look at the average score together with the

median score. We can see that some datasets have an average score of more than 1, but the median is 0, which means that most attributes are unlabeled, while only some may have a good score.

Distribution of attribute scores per dataset can be found in Appendix A.

We can see that scores vary in quality. To be able to benefit fully from this approach, it is necessary to model data based on the existing schema entity. Another approach would be to create a custom vocabulary, similar to Schema.org vocabulary, for describing the data.

7. Conclusion

The following chapter provides a conclusion about the research results, gained in this thesis. First, the key findings for each Research Question are summarized. Furthermore, possible limitations for this research are presented. Finally, a short insight into possible future work is given.

7.1. Retrospective

In the end, we can have a retrospective about research questions defined in the Introduction of this thesis:

- **RQ 1 - How are state-of-the-art solutions handling metadata?**

  We have shown and described several data-driven platforms. What we can conclude here is that the management of metadata is not an exact science. Every vendor models metadata based on their specific needs. Besides, we find that vendors did not identify semantic metadata as a key feature that could drive their business strategies. Initiatives like Open Data Initiative and Data Transfer Project show us that some big players in industry recognized the need for model consolidation, and with it, metadata handling will consolidate and Open Data initiative will be able to evolve further.

  We can also conclude that platforms do not want to bother end users with complex data or metadata management solutions. The focus is always on fast result delivery, whether that is analytics, pipeline processing or something else. Modularity is also an important aspect, not only from software engineering best practices but also from a business perspective. Every model can be seen as a new business opportunity to provide additional features to your client.

- **RQ 2 - How to add metadata to existing columnar data?**

  By analyzing well-established industry platforms, we can conclude that there is no best practice on how to approach this problem. It heavily depends on processes that specific vendor is using. We could derive enough knowledge from our analysis to define our own design decisions. We have decided to add metadata on a dataset level. In addition to that, we wanted to increase granularity and add metadata on

attribute level. With that, we can gain better flexibility for end users to store specific knowledge about their data.

We have also concluded that defining specific metadata properties can close some opportunities for the end users. That is why we decided to design our solution in a way that is possible to add arbitrary metadata properties (keys and values). In that way, users can leverage our solution concerning their already defined and well-known metadata.

- **RQ 3 - Where to store metadata?**

  In our analysis, we have seen different approaches to this problem. We have also discovered systematic classification for metadata location concerning where we store data. We have decided to externalize our metadata in a system that manages metadata separately concerning the system where we store data. We believe that this design decision can bring added value to the whole Midas platform. Now, the Midas platform manages metadata by leveraging REST APIs, open formats and fast Document based storage. That gives us an opportunity to enrich metadata separately from our data management. Also, we can decide if we want to use metadata in our solution, we can store different version, and at some point we will be able to provide some background processing that could improve our processes without interfering with end users - caching metadata, enriching of metadata in background, different entry point into metadata management system etc..

- **RQ 4 - How to leverage public vocabularies in metadata management?**

  We wanted to focus on Semantic Web as we strongly believe that the Internet is developing in that direction. Actions of big players in the IT world are evidence for our conclusion. Schema.org vocabulary is extensively used, mainly in search engines and for providing structured data on web pages. We wanted to leverage global knowledge, and that is why we have decided to evaluate possible usage of Schema.org vocabulary in our system. We have concluded that if the dataset contains general terms from everyday life, leveraging Schema.org entities can help us to describe our data faster semantically. The problem is that many datasets contain specific attributes (e.g., specific industry, research field, etc.). We have designed our solution in a way that it can similarly support other vocabularies. The downside to this approach that it will require some development effort to fully accommodate new features because the structure of vocabularies can slightly vary in structure.

7.2. Future Work

All of the design decisions are giving us the opportunity to work on the provided solution continuously. Based on the industry feedback, we can improve functionalities and expose

new ones via the REST API.

It would also be nice to have a web application that could be used explicitly for metadata management. It could be used for adding new metadata, editing existing metadata, managing of supported vocabularies and many more. It can be used by end users, or by back-office users who are managing data.

The interesting approach can be explored - matching datasets based on their metadata. With background processing of all metadata for different datasets, we can provide information which datasets are compatible for merge. We can also recommend how to get the largest dataset available based on steps for joining one dataset with other datasets.

Also, support for other formats can be added. In particular, we already can provide enough information for generating JSON-LD data. The problem that has to be solved is the feasibility and usability of such a feature for general use cases. Also, even though XML is considered old concerning JSON data format, many systems are still using XML in their internal system. XML is very powerful, and many specification and tools are created to support various business strategies.

Finally, by following industry best practices for software development, a newly developed system can provide the needed level of agility to support the evolving nature of data integration and data enrichment ecosystem.

List of Figures

List of Tables

List of Abbreviations

JSON-LD Javascript Object Notation for Linked Data

URI Unique Resource Identifier

CRUD Create Read Update Delete

REST Representational State Transfer

JSON Javascript Object Notation

WAR Web Application Archive

JAR Java Archive

XML Extensible Markup Language

CSV Comma Separated Values

IRI Internationalized Resource Identifier

URI Uniform Resource Identifier

SEO Search Engine Optimization

AI Artificial Intelligence

CLI Command Line Interface

ETL Extract Transform Load

Listings

Bibliography

[1] "Data integration reaches inflection point: Survey results," IBM, 2018, accessed 12-11-2018. [Online]. Available: https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=78013078USEN

[2] A. Basu, "Semantic web, ontology, and linked data," 01 2017.

[3] A. Lausch, A. Schmidt, and L. Tischendorf, "Data mining and linked open data - new perspectives for data analysis in environmental research," *Ecological Modelling*, vol. 295, pp. 5 – 17, 2015, use of ecological indicators in models. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304380014004335

[4] J. S. Ward and A. Barker, "Undefined by data: A survey of big data definitions." *CoRR*, vol. abs/1309.5821, 2013. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1309.html#WardB13a

[5] G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati, "Using ontologies for semantic data integration," pp. 187–202, 05 2018. [Online]. Available: https://rd.springer.com/chapter/10.1007/978-3-319-61893-7_11

[6] M. Lenzerini, "Data integration: A theoretical perspective," in *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '02. New York, NY, USA: ACM, 2002, pp. 233–246. [Online]. Available: http://doi.acm.org/10.1145/543613.543644

[7] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The Semantic Web*, K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 722–735.

[8] AwesomeData, "Awesome public datasets," 2014, accessed 28-02-2019. [Online]. Available: https://github.com/awesomedata/awesome-public-datasets

[9] T. Berners-Lee, "Is your linked open data 5 star?" 2009, accessed 26-01-2019. [Online]. Available: https://www.w3.org/DesignIssues/LinkedData.html

[10] P. Archer, L. Bargiotti, M. De Keyzer, S. Goedertier, N. Loutas, and F. Van Geel, "Report on high-value datasets from eu institutions," 2014, accessed 28-02-2019. [Online]. Available: https://ec.europa.eu/isa2/sites/isa/files/publications/report-on-high-value-datasets-from-eu-institutions_en.pdf

[11] D. Tosi and S. Morasca, "Supporting the semi-automatic semantic annotation of web services," *Inf. Softw. Technol.*, vol. 61, no. C, pp. 16–32, May 2015. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2015.01.007

[12] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, 2007. [Online]. Available: https://doi.org/10.2753/MIS0742-1222240302

[13] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Q.*, vol. 28, no. 1, pp. 75–105, Mar. 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=2017212.2017217

[14] 5startdata, "5-star open data deployment scheme," 2018, accessed 26-12-2018. [Online]. Available: https://5stardata.info/en/

[15] M. Peters, "Germany finally has an open data law," 2017, accessed 26-01-2019. [Online]. Available: https://www.opengovpartnership.org/stories/germany-finally-has-open-data-law

[16] M. Janssen, Y. Charalabidis, and A. Zuiderwijk, "Benefits, adoption barriers and myths of open data and open government." *Information Systems Management*, vol. 29, no. 4, pp. 258 – 268, 2012. [Online]. Available: http://search.ebscohost.com.eaccess.ub.tum.de/login.aspx?direct=true&db=bth&AN=82249540&site=ehost-live

[17] E. Wilde, C. Pautasso, and R. Alarcon, *REST: Advanced Research Topics and Practical Applications*. Springer, 01 2014.

[18] S. Subhashree, R. Irny, and P. Sreenivasa Kumar, "Review of approaches for linked data ontology enrichment," in *Distributed Computing and Internet Technology*, A. Negi, R. Bhatnagar, and L. Parida, Eds. Cham: Springer International Publishing, 2018, pp. 27–49.

[19] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web*, vol. 6, pp. 167–195, 2015.

[20] M. Lanthaler and C. Gütl, "On using json-ld to create evolvable restful services," in *Proceedings of the Third International Workshop on RESTful Design*, ser. WS-REST '12. New York, NY, USA: ACM, 2012, pp. 25–32, accessed 12-11-2018. [Online]. Available: http://doi.acm.org/10.1145/2307819.2307827

[21] R. Trypuz, "About schema.org initiative," 2016, accessed 11-02-2019. [Online]. Available: https://www.w3.org/community/meat/2016/02/04/more-about-our-mission/

[22] R. Guha, "Introducing schema.org: Search engines come together for a richer web," 2011, accessed 11-02-2019. [Online]. Available: https://googleblog.blogspot.com/2011/06/introducing-schemaorg-search-engines.html

[23] M. Rouse, "Metadata," 2014, accessed 28-02-2019. [Online]. Available: https://whatis.techtarget.com/definition/metadata

[24] E. Duval and W. Hodgins, "Metadata principles and practicalities," *D-Lib Magazine*, vol. 8, p. 2002, 2002.

[25] I. P. M. W. Group, "Embedded metadata manifesto," 2011, accessed 01-03-2019. [Online]. Available: http://www.embeddedmetadata.org/embedded-metatdata-manifesto.php

[26] J. Stander, "Managing data beyond boundaries: Third-party metadata collection," 2015, accessed 01-03-2019. [Online]. Available: https://blogs.sas.com/content/datamanagement/2015/12/23/managing-data-beyond-boundaries-third-party-metadata-collection/

[27] NISO, *Understanding metadata*. 4733 Bethesda Avenue, Suite 300, Bethesda, MD 20814 USA: NISO, 2004, iSBN: 1880124629. [Online]. Available: http://www.niso.org/publications/press/UnderstandingMetadata.pdf

[28] S. Handschuh, S. Staab, and A. Maedche, "Cream: creating relational metadata with a component-based, ontology-driven annotation framework." in *K-CAP*. ACM, 2001, pp. 76–83. [Online]. Available: http://dblp.uni-trier.de/db/conf/kcap/kcap2001.html#HandschuhSM01

[29] P. Cimiano, S. Handschuh, and S. Staab, "Towards the self-annotating web," in *Proceedings of the 13th International Conference on World Wide Web*, ser. WWW '04. New York, NY, USA: ACM, 2004, pp. 462–471. [Online]. Available: http://doi.acm.org/10.1145/988672.988735

[30] A. L. Egyedi, M. J. O'Connor, M. Martinez-Romero, D. Willrett, J. Hardi, J. Graybeal, and M. A. Musen, "Using Semantic Technologies to Enhance

Metadata Submissions to Public Repositories in Biomedicine," 12 2018. [Online]. Available: https://swat4hcls.figshare.com/articles/Using_Semantic_Technologies_to_Enhance_Metadata_Submissions_to_Public_Repositories_in_Biomedicine/7324175

[31] P. G. Kolaitis, "Schema mappings, data exchange, and metadata management," in *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '05. New York, NY, USA: ACM, 2005, pp. 61–75. [Online]. Available: http://doi.acm.org/10.1145/1065167.1065176

[32] M. Nagarajan, *Semantic Annotations in Web Services*. Boston, MA: Springer US, 2006, pp. 35–61. [Online]. Available: https://doi.org/10.1007/978-0-387-34685-4_2

[33] P. Wang, Y. He, R. Shea, J. Wang, and E. Wu, "Deeper: A data enrichment system powered by deep web," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: ACM, 2018, pp. 1801–1804, accessed 12-11-2018. [Online]. Available: http://doi.acm.org/10.1145/3183713.3193569

[34] V. Lopez, C. Unger, P. Cimiano, and E. Motta, "Evaluating question answering over linked data," *Journal of Web Semantics*, vol. 21, pp. 3 – 13, 2013, special Issue on Evaluation of Semantic Technologies. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S157082681300022X

[35] W. Alpha, "Wdf (wolfram data framework)," 2018, accessed 20-12-2018. [Online]. Available: https://reference.wolfram.com/language/guide/WDFWolframDataFramework.html

[36] P. Ristoski, C. Bizer, and H. Paulheim, "Mining the web of linked data with rapidminer," *Web Semant.*, vol. 35, no. P3, pp. 142–151, Dec. 2015, accessed 23-12-2018. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2015.06.004

[37] K. Sato, "An inside look at google bigquery," 2012, accessed 28-02-2019. [Online]. Available: https://cloud.google.com/files/BigQueryTechnicalWP.pdf

[38] J. Tigani and S. Naidu, *Google BigQuery Analytics*, 1st ed. Wiley Publishing, 2014.

[39] Microsoft, "Announcing the open data initiative," 2018, accessed 26-12-2018. [Online]. Available: https://www.microsoft.com/en-us/open-data-initiative

[40] "Data transfer project overview and fundamentals," Data Transfer Project, July 2018, accessed 12-11-2018. [Online]. Available: https://datatransferproject.dev/dtp-overview.pdf

[41] S. G. K. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *CoRR*, vol. abs/1503.06462, 2015. [Online]. Available: https://www.researchgate.net/publication/274012376_Normalization_A_Preprocessing_Stage

Appendix

A. Evaluation

A.1. Additional charts with score distribution per dataset

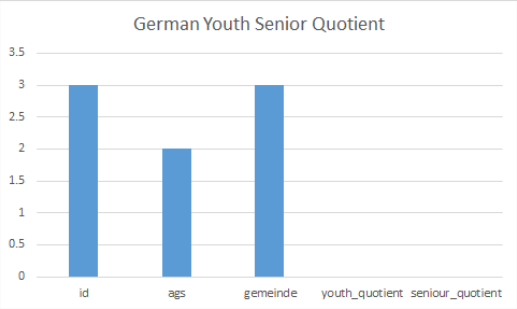


Figure A.1.: Score distribution *1*

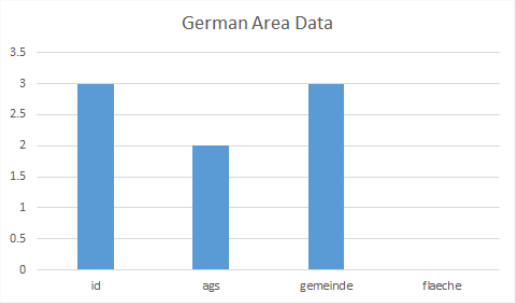


Figure A.2.: Score distribution *2*

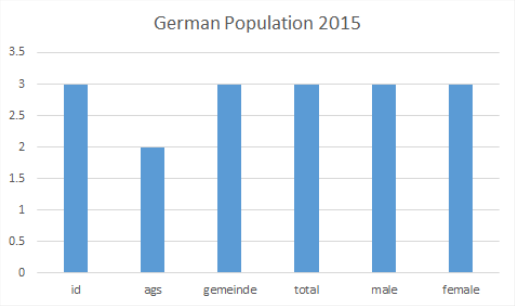


Figure A.3.: Score distribution *3*

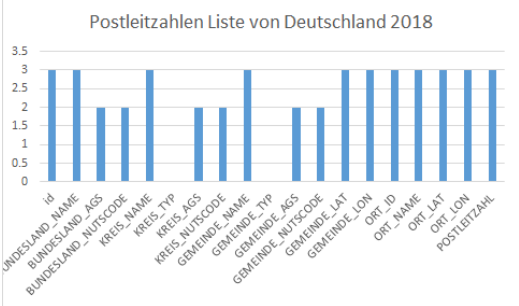


Figure A.4.: Score distribution *4*

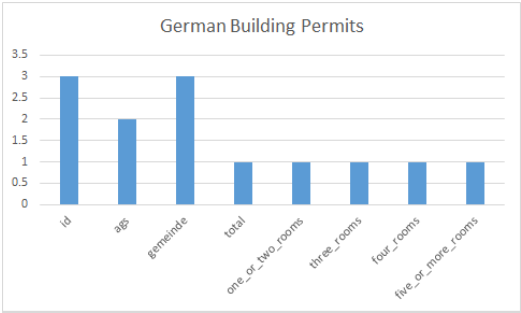


Figure A.5.: Score distribution *5*

Figure A.6.: Score distribution *6*

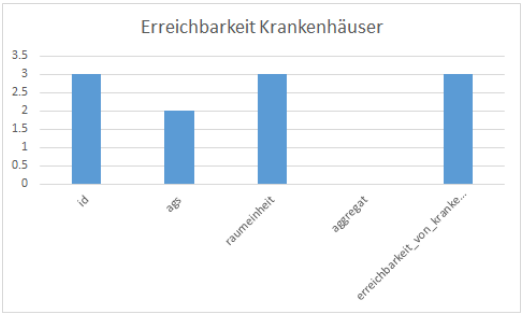


Figure A.7.: Score distribution *7*

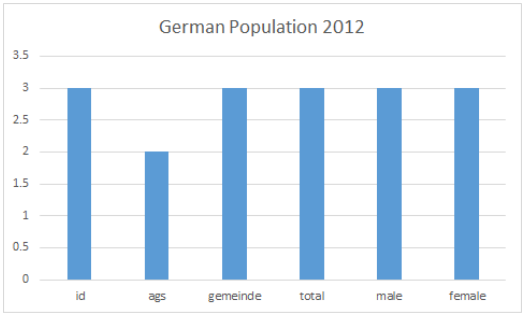


Figure A.8.: Score distribution *8*

Figure A.9.: Score distribution *9*

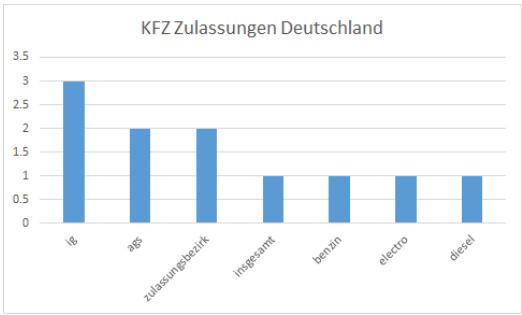


Figure A.10.: Score distribution *10*

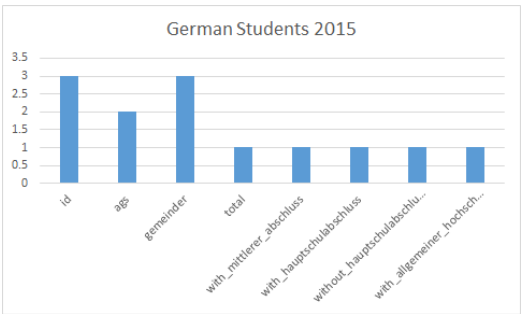


Figure A.11.: Score distribution *11*

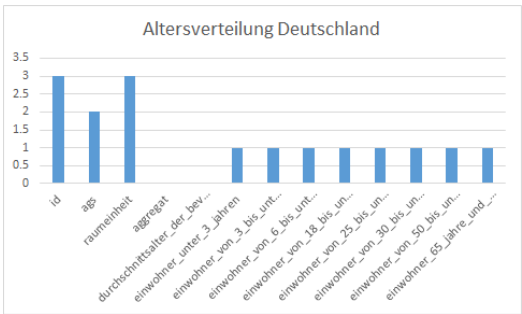


Figure A.12.: Score distribution *12*

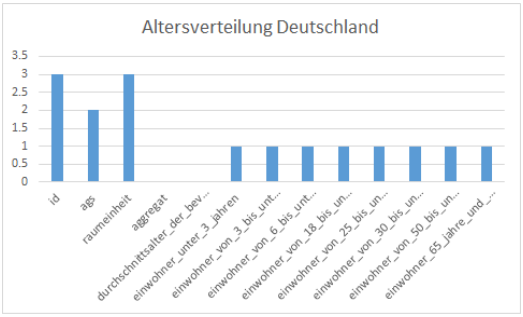


Figure A.13.: Score distribution *13*

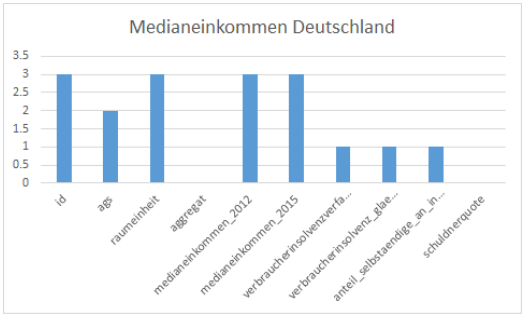


Figure A.14.: Score distribution *14*

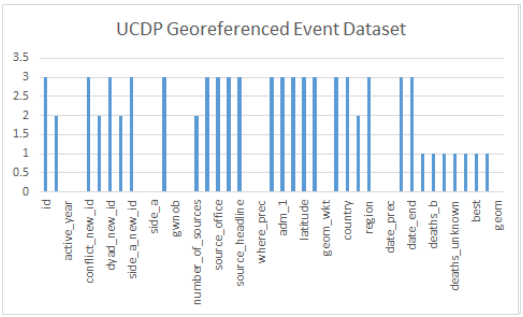


Figure A.15.: Score distribution *15*

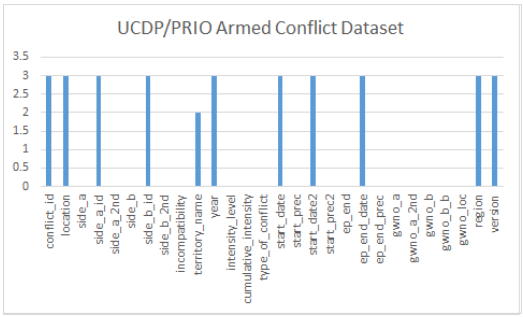


Figure A.16.: Score distribution *16*

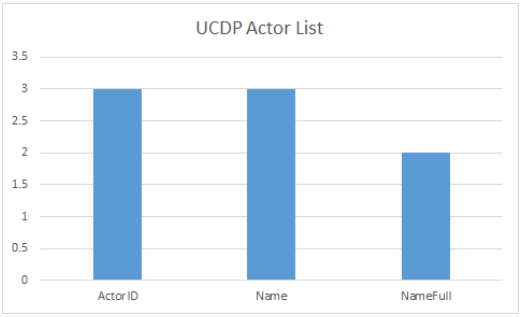


Figure A.17.: Score distribution *17*

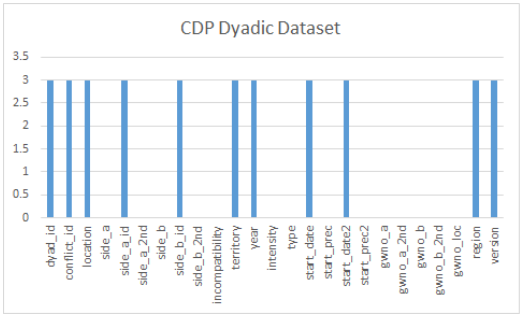


Figure A.18.: Score distribution *18*

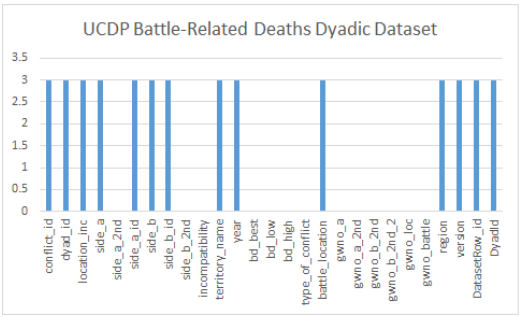


Figure A.19.: Score distribution *19*

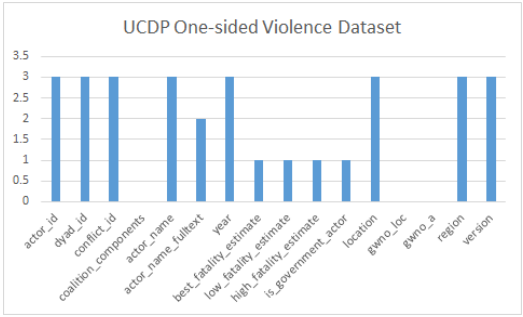


Figure A.20.: Score distribution *20*

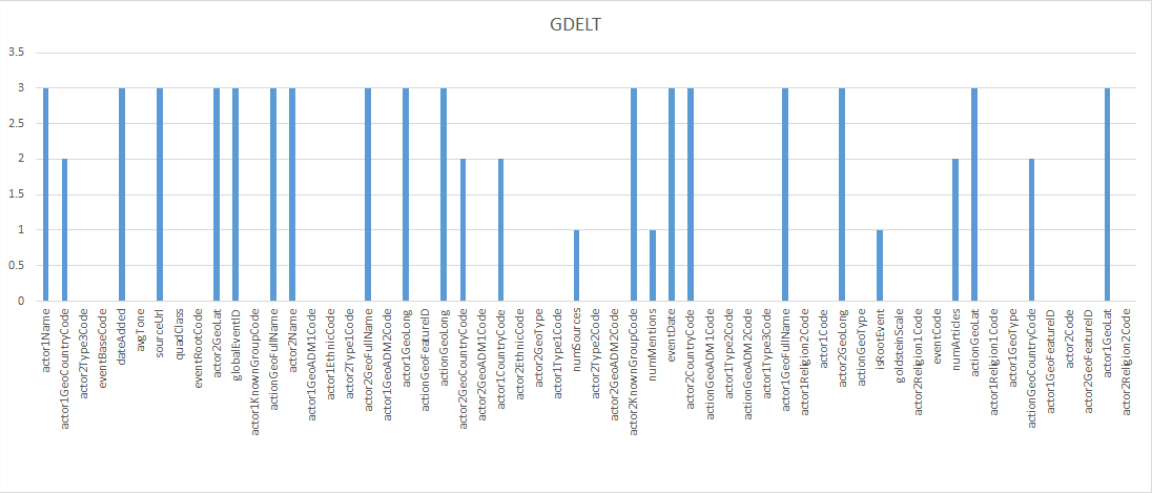


Figure A.21.: Score distribution *21*

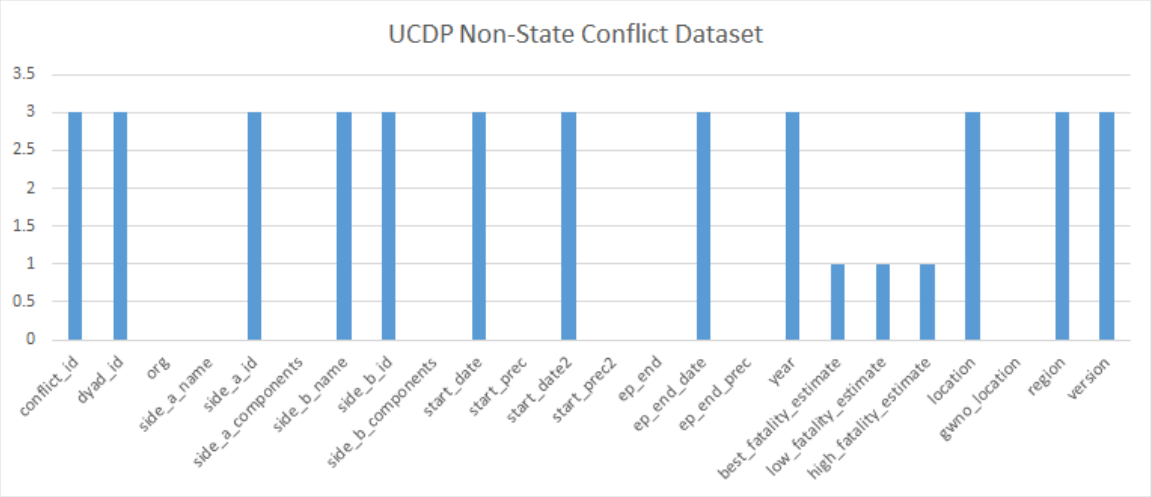


Figure A.22.: Score distribution *22*

B. Midatas Metadata Module

B.1. List of Dependencies and Versions

- **Group ID:** org.springframework
- **Artifact ID:** spring-core
- **Version:** 5.1.4.RELEASE

- **Group ID:** org.springframework
- **Artifact ID:** spring-web
- **Version:** 5.1.4.RELEASE

- **Group ID:** org.springframework
- **Artifact ID:** spring-webmvc
- **Version:** 5.1.4.RELEASE

- **Group ID:** javax.servlet
- **Artifact ID:** javax.servlet-api
- **Version:** 4.0.1

- **Group ID:** com.fasterxml.jackson.core
- **Artifact ID:** jackson-databind
- **Version:** 2.9.8

- **Group ID:** com.arangodb
- **Artifact ID:** arangodb-java-driver
- **Version:** 5.0.4

- **Group ID:** me.xdrop
- **Artifact ID:** fuzzywuzzy
- **Version:** 1.2.0

- **Group ID:** org.slf4j
- **Artifact ID:** slf4j-api
- **Version:** 1.7.25

- **Group ID:** org.slf4j
- **Artifact ID:** slf4j-log4j12
- **Version:** 1.7.25

- **Group ID:** io.springfox
- **Artifact ID:** springfox-swagger2
- **Version:** 2.9.2

- **Group ID:** io.springfox
- **Artifact ID:** springfox-swagger-ui
- **Version:** 2.9.2