Computer Science 5 - Information Systems and Databases

RWTH AACHEN UNIVERSITY

The present work was submitted to the
Institute of Information Systems and Databases

RWTH Aachen University

Faculty 1 - Mathematics, Computer Science and Natural Sciences
Course of Studies - Computer Science

Master's thesis

An Efficient Semantic Search Engine for Research Data in an RDF-based Knowledge Graph

communicated by Prof. Dr. Stefan Decker

Author:
Sarah Bensberg, 378908

Examiners:
Prof. Dr. Stefan Decker
Prof. Dr. Matthias Müller
Dr. Marius Politze

Aachen, October 16, 2020

Statutory Declaration in Lieu of an Oath

Bensberg, Sarah	378908
Last Name, First Name	Matriculation No. (optional)

I hereby declare in lieu of an oath that I have completed the present Master thesis entitled

An Efficient Semantic Search Engine for Research Data in an RDF-based Knowledge Graph

independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Aachen , October 16, 2020	
City, Date	Signature

Official Notification:

Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.

Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

I have read and understood the above official notification:

Aachen , October 16, 2020	
City, Date	Signature

Acknowledgements

At this point, I would like to thank everyone who supported me during the preparation of this master thesis.

First of all, I would like to thank my supervisor Dr. Marius Politze for giving me the opportunity to conduct my master thesis in the Process and Application Development Research group at the IT Center of RWTH Aachen University, where the infrastructure used in the thesis was provided. I am grateful for the challenging tasks, much support, and helpful advice.

My thanks go to Prof. Dr. Stefan Decker and Prof. Dr. Matthias Müller, who kindly agreed to supervise and review my work. I would like to thank them very much for their helpful suggestions and constructive criticism during the creation of this work.

Furthermore, I deeply thank all members of the department for their interest in my work, their patience, and their helpfulness.

Finally, I thank my family and my friends for supporting me, keeping me motivated, and always being there for me, not only during the time of my master thesis.

Abstract

An Efficient Semantic Search Engine for Research Data in an RDF-based Knowledge Graph

Sarah Bensberg

Digital transformation affects all areas of society: More data is produced and workflows rely on data analysis leading to new challenges in data management. Within the context of research data management, the National Research Data Infrastructure aims to systematize these data stocks and make them accessible. RWTH Aachen University supports this effort with the development of the research data management platform *CoScInE*. Research data is made accessible independent of the actual storage location and described with metadata that allows a structured search. The goal is to make the data findable, accessible, interoperable, and reusable, according to the *FAIR Guiding Principles*.

The W3C standards for the semantic web, such as the data model RDF, the corresponding query language SPARQL, and related technologies, provide the means to describe digital resources but require a significant amount of technical knowledge. This thesis deals with the implementation of a research data search in an RDF knowledge graph that is less dependent on the users' background in knowledge engineering.

Different approaches are considered under several evaluation criteria and it is investigated whether mapping the RDF data into a search index for use in a search engine improves the quality of the search and results. Such an implementation is opposed to the systematic generation of a SPARQL query.

In the course of the thesis, a transformation of RDF graphs into a search index using application profiles and rules was developed. This allows the use of all functionalities and search syntaxes provided by the search engine applied. The evaluation shows that in most cases an approach is either easy to use but slow or ineffective, or, on the contrary, fast and effective but difficult to use. With the presented transformation, a solution was found that combines these two contradictory properties.

Zusammenfassung

Eine effiziente semantische Suchmaschine für Forschungsdaten in einem RDF-basierten Wissensgraphen

Sarah Bensberg

Der digitale Wandel wirkt sich auf alle Bereiche der Gesellschaft aus: Es werden stetig mehr Daten produziert und Arbeitsprozesse sind auf Datenanalysen angewiesen. Dies führt zu neuen Herausforderungen im Datenmanagement. Im Rahmen des Forschungsdatenmanagements zielt die Nationale Forschungsdateninfrastruktur darauf ab, diese Datenbestände zu systematisieren und zugänglich zu machen. Die RWTH Aachen University unterstützt dieses Vorhaben mit der Entwicklung der Forschungsdatenmanagement Plattform *CoScInE*. Forschungsdaten werden unabhängig von ihrem eigentlichen Speicherort zugänglich gemacht und mit Metadaten beschrieben, die eine strukturierte Suche ermöglichen. Ziel ist es die Daten nach den *FAIR* Leitprinzipien auffindbar, zugänglich, interoperabel und wiederverwendbar zu machen.

Die W3C-Standards für das semantische Web, wie das Datenmodell RDF, die zugehörige Abfragesprache SPARQL und verwandte Technologien, bieten die Mittel zur Beschreibung digitaler Ressourcen, erfordern jedoch ein erhebliches Maß an technischem Wissen. Diese Arbeit befasst sich mit der Implementierung einer Forschungsdatensuche in einem RDF-Wissensgraphen, die unabhängig vom fachlichen Hintergrund des Nutzers ist.

Dazu werden verschiedene Ansätze unter mehreren Evaluationskriterien betrachtet und der Hypothese nachgegangen, ob die Abbildung der RDF Daten in einen Suchindex zur Verwendung in einer Suchmaschine die Qualität der Suche und Ergebnisse verbessern. Eine solche Implementierung steht der systematischen Generierung einer SPARQL Abfrage gegenüber.

Im Rahmen der Arbeit wurde eine Transformation von RDF Graphen in einen Suchindex mithilfe von Applikationsprofilen und Regeln entwickelt. Hierdurch können alle durch die verwendete Suchmaschine bereitgestellten Funktionalitäten und Suchsyntaxen verwendet werden. Die Evaluation zeigt, dass ein Ansatz in den meisten Fällen entweder einfach zu bedienen, jedoch langsam oder ineffektiv ist oder im Gegenteil schnell und effektiv, dafür aber schwierig zu benutzen ist. Mit der vorgestellten Transformation wurde ein Lösung gefunden, welche diese beiden konträren Eigenschaften vereint.

Contents

List of Abbreviations

AI	Artificial Intelligence
CoScInE	Collaborative Scientific Integration Environment
CSMD	Core Scientific Metadata Model
CWA	Closed World Assumption
DASH	DASH Data Shapes Vocabulary
DCAT	Data Catalog Vocabulary
DCMI	Dublin Core Metadata Initiative
DSL	Domain Specific Language
ES	Elasticsearch
FAIR	Findable, Accessible, Interoperable and Reusable
FOAF	Friend of a Friend Vocabulary
GOMS	Goals, Operators, Methods, and Selection rules
GRF	German Research Foundation
GRP	Good Research Practice
ID	Identifier
IFP	Inverse Functional Property
IR	Information Retrieval
IRI	Internationalized Resource Identifier
ISO	International Organization for Standardization
JISC	Joint Information Systems Committee in the UK
KLM	Keystroke-level Model
NFDI	National Research Data Infrastructure
NLP	Natural Language Processing
NL	Natural Language
OWL	Web Ontology Language
ORG	The Organization Ontology
OWA	Open World Assumption

PID	Persistent Identifier
RADAR	Research Data Repository
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RDLC	Research Data Life Cycle
RDM	Research Data Management
RIF	Rule Interchange Format
RWTH	RWTH Aachen University
SHACL	Shapes Constraint Language
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol And RDF Query Language
UNA	Unique World Assumption
URI	Uniform Resource Identifier
UI	User Interface
UX	User Experience
W3C	World Wide Web Consortium
WQS	Wikidata Query Service

1 Introduction

Within the context of the Research Data Management (RDM) at the RWTH Aachen University (RWTH), it is intended to create services that allow research data to be *Findable, Accessible, Interoperable and Reusable (FAIR)* [1]. Since research data is often fundamental for scientific discoveries, it should be stored on a long-term basis. Additionally, it should be reusable for future research projects. To motivate this topic, the background of this project and its relation to the National Research Data Infrastructure (NFDI) will be briefly discussed.

1.1 NFDI

The Council for Scientific Information Infrastructures established by the Joint Science Conference has recommended the establishment of a NFDI for the coordinated further development of the information infrastructure construction [2]. The background to this is the establishment of a coordinated and sustainable data infrastructure for the German science system, which strengthens the competitive position of research in Germany through the systematization of data stocks, good accessibility of research data, and continuous further development of services [3].

> "The aim of the National Research Data Infrastructure (NFDI) is to systematically manage scientific and research data, provide long-term data storage, backup and accessibility, and integrate the data both nationally and internationally. The NFDI will bring multiple stakeholders together in a coordinated network of consortia tasked with providing science-driven data services to research communities."
>
> — German Research Foundation [4]

1.2 Motivation

Kindling and Schirmbacher [5] describe RDM as the entire process that supports the allocation, generation, processing, enrichment, archiving, and publication of digital research data. The goal of RDM is the long-term utilization of research data according to the FAIR Guiding Principles.

Research data refers to data that serves as the basis for the results of a research project or to the results themselves. The German Institute for International Educational Research [6] describes research data in empirical educational research as the basis of scientific knowledge.

Especially for public and future access to research data, it is important that research data is sufficiently documented and described. Metadata, a data record describing the data, serves to make the data interpretable [7]. In the field of RDM, metadata can be used to clarify questions that occur regarding research data, e.g., the date of creation. As additional information is stored by the metadata, it is possible to search research data or to filter it according to certain properties. A standardized presentation of the data

makes the information machine-readable and thus a search possible. Through the usage of metadata, information can be linked across the entire World Wide Web [8], which is why it is particularly useful for accessibility and subsequent use.

1.3 RDM at RWTH Aachen University

To achieve the goal set by NFDI, the central university institutions, such as libraries or computing centers, have to be involved [9]. For this reason RWTH, as a renowned research institution, decided to invest in institutional RDM structures [10]. Further reasons, which led to the decision of RWTH to support researchers to do RDM and to deal with the topic comprehensively are described in the following paragraph[11]:

Overall, problems and difficulties during research work should be avoided. Another motivational aspect is the security of research data. RDM should ensure that data is not lost and that it is protected against misuse and theft. The visibility of research data also plays a major role – research data should be documented in a comprehensible way and assigned to the researcher so that queries regarding the data can be made. Another point is the reusability of research data. The goal is to be able to reuse research data for new research projects and to gain more efficient access to already existing research data. Besides, research data is often already stored in different systems and the combination and unification of this data records pose new challenges for researchers [12]. More and more public funding agencies such as the German Research Foundation (GRF) expect research data to be made available and data management plans to be developed. Furthermore, the access to research data is usually limited to one project or organization [13]. However, the intention here is to enable cross-disciplinary access.

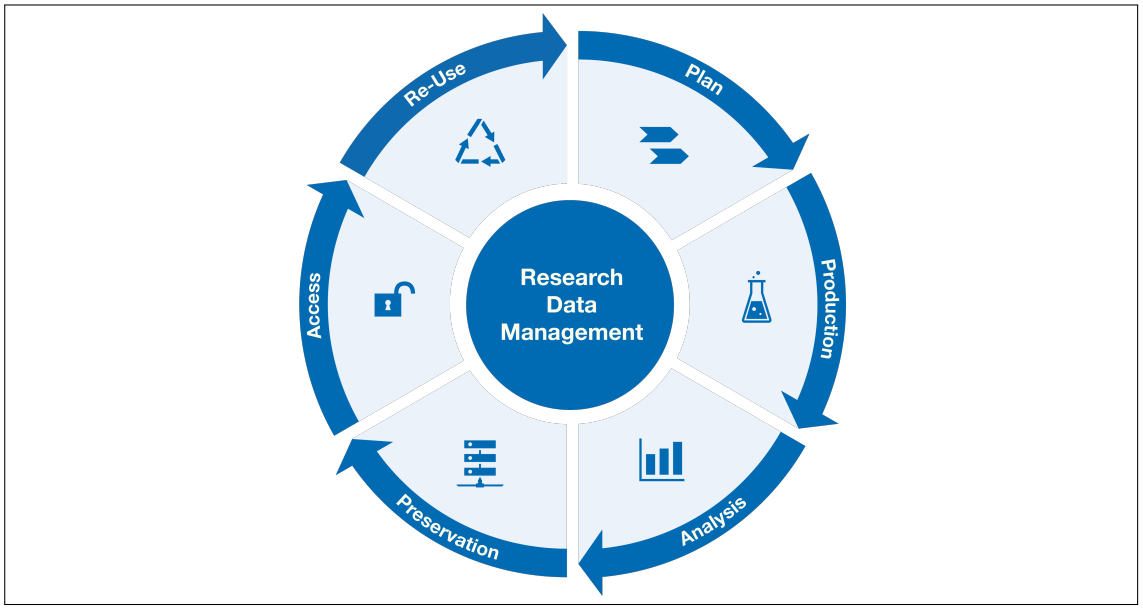


Figure 1.1: Research Data Life Cycle [9]

An important aspect that is relevant for the implementation of appropriate solutions is the diversity and heterogeneity of the various scientific disciplines as well as the decentralized nature of data. This results from the different research data infrastructures used by researchers and consisting different services. However, it should be possible to maintain the discipline-specific equipment and IT systems individually for each institute despite uniform support concerning RDM. The difficulty of reproducibility is thus increased even

further. Additionally, initial approaches already exist at some research groups, which is why the different stages of development represent a further challenge [9].

It should be noted that for the researchers themselves, the added value of RDM is indirect, so active integration into research processes is crucial [14]. Figure 1.1 shows the Research Data Life Cycle (RDLC), which is ideally supported throughout by suitable IT systems. Data records pass through this cycle again and again, which is why support during the transitions of the individual phases is particularly important. Besides, the role of a dedicated "Data Curator", who is familiar with certain techniques/standards and technologies, is progressively disappearing within the context of RDM; instead, every single researcher is under obligation. This trend indicates that many "IT skills" are becoming more and more "general education" in the context of digitization. To support the researcher in all phases of the RDLC, the goal of RWTH is to create a basic infrastructure of connected services [9]. To enable research groups to continue using individual components, technology-independent and process-oriented interfaces will be created. These ensure that different systems are bundled to generate added value for the researcher. Several components are used for this purpose: (i) the Persistent Identifier (PID) for the permanent and unique referencing of research data, (ii) external metadata stored with the PID, which enables the use of discipline-specific systems, (iii) a central archive system in which researchers can store their research data, and (iv) the support of the University Library to publish the results in a suitable repository. As the importance of descriptive and explanatory metadata is growing, an application has been developed that allows the creation of metadata for institution-specific application profiles. During the creation of research data, this metadata information is still very easily available, whereas during the further course of the RDLC this implicit knowledge is not passed on and is lost [15]. The application profiles are based on existing metadata standards. Moreover, an automated connection should be possible to be able to integrate a machine generation of metadata into existing work processes.

1.4 CoScInE

To support the RDLC the IT Center of RWTH is currently developing the research data management integration platform Collaborative Scientific Integration Environment (CoScInE) [12]. It is a software platform for the allocation and management of IT resources in research projects. It serves to map the basic processes of research data management: Versioning, archiving with the help of flexible metadata schemas, as well as reuse through a cross-source search. The main goal is to provide access to research data regardless of its storage location and to make existing data FAIR. CoScInE offers numerous functionalities, which can be divided into features for researchers (see Figure 1.2) and for organizations (see Figure 1.3) [16]. Researchers are supported by a project-based structure, a member administration, and access for cooperation partners (such as the use of ORCID [17]) in the project organization. Moreover, data management is facilitated by the integration of established services (ObjectStorage, GitLab, Sciebo [18], and the RWTH archive), the use of flexible metadata schemas and application profiles, a cross-source search, versioning, and archiving. Additionally, CoScInE provides assistance in order to meet the requirements of the Good Research Practice (GRP) [19] such as the guidelines on the use of standards and methods, documentation, allowing to make research results publicly accessible and archiving. Organizations are supported by individual adaptability, e.g., of organizational structures and roles or custom resource types, automatic provision of resources, and management ratios. More features are a transparent further development and participation in the community through code and

use cases through open source strategy. Furthermore, CoScInE supports in fulfilling the requirements of the GRP, as researchers are given infrastructural framework conditions to support work according to the GRP. Altogether it can be said that CoScInE is used for making research data across different storage services *FAIR* [16]: "Indexed and searchable metadata (F), Persistent identifiers for referencing and access (F), Access regulation and making data available (A), Standardized access to (meta-)data (A), Standardized application profiles and vocabularies (I), Project metadata and (flexible) metadata schemas (R), Provenance tracking (R)".

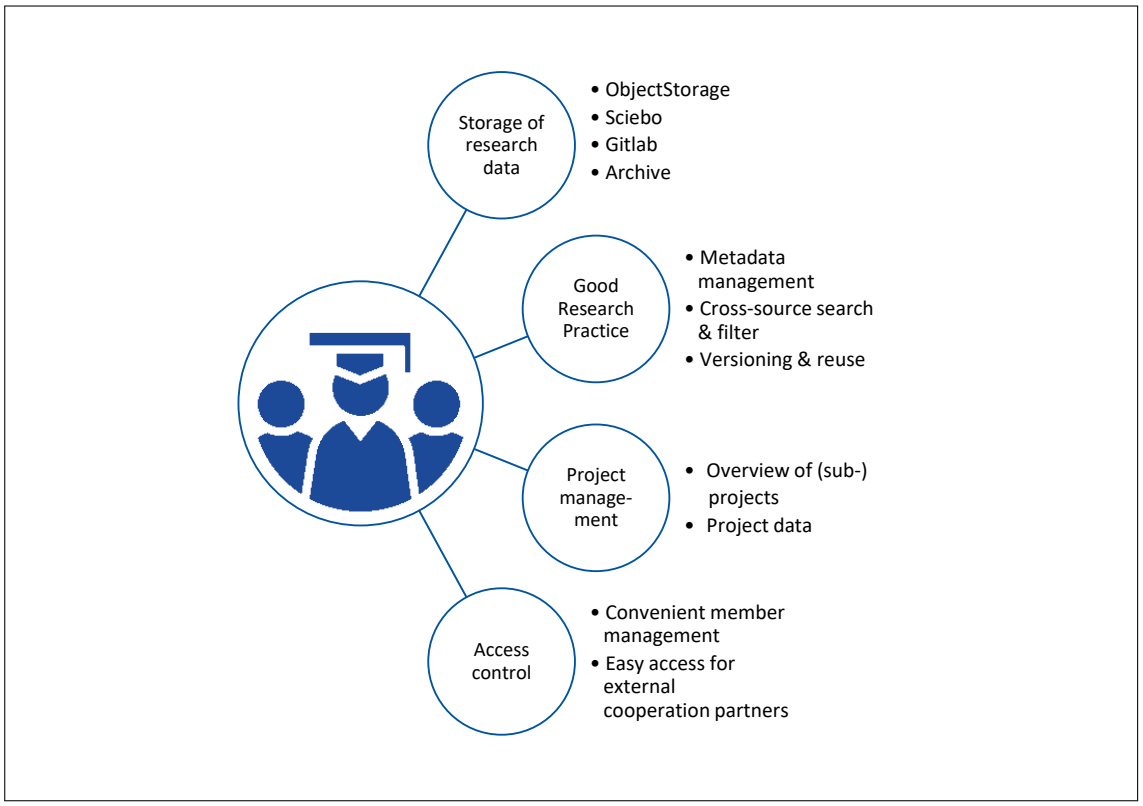


Figure 1.2: CoScInE – Features for researchers [16]

The platform is currently in a pilot phase for researchers at RWTH. The resulting user feedback will be continuously integrated into the existing system to improve and expand it. At the moment, the focus is on supporting more sophisticated RDM workflows such as data publication, archiving, and automatic extraction of metadata from the content of managed resources.

CoScInE itself does not store research data, but supports the structured storage of metadata on research data as described above. These are documents in the Resource Description Framework (RDF) model (see section 2.2.1) that are validated with Shapes Constraint Language (SHACL) application profiles (see section 2.2.3). Currently, there is no useful search implemented in these which limits the reusability of the data.

1.5 Research Goals and Questions

The development of a search in CoScInE is useful to find the stored metadata. It enables retrieval and thus the reproduction or further development of research projects. The background of the present master thesis is the elaboration and evaluation of different approaches to build such a search considering the data structure and requirements of

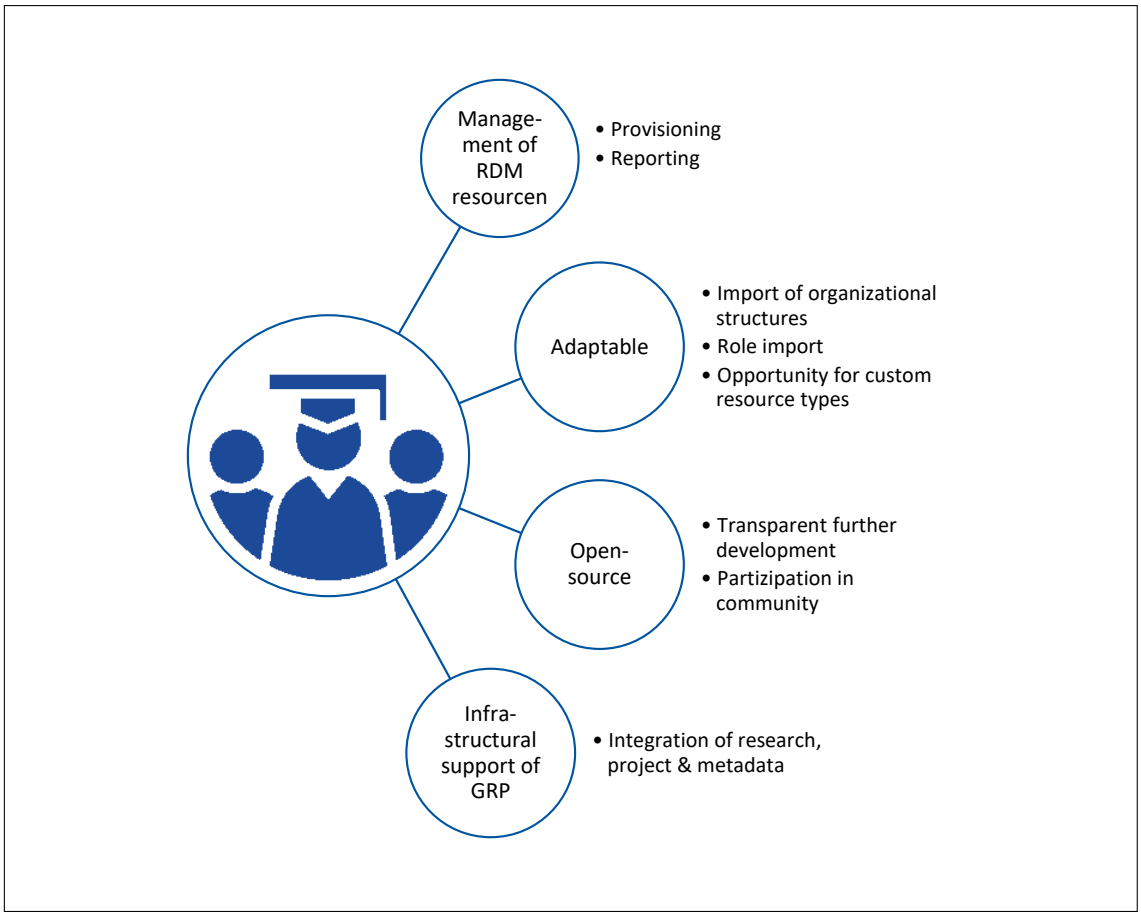


Figure 1.3: CoScInE – Features for organizations [16]

CoScInE (see section 3). Initially, the data is available as a RDF-based knowledge graph, which presents some general challenges described by Arnaout and Elbassuoni [20]:

- *Data incompleteness*, since the RDF triples contain a lot of information, but most knowledge is represented as free text.
- *Inflexible querying*, because triple-pattern queries are highly expressive, but at the same time restrictive, because they have to follow a certain structure and queries can only be made according to the underlying data structure and terms. A further restriction is that no query can be made regarding missing resources.
- *Missing result ranking*, since RDF graphs, or queries in such a graph may yield many results that can overwhelm a user. Therefore, a specific ranking is necessary, which can be based on different criteria. However, such a ranking is not provided by SPARQL Protocol And RDF Query Language (SPARQL) (see section 2.2.4), so it has to be implemented explicitly.
- *Result diversity*, because it is important that the topmost results give a broad overview of the results of a query and do not all contain similar aspects.

Since the data is stored in a triple store, the easiest way to access it is the SPARQL (see section 2.2.4) Endpoint that allows to query and filter data. However, a user must have some knowledge to do this: (i) RDF and SPARQL, (ii) the structure of the stored data, (iii) used vocabularies and terms as well as their corresponding Uniform Resource

Identifiers (URIs) or Internationalized Resource Identifiers (IRIs)[1]. Even if a user would have all the information, the use of SPARQL is very expressive and therefore challenging and thus can only be used by experienced Data Curators (see section 1.3) in a satisfactory manner. However, the goal is that every researcher should be able to use RDM tools. In addition, access to all data is technically possible, although not every user has all permissions (see section 3.4.1). A further aspect is the consideration of different search types or search options. In 2018 there were about 45000 students, 560 professors, 8500 employees at 260 institutes in 9 different faculties at RWTH. These numbers show that CoScInE needs to be prepared for handling large amounts of data. From this and from the requirements and general conditions of CoScInE described in section 3.4.1, the following research questions arise:

- How are access rights represented?
- How are the general challenges of querying RDF knowledge graphs addressed?
- Which search types (value ranges, and/or/not, date, full-text, ...) are meaningful and how can they be implemented?
- How to deal with large amounts of data, especially with regard to the runtime of search queries? How to get the highest possible performance?
- How must the data be stored or prepared for the search?
- What approaches exist to search in RDF data and which functions support them?
- Does a conversion into a search index have advantages for CoScInE? Which ones?
- What additional information can a SHACL application profile provide to enable a complex search?

Based on the state of research on RDF searches or general Information Retrieval (IR) and taking previous approaches into account, the work is oriented towards the following hypothesis, which is to be tested:

"Mapping RDF-based metadata records into a search index for use in a search engine improves the quality of the search and results for the user compared to using generated SPARQL queries."

In the context of the thesis, the word quality refers to the requirements presented in section 3.4, which are described in more detail in chapter 5 in the context of the evaluation. The quality of the search refers to the way from the search intention to the receipt of the results, i.e., very much to the usability for the user. The quality of the results focuses on how good the results returned by the search query are. Therefore, the goal of the master thesis is the evaluation of different approaches to create an efficient semantic search engine for metadata on research data in the RDF-based knowledge graph considering the requirements and constraints of CoScInE.

1.6 Methodology and Structure

The thesis is structured in the following way: Chapter 2 "Fundamentals" summarizes theoretical basics relevant to the developed solution. Furthermore, it gives an overview of the current state of research regarding the possibilities of searching in an RDF-based knowledge graph. Chapter 3 "Technical Details of CoScInE" deals with the technical details of the implementation as well as general conditions and requirements of CoScInE.

[1] An IRI is a generalization of an URI. Since both are used very interchangeably in general linguistic usage, only the term URI is used in the following, even if a IRI would be allowed.

Subsequently, in chapter 4 “Different Approaches” the approaches for the implementation of the search, which are considered in the context of the thesis, are presented. These approaches are examined under consideration of the CoScInE context. The data used for evaluation is presented at the beginning of chapter 5 “Evaluation and Results”. Thereupon, an evaluation set of search intentions is defined, which is applied to the different approaches and the results are discussed. In the last section of this chapter, the different evaluation criteria are considered and evaluated. Finally, chapter 6 “Conclusion” contains a discussion, a summary of the results, and a short outlook on the further procedure is given.

2 Fundamentals

This chapter lays the foundation on which the following work is built. Metadata, important technologies from the Semantic Web, basics of IR, and the term Semantic Search are described. Of special interest is the section 2.5 "Related Work", which contains the current state of research on the topic explained.

2.1 Metadata and Related Concepts

Various definitions of metadata exist in the literature. Steffen Staab [21] describes metadata as data that describes selected aspects of other data. The interactive course of Mantra [22], which is an online course for dealing with digital data as part of research projects, distinguishs three different types of metadata:

- *Descriptive* - general and domain-specific fields such as author, title, and devices used. They are often further subdivided into generic (easy to use and widely distributed) and domain-specific (richer in vocabulary and structure, but usually highly specialized and understandable only to researchers in the field) metadata [23].
- *Administrative* - metadata for preservation, rights management, and technical specifications.
- *Structural* - to represent the relationship between different components.

Usually, metadata is structured data. Terms are linked to a resource via generic categories (e.g., "title") and assigned to it. A resource describes a specific object using a link.

Therefore, metadata is information that says something about the creation, content, or context of a resource. They can be used to link data over the World Wide Web (see section 2.2) [8]. In general, metadata is used to understand data records in a broader context, so that people or users outside of their own project group or institution can interpret the data as well [22]. The following sentence in Example 2.1 represents a metadata in the form subject-predicate-object:

Example 2.1: Metadata

Research data X are from the author Y.

The Joint Information Systems Committee in the UK (JISC) (Organization for the Promotion of Digital Technologies in Research and Education) [24] describes a *metadata schema* as a collection of metadata fields that are combined into a set. A metadata field is a resource that is used as a property of a particular subject. For example, the metadata fields title, author, and subject can be combined in a metadata schema. There are many official *metadata standards*, which have been developed through a validation process of the metadata schema at a standard organization (such as the Dublin Core Metadata Initiative (DCMI)). Examples of such metadata schemas are Dublin Core [25], Data Catalog Vocabulary (DCAT) [26], DataCite [27], and RADAR [28].

Controlled Vocabulary can be a kind of linear keyword list, a hierarchically structured catalog or a taxonomy, thus limiting an input. According to the JISC [29], controlled vocabularies are used in metadata fields with the main goal of more efficient retrieval of resources through searching. They improve data consistency and reduce ambiguity in the language.

An *application profile* [30] is a specification for describing metadata in an application that uses controlled vocabularies and imposes further usage restrictions. The use of different metadata schemas is possible.

2.2 Semantic Web and Linked Data

The *World Wide Web Consortium (W3C)* (a committee for the standardization of techniques in the World Wide Web) [31] strives for a web of linked data, which leads to the term *Semantic Web*: An extension of the World Wide Web that serves to simplify data exchange and machine processing. Data is supplemented with additional information (metadata) to support machines in evaluation and processing. Technologies that are used for this purpose include RDF (see section 2.2.1), SPARQL (see section 2.2.4) and inference (see section 2.2.5). An important feature of the Semantic Web is its decentralization, i.e., there is no single knowledge base but everybody can make new explanations available in a public webspace [32].

Linked Data [33] describes the integration of data from different sources in the Semantic Web. Data and metadata are published on the Internet that are machine-readable and whose meaning has been defined beforehand. *Linked Data Web* means that the data from the different independent data sources are linked on the Web via their URIs and relationships can be created between the data to create a Web of data. The data are documents in the RDF model (see section 2.2.1) and are thus available as a graph.

Tim Berners-Lee [34] defines four design principles known as Principles of Linked Data for creating and linking Linked Data:

Definition 2.1: Linked Data Principles

1. *"Use URIs as names for things.*
2. *Use HTTP URIs so that people can look up those names.*
3. *When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).*
4. *Include links to other URIs. so that they can discover more things."*

2.2.1 RDF Data Model

The *Resource Description Framework (RDF)* is one of the W3C conceived data model for the representation of information in the Web [35]. It is a data model that is used in the area of the Semantic Web. Data is structured in the form of triples (subject-predicate-object) and combined into a graph. For this reason, RDF offers a reasonable way to map metadata. A resource (subject) is assigned a term or a value (object) via a category (predicate). Formally, an RDF triple is defined as follows [36]:

Definition 2.2: RDF triple

An RDF triple is represented as a tuple $\langle S, P, O \rangle \in (\boldsymbol{U} \cup \boldsymbol{B} \cup \boldsymbol{L}) \times \boldsymbol{U} \times (\boldsymbol{U} \cup \boldsymbol{B} \cup \boldsymbol{L})$ *where* S *is called the subject,* P *the predicate, and* O *the object and* $\boldsymbol{U}$*,* $\boldsymbol{B}$*, and* $\boldsymbol{L}$*, are used to represent* ***URIs****,* ***blank nodes*** *(a resource for which no URI exists), and* ***literals*** *respectively.*

Example 2.2 shows how to use the Dublin Core [37] vocabulary (see section 2.2.2) to map the statement `X` has creator `Y` as RDF triple.

Example 2.2: RDF triple

```
<X> dcterms:creator <Y>
```

A finite number of RDF triples can be combined into one RDF graph. Subjects and objects correspond to nodes and predicates to edges of the graph. Formally an RDF graph is defined as follows [36]:

Definition 2.3: RDF graph

Following the definition of an RDF triple, an RDF graph G *consists of a set of triples. The universe of* G*,* $universe(G)$*, is the set of elements in* $(\boldsymbol{U} \cup \boldsymbol{B} \cup \boldsymbol{L})$ *that occur in the triples of* G *and the vocabulary of* G*,* $voc(G)$*, is* $universe(G) \cap (\boldsymbol{U} \cup \boldsymbol{L})$*. We say that* G *is ground if and only if the* $universe(G) = voc(G)$*, i.e.,* G *does not contain blank nodes.*

Example 2.3 demonstrates how RDF can be used to represent additional information about `X`. The following triples state that X has author Y and subject Z.

Example 2.3: RDF graph

```
<X> dcterms:creator <Y>
<X> dcterms:subject <Z>
```

If the graph is additionally assigned an URI name, it is a so-called *Named Graph*, which is also called *context*. A graph consists of at least one triple. The individual elements can be blank nodes, data type literals or URIs. An URI element can be uniquely identified and is also called a resource. In principle, RDF can be used to describe properties of a resource and to represent relations between them. A collection of RDF graphs is called an RDF dataset and is formally defined as follows [36]:

Definition 2.4: RDF dataset

Given an RDF graph, an RDF dataset DS consists of a set of graphs, with exactly one ***default graph*** *possibly empty; and one or more* ***named graphs****, consisting of a pair* $\langle name, RDFGraph \rangle$*, where* $name \in (\boldsymbol{I} \cup \boldsymbol{B})$*.*

A *knowledge graph* is a database, which describes different entities and their relation to each other. It is available either as *Linked Data* (see section 2.2) or as RDF dataset [35].

2.2.2 RDF Vocabularies

A RDF vocabulary enriches data with additional meanings so that an ever-increasing number of people and machines can benefit from this data [31]. Vocabularies [38] are used

to classify terms that are applied in an application. They also provide possible restrictions on the use of these terms and can map relationships between them. Besides the word "vocabulary" there is the term "ontology", which is mostly used for a more complex and formal collection of terms. W3C [38] describes vocabularies as "basic building blocks for inference techniques on the Semantic Web". Due to different and diverse applications of vocabularies, there are several techniques (like RDF, Resource Description Framework Schema (RDFS), Simple Knowledge Organization System (SKOS), Web Ontology Language (OWL), and the Rule Interchange Format (RIF)) to describe different forms of vocabulary in a standard format. The most common use case of vocabularies is the definition of so-called controlled vocabularies (see section 2.1). In the following, the vocabularies RDFS and OWL for creating controlled vocabularies are introduced.

RDFS

The so-called *Resource Description Framework Schema* [39] is a semantic extension of the RDF vocabulary standardized by W3C. It describes groups of data and their relationship to each other. It is used to describe other resources and their characteristics to correctly interpret the information formulated in the RDF model. For example, the definition and value range for resources can be specified by the RDF schema. Sometimes the expression "RDF Vocabulary Description Language" is also used [40]. In summary, an RDF schema allows the grouping and description of the use of resources. Grouping includes creating hierarchies to define structures.

OWL

The W3C *Web Ontology Language* [41] is a logic-based language from the Semantic Web. Computer programs can use knowledge expressed in OWL, e.g., to check the consistency of this knowledge. The current version of OWL is "OWL 2" [42]. OWL allows the use of additional vocabulary to describe properties and classes as well as further functionalities like the use of cardinalities (e.g., "exactly one"), equivalence, characteristics of properties (e.g., symmetry) or relations between classes (e.g., disjunctness).

OWL can be divided into three increasingly-expressive sublanguages [43]. OWL Lite has the least formal complexity of the three since it mainly allows classification hierarchies for thesauri or other taxonomies as well as simple restrictions. OWL DL is based on the descriptive logic where the name comes from. It allows the use of all OWL language constructs, but with certain limitations to ensure computational completeness and decidability. OWL Full offers full expressive power at the expense of computational guarantees. For this reason, complete reasoning (see section 2.2.5) is not applicable.

2.2.3 Validate and Constrict RDF Data using SHACL and DASH

SHACL [44] is another W3C recommendation and serves the validation of so-called *Data Graphs* against *Shapes Graphs*. A shape graph is an RDF graph, which describes conditions and restrictions that the data graph needs to fulfill. For this reason, it is more suitable than RDFS to model the constraints in terms of a metadata schema. In addition to validation, the use of SHACL serves, among other things, to build user interfaces, generate code or to ensure data integration. SHACL can be divided into the two languages *SHACL Core language* and *SHACL-SPARQL*.

RDF Application Profiles

An RDF application profile is a SHACL shape and thus a possibility to implement an application profile (see section 2.1) with technologies of the Semantic Web. Example 2.4 shows a SHACL shape.

Example 2.4: SHACL shape (RDF application profile)

In this example, `ex:ExampleShape` is defined as SHACL shape via the definition `a sh:NodeShape`. Using `sh:property` a property is defined which in the example corresponds to `ex:author`. The values of `ex:author` must be instances of the class `ex:author`.

```
ex:ExampleShape
  a sh:NodeShape ;
  sh:property [
    sh:path ex:author ;
    sh:class ex:Author ;
  ] .
```

DASH

DASH Data Shapes Vocabulary (DASH) [45] is a standard compliant extension of SHACL. The vocabulary includes additional constraints and vocabularies for result validation, description of test cases, and for making suggestions when conditions are violated.

2.2.4 Query and Update RDF Data using SPARQL

The *SPARQL Protocol And RDF Query Language (SPARQL)* [46] is a collection of W3C specifications that make it easy to query and manipulate the contents of RDF graphs on the Web or in an RDF graph. SPARQL queries return a subgraph as result based on a graph pattern. A graph pattern consists of triple patterns, which can contain variables in all three positions. Formally, triple and graph patterns are defined as follows [36]:

Definition 2.5: Triple pattern, graph pattern

An RDF triple pattern is represented as a tuple $\langle S, P, O \rangle \in (\boldsymbol{I} \cup \boldsymbol{B} \cup \boldsymbol{L} \cup \boldsymbol{V}) \times (\boldsymbol{I} \cup \boldsymbol{V}) \times (\boldsymbol{I} \cup \boldsymbol{B} \cup \boldsymbol{L} \cup \boldsymbol{V})$ *where* $\boldsymbol{I}$, $\boldsymbol{B}$, *and* $\boldsymbol{L}$, *are used to represent* ***URIs***, ***blank nodes***, *and* ***literals*** *respectively and* $\boldsymbol{V}$ *represents a set of* ***variables*** *disjoint from* $(\boldsymbol{I} \cup \boldsymbol{B} \cup \boldsymbol{L})$. *A graph pattern is a set of triple patterns.*

The **SPARQL Query Language** [47] defines SPARQL as the query language for RDF. It allows searching for specific graphs or parts of graphs and returns the corresponding result. This search can be linked to the triple (whether subject, predicate, or object) with certain conditions. It is important to note that this is a pure query language, i.e., manipulation as well as deletion of data is not possible. Example 2.5 shows a SPARQL query which searchs all agents of the Friend of a Friend Vocabulary (FOAF) [48].

Example 2.5: SPARQL query

The following SPARQL query contains the triple pattern `?s a foaf:Agent`. Variable `?s` is used in the subject position, so the query matches all triples of type `foaf:Agent`.

Example 2.5 (continued)

```
SELECT ?s WHERE {
  ?s a foaf:Agent
}
```

SPARQL Update [49] describes the possibility to update RDF graphs. The language is based on the SPARQL Query Language. It provides functionality for editing, creating, and deleting data in a graph database.

2.2.5 Infer over RDF Data

Inference is a W3C standard [50] that describes the reasoning and discovery of new relationships between resources through the data as well as additional information and thus the acquisition of new knowledge. Vocabularies are mostly used to represent hierarchies or other relationships, whereas rule sets focus on defining and creating new relationships. A RDFS example for illustration is shown in Example 2.6.

Example 2.6: Inference with RDFS

The definition `foaf:person rdfs:subClassOf foaf:agent` from the vocabulary FOAF [48] states that each person is an agent. If now an instance `x` is defined as a person, the relation that `x` is also an agent can be deduced from it:

(foaf:person rdfs:subClassOf foaf:agent) ∧ (<x> a foaf:Person) → (<x> a foaf:Agent)

Since inference can be NP-complete [51] depending on the underlying logic, a restriction in the context of practical use cases is useful.

2.2.6 Open and Closed World Assumption

In the context of the Semantic Web, the term *Open World* is used, which is briefly introduced and differentiated from the meaning of *Closed World* [52].

CWA

Along with the **Closed World Assumption (CWA)** a system with complete access to all information about a subject exists. If a search is made for a specific piece of information within this system, the correct answer is found and only this answer exists. In this way, unambiguous statements can be made.

OWA

In case of the **Open World Assumption (OWA)**, the information in a system is or can be incomplete. Just because an answer to a question cannot be found in one system, does not imply that it cannot be found in another system, which might contain the necessary information. Thus, it cannot be concluded it is the only result or that there is none. Therefore, the correct and complete answer is unknown.

Difference between CWA and OWA

The main difference between the two assumptions is that the CWA includes the *Unique World Assumption (UNA)*. The UNA [53] states that two things with different names are in fact different. Example 2.7 illustrates this.

Example 2.7: Difference between CWA and OWA

Assuming the scenario [52] that a person can only live in one country and the following assertions are introduced: `Sarah lives-in Berlin` and `Berlin is-in Germany`. If the statement `Berlin is-in France` would also be added, the CWA system would result in a contradiction if it is assumed that persons can only live in one country and according to UNA Germany and France are not the same countries. The decisive point is that Berlin is the same city, i.e. the same resource is used, and not two different cities (which would have different URIs) with the same name:

$(\langle Sarah \rangle \langle lives\text{-}in \rangle \langle Berlin \rangle) \wedge (\langle Berlin \rangle \langle is\text{-}in \rangle \langle Germany \rangle) \wedge (\langle Berlin \rangle \langle is\text{-}in \rangle \langle France \rangle) \wedge (Germany \neq France) \rightarrow \bot$

In contrast, in an OWA system there would be no error, but the result would be the statement that Germany and France are the same things:

$(\langle Sarah \rangle \langle lives\text{-}in \rangle \langle Berlin \rangle) \wedge (\langle Berlin \rangle \langle is\text{-}in \rangle \langle Germany \rangle) \wedge (\langle Berlin \rangle \langle is\text{-}in \rangle \langle France \rangle) \rightarrow (\langle Germany \rangle = \langle France \rangle)$

However, an inconsistency could be created here, namely if the statement `Germany is-not France` would be added. The conclusion in OWA would then look like this: A person can only live in one country. Berlin is a city in Germany. Berlin is a city in France. It follows that Germany and France are the same things. But since Germany and France are two different things and therefore cannot be the same, something must be wrong:

$(\langle Sarah \rangle \langle lives\text{-}in \rangle \langle Berlin \rangle) \wedge (\langle Berlin \rangle \langle is\text{-}in \rangle \langle Germany \rangle) \wedge (\langle Berlin \rangle \langle is\text{-}in \rangle \langle France \rangle) \wedge (\langle Germany \rangle \neq \langle France \rangle) \rightarrow \bot$

OWA and the Semantic Web

The Semantic Web is a system with incomplete information. Missing information in this context means that the information has not yet been explicitly generated and that the Semantic Web follows the OWA, precisely because its main goal is to obtain new information [52]. Due to this fact, the Semantic Web can appear complicated at times, because the information and rules cannot be applied so intuitively and different meanings can occur in different applications. This is mainly since databases are normally based on CWA.

OWA and Metadata

Metadata is intended to provide additional information. Based on inference and the OWA, these statements are not unambiguous (see Example 2.7). If clear statements should be made in an application with metadata, these must be restricted by application profiles. This allows validating whether they follow a certain schema and use only certain controlled vocabularies.

2.3 Information Retrieval

Manning et al. [54] define IR as finding unstructured data (usually text in documents) in a large collection to satisfy an information need.

2.3.1 Unstructured, Structured, and Semi-Structured Data

Unstructured data is used to describe data that do not follow clear rules, such as free texts. A computer cannot structure such data.

Structured data, on the other hand, are understandable for machines. An example of this are relational databases.

Abiteboul [55] defines *semi-structured* data as a mixture of both. The data is neither clearly typed nor completely free or unstructured. RDF graphs belong to the semi-structured data because it is machine-interpretable, but the structure is not predefined and due to the OWA can be extended or is incomplete.

2.3.2 Search Queries

Manning et al. [54] divide search queries into three super-categories, which depend on the intention of the user: (i) *informational* queries are used to obtain general information on a specific topic, which may be contained in different sources, (ii) *navigational queries* aim to find the website/home page or similar of a specific entity and (iii) *transactional queries*, which are preliminary queries to initiate a user transaction.

2.3.3 Different Approaches of Information Retrieval

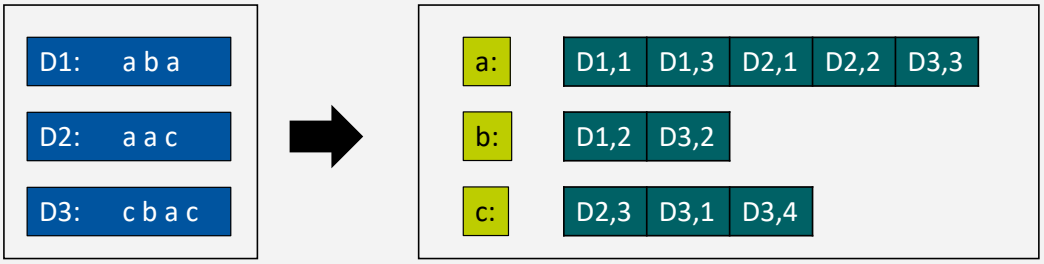
Since there are diverse kinds of data, as described in section 2.3.1, many different approaches exist in the literature.

Traditional IR

Büttcher et al. [56] describe that a basic task of search engines is to create an *inverted index* for the documents. This index enables searchability and is mainly used to compare the similarity or fit of a search query with a document to establish a ranking. The simplest form of an inverted index is the creation of a list of words for which the documents and positions of their occurrence are stored (see Example 2.8). There are further optimizations and techniques, but all of them lack the possibility to use structures of the data and deal with them [57].

Example 2.8: Inverted index

The following example shows that for each of the three words (*a, b, c*) of the three documents (*D1, D2, D3*), a list with occurrence and position in each document is created:



Information Retrieval on Structured Documents

As a result, more and more models have been developed which take into consideration not only the content but also the structure of documents. Baeza-Yates and Navarro [58] compared the hybrid model, PAT[1] expressions, overlapped lists, and proximal nodes. They have pointed out that no model is optimal in every case, but that the choice of a model depends on their application since they all have different advantages and disadvantages. However, their advantage over the former approaches is that more meaningful searches are possible since searches can be limited to individual areas and structures.

Information Retrieval of RDF Data

As explained in section 2.2.4, SPARQL provides a way to query and search RDF data. However, this requires SPARQL knowledge of a user as well as knowledge of the structure and vocabularies used, which is usually not given. Furthermore, this approach does not make use of the structural or content-related similarity between a data graph and a search query, which is why no ranking of results is possible [57].

2.4 Semantic Search

Semantic Search [59] means the use of Semantic Web technologies to implement a search. The goal is to improve the search with the use of semantics. Guha et al. [59] attribute two goals to Semantic Search: "The first is to augment traditional search results with data pulled from the Semantic Web. The second is to use an understanding of the denotation of the search term to improve traditional search."

2.4.1 Entity Retrieval Model for Web Data

Delbru et al. [57] describe an *Entity Retrieval Model* for Web Data to describe semi-structured information from heterogeneous and distributed sources for semi-structured information. The model consists of three components: *dataset*, *entity*, *view*. A *dataset* is a collection of entity descriptions and can be uniquely referenced by an URI. An *entity* is something that can be defined via an URI (such as documents, events or persons) and for which descriptions exist in the form of properties. A *view* is a piece of information accessible via an URI which is used to describe the dataset. Example 2.9 shows the difference between the three components *dataset*, *entity*, *view*.

Example 2.9: Entity retrieval model for web data

A *dataset* is a *database*, e.g., an RDF triplestore, which is available at an address like `http://example_database.de`. This database contains various *documents* (*views*), for example the following:

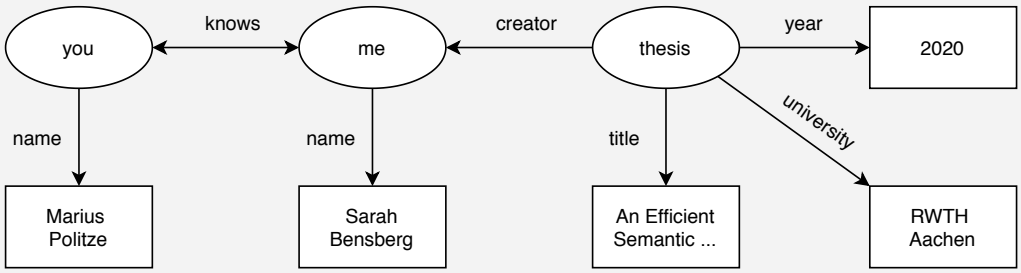
```
# person
<me>
  <knows> <you> ;
  <name> "Sarah Bensberg" .
<you>
  <knows> <me> ;
  <name> "Marius Politze" .
```

```
# publications
<thesis>
  <creator> <me> ;
  <year> 2020 ;
  <title>
    "An Efficient Semantic ..." ;
  <university> "RWTH Aachen" .
```

[1] PAT is a registered trademark of Open Text Corporation

Example 2.9 (continued)

From these documents the following *RDF graph* (*entity descriptions*) results:

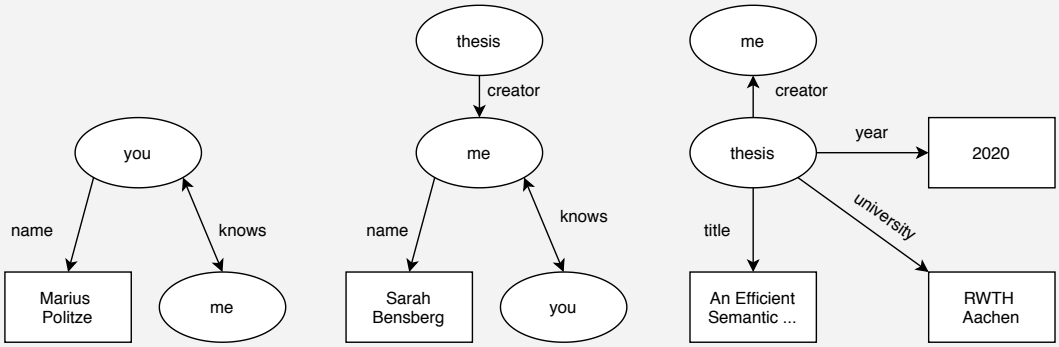


2.4.2 Entity Attribute-Value Model

The *Entity Attribute-Value Model* [57] is a model to describe the entities of a dataset. This is done with an Identifier (ID) describes the entity, a set of labeled properties, and a set of values for these properties (see Example 2.10).

Example 2.10: Entity attribute-value model

Related to the Example from 2.9 the data graph provides information about the entities *me*, *you* and *thesis* and their relations and textual attributes (literals). Consequently, the following subgraph can be extracted for each entity:



From these subgraphs the following entity attribut-value model is derived:

me

attribute	value
name	Sarah Bensberg
knows	you
knows^{-1}	you
creator^{-1}	thesis

you

attribute	value
name	Marius Politze
knows	me
knows^{-1}	me

thesis

attribute	value
title	An Efficient Semantic ...
year	2020
creator	me
university	RWTH Aachen

2.4.3 Search Model for Web Data

Furthermore, Delbru presents a search model for web data which, in contrast to conventional web search engines, does not represent entities as a bag of words but describes them as a set of attribute-value pairs. For the search, three different search types can be distinguished: (i) *full-text queries* which represent a search request with unknown structure,

(ii) *structural queries* which represent more complex queries in form of key-value-pairs if the structure is known and (iii) *semi-structural queries* as the combination of both.

2.5 Related Work

This section summarizes the current state of research on approaches to searching in RDF-based knowledge graphs that do not require SPARQL knowledge on the user's side. Table 2.1 shows an overview of the approaches. They are divided into different categories, whereby often not a clear separation is possible or ambiguous.

Retrieval of Ontologies and Documents in the Semantic Web Using Keywords

At first there are approaches like *Swoogle* [60] and *Sindice* [61]. *Swoogle* is a crawler-based indexing and retrieval system for RDF documents. For this purpose, an IR system is applied, which uses either character *n-grams* or URIs as terms to find documents given the search terms. Besides, a rank is calculated for each document, which indicates the importance of the document in the Semantic Web. On the contrary, *Sindice* indexes RDF documents based on resource URIs, Inverse Functional Propertys (IFPs) and keywords. However, the focus of these two approaches is on the retrieval of ontologies and documents in the Semantic Web. Similar examples are provided by *SWSE* [62] and *Falcons* [63]. The difference is that the focus is on an object-oriented search instead of a document-based search. *SWSE* is a search engine that allows finding and navigating in an object-oriented search space by keyword-based input. *Falcons* provides keyword-based search for concepts and objects on the Semantic Web by indexing all the terms of an entity's virtual document. "The virtual document of an entity consists of its names (local name and labels), other associated literals, and the names of its neighboring entities in RDF graphs, decoded from all the RDF documents on the Semantic Web" [63]. In the context of CoScInE, the metadata records already exist in an RDF triplestore and are not indexed from the Semantic Web. However, the approaches demonstrate the distinction between object-based and document-based search. CoScInE is about finding metadata records, i.e. documents described by properties or specific entities, namely instances of different application profiles. Furthermore, the meaning of names, labels, and literals, which are crucial for identification or search, can be derived from these approaches. Many approaches only allow a keyword-based search. The goal is to enable more search features like boolean queries to allow the user to perform more complex searches.

Other approaches can be divided into two major areas: One is the transformation of a user's input or interaction into a SPARQL query, which is finally executed. The other is the transformation of the knowledge graph into a search index, which is used by a search engine. The approach to convert to a SPARQL query can again be divided into different categories such as (i) *SPARQL Query Builder* and (ii) *Faceted Search*.

Generating a SPARQL Query

A very basic approach to generating a SPARQL query from the keyword-based search of a user is presented by Shekarpour et al. [64]. The construction of the query takes background information about the existing data structure into account. Predefined basic graph pattern templates and a mapping of keywords to possible URIs are used. From this approach, can be learned that it is helpful to use known background knowledge and structures to generate adequate queries.

SPARQL Query Builder

SPARQL query builder allow a step-by-step construction of SPARQL queries using cleverly chosen user interfaces. There is a wide variety of such query builders. Kuric et al. [65] have compared the best-known query builders regarding their usability for laypeople and divided them into three different categories, whereby these are partially mixed up and thus represent a hybrid approach. Their concepts are briefly described hereafter.

(i) *Form-Based Query Builders* use input fields to build the SPARQL query step by step. Classes and objects must either be advised or made available for selection by the user. This is one reason why only a limited number of queries are possible using such tools. *ExConQuer* [66], the *Linked Data Query Wizard* [67], *VizQuery* [68], and the *Wikidata Query Service (WQS)* [69] are examples of such approaches. However, the user can only use the queries that are constructable with the help of the user interface.

(ii) *Graph-Based Query Builders* make use of a visual and graphical user interface that guides the user in creating valid SPARQL queries. A disadvantage is that the rough structure must be known and in some cases SPARQL knowledge is necessary to be able to set up arbitrary queries. Some examples of these approaches are *iSPARQL* [70], *NITELIGHT* [71], *OptiqueVQS* [72], and *QueryVOWL* [73].

(iii) *Natural Language-Based Query Builders* build the query based on the natural language of the user. Therefore, linguistic considerations must be taken into account. Mostly, such approaches are limited by linguistic ambiguity and variability. Often, they are also limited to a certain domain or database because the development of such Natural Language (NL) approaches is costly and complicated. *NLP-Reduce* [74] and *SPARKLIS* [75] are examples for a NL-based approach. Also *DEANNA* [76] is based on NL approaches to generate a SPARQL query using phrase concept dictionaries for entities, classes, and relations. Another approach assigned to this category is *AskNow* [77], where the user can ask natural language questions, which initially are normalized into an intermediary canonical syntactic form, called Normalized Query Structure (NQS), and subsequently translated into SPARQL queries. *Swip* [78] (Semantic Web Interface using Patterns) is a system that also handles natural languages queries whereby queries are transformed to SPARQL requests in two steps: First, the input is translated into a pivot query and then into the formal query language. The *Semantic-based closed and open domain Question Answering System (ScoQAS)* [79] also belongs to the same category and uses Artificial Intelligence (AI) and Natural Language Processing (NLP) methods. Understanding NL questions or user input requires techniques to find out the intention of a query. This is a separate field of research, which is not the main focus of this master thesis. Instead, this thesis focuses on finding metadata records rather than answering a specific questions. Consequently, approaches that apply NLP are excluded from further consideration. This also applies to approaches that do not generate a SPARQL query but are answering questions over knowledge graph systems. *QAmp* [80] is an example of this which is based on an unsupervised message-passing algorithm. Zhu et al. [81] present a *Graph Traversal Based Approach to Answer Non-Aggregation Questions Over DBpedia*, which is also an example of such an approach. It first links the input to a resource and then looks at the subgraph of that resource to find the best possible path to the question.

Faceted Search

Faceted Search is used to allow flexible navigation through a large search space and to limit this space by so-called facets [82]. English et al. [83] describe that *hierarchical faceted*

metadata must be disclosed intuitively and appealingly to achieve this goal. Metadata is faceted (multiple orthogonal categories), hierarchical or flat, and single or multi-valued. Faceted search can also be applied to RDF knowledge graphs by offering properties (predicates) of entities (subjects) as facets with corresponding values (objects). *OSCAR* [84], the OpenCitations RDF Search Application, and *SemFacet* [85] are examples of such an approach.

Other Notable Approaches

Other notable approaches that generate a SPARQL query are the following: The *Assisted SPARQL Editor* [86] leverages the graph summary developed by Campinas in the context of the *Sindice* project to support the user in effectively formulating complex SPARQL queries. For this purpose, suggestions for classes, predicates, relationships between variables, and named graphs are provided to the user. This approach is not promising, because the user still needs to know the syntax of SPARQL queries. *SemSearch* [87] is a search engine that translates user input according to the syntax *subject:keyword1 and/or keyword2 and/or keyword3* into a formal SPARQL query. In principle, this creates its own search syntax, which is simpler but much less capable than the existing syntax SPARQL and therefore is not considered in detail. *SINA* [88] is a keyword-based search system, which uses a hidden markov model to transform a user's input into a SPARQL query. *AutoSPARQL* [89] uses active supervised machine learning to formulate a SPARQL query.

Generating a Search Index to use IR Techniques

Linked swissbib [90], *Open Semantic Search* [91], and *Semplore* [92] are examples for the approach of transforming a knowledge graph into a search index to use IR techniques. *Linked swissbib* converts bibliographic data into a RDF-based data model and divides it into six different concepts. By using the *JSON-LD* serialization of RDF, they are indexed in the search engine *Elasticsearch (ES)*. *Open Semantic Search* stores the associated labels of the RDF annotation to URIs for indexing data in Solr and allows a faceted search without generting a SPARQL query. *Semplore* also uses a hybrid query option that allows structured queries and faceted searches. For this purpose, Semantic Web data is translated into documents, fields, and terms properly so that the IR engine can index them in their inverted index structure and provide retrieval functions via the data. Rocha et al. [93] presented a hybrid approach to keyword-based search in the Semantic Web, where the focus is also on finding concepts. For this purpose, an instance graph is created for each concept based on the values of its properties, which is searched with traditional IR techniques. Afterwards, spread activation techniques are applied to find related concepts. A knowledge engineer has to weight the paths in the graph to the related concepts since they are always domain-dependent. In the context of CoScInE, the domain is not known in advance or can change by further application profiles and application areas. Thus, such an approach is not suitable. Changes and adjustments would have to be made again and again, which is why it is not pursued further. Consequently, the creation of the instance graphs is interesting, which store all terms from associated properties for a concept to make them referenceable.

Table 2.1: Overview of related approaches

Name	Ref.	Type/Principle		
		Retrieval in the Semantic Web (keyword-based)	**SPARQL Generator**	**Search Index and IR techniques**
Swoogle	[60]	Document-oriented, *N-grams* or URIs		
Sindice	[61]	Document-oriented, URIs or IFPs		
SWSE	[62]	Object-oriented		
Falcons	[63]	Object-oriented, Virtual document		
ExConQuer	[66]		Form-based	
Linked Data Query Wizard	[67]		Form-based	
VizQuery	[68]		Form-based	
WQS	[69]		Form-based	
iSPARQL	[70]		Graph-based	
NITELIGHT	[71]		Graph-based	
OptiqueVQS	[72]		Graph-based	
QueryVOWL	[73]		Graph-based	
NLP-Reduce	[74]		NL-based	
SPARKLIS	[75]		NL-based	
DEANNA	[76]		NL-based	
AskNow	[77]		NL-based	
Swip	[78]		NL-based	
ScoQAS	[79]		NL-based	
QAmp	[80]		NL-based, unsupervised message-passing algorithm	
Zhu et al.	[81]		NL-based, graph traversal	
OSCAR	[84]		Faceted search	
SemFacet	[85]		Faceted search	
Shekarpour et al.	[64]		Basic graph pattern templates, mapping of keywords to URIs	
Assisted SPARQL Editor	[86]		Graph summary	
SemSearch	[87]		Own search syntax	
SINA	[88]		Hidden markov model	
AutoSPARQL	[89]		Active supervised machine learning	
Linked swissbib	[90]			JSON-LD, Elasticsearch

Table 2.1: Overview of related approaches (continued)

Name	**Ref.**	**Type/Principle**		
		Retrieval in the Semantic Web (keyword-based)	**SPARQL Generator**	**Search Index and IR techniques**
Open Semantic Search	[91]			Solr, labels of RDF annotations, faceted search
Semplore	[92]			Translation into documents, fields, and terms, faceted search
Rocha et al.	[93]			Instance graph

3 Technical Details of CoScInE

This chapter deals with the relevant technical implementations of CoScInE. The focus is on the data and database model used for the metadata since this is relevant for the implementation of the search. Also, basic general conditions and requirements for the search are described. These must be taken into account when considering and evaluating the different approaches in the chapters 4 and 5.

3.1 Data Model

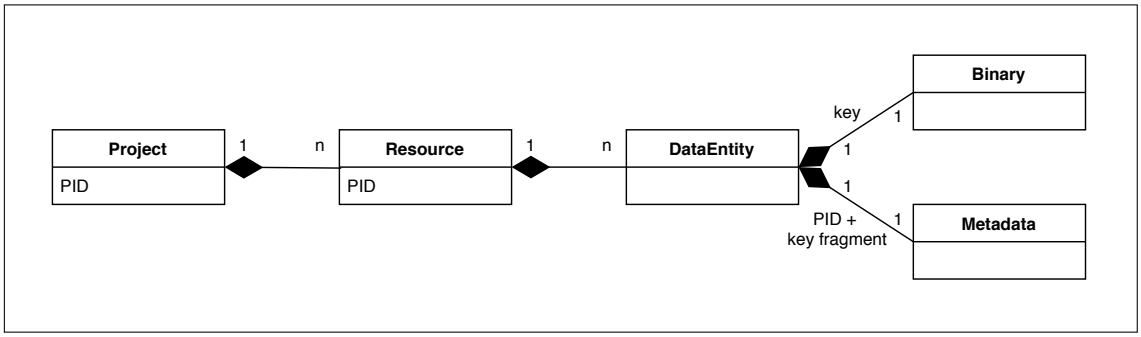


Figure 3.1: CoScInE – Data model

Since CoScInE [12, 16] is a project-based system, a predefined data model exists, which can be seen in Figure 3.1. Projects can be created by users and members can be added to these projects. It is also possible to define the visibility of projects: Either the data can be declared as public, i.e. everyone is allowed to view it, or only the project members are allowed to access it. For each project, any number of resources can be stored. A resource acts like a storage service and can further contain multiple data entities which each consist of a binary and a metadata record. The latter are the focus of this thesis. To each stored resource a unique PID is assigned, which can be used to add further data objects and which provides a resolution via the *handle* system [94]. Since the PID is an URI, it is suitable as a node of a *Linked Data knowledge graph*, which can represent metadata. Here, the PID with an additional key fragment (since a resource can have several files) is used to reference the corresponding metadata and to use it as subject of all associated metadata triples. The predicate is the corresponding metadata field and the object is the metadata value. As already mentioned, the PID is created via the *handle* system. However, for the values of the linked metadata the question arises how to map (new) entities and create unique IDs for these or other resources. The question is how to make sure that no URI exists to identify the entity or that this URI is reused. Besides, another aspect is the creation and publication of URIs in order to enable linking further information according to the principle of *Linked Data*. The answers to these questions are not part of the work but should be kept in mind.

RDF was chosen as data format because its schema-lessness provides flexibility to changes in research data and disciplines. Metadata graphs are thus individually extendable and adaptable and any vocabulary can still be used. RDF supports the approach of

Linked Data and uses the interoperability and reusability component of *FAIR Guiding Principles*. However, the flexibility of the RDF model leads to unstructured data and no uniform specifications for the metadata can be made. For this reason, SHACL is used to provide structural components to individual metadata graphs, therefore ensuring consistency and usability of the user interface.

SHACL enables clear structures and restrictions for RDF data. In comparison, ontologies can also define value ranges, but these cannot be validated or due to the OWA allow inconsistencies and supposed contradictions. In the past, ontologies have often been used incorrectly to narrow down knowledge and impose restrictions to ensure data quality. However, the use of ontologies only allows the reverse variant, namely to extract conclusions about existing data. Example 3.1 demonstrates this.

Example 3.1: Ontology use

The Organization Ontology (ORG) metadata standard [95] defines the property `org:memberOf`. `org:Agent` is defined as definition range and an `org:Organization` as value range:

```
org:memberOf
  rdfs:domain org:Agent ;
  rdfs:range org:Organization .
```

However, this does not ensure that each triple using the property `org:memberOf` as predicate has an `org:Agent` as subject and an `org:Organization` as object. But in contrast to this, a triple of the form `<x> org:memberOf <y>` is interpreted in such a way that `x` is an `org:Agent` and `y` is an `org:Organization`, although this might not be the case in reality:

(org:memberOf rdfs:domain org:Agent) ∧ (org:memberOf rdfs:range org:Organization) ∧ (<x> org:memberOf <y>) → (<x> a org:Agent) ∧ (<y> a org:Organization)

This is where SHACL comes in, since it supports the validation of RDF graphs against different constraints (see section 2.2.3). So, a `sh:NodeShape` validates that the data meets the conditions in any case, otherwise, the data graph does not fulfill the form of the *Node Shape*. In principle, this turns the Open World into a Closed World in which data follows certain principles and rules (see section 2.2.6). The result is a system of semi-structured data due to the fact that on the one hand there are certain (structured) fields and on the other hand there are different SHACL application profiles with different fields, vocabularies, and structures (unstructured). An application profile defines the structure for metadata to be stored. Within the scope of CoScInE, various application profiles are continously developed, each of which is geared to the subject-specific requirements of different disciplines. To ensure uniform use, the SHACL profiles are based on existing metadata schemas, whereby the metadata schema developed within the Research Data Repository (RADAR) project [28] is intended as a minimum [9].

Furthermore, according to the concept of *Linked Data* (see section 2.2) existing metadata standards and vocabularies can be used. Since the principle of the Semantic Web and the use of RDF allows the creation of arbitrary ontologies and vocabularies, there is a growing pool of such ontologies and vocabularies. Again and again, things, objects, relations, etc. are modeled redundantly by independent groups of knowledge engineers, since it is difficult to keep an overview of existing data sources and there is

no uniform rule for publishing them. However, there are already some approaches to counteract this problem like "Zazuko's Default Ontologies & Prefixes" GitHub Repository [96], which lists the most common ontologies, or the "VocabularyMarket" [97], which supports the search for suitable terms and vocabularies. To ensure the use of existing schemas and vocabularies, a knowledge engineer is required who has an overview of existing metadata standards in a variety of domains and who monitors the development process of a new application profile [13]. In CoScInE, Dublin Core [37], ORG [95], Core Scientific Metadata Model (CSMD) [98], FOAF [48], and DataCite [27] are already in use.

Because the most common application of metadata is to assign a value from a thesauri or a subject catalogue, ontologies are used to map these [13]. They define individuals as instances of a class that can be linked to the metadata record via an object property. Thus, a controlled vocabulary (see section 2.1) is generated. The vocabularies SPDX License List [99], DCMI Type Vocabulary [100], and the subject classification of the GRF [101] are currently used in CoScInE as controlled vocabularies for metadata fields.

File 3.1 contains an extract of the *EngMeta* metadata schema [102] for the description of engineering research data from the project *DIPL-ING* [103].

```
<https://purl.org/coscine/ap/engmeta/>
  a sh:NodeShape ;
  sh:targetClass <https://purl.org/coscine/ap/engmeta/> ;
  sh:property coscineengmeta:creator ;
  sh:property coscineengmeta:worked ;
  sh:property coscineengmeta:title ;
  sh:property coscineengmeta:subject ;
  sh:property coscineengmeta:created ;
  sh:property coscineengmeta:version .

coscineengmeta:creator
  sh:path dcterms:creator ;
  sh:order 1 ;
  sh:minCount 1 ;
  sh:name "Creator"@en, "Autor"@de ;
  sh:class foaf:Agent .

coscineengmeta:worked
  sh:path engmeta:worked ;
  sh:order 2 ;
  sh:maxCount 1 ;
  sh:datatype xsd:boolean ;
  sh:name "Worked"@en, "Hat funktioniert"@de .

coscineengmeta:title
  sh:path dcterms:title ;
  sh:order 3 ;
  sh:minCount 1 ;
  sh:minLength 1 ;
  sh:datatype xsd:string ;
  sh:name "Title"@en, "Titel"@de .

coscineengmeta:subject
```

```
  sh:path dcterms:subject ;
  sh:order 4 ;
  sh:maxCount 1 ;
  sh:name "Subject Area"@en, "Sachgebiet"@de ;
  sh:class <http://www.dfg.de/dfg_profil/gremien/fachkollegien/faecher/> .

coscineengmeta:created
  sh:path dcterms:created ;
  sh:order 5 ;
  sh:minCount 1 ;
  sh:maxCount 1 ;
  sh:datatype xsd:date ;
  sh:name "Creation Date"@en, "Erstellungsdatum"@de .
  
coscineengmeta:version
  sh:path engmeta:version ;
  sh:order 6 ;
  sh:minCount 1 ;
  sh:maxCount 1 ;
  sh:datatype xsd:integer ;
  sh:name "Version"@en, "Version"@de .
```

File 3.1: Extract of the *EngMeta* metadata schema [102]

In the file, `https://purl.org/coscine/ap/engmeta/` is declared as `sh:NodeShape`, i.e. as an application profile. In addition, six different `sh:property` are defined: `coscineengmeta:creator`, `coscineengmeta:worked`, `coscineengmeta:title`, `coscineengmeta:subject`, `coscineengmeta:created`, and `coscineengmeta:version`. For all these properties, there are further descriptions and restrictions in the profile. To access existing properties from metadata standards, the referenced property can be stored via `sh:path`. `sh:order` sets the order of the properties and `sh:minCount` or `sh:maxCount` determines the minimum and maximum number of values for this property. `sh:datatype` can be used to specify a data type and `sh:name` can be used to define a name for the property if the existing label of the ontology is to be overwritten. If the value of the property should not be a literal, but should be described as an instance of a class in the form of a vocabulary, `sh:class` can be used to specify this class and thus all instances of the class are allowed as values. In this case, the stored value is an URI. Besides, DASH can specify further definitions and restrictions, e.g., `dash:singleLine` [45] determines that no line breaks are allowed in the lexical form of literal value nodes.

Using the SHACL application profiles, a form is generated [104], which hides all technical details about the definition and structure of the metadata. Restrictions are mapped automatically so that, e.g., mandatory fields are marked as such or a maximum of one element can be selected. Furthermore, the user gets easy access to the ontology, e.g. to the vocabularies. Figure 3.2 shows the input mask generated from the application profile of File 3.1. The illustrations shows that class constraints are automatically transformed into drop-down boxes and the user can select an instance without having to know the URI behind it. Data types are also transformed into corresponding form fields.

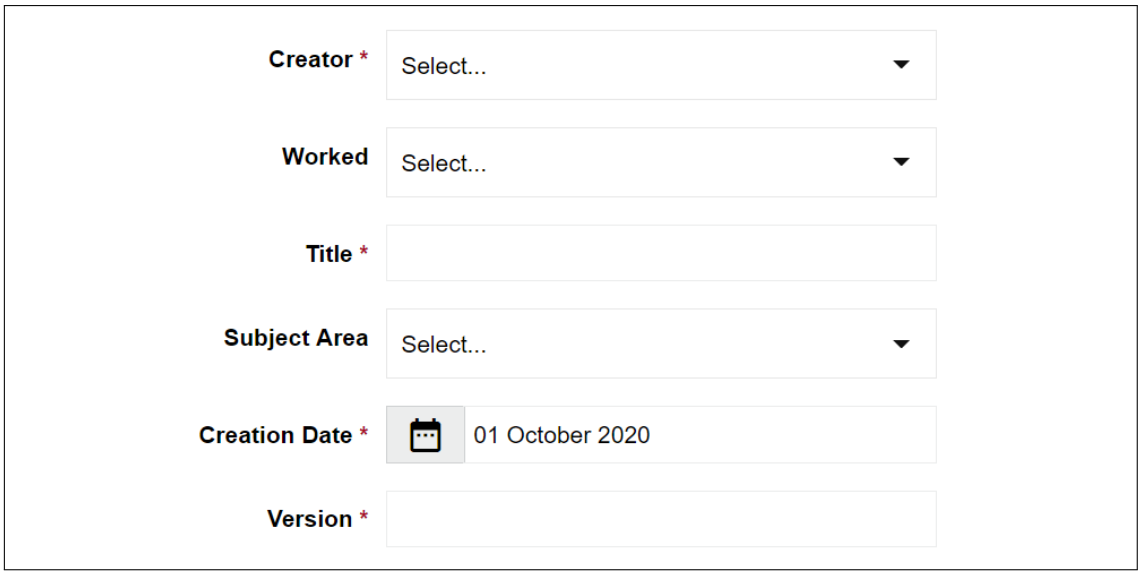


Figure 3.2: CoScInE – Generated form from the application profile of File 3.1

3.2 Database Model

To store the metadata of CoScInE, different triple stores, which are graph databases for storing and retrieving RDF triples, were evaluated in advance and finally Virtuoso [105] was selected as database management system [106]. At first, the open-source variant is used. Virtuoso provides a SPARQL endpoint to query and manipulate the included RDF data. A database can store several *contexts* (see section 2.2.1). These are used to structure the necessary data for administration as well as the actual metadata in a meaningful way. In case of changes, the division serves the simple exchange or deletion of graphs. The following graphs are created as *Named Graph* and thus form the *knowledge graph* of CoScInE: (i) any necessary ontology explicitly used for inference rules in Virtuoso or including the definition of individual properties and their associated labels, (ii) each application profile, (iii) each metadata record for a resource, (iv) each vocabulary, and (v) all project management information (resources, organizations, projects, and their members).

3.3 Components and Processes

Figure 3.3 shows the most important components and interrelationships of CoScInE. First, a SHACL application profile with the desired metadata fields and restrictions is created as described above. These are adapted to the requirements of an institute, an organization or a project. *Vocabularies* and *Metadata Standards* are reused and created. In the future, this will be handled by the *Application Profile Generator*. The profile is stored in the *Application Profile Repository* so that it can be called up in CoScInE and a corresponding input mask can be generated if the user wants to create metadata for a file. Therefore, the form in the *Metadata Interface* has to be filled and saved. The data is sent to the backend via HTTP request using a REST API and validated there using the SHACL profile. Another possibility is to connect instruments directly to CoScInE to generate metadata automatically. As a final step, the resulting *Formalized RDF Metadata Graph* is stored in the *Metadata Repository* which is assigned via the PID to a resource and the associated research data that are located on a server. File 3.2 shows an exemplary metadata record for the previously presented extract of the application profile of File 3.1.

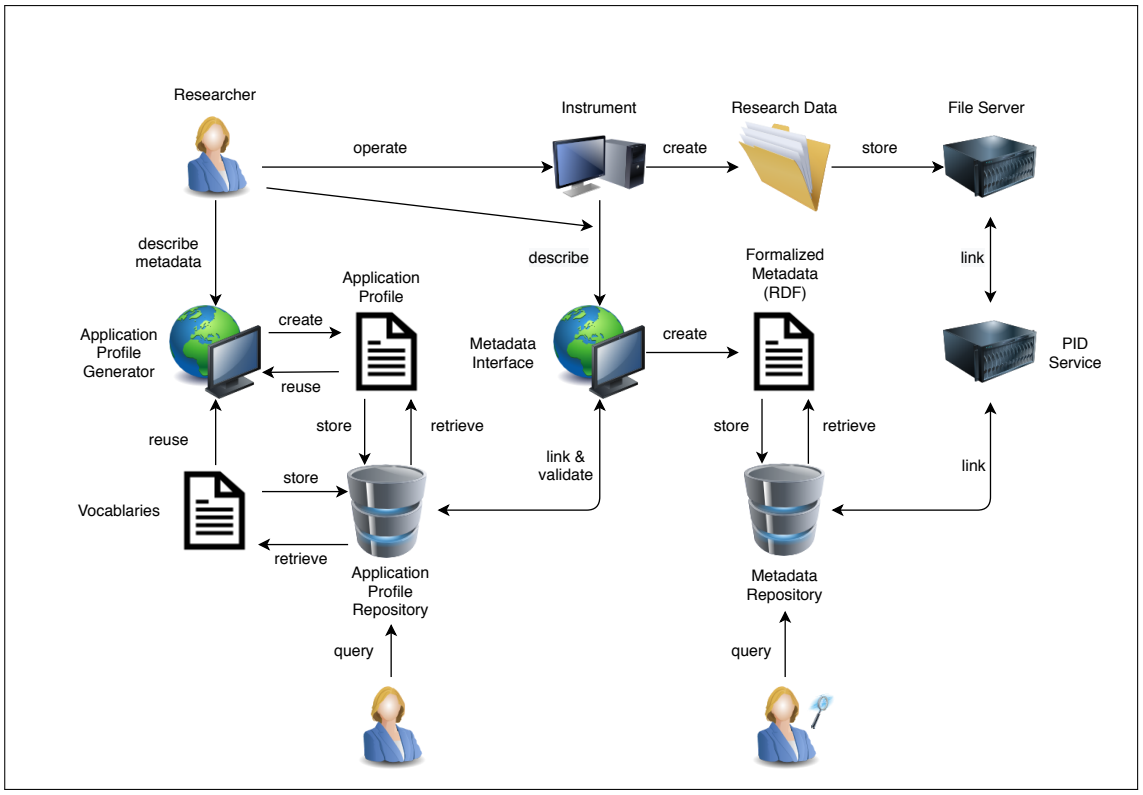


Figure 3.3: CoScInE – Components and processes [107]

```
<https://hdl.handle.net/21.11102/61aT5b993PMMy@path=example.txt>·
····dcterms:creator·<http://dbpedia.org/resource/Thomas_Pynchon>·;
····engmeta:worked·true·;
····dcterms:title·"Design·Patterns"·;
····dcterms:subject·<http://www.dfg.de/.../liste/index.jsp?id=409#409-02>·;
····dcterms:created·"2020-01-01+02:00"^^xsd:date·;
····engmeta:version·3·;
····a·<https://purl.org/coscine/ap/engmeta/>·.
```

File 3.2: Example metadata record of application profile of File 3.1

3.4 Search Requirements

Since this master thesis deals with the evaluation of different approaches to enable a search in the metadata records of CoScInE, the following section defines some requirements for the realization. The general conditions are a basic requirement for the use of an approach. The quality of the results is evaluated by the categories usability and complexity for the user and the quality of the search by the category effectiveness (see section 1.5).

3.4.1 General Conditions

Visibility
The visibility for the metadata files results from the visibility of their corresponding project. For the search, this means that users may only find metadata that is accessible and approved for them.

Language
In the future, the system will not only be used at RWTH. For this reason, it should be possible to set up the system in different languages. Currently, the implementation for English and German is prioritized. Within the scope of the work, the focus is on the English language implementation, whereas other languages are equivalent to it.

User Background
The general user who ends up searching the data records is a researcher who does not necessarily come from a technical environment. Besides, usually no RDF or SPARQL knowledge is present and users are not aware of the structure of the data or the database model.

3.4.2 Effectiveness

According to International Organization for Standardization (ISO), effectiveness describes the "extent to which planned activities are realized and planned results are achieved" [108]. In this context, this refers to the quality of the search results. The user searches CoScInE with a certain intention. The goal is either to find concrete data records (see *Navigational Queries* 2.3.2) or to obtain information about existing data (see *Informational Queries* 2.3.2). The system should find the correct data records and thus satisfy the user's intention. This also means that data records should be ranked accordingly and *false positives* and *false negatives* should be avoided.

3.4.3 Usability

According to ISO, the word "usability" is used "as a qualifier to refer to the design knowledge, competencies, activities and design attributes that contribute to usability" [109]. In the context of the search, it is therefore primarily a matter of making the search interface appealing and understandable to establish a pleasant search for the user.

3.4.4 Complexity of Search Request for the User

This requirement is about how complex the creation of the search query is for the user. The first step is to compare whether the user is expected to have certain knowledge. Furthermore, it is checked whether the user needs to adhere to a certain search syntax to make successful search queries or whether any user input can be evaluated. Finally, the comprehensibility or complexity of this search syntax is examined.

3.4.5 Efficiency

The ISO defines efficiency as "relationship between the result achieved and the resources used" [108]. In the context of search, this ratio refers to the effort a user has put in (input) compared to the generated results (output). This effort includes knowledge that a user must have or activities that a user must perform to start the search.

3.4.6 Response Time

Nielsen [110] differs several types of response times: One limit is set to 0.1 seconds so that the user feels that the system is responding directly to the interaction. Another limit is 1.0 second: the user notices the delay but remains uninterrupted. This concludes that the search results (i.e., from entering the search query to displaying the found metadata) should be found within 0.1 seconds if possible and within one second at most.

3.4.7 Scalability

CoScInE is a system that will grow steadily the more metadata about research data is stored and collected. That means that the system should be able to handle large amounts of data and enable searching in this amount of data.

3.4.8 Additional Effort

In this category, the effort is considered, which must be operated additionally to the actual storage and search. On the one hand, it is about whether data is stored redundantly, i.e. whether the use of additional storage is necessary. On the other hand, it is about whether additional calculations are necessary and how time-consuming these calculations are.

4 Different Approaches

This chapter presents the approaches that have been examined in detail in the context of the master thesis and applied to the context of CoScInE. As described in section 2.5, there is a variety of different approaches, some of which have already been excluded for further consideration. Furthermore, many of the presented approaches are outdated, no longer available or there is no code or exact description for their implementation and realization. Ironically, this is exactly the phenomenon, which should be solved by a RDM system. In addition, many approaches are domain-specific and would have to be adapted for use in CoScInE. All approaches with natural language questions are further excluded because the focus of CoScInE is not on answering a question but on finding documents or in this case metadata records. Furthermore, it is not about finding data in the entire Semantic Web that needs to be collected first, but the data is already stored in an RDF database and through the SHACL profiles follows a certain structure that is known to us. For these reasons, it was decided to take a closer look at the following approaches to CoScInE. All of them taking into account that the user has no knowledge of SPARQL, the data structure, used terms, and vocabularies.

4.1 Literal and Additional Rules

In the context of the master thesis, two different types of rules were established, which are used in some approaches to implementing the search and are presented in the following.

Literal Rules

As introduced in chapter 3, the metadata records to be found are structured in such a way that, due to the SHACL profiles, they contain an URI as the subject, the metadata field as the predicate, and either a literal or an URI as the object. It is assumed that a user does not search directly for an URI or does not even know it but the corresponding label. The matching literal to the URI can then be found elsewhere depending on the instance or class. There is also not always a suitable `rdfs:label` property available, as described in the approach [64], which could be used to find suitable URIs. But since it is also possible that the object is directly the literal, the previous mapping to an URI is not used. Also, all used classes are known and can therefore be used to find literals with the help of the SHACL profiles. Besides, further literals for the description of a resource may exist in the knowledge graph, in which the user should also be able to search. Compound literals or the direct application of inferences to represent human knowledge as well as relations and to improve the search would also be conceivable. If this implicit information is stored additionally as SHACL rule [111] for each class and thus make it explicit, the user's search query can automatically be applied to the matching literals. In the following, these rules are called *literal rules*. Example 4.1, 4.2 and 4.3 serve as demonstration of these. As can be seen from the two examples, SHACL rules can be used to map literals and structures of any complexity. Since the mapping may need to differ for different application profiles, they could be added and applied per profile.

Example 4.1: Literal rule for the class `foaf:Person`

Suppose it exists the following input graph which describes the person Thomas Pynchon:

```
<http://dbpedia.org/resource/Thomas_Pynchon>
  a foaf:Person ;
  foaf:name "Thomas Pynchon"@en .
```

From this, the following graph is to be generated for the literals:

```
<http://dbpedia.org/resource/Thomas_Pynchon>
  rdfs:label "Thomas Pynchon"@en .
```

The following SHACL rule shows the generation of the matching literal of a `foaf:Person`, where `$this` is the focus node, which is replaced by the respective instance of the class. The property `sh:prefixes` can be used to declare the prefixes for the SPARQL query.

```
<http://xmlns.com/foaf/0.1/Person>
  sh:rule [
    a sh:SPARQLRule ;
    rdfs:label "Infer literal for a person" ;
    sh:prefixes prefixes:personPrefixes ;
    sh:construct """
      CONSTRUCT {
        $this rdfs:label ?label
      } WHERE {
        $this foaf:name ?label
      }
    """ ;
  ] .
```

Example 4.2: Advanced literal rule for the class `foaf:Person`

Regarding Example 4.1, if not only the name but also the first and last name individually or the corresponding organizational unit and organization should be matched, the literal rule used is the following:

```
<http://xmlns.com/foaf/0.1/Person>
  sh:rule [
    a sh:SPARQLRule ;
    rdfs:label "Infer additional literals of a person" ;
    sh:prefixes prefixes:personPrefixes ;
    sh:construct """
      CONSTRUCT {
        $this rdfs:label ?label
      }
      WHERE {
        {
          $this foaf:name ?label
        }
        UNION
        {
          $this foaf:givenName ?label
        }
        UNION
        {
          $this foaf:familyName ?label
        }
```

Example 4.2 (continued)

```
        UNION
        {
          ?membership a org:Membership .
          ?membership org:member $this .
          ?membership org:organization ?organization .
          ?organization rdfs:label ?label
        }
        UNION
        {
          ?membership a org:Membership .
          ?membership org:member $this .
          ?membership org:organization ?unit .
          ?organization org:hasUnit ?unit .
          ?organization rdfs:label ?label
        }
      }
    """ ;
  ] .
```

Example 4.3: Literal rule for the subject classification of the GRF

For instances of the GRF subject classification, the literals of the superclass are also applied because an instance of the subclass is also automatically an instance of the superclass. The following graph is assumed:

```
<http://www.dfg.de/.../liste/index.jsp?id=409#409-02>
  a <http://www.dfg.de/.../liste/index.jsp?id=409#409-02> ;
  rdfs:label "Software Engineering and Programming Languages"@en ;
  rdfs:subClassOf <http://www.dfg.de/.../liste/index.jsp?id=409> .

<http://www.dfg.de/.../liste/index.jsp?id=409>
  a <http://www.dfg.de/.../liste/index.jsp?id=409> ;
  rdfs:label "Computer Science"@en .
```

From this, the following graph for the literals should be generated:

```
<http://www.dfg.de/.../liste/index.jsp?id=409>
  rdfs:label "Software Engineering and Programming Languages"@en,
      "Computer Science"@en .
```

The following SHACL rule shows the generation of the matching literals:

```
<http://www.dfg.de/dfg_profil/gremien/fachkollegien/faecher/>
  sh:rule [
    a sh:SPARQLRule ;
    rdfs:label "Infer literals of a GRF instance" ;
    sh:prefixes prefixes:GRFPrefixes ;
    sh:construct """
      CONSTRUCT {
        $this rdfs:label ?label
      }
      WHERE {
        ?class rdfs:subClassOf* ?superclass .
        ?superclass rdfs:label ?label .
        $this a ?class .
      }
    """ ;
  ] .
```

Additional Rules

Furthermore, it is possible to use *additional rules*, which enable the automatic generation of further triples for specific metadata records. A human being is aware of further knowledge or information due to the specified properties of a metadata record, which are withheld from a machine and thus from the search. The rules are used to slightly close this gap. Example 4.4 contains a simple additional rule to create a triple indicating whether the referenced data is in the last step.

Example 4.4: Additional rule for last step

It is assumed that a metadata field `engmeta:step` is filled by instances of the self-created vocabulary "steps" (see section G.6 "Published Research Data"). The instances are processing steps that follow a certain order. An additional triple should be created, which uses a boolean value and the predicate `coscinesearch:isLastStep` to indicate if a metadata record is the last step. The following graph is given:

```
<ID> engmeta:step coscinestep:storage .

coscinestep:storage
  rdfs:label "Data storage"@en ;
  a <https://purl.org/coscine/terms/step/> .
```

From this, the following graph should be generated:

```
<ID> coscinesearch:isLastStep true .
```

The following rule creates this additional triple. The focus node `$this` is replaced by the ID of the metadata record.

```
<https://purl.org/coscine/ap/engmeta/>
  sh:rule [
    a sh:SPARQLRule ;
    rdfs:label "Infer if step is the last" ;
    sh:prefixes prefixes:lastStepPrefixes ;
    sh:construct """
      CONSTRUCT {
        $this  coscinesearch:isLastStep ?value
      } WHERE {
        $this engmeta:step ?step .
        BIND (not exists{?step schema:predecessorOf ?otherStep} as ?value)
      }
    """ ;
  ] .
```

In Example 4.4, only a triple is created for the given metadata record. However, the rules can be more complex and also influence other metadata records as shown in Example 4.5.

Example 4.5: Additional rule for newest version

We assume a metadata field `engmeta:version` that contains the version number of a metadata record in the form of an integer value. It is assumed that the associated resource is used as a kind of backup, which contains the same data in different versions. Thus, it is possible for a human being to identify the metadata record with the most current version in a resource. However, this information has to be created explicitly for the search because a machine cannot automatically develop

Example 4.5 (continued)
such connections. Suppose the following input graph exists:

```
<ID1>
 engmeta:version 1 ;
 coscineprojectstructure:isFileOf coscineresource:resource1 .

<ID2>
 engmeta:version 2 ;
 coscineprojectstructure:isFileOf coscineresource:resource1 .
```

From this, the following graph should be generated:

```
<ID1> coscinesearch:isNewestVersion false .

<ID2> coscinesearch:isNewestVersion true .
```

The rule for creating a field indicating the newest version in a resource based on the property `engmeta:version` is as follows. Furthermore, this rule does not only affect a single metadata record but must always be considered in the context of all metadata records of a resource.

```
<https://purl.org/coscine/ap/engmeta/>
  sh:rule [
    a sh:SPARQLRule ;
    rdfs:label "Infer if resource version is the newest one" ;
    sh:prefixes prefixes:newestVersionPrefixes ;
    sh:construct """
      CONSTRUCT {
        ?s coscinesearch:isNewestVersion ?value
      } WHERE {
        ?s engmeta:version ?version .
        ?s coscineprojectstructure:isFileOf ?resource .
        $this coscineprojectstructure:isFileOf ?resource .
        {
          SELECT ?newestVersion
          WHERE {
            ?file engmeta:version ?newestVersion .
            ?file coscineprojectstructure:isFileOf ?resource .
            $this coscineprojectstructure:isFileOf ?resource .
          } ORDER BY DESC(?newestVersion) LIMIT 1
        } .
        BIND( ?version = ?newestVersion AS ?value)
      }
    """ ;
  ] .
```

Also, the additional rules do not have to generate boolean objects. With such rules, any additional information can be generated automatically or additionally to extend or improve the search options for the user. It is possible to define general or specific (for an application profile) additional rules.

4.2 Full-Text Search in RDF Literals

Since Virtuoso is used as triplestore in CoScInE, two different approaches can be distinguished for direct full-text search in single RDF literals with the use of SPARQL. In this thesis, both are considered in the evaluation but are not fully implemented, since it is sufficient to consider the theoretically generated SPARQL queries that result from these.

Full-Text Search in RDF Literals using *regex*

SPARQL offers the possibility to filter data using regular expressions. However, a SPARQL query must be generated internally, which applies the user input to all possible occurrences of a literal by applying the literal rules (see section above). Due to the use of regular expressions, the user is thus not only able to perform a full-text search in a literal but also to use a regular expression for the search. Example 4.6 shows a generated SPARQL query which follows this approach and uses a case insensitive variant for the regular expression.

Example 4.6: Generated SPARQL query using regex for search

The following SPARQL query includes the search if the object is directly the literal or if the literal rule from Example 4.3 is applied. The first two *triple pattern* (`?s a ?profile` and `?profile a sh:NodeShape`) narrow down the search to metadata records. For each literal rule there is a select statement, which is linked by a SPARQL `UNION`. So that the user's input ("query") can be searched in all corresponding literals. Furthermore, the additional rules could also be added using the same principle, but this only makes sense if additional text is inserted because this is a full-text search.

```
SELECT DISTINCT ?s WHERE {
  ?s a ?profile.
  ?profile a sh:NodeShape .
  {
    SELECT ?s {
      ?s ?p ?label .
      FILTER regex(STR(?label), "{query}", "i")
    }
  }
  UNION {
    SELECT ?s {
      ?s ?p ?o .
      ?class rdfs:subClassOf* ?superclass .
      ?superclass rdfs:label ?label .
      ?o a ?class .
      FILTER regex(STR(?label), "{query}", "i")
    }
  }
}
```

Full-Text Search in RDF Literals using *bif:contains*

Virtuoso also offers the option to do a case insensitive full-text search with the *bif:contains* function [112]. Example 4.7 shows the corresponding SPARQL query. The advantage of *bif:contains* over regular expression filtering is that the SPARQL/SQL optimizer determines whether the text pattern is used to control the query or whether the results are filtered after other conditions are applied first. This means that the query could be more efficient because an index is used, which is never the case when using *regex*. *bif:contains* finds only matches in literals, but it allows the use of *and/or/not* operations and furthermore the use of the operator "*" and a *NEAR* operator is possible. Additionally, a *score* is provided at the same time, which makes it possible to rank search results. When using *bif:contains* multiple times, multiple scores must be combined to form a final score. Here, a strategy on how to derive the final score would need to be defined.

Since only indexed literals can be searched with *bif:contains*, it is not possible to use additional rules as well as literal rules which compound the literals (see Example

4.8) in the constructed SPARQL query. Therefore not all literals can be searched or rules can be mapped. There would be the possibility to store the constructed literals and additional triples directly in the triplestore so that they are also indexed. For this, however, a strategy would have to be considered how the original metadata record can be restored, which is why this variant is not considered further in this work.

Example 4.7: Generated SPARQL query using *bif:contains* for search

The following SPARQL query is the same as in Example 4.6 with the difference that *bif:contains* was used instead of *regex*.

```
SELECT DISTINCT ?s WHERE {
  ?s a ?profile.
  ?profile a sh:NodeShape .
  {
    SELECT ?s {
      ?s ?p ?label .
      ?label bif:contains "{query}"
    }
  }
  UNION
  {
    SELECT ?s {
      ?s ?p ?o .
      ?class rdfs:subClassOf* ?superclass .
      ?superclass rdfs:label ?label .
      ?o a ?class .
      ?label bif:contains "{query}"
    }
  }
}
```

Example 4.8: Literal rule which compounds literals For an instance of the variable class the compounds literals from value and unit as well as value and symbol should be used besides the label. Suppose it exists the following input graph:

```
coscinevariable:variable1
  rdfs:label "Variable 1"@en ;
  qudt:unit unit:M ;
  qudt:value "2" ;
  a <https://purl.org/coscine/terms/variable/> .

unit:M
  qudt:symbol "m" ;
  rdfs:label "Meter" .
```

From this the following graph is to be generated for the literals:

```
coscinevariable:variable1
  rdfs:label "Variable 1"@en, "2 Meter", "2 m" .
```

The following literal rule applies:

```
<https://purl.org/coscine/terms/variable/>
  sh:rule [
    a sh:SPARQLRule ;
    rdfs:label "Infer literals for a measured variable" ;
    sh:prefixes prefixes:variablePrefixes ;
    sh:construct """

      CONSTRUCT {
        $this rdfs:label ?label
```

Example 4.8 (continued)

```
      } WHERE {
        {
          $this rdfs:label ?label
        }
        UNION
        {
          $this qudt:value ?value .
          $this qudt:unit ?unit .
          ?unit rdfs:label ?unitLabel .
          BIND(CONCAT(STR(?value), " ", ?unitLabel) as ?label) .
        }
        UNION
        {
          $this qudt:value ?value .
          $this qudt:unit ?unit .
          ?unit qudt:symbol ?symbol .
          BIND(CONCAT(STR(?value), " ", ?symbol) as ?label) .
        }
      }
    """;
  ] .
```

If the functions AND/OR/NOT are to be used, it should be noted that they can only be applied to a single literal and not over several literals (see Example 4.9).

Example 4.9: Boolean use of *bif:contains*

If there was the following metadata record available that has *computer science* defined as the *subject* and *simulation* as the *mode*, “"computer science" AND simulation” would not lead to a result, because both terms are searched in the same literal instead of in different.

```
<ID>
  <subject> "computer science" ;
  <mode> "simulation" .
```

Theoretically, the user input could be separated at the operator AND and be used separately. The overall result would then be the combination of all single hits, but this raises, e.g., the question of how to handle the scoring. A hit in the same literal should be scored differently than a hit in two different literals. For this reason, this possibility is not analyzed further.

4.3 SPARQL Query Builder

As introduced in section 2.5 there are a lot of different query builders, which all work a bit differently or have a different focus. They will be evaluated in the work because they can provide the most optimal and direct representation in the knowledge graph by gradually assembling a SPARQL query. Important for the evaluation is the generated SPARQL query at the end, which has to be considered. For this reason, no SPARQL query builder is fully implemented but abstracted to a point that the actual SPARQL query is given. Furthermore, various such approaches have already been extensively evaluated [113, 65, 114].

4.4 Faceted Search

The approach of a faceted search is especially interesting because it allows for a clever refinement of data. The important thing about this approach is that a SPARQL query is generated that, by selecting facets and values, directly contains concrete URIs and data structures of the knowledge graph. Therefore, the resulting knowledge graph can be searched for specific content in the right place. For an implementation, further aspects have to be considered like which facets are offered to the user for restriction and how they are represented. In the case of CoScInE these would certainly be the metadata fields, but the question arises how exactly to name them and how to deal with different properties with the same name. For the comparison, the focus should not be on the implementation of the user interface, but on the result which can be achieved with such an approach. For this reason, only the SPARQL query generated by such an approach is considered again.

4.5 Using Elasticsearch as Search Engine

The last approach considered is based on creating a search index from the RDF graphs to be able to use existing search engines. As advantages for the search, it is possible to benefited from functionalities and optimizations for user input and search features. Examples are the handling of grammatical differences or spelling and the use of auto-completions or search suggestions. This is exactly why the focus of the work is on this approach. The possibilities that a search index provides for the user and generally the optimization of the search sound very promising and therefore shall be examined in this thesis. Positively evaluating this approach would therefore support the core hypothesis of the thesis (see section 1.5).

ES [115] could be used as a search engine because it is scalable and with its document-based approach it fits very well with the mapping of metadata records. It also provides a near-real-time search for different types of data. This meets the requirements of a structured search as well as advanced search types, e.g., range queries, as it should be possible in CoScInE.

The crucial question is how to transform the CoScInE knowledge graph into a search index. This is especially important because it is to be benefited the from structure and inference rules, i.e., from the advantages of the RDF model. The approach of *Linked swissbib* [90] precedes as a good example for mapping RDF data into a search index. However, it is outdated, since *ES* no longer supports the use of different types in a search index [116]. Furthermore, the pure use of the *JSON-LD* serialization in our context does not make sense, because otherwise URIs would be indexed for which the users do not search. For this reason, *Open Semantic Search* [117] additionally indexes the corresponding `rdfs:label` to a URI. Nevertheless, this is not sufficient since it is not always set or can be found in other properties depending on the data structure.

4.5.1 Transformation of a Metadata Record into an Elasticsearch Document

In *ES* a document consists of properties/fields and associated values. Furthermore, different data types like *booleans*, *integers*, *dates*, and *texts* are supported for the values. The used mapping of metadata records into the search index is oriented on the presented concept of Semplore [92] (see section 2.5). The differences are the concrete generation of the mapping, which in the considered approach can contain, for example, inference rules, and the possibility to use ES to benefit from advanced search types like value range queries. Resources (in the context of CoScInE only the metadata records) are considered as docu-

ments, the corresponding properties (metadata fields) as fields, and all objects as values. In principle, the *Entity Attribute-Value Model* and the corresponding *Search Model for Web Data* (see section 2.4.2 and 2.4.3) are applied here, where the metadata records are the entities. For all metadata fields, an appropriate value must be created to describe the corresponding field in ES. In case the object of a metadata record is a literal, the literal is taken as value. Again (as in full-text search), the literal rules are used for the objects that are not literals. According to the same principle, additionally generated triples can be stored in the ES document by using the additional rules. Thus, any arbitrarily complex graph is transformed into a flat object, which consists only of field-value(s) pairs, as specified in the following definition:

Definition 4.1: Mapping RDF metadata records into ES documents

Given is any metadata record G, which is an RDF graph. The root node is the ID of the metadata record, the edges correspond to the metadata fields and the objects to the values of these. For the RDF triple $\langle S, P, O\rangle$ of G, $S \in \boldsymbol{I}$, $P \in \boldsymbol{U}$ and $O \in (\boldsymbol{U} \cup \boldsymbol{L})$ applies, where $\boldsymbol{I}$ are IDs, $\boldsymbol{U}$ are URIs, and $\boldsymbol{L}$ are literals. G is extended by the knowledge graph with all additional information about the respective URIs. The mapping in an ES document is a tree G' with exactly height one. Formally, for each pair of triples $(\langle S'_i, P'_i, O'_i\rangle, \langle S'_j, P'_j, O'_j\rangle)$ from G' $S'_i = S'_j$ applies. For all triples $\langle S', P', O'\rangle$ of G', $S' \in \boldsymbol{I}$, $P' \in \boldsymbol{S}$, and $O' \in \boldsymbol{L} \cup \boldsymbol{L^n}$ is valid, where $\boldsymbol{S}$ are strings, and $\boldsymbol{L^n}$ are lists of literals.

To demonstrate the mapping Example 4.10 is considered. It is important that all URIs are transformed into literals or lists of literals during this transformation. For this reason, the mapping of Definition 4.1 is not injective. It may happen that two different RDF metadata records map to the same ES document as shown in Example 4.11.

Example 4.10: Mapping of an RDF metadata record into an ES document

Again, the example metadata record of File 3.2 from section 3.3 is considered. Furthermore, it is assumed that the following additional information is stored in the knowledge graph:

```
# additional information for subject
<http://www.dfg.de/.../faecher/#4>
  a <http://www.dfg.de/.../faecher/#4> ;
  rdfs:label "Engineering Sciences"@en ;
  rdfs:subClassOf <http://www.dfg.de/.../faecher/> .

<http://www.dfg.de/.../liste/index.jsp?id=409>
  a <http://www.dfg.de/.../liste/index.jsp?id=409> ;
  rdfs:label "Computer Science"@en ;
  rdfs:subClassOf <http://www.dfg.de/.../faecher/#4> .

<http://www.dfg.de/.../liste/index.jsp?id=409#409-02>
  a <http://www.dfg.de/.../liste/index.jsp?id=409#409-02> ;
  rdfs:label "Software Engineering and Programming Languages"@en ;
  rdfs:subClassOf <http://www.dfg.de/.../liste/index.jsp?id=409> .

# additional information for person
<https://www.rwth-aachen.de/>
  a org:FormalOrganization ;
  rdfs:label "RWTH Aachen University" ;
  org:hasUnit <https://www.rwth-aachen.de/22000> .

<https://www.rwth-aachen.de/22000>
```

Example 4.10 (continued)

```
  a org:OrganizationalUnit ;
  rdfs:label "IT Center" .

<http://dbpedia.org/resource/Thomas_Pynchon>
  a foaf:Person ;
  foaf:givenName "Thomas"@en ;
  foaf:familyName "Pynchon"@en ;
  foaf:name "Thomas Pynchon"@en .

[]
  a org:Membership ;
  org:member <http://dbpedia.org/resource/Thomas_Pynchon> ;
  org:organization <https://www.rwth-aachen.de/22000> .
```

If it is assumed that only the presented rules from the Examples 4.1 and 4.3 from section 4.1 and no other additional rules are applied, the following JSON mapping results:

```
{
  "creator": [
    "Thomas",
    "Pynchon",
    "Thomas Pynchon",
    "IT Center",
    "RWTH Aachen University"
  ],
  "worked": true,
  "title": "Design Patterns",
  "subject": [
    "Software Engineering and Programming Languages",
    "Computer Science",
    "Engineering Sciences"
  ],
  "created": 2020-01-01,
  "version": 3
}
```

Example 4.11: Mapping into an ES document is not injective

Given are the two RDF triples:

```
<http://www.example.de/subject#SoftwareEngineeringAndProgrammingLanguages>
  rdfs:label "Software Engineering and Programming Languages" .

<http://www.dfg.de/.../liste/index.jsp?id=409#409-02>
  rdfs:label "Software Engineering and Programming Languages" .
```

Assuming there are two metadata records which differ only in one metadata value (the two URIs) and no other rules are applied. The corresponding mapping could not be distinguished because the URIs are mapped to the same literal and thus the same ES document is created.

5 Evaluation and Results

First, this chapter presents the sample data generated for the evaluation and the search intentions that have been devised to evaluate the approaches listed in chapter 4. Then the test environment is described and the requirements and criteria presented in section 3.4 are considered.

5.1 Generated Metadata Records for Evaluation[1]

For the following evaluation of the different approaches, 10,000 exemplary metadata records were generated. All of them are based on a modified form of the EngMeta application profile. Some classes and data types of the profile have been modified to demonstrate the different possibilities that the SHACL definitions provide. Furthermore, the data does not represent real metadata of research data records but serves for illustration. Initially, five exemplary projects were generated, each with an associated resource. In addition, there are two different persons, the first of which is a member of projects 2 and 3 and the second of projects 2 and 5. Furthermore, projects 1 and 4 are each declared as public in order to map different visibility and authorization rules. When generating the data, all properties marked as mandatory fields were filled in and all others were randomly filled in or omitted. For the fields `dcterms:publisher` and `dcterms:creator` resources of the type `foaf:Agent` could be selected. For this, some organizations from the created "ror" vocabulary (`foaf:FormalOrganization`) or the personal data record (`foaf:Person`) of *DBpedia* [118] (joint project to extract structured content from Wikimedia projects) were available. In some cases also the `dcterms:creator` was taken over as `dcterms:publisher`. The `dcterms:subject`, which is an instance of the GRF subject classification, was chosen randomly. An element from *DBpedia* was chosen for the `dcterms:title`, which has a `rdfs:label` and a `dbo:abstract`. This abstract contains the individual words of the topic, in order to be chosen to be reasonably appropriate to the title. In addition, the abstract has been saved to enrich the data with free texts, which do not contain any research relevant data but can be used for the evaluation of search criteria. The `csmd:investigation_keyword` were also generated from the words of the selected title. For the two date fields `dcterms:created` and `dcterms:issued`, a date between today and the last 15 years was randomly selected. For the field `dcterms:issued`, today's date was selected with a higher probability. The date field `dcterms:available` was randomly assigned a date between today and two years from now, with today's date being preferred. For `engmeta:workednote` exemplary strings were available. For the `dcterms:type` the *DCMI Type Vocabulary* was used [119] for selection. For the `engmeta:version` any integer between 0 and 10 was chosen. For the metadata fields `engmeta:mode`, `engmeta:measuredVariable`, `engmeta:controlledVariable`, and `engmeta:step` the self created vocabularies of modes, variables, and steps were used. It should be noted that normally there would be no class instances for variables with a concrete value because otherwise there would have to be infinitely many instances for continuous values. For the field `dcterms:format`

[1] All data (metadata records, application profiles, vocabularies, etc.) used for the evaluation are published (see section G.6 "Published Research Data")

the *Media Types* [120] vocabulary is in use. The last field used in the evaluation is `dcterms:license`, for which the generated vocabulary from *SPDX License List* [99] was used.

As shown in the modified EngMeta application profile, a rule, which is specific to this profile, has been created to construct literals for instances of the variable class. This rule generates a string for value and unit (once with the symbol of the unit and once with the unit label) in addition to the name. Furthermore, the profile contains a rule for the additional generation of a field `coscinesearch:isNewestVersion`, which was already introduced in section 4.1. The general (i.e. used for all profiles) rules for the generation of literal values for classes and the general rules for additional triples can also be seen in the published data. The general additional rules are only relevant for the ES approach since they cover administrative information which is already contained in the knowledge graph of CoScInE.

In order to check whether all relevant metadata records have been found, the evaluation set is slightly restricted and only 35 metadata records are used. The focus is mainly on the five records which are also highlighted in the published data. During the selection process, care was taken to ensure that there are no duplications of the titles or abstracts, which cannot be completely ruled out in the large data record. Furthermore, the metadata records were selected in such a way that various features and characteristics of the different approaches can be identified.

5.2 Considered Search Queries for Evaluation

In the context of the evaluation and for testing the quality of the search results some search queries were considered. Behind each query is the intention to find one or more data records (which are described by metadata). They were selected to cover as many different search types as possible and to be related to the RDM. In the examples the user wants to find or get information about the following things:

- Data records which were published at the IT Center
- Data records with version number 10
- Data records which were created in 2007
- Data records about design patterns and with version number less than 5
- Data records about computer science
- The newest data record of resource 2
- Data records which are experiments or simulations and not analysis
- Data records about non computability of the human consciousness
- Data records which are available since 03.07.2020
- Data records about political left
- Data records which contain a variable with the value 2 meters
- Data records about object-oriented software which are published before 2015
- Data records which are published by Thomas Pynchon, created in 2016 and about object-oriented software

Depending on the approach, this intention has to be formulated differently as a request, because a certain syntax has to be followed. For the tests it is assumed that the user does not make any mistake when translating the intention into the query. Furthermore, the search query is formulated in such a way that it is at least well-fitted or close to optimal for the corresponding approach. This means that the intention “published on IT Center”,

e.g., in the query is only translated to “IT Center” if it is known that properties’ labels (in this case “published”) are not searchable in the approach. Also, it is assumed that the user searches for the correct words, i.e. those that are present in the text and not for variants or synonyms of them.

5.3 Test Environment

For the evaluation, the approaches presented in chapter 4 were implemented or an environment for testing them was created. *.NET Framework 4.8* was used with the programming language *C#* and the library *dotNetRDF* [121] was included in the version 2.6. As RDF database *Virtuoso* in the open-source variant *7.20.3217* with the default settings was applied. Only the parameter *NumberOfBuffers* was changed to 660000 and *MaxDirtyBuffers* to 495000 in the *virtuoso.ini* file. For the implementation ES version *7.6.2* was used. In addition, all default settings have been adopted, i.e., in particular only one *node*, one *cluster*, one *shard*, and one *replica* have been created. In order to keep the conditions, especially for the measurement of times and their comparison, as similar as possible, the tests were carried out under the same conditions and the same hardware. A 64-bit system with the operating system *Microsoft Windows Server 2016 Standard*, 16GB RAM and two *Intel(R) Xeon(R) CPU E5-2695 v3 @ 2.29GHz* processors was used.

Test Environment for Full-Text Search in RDF Literals

For the two variants of the full-text search, all rules used for the evaluation were mapped into a SPARQL query according to the principle of the Examples 4.6 and 4.7. Since each rule is composed in the same way, the query could be generated automatically. Only administrative information or boolean specifications were used as additional rules, so they were omitted in the generated SPARQL query, although the *regex* approach supports them in principle. The queries were executed using the *VirtuosoManager*, a class for accessing the Native Virtuoso Quad Store. To avoid the user having to specify complicated regular expressions for the *regex* approach when executing boolean queries, the input is processed before execution. Words separated by spaces are automatically translated into a regular expression that links them with the logical operation `OR`. Text written in quotation marks is interpreted as one word. It would also be conceivable to link the words with `AND` instead. Because of the well-known Google search, which interprets keywords first with the logical operation `OR`, this variant was chosen. An arbitrary mixture of `AND/OR` was omitted to minimize the complexity of the regular expression.

Test Environment for SPARQL Query Builder

For testing the SPARQL query builder approach, SPARQL queries were also executed using the *VirtuosoManager*. The creation of the queries was based on the possibilities offered by WQS. WQS was selected because Kuric et al. [65] emphasized it as one of the most promising SPARQL query builders. It offers the possibility to select triple patterns via “filter” dropdowns, to determine which information is displayed and to specify a limit. For the evaluation, it is necessary to display all results and only metadata records, what WQS supports. The interface does not support SPARQL filters like date restrictions or to check if a label contains a word. Furthermore, it does not provide support for sorting the results. This means that these options have to be manually entered into the query based on the existing examples provided by WQS. Thus, it is assumed that they are possible but that it requires a search and a copy operation. *SPARKLIS* [75] another well-known approach according to Kuric et al. [65] shows that such filters can be clicked together

with appropriately selected controls, but that they require more effort than simple triple patterns.

Test Environment for Faceted Search

The faceted search approach is divided into two steps and is also based on the functionalities of known examples from section 2.5. First, a full-text search (*regex* approach) is used to limit the search space. In the context of CoScInE, this means a restriction to a few metadata records. For these, different facets and their values are then queried, which are given to the user for selection. However, only those facets are displayed that have URIs, integers or dates (whereby for reasons of clarity only the year is used) as objects, to restrict the value range and offer the user concrete values to choose from. The user now can further restrict a full-text search via the facets by clicking on specific values. The various metadata fields with the values clicked on are linked by `AND`. If several values are clicked for a metadata field, they are linked with the logical operation `OR`. In addition to that, like in the *SemFacet* [85] approach, a field "ANY" is offered which can be selected and serves as a variable, which can be restricted by further attributes. Through the facets, concrete URIs are known, which contain the previous SPARQL query and can be added to the full-text search. As before, all SPARQL queries are executed via the *VirtuosoManager*.

Test Environment for Using Elasticsearch as Search Engine

This approach has been fully implemented in the master thesis because it is identified as the most promising approach. Within this test environement, the hypothesis is tested: *Mapping RDF-based metadata records into a search index for use in a search engine improves the quality of the search and results for the user compared to using generated SPARQL queries.* In ES different indices can be created, which can be searched individually or together. Each index is independent and stores documents. A data type can be defined for each field. This data type must not change within an index. In addition, different *analyzer* and *tokenizer* can be used per index to index the data. There are also language-specific analyzers available. For example, they can remove stop words (words that are not considered during indexing, e.g., "the") of the corresponding language and apply *stemming*. Stemming [122] is a procedure to reduce all words with the same root to a common form. It is used to match variants of a word during a search from which the quality of a search result benefits.

However, it must be ensured that the language is taken into account. For this reason, an index is created for each language, which applies the corresponding analyzers and filters. A search can also be performed in the language-specific index. Theoretically, it is also possible to search in several indices at the same time to enable a language-independent search, which at the same time respects language-specific indexing. For the transformation of the RDF-based metadata record, it is important to note that the corresponding literals of the desired language are used. Here, the same principle applies that when executing SPARQL queries or rules, the language is filtered accordingly via the language tag. Using a function, it can be specified how the ranking of the results is calculated. It was decided to use the default variants, which are defined in the ES documentation [115]. Altogether, ES offers many different *analyzer*, *tokenizer*, and *settings*. This work was limited to the presented variant but the analysis of different approaches and settings could also be interesting and optimize the search.

There are four use cases for implementing the search with a search engine like ES. All of them are used to keep the data of the RDF database and the search index in sync

so that queries to the search index also reflect the correct data of the RDF database. Their significance and procedure are discussed below.

1. Initialize Index
The first use case is the creation of an index with the associated indexing of all existing metadata. This task is only necessary once when the search index is empty and the RDF database already has metadata records. Beside general settings, such as the specification of the indexing and search analyzer, the main task is to identify all properties and their names as well as the data types based on the application profiles. The names of the properties are generated automatically: First, it is checked whether there is an `rdfs:label` for the property in the knowledge graph. Next, the name is guessed based on the URI and as of the last attempt the name of the property from an application profile is used. If the name consists of several words, they are all connected with underscores and used in lowercase. If still no name could be generated for the property this way, it is skipped and cannot be indexed. This is not problematic because the search index is only used for the search and the consistency and completeness of the data must only be maintained in the used RDF triplestore. Ideally, the same properties are defined in different application profiles with the same data type or are described using instances of one class. In this case, they can receive exactly this defined data type as type in ES (*date*, *text*, *boolean* or *integer*) or instances of a class are always mapped as *text*. If this is not the case, the property is also mapped to the type *text*, because as mentioned before the same property in ES may only have one unique type per index. Table 5.1 shows all possible mappings of data types between the existing application profiles and ES.

Table 5.1: Data types mapping between application profiles and ES where * means that the same properties were described in different profiles with the same data type.

Application Profile	**ES**
date*	date
string*	text
boolean*	boolean
integer*	integer
class*	text
different data types	text

2. Create, Update or Delete Metadata
If a new metadata record is created, it must also be added to the ES index. For this, the mapping described in section 4.5.1 and literal rules (see section 4.1) are used. Properties of type date and boolean are special cases. Since it is not possible to search a date field in ES only by a year, three additional fields for day, month, and year are stored for each date, whereby the month is represented as a string. Boolean properties are additionally stored in a written variant (see Example 5.1), because the labels of properties are not searchable within the simple search syntax and a pure search for a boolean value does not add value.

Example 5.1: Written variant of boolean property `engmeta:worked`

Given are the following RDF triples:

```
<ID> engmeta:worked true .
```

Example 5.1 (continued)

```
engmeta:worked rdfs:label "worked"@en .
```

The triple is mapped to the following key-value pairs for the ES document:

```
{
  "worked": true,
  "worked_written": "worked true"
}
```

The values of the mapping are based on the previously defined types from the mapping of the index. It is possible to define general rules for each class or individual rules for each application profile. When applying these rules, the first step is to look at the direct classes of an instance (which is specified as the value of a property). If there is no rule for this class that matches the application profile used, the system checks whether there is a general rule for this class and executes it. This is done for all direct classes. If no rules could be found, the parent classes of these classes are looked at, which continues recursively until there is no more parent class and the root node is reached which applies the simple rule to use the `rdfs:label` of the instance (see Example 5.2).

Example 5.2: Literal rule for root class

```
coscinesearch:rootClass
  sh:rule [
    a sh:SPARQLRule ;
    rdfs:label "Infer literal by rdfs:label for the root class" ;
    sh:prefixes prefixes:prefixes ;
    sh:construct """
      CONSTRUCT {
        $this rdfs:label ?label
      }
      WHERE {
        $this rdfs:label ?label
      }
    """ ;
  ] .
```

In case no rules are defined for a class and the rule of the root node does not produce any literals either, the URI is used again to guess a literal. Otherwise the corresponding triple is excluded from indexing.

Moreover, specific additional triples as well as some general administrative properties are generated for each metadata record through the use of general and specific additional rules (see section 4.1). Examples for general administrative properties are the project affiliation to map visibility as well as the graph name to identify the associated RDF metadata graph in the triplestore. The ID of the graph cannot be used as the ID of a document due to the specifications of ES, so it must be saved as a field to be able to retrieve the actual metadata record later. The additional rule with the newest version (see Example 4.1) also contains the special feature that not only a property is created for the newly added metadata record, but also existing documents (mapping of other metadata records in ES) are modified. The reason for this is that adding a metadata record with a newer version will result in the previous most recent version no longer being the most current one, resulting in a change in that set. This means that changing, adding or deleting a metadata record can always affect other documents, so the additional rules of such a profile must always be executed.

Special attention must be paid to the deletion process. If the record is deleted directly from the RDF database, it is no longer possible to conclude the possibly modified current version of a resource. If the graph is deleted afterwards and the rule is applied, the version of the deleted metadata record is still considered. To prevent this, a metadata record is marked as deleted (`coscinesearch:isDeleted`) and the rule is extended by the following filter: `FILTER NOT EXISTS { ?file coscinesearch:isDeleted true }`. This allows drawing conclusions about the affected resource and still exclude the deleted metadata record when the rule is executed. Once the rule has been applied, the metadata record can be permanently deleted.

3. (Full) Re-Index
Re-Indexing is necessary if the existing application profiles and thus the already indexed data or their mappings change. In this case, a new index is created using the new application profiles and then all existing data is re-indexed. This can take some time despite using the bulk request of ES. To avoid downtime during the search, the previous index remains in place and is used for the search until the new index is fully indexed with all old documents. For this purpose aliases are used which point to an index and are added or deleted accordingly. There are some cases where a complete re-indexing can be avoided. However, this assumes that the change made is known. Although existing properties cannot be changed in ES, others can be added dynamically. Hence, if the change is only to add more properties, they can simply be added in the ES mapping. The same applies to the addition of rules for generating additional triples. On the other hand, if an existing rule for the creation of new triples or an existing property is changed, a new triple must be initialized. However, if a completely new indexing is performed, it is not necessary to take care that a document changes a property of another document. This is because all information is already available in the triplestore and is therefore taken into account when passing through each metadata record.

4. Search
To define queries, ES provides a full Query Domain Specific Language (DSL) based on JSON. For full-text queries, ES offers two different search syntaxes to directly convert user inputs into a corresponding search query. There is a simpler variant in which syntactically incorrect input is ignored and all remaining interpretable queries are executed. The second variant offers the user further functionalities but is strict in contrast to the simpler variant and does not return partial results in case of incorrect input. The user can choose the preferred search option via a flag. Moreover, the created field names (metadata fields of all application profiles) are made available to the user to enable searching in concrete fields in order to refine searching and enable a structured search.

Figure 5.1 shows the rough class diagram of the developed approach (see section G.6 “Published Research Data”). The *RdfSearchMapper* contains the main logic to perform the four different use cases for mapping RDF graphs in ES documents. It uses an instance of the *ElasticsearchSearchClient* to execute the requests against ES. The interface *ISearchClient* allows exchanging the search engine. Furthermore, the *RdfGraphMapper* uses the *RdfClient*, which is for querying RDF specific data. It connects to the triplestore via the interface *IRdfConnector*, in this case the *VirtuosoRdfConnector*. The *DataTypeParser* serves to parse the data according to its data type or class. Additionally, the *SpecificApplicationProfile* is used to execute the corresponding literal and additional rules for an application profile. Since there are also general rules, this class extends the *Profile*.

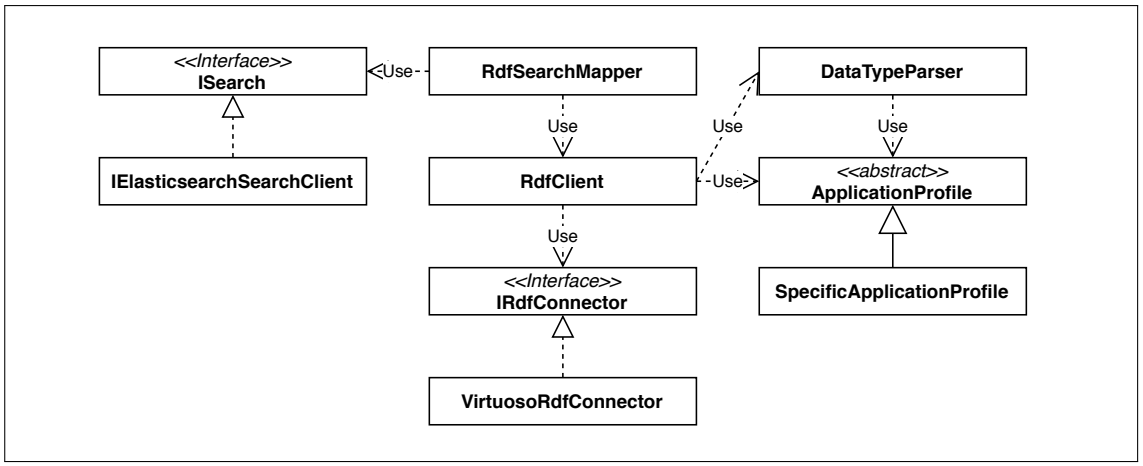


Figure 5.1: Class diagram for implementing the ES approach

5.4 Evaluation[1]

This section contains the evaluation of the criteria presented in section 3.4. First, a short explanation of the procedure or method of measuring the criterion is given and then the approaches presented in chapter 4 are considered under these aspects. The following results are based on the assumptions made in the context of the thesis, the presented test environment or implementation, and the generated sample data.

5.4.1 General Conditions

Of the three aspects presented in section 3.4.1 the *user background* condition is directly fulfilled due to the choice of approaches. Even if the approaches are differently complex for the users, they do not need to be familiar with SPARQL. In contrast, the categories *visibility* and *language* influence the implementation of the search.

General Conditions for SPARQL Queries

For all approaches where a SPARQL query is generated(*regex*, *bif:contains*, SPARQL query builder, and faceted search), the implementation of the visibility and language constraint is identical. For the *visibility* the SPARQL query is completed with the *triple pattern* from File 5.1. Since the data on the project structure and visibility is also stored in the triplestore, a corresponding filter must be implemented that only allows the return of public data or data on a user's projects. The list of projects is accordingly replaced with the valid project URIs, which were previously queried for a person via SPARQL query.

```
?s coscineprojectstructure:isFileOf ?resource .
?resource coscineprojectstructure:isResourceOf ?project .
?project coscineprojectstructure:isPublic ?public .
FILTER (?public = true || ?project IN ('{project1}', '{project2}')) .
```

File 5.1: Visibility supplement for *regex* and *bif:contains* SPARQL query

Regarding the *language* condition, the SPARQL language tag is appended to the literals of the query, which allows that only literals of a certain language are searched. Since not

[1] In the following, the SPARQL query builder is abbreviated to query builder in some places.

for all literals in the knowledge graph language tags are always present, the search is also done in the literal without language tag. The procedure can be seen in File 5.2 for English.

```
SELECT ?s {
  ?s ?p ?label .
  FILTER (lang(?label) = 'en' || lang(?label) = '') .
}
```

File 5.2: Language tag supplement for *regex* and *bif:contains* SPARQL query

If this filter is omitted, the entire knowledge graph is searched independently of the language. Whether it makes sense to limit the search to one language, because it might reduce the query time, is another question, which will not be discussed here. However, Kurz [123] describes that filters reduce costs, so limiting the search to one language sounds promising.

General Conditions for Using Elasticsearch as Search Engine

For the implementation of visibility in ES, a filter can be attached to the two presented search variants (see section 5.3). The filter allows only public or the project's own metadata records and is not included in the scoring of ES. This filter can be seen in File 5.3.

```
{
  "bool": {
    "filter": {
      "bool": {
        "must": {
          "bool": {
            "should": {
              "terms": {
                "belongsToProject": [
                  "{project1}",
                  "{project2}"
                ]
              },
              "term": {
                "isPublic": true
              }
            }
          }
        }
      }
    }
  }
}
```

File 5.3: Visibility supplement for ES search query

Using the filter function in the outer *bool* function ensures that the filter is not included in the scoring. Afterwards, a boolean query is generated because the metadata records must be either public or belong to the corresponding projects. ES allows to specify conditions in `must` and `should`. All terms which are linked with the logical operation `OR`, in this

case `coscinesearch:isPublic` or `coscinesearch:belongsToProject`, must be included in the `should` statment. However, this statement must be stored in a `must` statement, because one of both must be fulfilled. The fields `coscinesearch:isPublic` and `coscinesearch:belongsToProject` were generated as general additional rules for each metadata record.

Comparison of the General Conditions

Due to the presented possibilities to map and include the *visibility* and the *language*, all considered approaches fulfill the general conditions and are therefore suitable for the implementation of the search.

5.4.2 Effectiveness

To compare the effectiveness of the different approaches, the search intentions presented in section 5.2 were considered. First, the search inputs for each approach were created and then executed in the corresponding English language approach on 35 selected metadata records (see section G.6 "Published Research Data"). To get a better overview of the data and possibilities, the *visibility* filter was omitted. To see which metadata records are relevant, the query was translated as a guideline into a pure SPARQL query (see section G.1) considerung the existing structure and data. A full-text search, i.e., if searching in strings and no corresponding URI exists, is only possible by using *regex* or *bif:contains*. Based on the results of the query it was decided which records should be found for the respective search intention. Afterwards, confusion matrices were set up, which compare the relevant search results with the found ones and thus make a concrete analysis possible. The approach-specific search inputs and all confusion matrices can be found in "Appendix" G.2. Table 5.2 contains all summed up matrices, which add up the results of the different search intentions.

In the following the observations for each approach are briefly explained. It should be mentioned in advance that the search intention "left (political)" will be dealt with in the comparison at the end since for all approaches two irrelevant records were found for this query. Furthermore, there are no results for the search intention "published before 2015 and about object-oriented software".

Table 5.2: Summed up confusion matrices

		Relevant	
Received		+	-
	+	51	34
	-	0	370

Regex

		Relevant	
Received		+	-
	+	34	10
	-	17	394

Bif:contains

		Relevant	
Received		+	-
	+	51	2
	-	0	402

SPARQL Query Builder

		Relevant	
Received		+	-
	+	46	6
	-	5	398

Faceted Search

		Relevant	
Received		+	-
	+	50	26
	-	1	378

ES Simple

		Relevant	
Received		+	-
	+	51	2
	-	0	402

ES Advanced

Effectiveness of Full-Text Search in RDF Literals

Since the syntax of *regex* and *bif:contains* is different, they are considered one after the other. The confusion matrix of *regex* shows that the approach finds all relevant metadata records in any case. Besides, it also finds many other documents that are not relevant to the search intention. As already mentioned, it is not possible to find combinations in different fields or to limit the search to specific fields. Since the different terms are linked with the logical operation `OR`, further non-relevant records are found. Here it might be considered to limit the search to the most specific search term to minimize the search results. However, this criterion depends on the existing database and is therefore not known to the user. More complex queries that refer to other metadata records, such as the search for the most recent metadata record, are not possible due to the independence of the search from other records. Also, only those queries that can be expressed in the form of a regular expression are possible. The search for records before 2015 is difficult because although it is possible to search for numbers such as X-2014, the question is where to draw the line to not find irrelevant data since the search is not limited to date fields. A similar problem is a general search for numbers. If the version number were not a pure integer field, it could not be found using the regular expression "^10$". If only "10" would be searched for, a lot of irrelevant records would be found, because then, for example, date fields are also looked at. If there were more integer fields, which have the same value, several irrelevant records would be found.

With the *bif:contains* variant, as with the *regex* approach, it is not possible to search on concrete fields or multiple fields. In this case, the different terms are linked, which can lead to finding irrelevant records. Due to the underlying database this happens only with the last search intention. As already mentioned in the *regex* variant, it might be considered to limit the search to the most specific keyword to minimize the search results. Furthermore, the search is also independent of other data records. Most noticeably, no search is possible in numbers, dates, and in compound literals. As a result, the confusion matrix shows that some relevant documents cannot be found. At the same time, a reduced number of irrelevant records is found. The possibility to cover value ranges or similar is also completely omitted.

Effectiveness of SPARQL Query Builder

The confusion matrix shows that the SPARQL query builder gives very good results regarding the effectiveness. Only two irrelevant and all relevant documents were found. Since this approach allows the arbitrary assembly of SPARQL queries, more complex queries like the latest version as well as queries concerning different metadata fields of a record can be mapped. However, this only applies to the extent that corresponding functionalities are provided in the user interface or a SPARQL endpoint with examples is available for orientation.

Effectiveness of Faceted Search

The faceted search consists of two steps. It was decided to implement the first step with the *regex* approach because it has a recall of 1 and consequently will find relevant documents in any case and can restrict the non-relevant one by using the facets. Therefore, the first step does not differ from the *regex* approach. For the second step, the SPARQL query generated by the *regex* approach is refined by the selection of the user.

The faceted search also performs well within the scope of the data and the queries presented. Except for five documents all relevant documents could be found and only six

irrelevant. The disadvantages of this approach are that only years can be selected when selecting the facets of date fields and that text fields cannot be displayed and further restricted. For this reason, the approach benefits when many metadata fields in the SHACL profiles use classes and controlled vocabularies. The problem occurs, for example, in the considered search intentions when searching for data records about design patterns with a version number less than 5. In the data and concerning the previous restriction via the regex full-text search, the version number 3 can be clicked in the facets. However, now the version number is mandatory and therefore fulfills the logical `OR` operation without containing the words "design patterns", which is why an irrelevant document is found. Another disadvantage is that the faceted search, as described in section 5.3, does not allow any logical `OR` operation between different metadata fields, which is why the five documents were not found when asking for variables with two meters. It is possible to ask for a variable with the value "2 meters", but not the `engmeta:measuredVariable` `OR` the `engmeta:controlledVariable` can be selected. Since both were selected, but only one of them delivers the corresponding value, they could not be found. If only one of the two variables metadata fields had been selected, records would have been found. However, here again, the question arises, which one should be selected best. But this shows that it would be possible to find the correct data records in two steps: once via the `engmeta:controlledVariable` and once via the `engmeta:measuredVariable` metadata field. As with the SPARQL query builder, one advantage is the ability to query in concrete metadata and across multiple fields. Regarding the example of object-oriented software before 2015, the first limitation by regex returns only a document created after 2015. For this reason, the user cannot select any fields smaller than 2015 in the user interface but knows without a further search that no matching metadata records exist.

Effectiveness of Using Elasticsearch as Search Engine

Since the syntax for ES differs depending on the selected search variant (see section 4.5), they are considered one after the other. For the simple variant of the ES approach, all relevant data except one are found. This is because boolean queries refer to the whole data record and not to single fields. In the search intention "experiment or simulation and not analysis" exactly the problem occurs that not only the metadata field `engmeta:mode` is considered, but also the other fields. The record that was not found contains the word "analysis" in the `dcterms:abstract`, so it is excluded. Furthermore, some irrelevant records are found. This is because it is not possible to search in concrete fields or value ranges or similar. It is noticeable that the search for "IT Center" found two more irrelevant metadata records. This is because the word "it" counts as a stop word and thus only the word "center" is searched for. It should also be noted that this approach only finds the most recent version of resource 2, because it was decided to put this information in a seperate field during implementation. Without this modification this query would not be possible. The same applies to searching for concrete years. Because the evaluation also uses the day of a date as a field, a lot of irrelevant data is found when searching for a number (see query "10"). Here it might be considered that only month and year are mapped to improve the effectiveness considerably, because it is rather unlikely that a user will search for the day of a date only. An important advantage, however, is that the results are ranked according to their relevance and queries can be mapped across multiple fields.

In the advanced variant of the ES approach all relevant records were found and only the two records each approach found were irrelevant. The advantages of this approach are that a search can be done in concrete fields, any combination of fields is possible and specification of value ranges is allowed. Furthermore, the year and boolean

values (due to the search in concrete fields) can be found in contrast to the simple variant without additional created fields. Besides, a ranking of the found records is also available. As with the simple variant (possibly limited to certain fields only), additional metadata records could be found for specific phrases containing stop words. This is not shown in the example queries or data but would be a disadvantage. A big difference between ES and the other approaches is that in ES queries are searched document based and the other data only allows searching in single fields.

Comparison of the Effectiveness

As mentioned in the beginning, all approaches for the search intention "left (political)" find two irrelevant records. This is because a pure search for "left" leads to the fact that in addition to the political left, the left side or the past tense of the word "leave" can also be meant. For a computer, this semantic difference is not recognizable. However, in every approach the search could be extended by the word "political" or a wildcard variant "politic*" and would in this case (except for the *regex* approach by the implementation with the logical operation `OR`) only find the one relevant data record. However, this requires that the word also occurs in the same string. If this is not the case, it is not possible to distinguish between them. If the topic would be political science or similar (which is not the case in the data), the search could also be further limited by this information, but this would not work for the two full-text approaches *regex* and *bif:contains*.

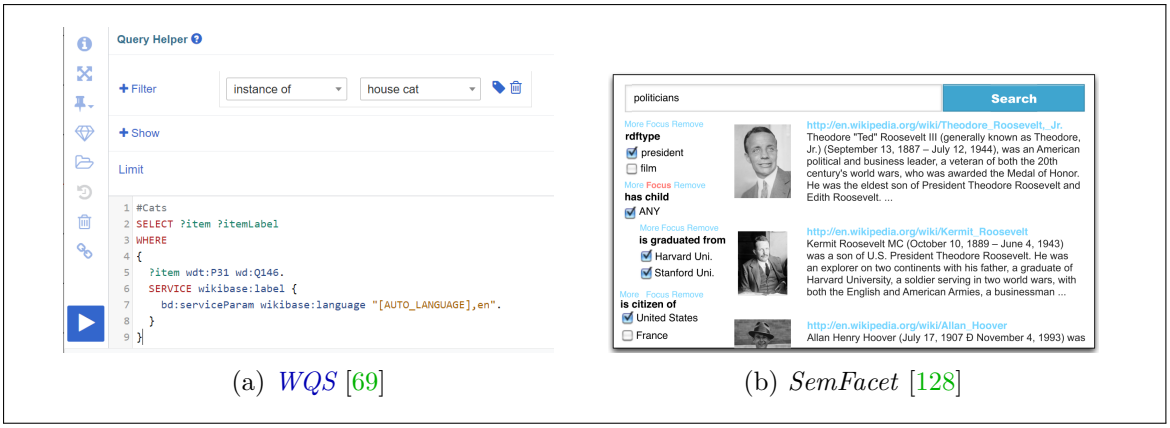
Table 5.3: Precision, recall, and F1 score (rounded to two digits)

	Regex	**Bif:contains**	**SPARQL Query Builder**	**Faceted Search**	**ES Simple**	**ES Advanced**
Preciscion	0,6	0,77	0,96	0,88	0,66	0,96
Recall	1	0,67	1	0,9	0,98	1
F1 Score	0,75	0,72	0,98	0,89	0,79	0,98

Table 5.3 summarizes the results by calculating precision, recall, and F1 score of the total confusion matrices of the respective approaches. The advanced variant of the ES approach and the SPARQL query builder perform best in terms of all three categories regarding effectiveness, as they are the most expressive.

5.4.3 Usability

In the context of the master thesis, it is sufficient to roughly estimate the usability of the approaches. For this reason, no formal user tests were made. In section 3.4 it has already been described that for the usability it is decisive whether the search interface is simple and understandable for the user. It is difficult to accurately evaluate or measure a User Interface (UI). However, there are many design principles and criteria that have proven to be useful in the past such as feedback and natural mapping [124]. Wilson [125] has worked out ten design principles especially for search interfaces. Most of them refer to additional options and functionalities for the creation of a search interface, which are not the focus of this thesis and for which approaches could be implemented identically. To differentiate between the approaches the principle *aesthetics and minimalism* is suitable, which says that a clean and clear design is important, as, e.g., the Google search bar is a well-known and familiar example. Furthermore, the *task execution time* was considered, which is estimated by means of the *Keystroke-level Model (KLM)*. This model is the simplest of the Goals, Operators, Methods, and Selection rules (GOMS) family [126].



(a) *WQS* [69] (b) *SemFacet* [128]

Figure 5.2: User Interfaces of *WQS* and *SemFacet*

The KLM [127] describes various actions with their associated physical operations and specifies an execution time for them. Thus, tasks can be converted into operators and the execution time can be added up.

Consideration of the User Interface to Determine the Usability

The different approaches require different UIs. The two full-text searches as well as the simple ES variant each consist of a simple search bar known from Google. In the advanced ES version, an overview of existing properties is added to this search bar. As described in section 5.3, the SPARQL query builder is based on the UI of the *WQS*. Figure 5.2a shows the corresponding interface. For the faceted search, the *SemFacet* approach was mainly used. Figure 5.2b shows an excerpt of the corresponding UI.

Based on these descriptions, the aesthetics and minimalism of the approaches can be evaluated directly (see Table 5.4). Within this process, three gradations were subdivided . The division of the faceted search into the lowest category is due to the fact that

Table 5.4: Aesthetics and minimalism ($\surd$ = well, $\sim$ = medium, $-$ = bad)

Regex	Bif:contains	SPARQL Query Builder	Faceted Search	ES Simple	ES Advanced
$\surd$	$\surd$	$-$	$-$	$\surd$	$\sim$

with arbitrarily large data records or if the first step does not bring a large restriction of the data records, the navigation becomes very unclear because there are many options. But here it might be considered to use dropdowns instead of checkboxes. The SPARQL query builder also falls into the worst category, since many controls must first be selected and operated for the search. Kuric et al. [65] have also tested the usability of different SPARQL query builders and found that all problems in this category occurred.

Consideration of the Task Execution Time to Determine the Usability

For the application of the KLM to calculate the task execution time the presented search intentions from section 5.2 were analyzed for the individual approaches. It can be assumed that the interfaces are implemented as described above. Furthermore, only physical operators with their respective times from Table 5.5 are considered, because the mental operators are described separately in the section 5.4.4 “Complexity of Search Request for the User”.

Table 5.5: Operators with time in s used for the KLM [127, 129]

Operator	Description	Time
K	pressing a key	0.28
P	point with mouse to a target on the display	1.1
BB	click mouse button	0.2
H	home hands to keyboard or mouse	0.4
CP	copy and paste	4.51
PDL	pull-down list	3.04

In addition, it is assumed that the user immediately finds the example during copy and paste operations for the SPARQL query builder approach, i.e., the search time for this is omitted. The same applies to scrolling a page and searching for specific fields or properties. It is also important that the processes are always seen in the context of the database and environment. This means, for example, for the faceted search, that only those facets and values are selectable that are possible by restricting the first step. It is assumed that the user must always first move his hands to the keyboard (H) and then aim at an element of the UI (P). After that, the SPARQL query builder requires the selection of elements from the dropdowns (PDL) and the application of copy operations (CP), which still need to be revised (H, P, BB, K). It is assumed that the copy and paste operation does not require switching from keyboard to mouse or vice versa. At the end of the input, the search is executed by clicking the search button (H-P-BB). For the advanced ES approach, depending on the intention, the corresponding properties, which are used in the input, must first be read from a dropdown (PDL). If several properties are used, multiple drop-down selections can be made without any additional pointing in between, since the same UI element is used. For the remaining approaches, the search bar is clicked (BB) and a string is entered (n*K), which is confirmed with pressing Enter (K) and triggers the search. For the faceted search, the hand must be moved to the mouse (H), one of the values of a facet must be targeted (P) and clicked (BB). Example 5.3 serves to explain the procedure using a concrete example. The assumed steps for the remaining search queries and approaches are listed in "Appendix" Table G.12.

Example 5.3: Application of the KLM

In the following example, the operators that a user must perform in the SPARQL query builder to execute the search intention "2007 created" are considered. First, the user has to move his hand to the mouse (H) and aim at a drop down field (P). Then the user selects the property "date created" from this field (PDL). Since this date is to be filtered to the year 2007, the user has to find out in the examples (P) of how this works and copy the corresponding lines (CP). Afterwards, the user has to select (P) and click (BB) on the comparison operator which indicates the lower limit. Subsequently, the user has to move his hands to the keyboard (H) to change it to ">=" (2K). Afterwards, the same procedure is done to click on the first date limit (H-P-BB-H) and enter the lower limit "2007-01-01" (10K). The user repeats the last steps exactly for the input of the comparison operator "<", this time consisting of only one key click and for the upper limit "2008-01-01". Finally, he moves his hands back to the mouse (H), aims at the search button (P) to press it (BB), and thus execute the search query. The resulting operators of this search query are the following: H-P-PDL-P-CP-P-BB-H-2K-H-P-BB-H-10K-H-P-BB-H-K-H-P-BB-H-10K-H-P-BB

Table 5.6: Calculated task execution time in s

Intention	Regex	Bif:contains	Query Builder	Faceted Search[1]	ES Simple	ES Advanced
Published at IT Center	5.46	5.46	33.62	3.0	5.46	12.4
Version 10	3.5	3.5	8.5	1.7	2.94	9.32
Created in 2007	3.5	4.06	26.29	1.7	3.5	17.44
Design Patterns and Version < 5	9.38	7.14	52.94	1.7	7.14	15.48
Computer Science	7.42	7.42	38.49	-	7.42	7.42
Newest Version Res. 2	5.74	5.74	32.06	1.7	19.64	16.38
Experi./Simu. not Anal.	14.42	4.06	49.16	3.0	11.34	20.52
Non Comput. Human Con.	12.74	13.86	25.03	-	13.3	21.08
Av. s. 03.07.20	5.18	5.74	15.78	1.7	5.18	13.52
Left (political)	3.5	3.5	14.67	-	3.5	3.5
2 Meters	4.9	4.9	55.81	8.2	4.9	26.08
Pub. < 2015 Oo. Softw.	9.66	9.66	37.86	-	9.66	28.32
Pub. T. Pyn. Created 2016 Oo. softw.	15.82	17.5	53.7	3.0	16.66	38.64
∅	7.79	7.12	34.15	2.86	8.51	17.7

Table 5.6 contains the calculated times for the task execution time and the average to show which approach can be executed fastest by the user. The *bif:contains* approach can be executed the quickest, closely followed by *regex* and the simple ES variant. This is followed by the faceted search. It is noticeable that the extended ES approach is at the second last place, although it consists of a simple search bar. This is because the user has to search the fields first and the search input is lengthened by the additional specification of fields and operators. The SPARQL query builder comes off worst, which also fits its complex UI.

Comparison of the Usability

If the usability is summarized using *aesthetics and minimalism* and *task execution time*, the approaches *regex*, *bif:contains* and the simple ES variant perform best, because their estimated execution time is the fastest and they also have a very simple UI. After that the extended ES variant and the faceted search follow. The advanced ES variant is in the middle range for both criteria and the faceted search has a fast task execution time but a

[1]Only the time of the second step is calculated. For the first step, the same time is valid as for the respective search intention in the regex approach.

bad UI. The SPARQL query builder comes off worst regarding *aesthetics and minimalism* and *task execution time*.

5.4.4 Complexity of Search Request for the User

Table 5.7 shows an overview that lists the user's required knowledge to perform the search queries. For the SPARQL query builder, SPARQL is shown as knowledge in brackets, because it does not actually require any specific knowledge. Nonetheless, by copying parts of queries and inserting a query directly into an endpoint, it is necessary or helpful to have a basic knowledge. In the faceted search, regular expressions are in brackets, because the first step is regulated by the *regex* approach. However, it is not necessary to enter regular expressions, since the search is specified more precisely by the facets anyway.

Table 5.7: Knowledge user needs to perform search requests

Regex	**Bif:contains**	**SPARQL Query Builder**	**Faceted Search**	**ES Simple**	**ES Advanced**
Regular expressions	*Bif:contains* operators	(SPARQL)	(Regular expressions)	ES simple search syntax	ES advanced search syntax

In order to evaluate the complexity of the search syntax and the effort for the user to perform the query, the individual search queries were considered per approach and classified into three categories (see Table 5.8). Afterwards, the most often selected category was used as an indicator for a final comparison of the complexity. Initially, all search queries are classified into the simple category, which only requires keywords to be entered or selections to be made in certain fields.

In the *regex* approach a query is classified as partially complex if a regular expression is used that goes beyond the simple listing of words. In the *bif:contains* approach, a query is considered partially complex if date fields or integer values unexpectedly need to be quoted due to the *bif:contains* syntax. In a faceted search, an approach is classified as partially complex if, in addition to the simple selection of values, it must be taken into account that this value may occur in different facets. This is the case, for example, when selecting the two facets *controlledVariables* and *measuredVariables* when searching for metadata records with a variable value of 2 meters. In addition, such queries are classified as partially complex, where the user must have additional knowledge about the content, e.g., within the search intention for *publications of the IT Center*. In this case, the user must also select the *persons employed by the IT Center*. For the simple ES variant hardly any syntax has to be considered. Only one of the search intentions was classified as partially complex, since the correct syntax for the query "experiment or simulation and not analysis" is not quite trivial. With the extended ES variant such queries were classified as partially complex, where not only concrete fields are searched, but the syntax for value ranges or comparisons must be used. Since the SPARQL query builder does not only have to follow a concrete syntax, some queries are classified as complex. Sometimes the user is expected to have certain knowledge (such as *employment at the IT center*) or several copy operations from the examples are necessary. Here the user must first find a suitable example, understand it and apply it to the corresponding search intention, as is the case with the search intention *topic design patterns and version number less than 5*. This is because here, in addition to a copy of *bif:contains*, the understanding of a SPARQL `UNION` operation is also necessary. All queries that require the SPARQL query builder to simply map *triple pattern* were classified as simple, the application of filters to individual variables as partially complex and everything beyond that, especially the

use of SPARQL `UNION`s, as complex. The result is that all approaches except for the SPARQL query builder do not pose a major challenge to the user because they follow a simple syntax.

Table 5.8: Complexity for user to perform search requests (√ = easy, ∼ = partially complex, − = complex)

Intention	Regex	Bif:contains	Query Builder	Faceted Search	ES Simple	ES Advanced
Published at IT Center	√	√	−	∼	√	√
Version 10	∼	∼	√	√	√	√
Created in 2007	√	∼	∼	√	√	∼
Design Patterns and Version < 5	∼	√	−	√	√	∼
Computer Science	√	√	−	√	√	√
Newest Version Res. 2	√	√	−	√	√	√
Experi./Simu. not Anal.	√	√	−	√	∼	√
Non Comput. Human Con.	√	√	∼	√	√	√
Av. s. 03.07.20	√	∼	√	√	√	√
Left (politicial)	√	√	∼	√	√	√
2 Meters	√	√	−	∼	√	√
Pub. < 2015 Oo. Softw.	√	√	∼	√	√	∼
Pub. T. Pynt. Created 2016 Oo. Softw.	√	∼	∼	√	√	√
In Total	√	√	−	√	√	√

5.4.5 Efficiency

Since efficiency is defined as the relationship between a user's effort and the results achieved, the categories of effectiveness, usability, and complexity for the user are included in the evaluation of efficiency and put in relation to each other. The ranking for the efficiency results from the ratio output/input, where output is the effectiveness and input is the combination of usability and complexity. Six approaches are considered, which is why in the category effectiveness the points from 6-1 are distributed according to their placement. For usability and complexity, 1-3 points are distributed for each of them, so that when summed up they correspond to the points of effectiveness. The efficiency for an approach that scores best in all three categories would thus be calculated from $\frac{6}{1+1} = 3$. The exact score is not important as only the comparison between the approaches is of interest. Due to the chosen procedure, an approach with good output and little effort

is rated best. After that, good output with a lot of effort, bad output with little effort, and bad output with a lot of effort follow in order, since good output is prioritized over low effort in terms of effectiveness. In "Appendix" Table G.13 the calculated values for efficiency are shown. The rankings of the considered categories effectiveness, usability, and complexity as well as the resulting ranking for the efficiency are shown in Figure 5.3. Both ES approaches perform best in terms of efficiency. Next comes the faceted search, next the *regex* approach and then the SPARQL query builder. The *bif:contains* approach records the lowest efficiency.

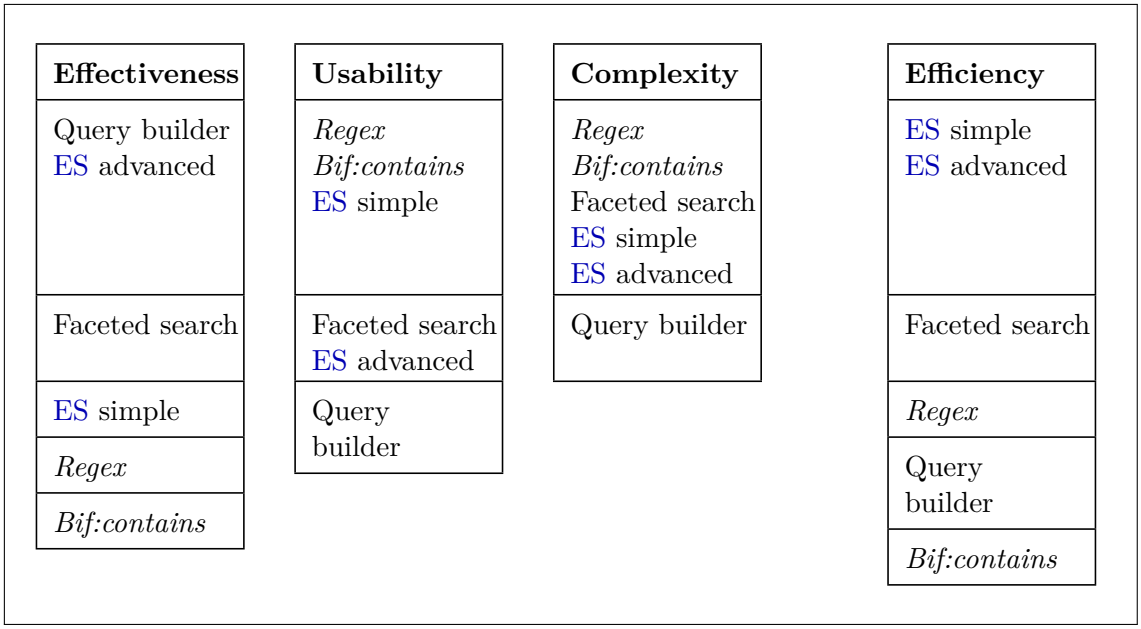


Figure 5.3: Ranking of effectiveness, usability, complexity, and efficiency whereby the approaches in a higher row are better ranked

5.4.6 Response Time

The response time of the different approaches was measured for all presented search intentions on the search data record. To make the comparison of the reponse times as fair and realistic as possible, an executable file was created for each approach, which receives the query (see section 5.4.2) and performs the corresponding function with it. The function returns a list with the IDs of the found metadata records (possibly with associated scoring). It was omitted to run through and output of these results to avoid that large result lists influence the runtime. With the executables, an attempt is made to map a complete user request, which goes beyond simply executing the request against a database. The search input of the user has to be sent to the backend to add the visibility restrictions. However, in the future, a web service will be running, and not every time a process will be started that reloads the used libraries. Therefore a basic process was created for comparison, which loads the libraries for *dotNetRDF.Data.Virtuoso* needed in all approaches and queries a simple graph in Virtuoso. An execution time of 221 ms was measured. Also, this type of measurement was chosen because it is not about comparing different queries within an approach, but about the differences between the approaches and the time elapsed for the user. To get a more accurate measurement, in each approach the executable was executed for each query 100 times in a row and then the average value was calculated. The results are shown in Table 5.9. Besides, the standard deviation was calculated to illustrate the inaccuracies of the measurements. For the faceted search approach, only the time of

the second search query was listed. Actually, the time of the *regex* query of the respective search intention from the first step has to be added. No time was measured if the search could not be further restricted.

Table 5.9: Response time in ms (rounded to a full number) of user request

Intention	**Regex**	**Bif:contains**	**Query Builder**	**Faceted Search**[1]	**ES Simple**	**ES Advanced**
Published at IT Center	401 ± 56	347 ± 22	323 ± 44	501 ± 31	556 ± 54	572 ± 78
Version 10	395 ± 21	354 ± 53	313 ± 18	404 ± 36	558 ± 23	550 ± 21
Created in 2007	393 ± 19	367 ± 46	302 ± 13	348 ± 31	552 ± 22	552 ± 23
Design Patterns and Version < 5	399 ± 16	335 ± 16	306 ± 12	404 ± 17	551 ± 23	552 ± 24
Computer Science	394 ± 20	336 ± 20	304 ± 15	-	563 ± 63	548 ± 24
Newest Version Res. 2	395 ± 24	336 ± 13	321 ± 15	398 ± 17	551 ± 30	558 ± 29
Experi./Simu. not Anal.	409 ± 29	342 ± 24	327 ± 19	450 ± 17	551 ± 29	555 ± 22
Non Comput. Human Consci.	414 ± 23	340 ± 21	307 ± 19	-	556 ± 33	556 ± 26
Av. s. 03.07.20	411 ± 31	329 ± 20	304 ± 18	352 ± 19	554 ± 28	556 ± 18
Left (politicial)	403 ± 34	341 ± 22	305 ± 19	-	549 ± 19	553 ± 25
2 Meters	399 ± 25	332 ± 16	323 ± 22	1023 ± 30	549 ± 21	555 ± 29
Pub. < 2015 Oo. Softw.	396 ± 14	344 ± 22	301 ± 23	-	547 ± 21	551 ± 23
Pub. T. Pyn. Created 2016 Oo. Softw.	402 ± 25	339 ± 18	309 ± 25	368 ± 19	554 ± 26	556 ± 27
∅	401 ± 26	346 ± 24	311 ± 20	327 ± 24	553 ± 30	555 ± 28

When comparing the response times of the approaches, it must be taken into account that they are not equally expressive and that in some approaches certain information could not be further restricted (see section 5.4.2) leading to a change in the duration. Therefore, the main focus is on the overall consideration of the reponse time per approach. The SPARQL query builder performs best in query time, followed by the *bif:contains* variant. The latter being faster than the *regex* approach was already foreseeable due to the indexing in the triplestore. At position four is the simple variant of ES, closely followed by the extended variant of the ES approach. The slowest approach is that of the faceted search because of the additional time of the first step to reduce the size of the search record. Not for all requests a further restriction is possible and thus no new query time is added but in

[1]Only the time of the second query step is listed. For the first step, the same time is valid as for the respective search intention in the regex approach.

most cases. Although in the faceted search with the intention "2 meters" only simple graph patterns are added compared to the coresponding *regex* query, the measured value is remarkably high. Thus, it is the only one with a total runtime of more than 1 second. This behavior could be since a string was used for the value instead of an integer. Here it becomes clear that string comparisons are much more complex and that such aspects should be considered when creating vocabularies. Furthermore, if the test environment would be chosen in such a way that the resulting data records of the first step are used as a filter in the second step instead of running the full-text search again, the second step would be much faster. Overall, it can be said that all tested search queries pass through less than 1.0 second. Consequently, they all meet the criterion of response time, which was set up in section 3.4.

5.4.7 Scalability

To measure the scalability of the approaches, the search queries presented in section 5.2 were executed on the entire 10,000 metadata records the same way described the section above. The results of the response times on the large data record are shown in Table 5.10.

Table 5.10: Response time in ms (rounded to a full number) of user requests in large search data record

Intention	**Regex**	**Bif: contains**	**Query Builder**	**Faceted Search**[1]	**ES Simple**	**ES Advanced**
Published at IT Center	2276 ± 542	344 ± 55	358 ± 72	1986 ± 116	561 ± 84	557 ± 69
Version 10	2119 ± 50	340 ± 16	393 ± 162	408 ± 19	547 ± 21	549 ± 20
Created in 2007	2131 ± 36	348 ± 20	327 ± 27	506 ± 34	549 ± 21	551 ± 20
Design Patterns and Version < 5	3483 ± 211	339 ± 19	314 ± 32	756 ± 22	544 ± 35	548 ± 23
Computer Science	2172 ± 35	342 ± 14	308 ± 18	-	560 ± 35	549 ± 29
Newest Version Res. 2	2155 ± 36	360 ± 11	319 ± 17	410 ± 14	548 ± 28	550 ± 18
Experi./Simu. not Anal.	3213 ± 51	492 ± 35	432 ± 23	2599 ± 36	551 ± 18	554 ± 18
Non Comput. Human Consci.	3477 ± 47	336 ± 23	319 ± 43	-	564 ± 79	556 ± 31
Av. s. 03.07.20	2199 ± 35	329 ± 15	344 ± 20	1430 ± 31	554 ± 27	556 ± 20
Left (politicial)	2148 ± 44	344 ± 18	308 ± 19	-	547 ± 17	558 ± 33
2 Meters	2118 ± 36	327 ± 15	325 ± 14	1291 ± 42	552± 25	549 ± 23
Pub. < 2015 Oo. Softw.	2208 ± 53	338 ± 16	299 ± 21	-	547 ± 18	549 ± 21

[1]Only the time of the second step is calculated. For the first step, the same time is valid as for the respective search intention in the regex approach.

Table 5.10: Response time in ms of user requests in large data record (continued)

Intention	Regex	Bif: contains	Query Builder	Faceted Search	ES Simple	ES Advanced
Pub. T. Pyn. Created 2016 Oo. Softw.	4148 ± 89	397 ± 17	309 ± 22	370 ± 17	546 ± 21	550 ± 26
∅	2604 ± 97	357 ± 21	335 ± 38	1084 ± 37	552 ± 33	552 ± 27

In contrast to the measurement of the response times on the small data record, the order changes. As before, the SPARQL query builder does best, followed by the *bif:contains* approach. Now, the simple and advanced ES variant follow whose response times do not differ in the measurements. After all, the data is indexed in the same way and just executed with different search syntaxes. The reason for the changed order is that prior indexing as with ES and *bif:contains* or the direct use of SPARQL without a full-text search can be performed more efficiently. This fact was not visible in the section before due to the small data record. The *regex* approach is on the second last place and the faceted search does the worst because again the time of the *regex* approach has to be added up. The second step of the faceted search performs the same queries as the *regex* approach, but with some search intentions that are not only full-text searches, additional triple patterns are used to narrow the search. As expected, this speeds up the execution time, which first becomes ovious on the larger data record. However, some restrictions reduce the response time noticebly more than others. `UNION` operations, the application of filters or string comparisons are more time consuming than simple triple patterns.

Furthermore, it is noticeable that the response times of the four leading approaches are still below 1.0 seconds, thus supporting a meaningful search (see section 3.4). Most importantly, they have hardly changed at all. The *regex* and thus also the approach of the faceted search exceed the limit of one second for all considered search intentions. For this reason, they do not meet the requirement to be usable even with a large amount of data. Consequently, indexing data and using the specially for RDF data constructed query language are most suitable. Furthermore, the RDF specific approaches dominate over external variants.

For the assessment of the scalability, the factor by which the execution times have increased are considered since they indicate how the response times will behave on even larger data sets. Table 5.11 shows the factor between the respective values of the average times from the tables 5.9 and 5.10.

Table 5.11: Factor (rounded to two digits) by which the response time has increased compared to the small data record

Regex	Bif:contains	SPARQL Query builder	Faceted Search[1]	ES Simple	ES Advanced
6.49	1.03	1.08	3.31	1	0.99

Regarding the factors, also the approaches with the fastest response times on the large data record are leading. The *regex* and thus also the faceted search shows a much higher scaling factor, which does not allow a comfortable search for users.

[1]Only the time of the second query step is listed. For the first step, the same time is valid as for the respective search intention in the regex approach.

5.4.8 Additional Effort

This category describes the additional effort (storage and calculations) required to implement the respective approach which is necessary in addition to the previous storage of metadata records in the RDF database. For the SPARQL query builder and the faceted search, no further arrangements need to be made. For the *regex* and *bif:contains* approach, only the literal rules described in section 4.1 need to be specified and stored. For ES, in addition to this storage of the literal rules, the implementation of the additional administrative triples and the mapping of the metadata records for the search index from section 4.5.1 is required. This results in completely redundant storage of the ES mappings (see section 4.5.1) of all metadata records. Besides the additional memory consumption, this also leads to necessary synchronization steps (see section 5.3). The already existing data must be indexed at the beginning. Besides, new metadata records must be created, updated, and deleted synchronously. If application profiles change or new ones are added, in the worst case a new index must be created.

The times to perform these actions are shown in Table 5.12. As with the measurement of the response times, executable files were created for this purpose, whose execution time and standard deviation were measured. Since the initialize indexing has to be done only once at the beginning and the reindexing differs only by deleting the old index and switching the alias, which are fast queries, only the reindexing was measured. Due to its duration, it was executed ten times independently with the 10,000 test data and the average was given. The creation, updating, and deletion of individual records were performed 100 times in a row and then the average value was calculated. The additional rule for saving the latest version (see Example 4.5) was used once and once without it, because it can affect other metadata records and thus increases the execution time. When (re)-indexing, the additional rules are always executed, but they do not affect the other documents every time, since all knowledge is already available in the knowledge graph and thus can be directly queried correct and complete. To see the effects of the additional rules, the actions were executed on the large data set. For adding, 9900 records already existed and the remaining 100 were added. When updating and deleting, all 10,000 records were already indexed.

It is important that these delays are not noticeable for the user, because they can run parallel in the background without limiting the search possibilities. Furthermore, it is clearly visible that creating, updating and deleting is faster if the other metadata records are not affected. In order to see the relation between the additional time required to add a metadata record and the time required to store and validate it in Virtuoso, the execution time for running an executable with this function was measured. The average value for 100 consecutive executions is 3217 ms. This time was calculated using a small metadata record that only fills the metadata fields `dcterms:author`, `dcterms:title`, `dcterms:subject`, and `dcterms:created`. It follows that the additional mapping in ES results in a further time expenditure of 26%. This number shows that the main effort lies in the necessary validation and storage in Virtuoso.

Table 5.12: Times in ms (rounded to a full number) for the actions of the additional effort in the ES approach

(Re)-Indexing	Add[1]	Add	Update[1]	Update	Delete[1]	Delete
1646706 ± 53376	3055 ± 233	837 ± 143	2996 ± 97	839± 129	2901 ± 148	735 ± 121

[1] Additional rules which influence other metadata records were used.

5.5 Results

In Figure 5.4 all considered evaluation criteria with the appropriate ranking of the individual approaches are listed. The order is a direct result of the measured times or classifications without normalizing them or considering further gradations (i.e., to classify similar values equally) in the individual categories. Furthermore, the general basic conditions were omitted, since they are the same for all approaches.

5.5.1 Final comparison

There are different methods to create a total ranking based on different individual rankings. Depending on the chosen method or previous normalization of a ranking, a different overall ranking is created. To still get a better overview of the results, a preference matrix is used to look at how often one approach is better or worse than another. Then these frequencies are added up and the difference is calculated to get a ranking.

Table 5.13: Preference matrix to compare different approaches where the numbers indicate how often the approach of the left side beats the one to be compared

	Regex	**Bif: contains**	**Query Builder**	**Faceted Search**	**ES Simple**	**ES Adv.**	**Wins**	**Loses**	**Diff.**
Regex		1	3	3	2	2	11	15	-4
Bif: contains	2		3	3	2	3	13	14	-1
Query Builder	4	4		3	3	2	16	17	-1
Faceted Search	3	3	3		2	1	12	17	-5
ES Simple	3	3	4	4		2	16	11	5
ES Adv.	3	3	4	4	2		16	10	6

According to this ranking, the extended ES approach performs best. Next comes the simple variant of ES, and after that the SPARQL query builder together with the *bif:contains* approach. The next one is the *regex* approach and the last one the faceted search. However, this ranking does not include any weighting of the different categories and only refers to the data records, search queries, and test environments examined in the course of the work with the corresponding general conditions for implementation. Furthermore, this ranking does not include any indication of the ratio of how much better an approach is than another in a particular category. Thus, it counts equally whether, for example, concerning response time, one approach is a few milliseconds or a few hundred milliseconds faster. Graduations could have been considered, so that similarly good approaches are ranked equally. However, the limits for classifying into the same category also influence the results.

Another method is the collection of points for approaches using the results from Figure 5.4 and assign each approach the number of the rank in reverse order. However, since the placements and thus the range of values vary, the min-max normalization ($X' = \frac{X - X_{min}}{X_{max} - X_{min}}$) is applied to bring all data to a uniform scale and thus avoid the unintentional weighting of different categories. The results are shown in Table 5.14 and

the calculations can be found in "Appendix" Table G.14. In this illustration, the simple ES approach prevails over the advanced one, over *regex*, over the faceted search, over the *bif:contains*, and last over the SPARQL query builder. The two comparisons show that the result depends strongly on the chosen method. However, the scores achieved by the approaches within each comparison respectively show that they do not differ substantially. For this reason, the two comparisons are only used to provide a rough overview of the results rather than a final recommendation.

Table 5.14: Ranking based on the min-max normalization of the different categories

Regex	**Bif:contains**	**SPARQL Query Builder**	**Faceted Search**	**ES Simple**	**ES Advanced**
4.05	3.9	3.65	4	4.9	4.7

From the category of scalability it is known that the *regex* approach and the faceted search do worst when searching in a large data record. The main reason for this is that there is no indexing in advance and therefore the search is very slow. Furthermore, the search with a regular expression in all literals leads to a lot of irrelevant data. In the second step of the faceted search, these data can be limited by specifying certain fields, but this second step is a disadvantage and additional work for the user. The four other approaches are the most performant since the data is either indexed in advance or directly in the ideal RDF form respectively in the corresponding structure by using the URIs. From this, it can be concluded that in any case an indexing approach should be chosen for implementing a search in RDF-based knowledge graphs.

The simple ES variant is very similar to the *bif:contains* variant, because in both cases literals are indexed and made searchable without being able to search in concrete fields. One difference is that ES, in contrast to *bif:contains*, allows a cross-field search. This is an advantage especially regarding the later applicability because search fields can be combined and thus more specific queries are possible. Besides, *bif:contains* does not allow searches in date and integer fields or in compound literals. Since date and number fields play an important role in research data, an implemenation would not make sense without their searchability.

The results of the evaluation show that the both ES variants hardly differ in most categories. Regarding response time and scalability the difference is almost indistinguishable and therefore negligible. They only vary in the search syntax, thus the advanced version scores better in terms of effectiveness, but worse in terms of usability. Since the search results of the extended approach is significantly higher and the quality of search for the user differs only slightly, this approach should be preferred. However, the advantage of the simple variant is that it can still display results if a user enters the wrong search string and that the disclosure of properties is not necessary. It is possible to offer the user both variants by setting a flag. The only aspect to keep in mind is that the search syntax is different and incompatible even if the intention is the same.

In principle, this leaves the extended ES variant and the SPARQL query builder. The SPARQL query builder scores above all with its speed. Even though the extended ES variant is inferior to this approach in terms of speed, it also clearly meets the maximum response time of one second. Both approaches score the same in terms of effectiveness. Most importantly, they both allow cross-field and field-specific searches and the specification of value ranges. Thus, they are especially useful when more complex queries are to be enabled as a keyword or full-text search in single literal queries.

Since the user's search experience is the main focus, efficiency is also an important category for evaluation. It contains the effort and complexity for the user compared to the delivered results. Here, the extended ES variant scores much better. The user interface and search syntax are much simpler and clearer. It is also very understandable and well documented (see the documentation of ES's advanced search syntax [130]). The SPARQL query builder has the additional disadvantage that a basic understanding of SPARQL can be crucial to map certain queries. As already mentioned by Kuric et al. [65], SPARQL query builder approaches all encounter usability problems. This leads to user dissatisfaction and eventually even to the failure of certain queries. ES also offers the advantage of being able to map synonyms and find results by using stemming without having to map them literally in the data. Just the possibility to enter singular or plural words leads to an extreme increase in user satisfaction and error tolerance. In this case, it is no longer necessary to assume that the search is optimal in every case, so that, for example, a search can be made for "2 meters" instead of "2 meter".

Furthermore, ES brings along a ranking compared to SPARQL. This is especially helpful if a search query returns many results. This can be used to highlight metadata records for which a certain search query is more specific than others. Two disadvantages of the extended ES variant compared to the SPARQL query builder are its additional resource consumption regarding memory and computing time as well as the additional mapping of rules. Since both are not directly visible to the users and do not bring any disadvantage for them, this point of criticism is not very serious. The rules only have to be created once and can be reused. They also allow a very flexible and customizable extension, which has a great influence on the search results. Also, the rules have the advantage that a user no longer has to think about such structures and connections of the RDF data like the SPARQL query builder does. Here the user has to map, e.g., the transitive relationship of the subclasses for the GRF subject classification or the affiliation of a person to an institute (see Example 4.1, 4.3 and section G.2). The user might not even be aware of these relations, so it is advantageous to include them from the beginning. But this is also a disadvantage because only such relations can be found in the search, which are mapped by rules for the search index. For example, if the transitive rule for the subclasses of the GRF subject classification is not used, no subclasses can be found in the search for a superclass. This means that all knowledge or structures must be aware of and considered in advance.

These comparisons and considerations support the ranking result of the two methods considered. The only obvious conclusion was the superiority of the ES approach under the overall consideration of all criteria. The evaluation shows that in most cases an approach is either easy to use but slow (*regex*, faceted search) or ineffective (*bif:contains*), or on the contrary, fast and effective but difficult to use (SPARQL query builder). With the transformation of metadata graphs into a search index by application profiles and for use in a search engine like ES, an approach was found that combines these two contradictory properties. The only drawback is the additional memory and computing time consumed by the mapping, which, however, does not directly affect the search.

5.5.2 Answering the Research Questions and Hypothesis

In the following, the answers to the research questions posed in section 1.5 are briefly discussed. Since the final comparison already shows that the ES approach and the SPARQL query builder are the only appropriate implementation options considerung all categories, often only these two variants are considered.

What approaches exist to search in RDF data and which functions support them?
The existing approaches for searching in RDF data are discussed in detail in the section 2.5 "Related Work". The approaches considered in the master thesis are explained in chapter 4. In chapter 5 "Evaluation and Results" they are analyzed and thus their functionalities are clarified. They can be roughly divided into two categories. One generates a SPARQL query and the other maps the data into a search index in order to use a search engine.

How are access rights represented?
The mapping of access rights is possible in all considered approaches. It can be ensured either by mapping the visibility as additional properties in the ES document and using an additional filter when executing ES queries or by using the in the RDF data included project visibility and SPARQL (see section 5.4.1).

How are the general challenges of querying RDF knowledge graphs addressed?
The main challenges when querying RDF knowledge graphs are the unknown structure and URIs. In the ES approach, these challenges have been eliminated due to the mapping and the application of the literal rules, as more complex graph structures and URIs are thus resolved. In the SPARQL query builder, these challenges were handled in such a way that structures can be mapped and selected using different form fields. These reveal the structure to the user and enable the use of URIs without having to know it himself.

Which search types (value ranges, and/or/not, date, full-text, ...) are meaningful and how can they be implemented?
The more search types an approach allows, the better its effectiveness since more complex and diverse search intentions can be mapped as shown in section 5.4.2. Both ES and the SPARQL query builder allow different search types like keyword search, value ranges, boolean queries, and full-text search.

How to deal with large amounts of data, especially with regard to the runtime of search queries? How to get the highest possible performance?
For the handling of large amounts of data, it was considered how the approaches work with large amounts of data (see section 5.4.7). Both approaches deliver successful results and reasonable times. In ES this is due to the previous indexing of the data. Besides, ES is specifically designed for scalable and fast searches [115]. In the SPARQL query builder, this is because SPARQL is specially written as a query for RDF graph was designed. This is exactly the reason why ES cannot fully keep up with the speed of SPARQL.

How must the data be stored or prepared for the search?
The metadata records are already included in the RDF database. Only for the ES approach, they have to be mapped into an ES document to create a search index as described in section 4.5.1.

Does a conversion into a search index have advantages for CoScInE? Which ones?
The advantages of mapping to a search index are the advantages of the extended ES approach over SPARQL query builder, as described in the evaluation and results (see sections 5.4 and 5.5). These are above all the much smoother and clearer UI, the comprehensibility of the search syntax, and the complete hiding of the data structure, vocabularies, and schemas.

What additional information can a SHACL application profile provide to enable a complex search?
The SHACL application profiles are an essential component in the implementation of the ES approach. Only through them, the mapping of a metadata record into an ES document (see section 4.5.1) as well as the storage of (specific) literal and additional rules (see section 4.1) and thus a meaningful search in the RDF knowledge graph is possible. Only the profiles can ensure that the metadata graphs follow specific metadata fields and their associated data types. In the SPARQL query builder the application profiles could be used to generate dropdowns that are limited to the metadata fields or instances of an associated class.

Hypothesis: Mapping RDF-based metadata records into a search index for use in a search engine improves the quality of the search and results for the user compared to using generated SPARQL queries.
The final comparison (see section 5.5.1) after the evaluation as well as the differentiation of the extended ES variant with the SPARQL query builder largely confirms the hypothesis in the considered context when using application profiles. The mapping of RDF-based metadata records into a search index for use in a search engine improves the quality of the search for the user, compared to the use of generated SPARQL queries. The same results can be achieved with much less effort for the user.

Effectiveness
Query builder ES advanced
Faceted search
ES simple
Regex
Bif:contains

Usability
Regex *Bif:contains* ES simple
Faceted search ES advanced
Query builder

Complexity
Regex *Bif:contains* Faceted search ES simple ES advanced
Query builder

Efficiency
ES simple ES advanced
Faceted search
Regex
Query builder
Bif:contains

Resp. Time
Query builder
Bif:contains
Regex
ES simple
ES advanced
Faceted search

Scalability
ES advanced
ES simple
Bif:contains
Query builder
Regex
Faceted search

Add. Effort
Query builder Faceted search
Regex *Bif:contains*
ES simple ES advanced

Figure 5.4: Resulting ranking in all categories whereby the approaches in a higher row are better ranked

6 Conclusion

This chapter starts with some aspects worthy of discussion that have been noticed in the context of the master thesis. Then the most important results and steps of the master thesis are summarized once again. This is followed by a short outlook on the proposed further procedure in the context of CoScInE and the use of the ES approach.

6.1 Discussion

First, the question arises whether the implemented ES approach creates an efficient semantic search engine, as described in the title of the thesis. In the context of the work, efficiency was defined as the ratio of the effort for the user compared to the precision and recall of the search results. Efficiency was also considered as a separate category in the evaluation, in which the ES approach scores best. As described in section 2.4, a semantic search engine uses Semantic Web technologies. In this work RDF data, SPARQL, and SHACL are used. SPARQL queries were used to map semantic knowledge in the form of SHACL rules. This knowledge includes, e.g., inference rules like subclasses or the mapping of structures to search in the corresponding literals of URIs. Furthermore, in a semantic search engine it is important to understand the search intention, which is made possible by the structured search in the metadata fields.

As described at the beginning, a big problem of RDF is that although much information can be displayed via RDF triples, most of the information is still available as full-text, as in CoScInE and the description of research data. For this reason, a good approach must support full-text search. Of the approaches considered, *bif:contains* and ES are the only ones that adequately address this issue.

In this work, an evaluation of six approaches to an semantic search engine is derived. These were prepared as described in section 5.3 and were assumed to serve as a basis for the evaluation criteria. To be able to make a generally valid statement about the best approach, all possible approaches as well as all combinations and variants of these would have to be considered. This number is arbitrarily large, since only small changes, such as the additional enabling of the logical `AND` operation in the *regex* approach or the use of *bif:contains* instead of *regex* in the faceted search, would yield different results.

As described in the evaluation, no formal user tests were conducted in the context of the master thesis, since the focus of the thesis was the evaluation of different approaches considering several criteria. The heuristics and estimates used in the respective criteria serve to roughly classify the approaches. For concrete statements about the usability and also the User Experience (UX), user tests would have to be conducted, which could also evaluate the satisfaction and fault tolerance of the approaches.

ES allows a ranking for the use of the data. According to the configuration of the similarity [131], the ranking depends, for example, on how often certain terms are used in all records and occur in a metadata record. Therefore, researchers can prepare their metadata records in such a way that they are ranked as high as possible and thus influence

the search results. Depending on the settings, prevention may be necessary. At the same time, the ranking also leads to a weighting of the results found. Thereby in some cases, relevant data can be distinguished from irrelevant data. In the example after searching for data published at the IT center on the small data set, the ranking leads, for example, to the fact that the actual desired results are ranked higher. Thus, the ranking provides additional support for the user. Furthermore, the extended syntax allows a so-called boosting [130] to increase the relevance of one search term over another.

Querying structured data is generally characterized by a trade-off between expressivity and usability [132]. The SPARQL query builder is a classic example of this. It is very expressive due to the transformation into a SPARQL query, but its usability is very difficult and confusing. The presented ES approach breaks this statement in the considered context of CoScInE and through the use of application profiles. The search syntax and UI of the ES approach is very user-friendly. It scores very well in both categories usability and complexity. Nevertheless, it is very expressive because the rules can be used to map any complex information via SPARQL. The considered search results are just as good as with the SPARQL query builder, but much easier to query.

6.1.1 Limitations of Using Elasticsearch as Search Engine for RDF Data

Since the use of ES for CoScInE has proven to be very suitable within the scope of the work a semantic search engine based on ES was implemented in this thesis. Thereby, restrictions to be considered in the application are discovered.

The implementation is limited to SHACL profiles being available to specify corresponding fields and data types. Only with this information an index can be created in ES in advance. Furthermore, when creating different profiles, it is important to ensure that the same data types are used for the same metadata fields, to provide a data type specific mapping in ES instead of a string representation. The same properties should be used for the same metadata fields, otherwise there will be different mappings for the same property, or there will be a conflict when creating the index for the second property due to the identical name (see Example 6.1). In any case, the reuse of existing metadata schemas corresponds to the principle of Linked Data.

Example 6.1: Problems when using different properties for the same field

In an application profile, the metadata field of the author is described via the property `dcterms:creator`, where the associated label is “creator”. In the second application profile, the property `example:author` is defined with the associated label “author”.

```
coscineengmeta:creator
  sh:path dcterms:creator ;
  sh:name "Creator"@en, "Autor"@de .

dcterms:creator rdfs:label "creator" .

coscineengmeta2:creator
  sh:path example:author ;
  sh:name "Creator"@en, "Autor"@de .

example:creator rdfs:label "author" .
```

This leads to the fact that in ES there will be a field “creator” and a field “author”, although the same metadata field is described. If the label of the second property

Example 6.1 (continued)
would also be “creator”, a conflict would occur when creating the index in ES because a field with the same name already exists.

In this example it might be considered to check if such a field already exists in the ES index and use it for both properties. This has to be considered when creating the data types. However, it would have to be ensured that both properties are the description of the same metadata field. This decision is not necessarily unambiguous and trivial, so it cannot simply be made by a machine. If two different metadata fields (which also have a different semantic meaning) are identified with the same label, which can occur due to homonyms, a workaround must be developed so that the fields can be clearly distinguished and offered to the user as search fields. This problem also exists in the faceted search and the SPARQL query builder, because there the labels and not the URIs are used for display. In the full-text search with *bif:contains* and *regex*, it is not even possible to use the metadata fields. The whole example also shows the problem of the semantic gap [132], namely that the same data or information can be represented using different vocabularies and RDF terms. An arbitrary number of different graphs exists, which represent the same semantic information. Within the ES approach, this problem is countered by the uniform creation of application profiles, the reuse of both metadata schema and vocabularies, as well as the controlled creation of literal and additional rules.

For the creation of the literal and additional rules a knowledge engineer with SPARQL knowledge is required. Although this saves the user a lot of cognitive work, the creation of application profiles is more complex. When creating the rules, it is especially important to pay attention to their meaningfulness. The rule from the example 4.2, for example, also creates the corresponding institutional organization for a person as value. This makes sense in the context of a publisher metadatafield. However, it must also be taken into account that the associated institutional organization can change in the knowledge graph. Consequently, the new institute is stored instead of the previous specified at the time of publication, which makes less sense.

Another aspect worthy of discussion is the arbitrary extensibility of the rules. The question arises, if any rule can be constructed. Anything can be mapped that can be constructed using the SPARQL `CONSTRUCT`s based on the database, if the corresponding instance of a class (literal rule) or the ID of a metadata record (additional rule) is used as focus node. In general, the rules are very generic and can be used across all application profiles. It is crucial that a knowledge engineer decides on the basis of the rules which information of the metadata graph is mapped and thus is searchable and which information is lost through the mapping according to ES and thus cannot be searched. Furthermore, in the evaluation it was found that the use of additional rules that influence other metadata records essentially increases the indexing time of a new metadata record. For this reason, they should only be used very rarely and with caution.

The fact that it is not possible in ES to search for a year number in a date field without using the advanced syntax is a disadvantage that was circumvented during implementation by additionally generated fields for day, month, and year. The conversion of the year and the written out month should be kept. However, the day should not be indexed, as otherwise too many irrelevant data can be found in the unstructured search and usually not only certain days (independent of the entire date) are searched for. Also the written representation of boolean values can be omitted, because the advanced search allows the concrete query of this information.

6.1.2 Future Research Suggestions

The entire evaluation and selection of approaches refer to the context of research data and CoScInE (especially the use of application profiles). The sample data was generated and the search intentions were devised with exactly this focus. In addition, the search for metadata records is also a specific issue. Other entities like single persons should not be found. In addition, the most optimal search queries were assumed, i.e., that the user does not make any mistakes during the input and that the structure and appropriate choice of words for the approach were adhered to. To what extent such an approach is transferable to other use cases and whether ES can be generally considered a suitable search engine for arbitrary RDF knowledge graphs remains an open question for future research projects.

Up to now, a knowledge engineer is required to map the literal and additional rules. An interesting question is whether it is possible to have these rules created by a user without the appropriate knowledge. For example, would a user interface be conceivable in which rules can be clicked together based on the vocabularies, metadata standards, and ontologies used?

6.2 Summary

The goal of this thesis was to make the RDF based knowledge graph of CoScInE searchable for a user without deep knowledge of the underlying data models, metadata schemas, and SPARQL. For this purpose, existing approaches were first considered and the four different approaches *full-text search in RDF literals*, *SPARQL query builder*, *faceted search*, and *ES* were selected. In detail, six approaches were considered, since the full-text search is divided into the *regex* and *bif:contains* variants and ES allows two different search syntaxes. In the context of the master thesis, all six approaches were examined more closely and evaluated on the basis of their effectiveness, usability, complexity for the user, efficiency, response time, scalability, and additional effort.

The evaluation of the approaches reveals the SPARQL query builder and the extended ES variant as the most promising approaches according to the defined evaluation criteria. Based on this finding, a search method that reaches the goal of searchability, and therefore reusability of data was found. Furthermore, the evaluation shows that in most cases an approach is either easy to use but slow (*regex*, faceted search) or ineffective (*bif:contains*), or, on the contrary, fast and effective but difficult to use (SPARQL query builder). With the transformation of metadata graphs into a search index by application profiles and for use in a search engine like ES, a solution was found that combines these two contradictory properties. For this reason, it was decided to use a combination of the simple and extended ES variant for CoScInE. In addition, the user benefits from the fact that ES is a search engine that has been implemented exactly for this purpose and thus contains many search features.

In conclusion, the main finding of this work is that the use of search engines can also be suitable for searching in RDF-based knowledge graphs if a skillful mapping of the data and the knowledge contained in its structure is applied. In the context of the thesis, a mapping using SHACL application profiles, literal, and additional rules was developed. In addition, an appropriate synchronization with the data from the RDF triplestore and the handling when using different SHACL application profiles were presented.

6.3 Outlook

The next step should be the full integration of the search functionality in CoScInE. Until now, the search was implemented independently of the existing code base and was built in using command-line programs. In addition, an appropriate user interface for displaying the found metadata records must be considered. One possibility is the use of ES specific search interfaces [133]. The interface could also include ES features like auto-complete or search suggestions [134], which reduce the error rate and help the user to enter the most optimal search query.

Additionally, some tests or considerations for the optimal configuration of ES could still be done. Here, for example, the indexing and search analyzers, tokenizer, and ranking functions could be looked at to get even better results. Furthermore, synonyms could be built in, e.g., to find metadata records with the entry "image" when searching for "picture". However, stemming is not possible in this case. In order not to have to decide between both variants, ES also offers the possibility to index single fields multiple times. So stemming on the one hand and the synonym filter on the other hand could be used.

Depending on the vocabularies and data used, it might also be interesting to enrich the knowledge graph with information from other data sources according to the principle of Linked Data. In this way, further rules could be mapped.

ES offers besides the presented data types also the use of *nested field types* [135]. These allow objects to be saved. For example, the name of a person could be stored as an object with first and last name instead of a simple string. This construction could be integrated into the approach. However, for a specific metadata field, it must then be ensured that the selected instance has a first and last name. Again, SHACL shapes could be used to validate that a specific class, in this case, a person, has a first and last name property. This could also be used to automatically generate subfields in the form, which allow us to describe a person by first and last name. Again, it becomes problematic if, for example, not only a `foaf:Person` but also a `foaf:Agent` is allowed as value. This is because a `foaf:Organization` is also a `foaf:Agent` but does not have a first and last name. In this case, the data type would have to be broken down into a string. The extent to which this variant will provide further progress remains to be seen.

Based on the search engine implemented in this thesis additional tests and use cases can be evaluated in further research. As a concrete next step, the implemented approach could be applied to real metadata records of research data currently collected in CoScInE.

A List of Tables

B List of Figures

C List of Files

D List of Definitions

E List of Examples

F References

[1] Mark D. Wilkinson et al. “The FAIR Guiding Principles for scientific data management and stewardship”. In: *Scientific data* 3 (2016), p. 160018. ISSN: 2052-4463. DOI: 10.1038/sdata.2016.18. URL: https://pubmed.ncbi.nlm.nih.gov/26978244/.

[2] NFDI. *Nationale Forschungsdateninfrastruktur NFDI*. German. URL: https://www.nfdi.de/informationen (visited on 08/19/2020).

[3] RfII. *Themen - RfII*. German. URL: http://www.rfii.de/de/themen/ (visited on 08/19/2020).

[4] DFG. *DFG, German Research Foundation - National Research Data Infrastructure*. URL: https://www.dfg.de/en/research_funding/programmes/nfdi/index.html (visited on 08/09/2020).

[5] Maxi Kindling and Peter Schirmbacher. “„Die digitale Forschungswelt“ als Gegenstand der Forschung / Research on Digital Research / Recherche dans la domaine de la recherche numérique”. In: *Information - Wissenschaft & Praxis* 64.2-3 (2013). ISSN: 1619-4292. DOI: 10.1515/iwp-2013-0017. URL: https://www.researchgate.net/publication/270550641_Die_digitale_Forschungswelt_als_Gegenstand_der_Forschung_Research_on_Digital_Research_Recherche_dans_la_domaine_de_la_recherche_numerique.

[6] *Forschungsdaten — DIPF | Leibniz-Institut für Bildungsforschung und Bildungsinformation*. German. URL: https://www.dipf.de/de/forschung/forschungsdaten (visited on 08/19/2020).

[7] *Forschungsdatenmanagement von A - Z - RWTH AACHEN UNIVERSITY - Deutsch*. German. URL: https://www.rwth-aachen.de/cms/root/Forschung/Forschungsdatenmanagement/~svkj/A-bis-Z/ (visited on 08/19/2020).

[8] *Describing metadata | Jisc*. URL: https://www.jisc.ac.uk/guides/metadata/describing-metadata (visited on 08/13/2020).

[9] Dominik Schmitz and Marius Politze. “Forschungsdaten managen – Bausteine für eine dezentrale, forschungsnahe Unterstützung: 76-91 Seiten / o-bib. Das offene Bibliotheksjournal / herausgegeben vom VDB, Bd. 5 Nr. 3 (2018) / o-bib. Das offene Bibliotheksjournal / herausgegeben vom VDB, Bd. 5 Nr. 3 (2018)”. In: *o-bib. Das offene Bibliotheksjournal / Herausgeber VDB* 5.3 (2018), pp. 76–91. ISSN: 2363-9814. DOI: 10.5282/o-bib/2018H3S76-91. URL: https://www.o-bib.de/article/view/5339.

[10] RWTH Aachen. *Leitlinie zum Forschungsdaten-management an der RWTH Aachen - RWTH AACHEN UNIVERSITY - Deutsch*. German. URL: https://www.rwth-aachen.de/cms/root/Forschung/Forschungsdatenmanagement/~ncfw/Leitlinie-zum-Forschungsdatenmanagement/ (visited on 08/19/2020).

[11] RWTH Aachen University. *Forschungsdatenmanagement an der RWTH Aachenf*. Youtube. 2016. URL: https://www.youtube.com/watch?v=6XLcJxPcrFQ (visited on 08/19/2020).

[12] Marius Politze et al. "How to Manage IT Resources in Research Projects? Towards a Collaborative Scientific Integration Environment". In: *European Journal of Higher Education IT 2020-2*. Ed. by Yves Epelboin et al. in Press.

[13] Marius Politze and Bernd Decker. "Ontology Based Semantic Data Management for Pandisciplinary Research Projects". In: *Proceedings of the 2nd Data Management Workshop*. Ed. by Constanze Curdt and Christian Wilmes. Kölner Geographische Arbeiten. Cologne, Germany, 2016. DOI: 10.5880/TR32DB.KGA96.10.

[14] Marius Politze and Thomas Eifert. "On the Decentralization of IT Infrastructures for Research Data Management". In: (2019).

[15] Marius Politze, Sarah Bensberg, and Matthias Müller. "Managing Discipline-Specific Metadata Within an Integrated Research Data Management System". In: *Proceedings of the 21st International Conference on Enterprise Information Systems ICEIS 2019, Heraklion, Crete - Greece, May 3 - 5, 2019*. Ed. by Joaquim Filipe. ICEIS (Setúbal). [S. l.]: SciTePress, 2019, 253-260UR –https://www.scitepress.org/Link.aspx?doi=10.5220/0007725002530260. ISBN: 978-989-758-372-8. (Visited on 05/28/2019).

[16] Bela Brenger. *CoScInE – Product Features*. 2020.

[17] Laurel L. Haak et al. "ORCID: a system to uniquely identify researchers". In: *Learned Publishing* 25.4 (2012), pp. 259–264. ISSN: 09531513. DOI: 10.1087/20120404.

[18] Raimund Vogl et al. ""sciebo — theCampuscloud" for NRW". In: *Proceedings of the 21st EUNIS Congress*. Ed. by Michael Turpie. Dundee, Scotland, 2015.

[19] *DFG - Deutsche Forschungsgemeinschaft - Gute wissenschaftliche Praxis*. German. URL: https://www.dfg.de/foerderung/grundlagen_rahmenbedingungen/gwp/ (visited on 08/24/2020).

[20] Hiba Arnaout and Shady Elbassuoni. "Effective Searching of RDF Knowledge Graphs". In: *SSRN Electronic Journal* (2018). ISSN: 1556-5068. DOI: 10.2139/ssrn.3199315. URL: https://www.researchgate.net/publication/326337968_Effective_Searching_of_RDF_Knowledge_Graphs.

[21] Steffen Staab. "Wissensmanagement mit Ontologien und Metadaten". In: *Informatik Spektrum* 25.3 (2002), pp. 194–209. ISSN: 0170-6012. DOI: 10.1007/s002870200226. URL: https://www.researchgate.net/publication/220351556_Wissensmanagement_mit_Ontologien_und_Metadaten.

[22] *Documentation, metadata, citation*. URL: https://mantra.edina.ac.uk/documentation_metadata_citation/ (visited on 08/13/2020).

[23] *Standards/Schema - Metadata for Data Management: A Tutorial - LibGuides at University of North Carolina at Chapel Hill*. URL: https://guides.lib.unc.edu/metadata/standards (visited on 09/14/2020).

[24] *Metadata management | Jisc*. URL: https://www.jisc.ac.uk/guides/metadata/management (visited on 08/13/2020).

[25] *DCMI: Dublin Core™*. URL: https://www.dublincore.org/specifications/dublin-core/ (visited on 08/13/2020).

[26] *Data Catalog Vocabulary (DCAT) - Version 2*. URL: https://www.w3.org/TR/vocab-dcat-2/ (visited on 08/13/2020).

[27] *DataCite Schema*. URL: https://schema.datacite.org/ (visited on 08/13/2020).

[28] Angelina Kraft et al. "The RADAR Project—A Service for Research Data Archival and Publication". In: *International Journal of Geo-Information* 5.3 (2016), p. 28. ISSN: 2220-9964. DOI: 10.3390/ijgi5030028. URL: https://www.researchgate.net/publication/297607606_The_RADAR_Project-A_Service_for_Research_Data_Archival_and_Publication.

[29] *Controlled vocabulary | Jisc*. URL: https://www.jisc.ac.uk/guides/metadata/controlled-vocabulary (visited on 08/16/2020).

[30] *DCMI: Application Profile*. URL: https://www.dublincore.org/resources/glossary/application_profile/ (visited on 09/14/2020).

[31] W3C. *Semantic Web - W3C*. URL: https://www.w3.org/standards/semanticweb/ (visited on 08/09/2020).

[32] T. I.M. BERNERS-LEE, JAMES HENDLER, and O. R.A. LASSILA. "THE SEMANTIC WEB". In: *Scientific American* 284.5 (2001), pp. 34–43. ISSN: 00368733. URL: www.jstor.org/stable/26059207.

[33] *Linked Data - W3C*. URL: https://www.w3.org/standards/semanticweb/data (visited on 08/11/2020).

[34] Tim Berners-Lee. *Linked Data - Design Issues*. URL: https://www.w3.org/DesignIssues/LinkedData.html (visited on 08/11/2020).

[35] W3C. *RDF 1.1 Concepts and Abstract Syntax*. 2014. URL: https://www.w3.org/TR/rdf11-concepts/ (visited on 08/09/2020).

[36] Sabrina Kirrane. "Linked data with access control: Linked data with access control". PhD thesis. URL: https://aran.library.nuigalway.ie/handle/10379/4903.

[37] *DCMI: DCMI Metadata Terms*. URL: https://www.dublincore.org/specifications/dublin-core/dcmi-terms (visited on 08/24/2020).

[38] *Ontologies - W3C*. URL: https://www.w3.org/standards/semanticweb/ontology (visited on 08/16/2020).

[39] *RDF Schema 1.1*. URL: https://www.w3.org/TR/rdf-schema/ (visited on 08/16/2020).

[40] Brian McBride. "The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS". In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. International Handbooks on Information Systems. Berlin and Heidelberg: Springer, 2004, pp. 51–65. ISBN: 978-3-540-24750-0. DOI: 10.1007/978-3-540-24750-0_3.

[41] *OWL - Semantic Web Standards*. URL: https://www.w3.org/OWL/ (visited on 08/16/2020).

[42] *OWL 2 Web Ontology Language Document Overview (Second Edition)*. URL: https://www.w3.org/TR/owl2-overview/ (visited on 08/16/2020).

[43] *OWL Web Ontology Language Overview*. 2020. URL: https://www.w3.org/TR/2004/REC-owl-features-20040210/ (visited on 10/06/2020).

[44] H. Knublauch and D. Kontokostas. *Shapes Constraint Language (SHACL)*. URL: https://www.w3.org/TR/shacl/ (visited on 08/13/2020).

[45] *DASH Data Shapes Vocabulary*. URL: http://datashapes.org/dash (visited on 08/13/2020).

[46] W3C SPARQL Working Group. *SPARQL 1.1 Overview*. URL: https://www.w3.org/TR/sparql11-overview/ (visited on 08/10/2020).

[47] W3C SPARQL Working Group. *SPARQL 1.1 Query Language*. URL: https://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (visited on 08/10/2020).

[48] *Linked Open Vocabularies*. URL: https://lov.linkeddata.es/dataset/lov/vocabs/foaf (visited on 08/24/2020).

[49] W3C SPARQL Working Group. *SPARQL 1.1 Update*. URL: https://www.w3.org/TR/2013/REC-sparql11-update-20130321/ (visited on 08/10/2020).

[50] *Inference - W3C*. URL: https://www.w3.org/standards/semanticweb/inference (visited on 08/11/2020).

[51] Venkat Chandrasekaran, Nathan Srebro, and Prahladh Harsha. *Complexity of Inference in Graphical Models*. 2012. URL: https://www.researchgate.net/publication/226423110_Complexity_of_Inference_in_Graphical_Models.

[52] Juan Sequeda. *Introduction to: Open World Assumption vs Closed World Assumption - DATAVERSITY*. Nov. 2012. URL: https://www.dataversity.net/introduction-to-open-world-assumption-vs-closed-world-assumption/ (visited on 08/13/2020).

[53] Nick Drummond and Rob Shearer. "The open world assumption". In: *eSI Workshop: The Closed World of Databases meets the Open World of the Semantic Web*. Vol. 15. 2006.

[54] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. "Introduction to Information Retrieval". In: *Natural Language Engineering* 16.1 (2010), pp. 100–103. URL: http://eprints.bimcoordinator.co.uk/35/.

[55] Serge Abiteboul. “Querying semi-structured data”. In: *Lecture Notes in Computer Science* 1186 (1970), pp. 1–18. ISSN: 0302-9743. DOI: 10.1007/3-540-62222-5_33. URL: https://www.researchgate.net/publication/2598737_Querying_Semi-Structured_Data.

[56] Stefan Buettcher, Charles L. A. Clarke, and Gordon V. Cormack. *Information Retrieval: Implementing and Evaluating Search Engines*. 2010. URL: https://research.google/pubs/pub36555/.

[57] Renaud Delbru, Stephane Campinas, and Giovanni Tummarello. *Searching Web Data: An Entity Retrieval and High-Performance Indexing Model*. 2012. DOI: 10.2139/ssrn.3198931.

[58] Ricardo Baeza-Yates and Gonzalo Navarro. “Integrating contents and structure in text retrieval”. In: *ACM SIGMOD Record* 25.1 (1996), pp. 67–79. ISSN: 01635808. DOI: 10.1145/381854.381890.

[59] R. Guha, Rob McCool, and Eric Miller. “Semantic search”. In: *Proceedings of the 12th international conference on World Wide Web*. Ed. by Gusztáv Hencsey. New York, NY: ACM, 2003. ISBN: 1581136803. DOI: 10.1145/775152.775250.

[60] Li Ding et al. “Swoogle”. In: *Proceedings of the Thirteenth ACM conference on Information and knowledge management - CIKM ’04*. Ed. by David Grossman et al. New York, New York, USA: ACM Press, 2004, p. 652. ISBN: 1581138741. DOI: 10.1145/1031171.1031289.

[61] Eyal Oren et al. “Sindice.com: a document-oriented lookup index for open linked data”. In: *International Journal of Metadata Semantics and Ontologies* 3.1 (2008), p. 37. ISSN: 1744-2621. DOI: 10.1504/IJMSO.2008.021204. URL: https://www.researchgate.net/publication/220094840_Sindicecom_A_document-oriented_lookup_index_for_open_linked_data.

[62] Andreas Harth et al. *SWSE: Objects before documents*. 2012. URL: https://www.researchgate.net/publication/253250801_SWSE_Objects_before_documents.

[63] Gong Cheng, Weiyi Ge, and Yuzhong Qu. “Falcons: searching and browsing entities on the semantic web”. In: 2008, pp. 1101–1102. DOI: 10.1145/1367497.1367676. URL: https://www.researchgate.net/publication/221022206_Falcons_searching_and_browsing_entities_on_the_semantic_web.

[64] Saeedeh Shekarpour et al. “Keyword-Driven SPARQL Query Generation Leveraging Background Knowledge”. In: 2011, pp. 203–210. DOI: 10.1109/WI-IAT.2011.70. URL: https://www.researchgate.net/publication/224261676_Keyword-Driven_SPARQL_Query_Generation_Leveraging_Background_Knowledge.

[65] Emil Kuric, Javier D. Fernández, and Olha Drozd. “Knowledge Graph Exploration: A Usability Evaluation of Query Builders for Laypeople”. In: *Semantic Systems. The Power of AI and Knowledge Graphs*. Ed. by Maribel Acosta, Philippe Cudré-Mauroux, and Maria Maleshkova. Information Systems and Applications, incl. Internet/Web, and HCI. Cham: Springer International Publishing, 2019, pp. 326–342. ISBN: 978-3-030-33220-4.

[66] Judie Attard et al. “ExConQuer: Lowering barriers to RDF and Linked Data re-use”. In: *Semantic Web* 9.Nr.2 (2018), pp. 241–255. ISSN: 1570-0844.

[67] Patrick Hoefler et al. “Linked data query wizard: A novel interface for accessing sparql endpoints”. In: *7th Workshop on Linked Data on the Web 2014* (2014). URL: https://research.utwente.nl/en/publications/linked-data-query-wizard-a-novel-interface-for-accessing-sparql-e.

[68] *VizQuery - Hay's tools*. URL: https://hay.toolforge.org/vizquery/ (visited on 08/28/2020).

[69] *Wikidata Query Service*. URL: https://query.wikidata.org/ (visited on 08/28/2020).

[70] *OpenLink iSPARQL*. URL: http://dbpedia.org/isparql/ (visited on 08/28/2020).

[71] Paul R Smart et al. “A Visual Approach to Semantic Query Design Using a Web-Based Graphical Query Designer”. In: 2008. DOI: 10.1007/978-3-540-87696-0_25. URL: https://www.researchgate.net/publication/39996370_A_Visual_Approach_to_Semantic_Query_Design_Using_a_Web-Based_Graphical_Query_Designer.

[72] Ahmet Soylu et al. “OptiqueVQS: A visual query system over ontologies for industry”. In: *Semantic Web* 9.5 (2018), pp. 627–660. ISSN: 1570-0844. DOI: 10.3233/SW-180293. URL: https://content.iospress.com/articles/semantic-web/sw293.

[73] Florian Haag et al. “QueryVOWL: Visual Composition of SPARQL Queries”. In: *The semantic web: ESWC 2015 satellite events*. Ed. by Fabien Gandon et al. Lecture notes in computer science Computer communication networks and telecommunications. Cham, Heidelberg, and New York: Springer, 2015, pp. 62–66. ISBN: 978-3-319-25638-2. DOI: 10.1007/978-3-319-25639-9_12. URL: https://www.researchgate.net/publication/305856773_QueryVOWL_Visual_Composition_of_SPARQL_Queries.

[74] Esther Kaufmann, Abraham Bernstein, and Lorenz Fischer. “NLP-Reduce: A naive but domainindependent natural language interface for querying ontologies”. In: *4th European Semantic Web Conference ESWC*. 2007, pp. 1–2.

[75] Sébastien Ferré. “Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language”. In: *Semantic Web* 8.3 (2016), pp. 405–418. ISSN: 1570-0844. DOI: 10.3233/SW-150208. URL: https://www.researchgate.net/publication/311481197_Sparklis_An_expressive_query_builder_for_SPARQL_endpoints_with_guidance_in_natural_language.

[76] Mohamed Yahya et al. *Natural Language Questions for the Web of Data*. 2012. URL: https://www.researchgate.net/publication/267557990_Natural_Language_Questions_for_the_Web_of_Data.

[77] Mohnish Dubey et al. “AskNow: A Framework for Natural Language Query Formalization in SPARQL”. In: *The semantic web*. Ed. by Harald Sack et al. Lecture notes in computer science Theoretical computer science and general issues. Cham and Heidelberg: Springer, 2016, pp. 300–316. ISBN: 978-3-319-34128-6. DOI: 10.1007/978-3-319-34129-3_19. URL: https://www.researchgate.net/publication/303098348_AskNow_A_Framework_for_Natural_Language_Query_Formalization_in_SPARQL.

[78] Camille Pradel, Ollivier Haemmerlé, and Nathalie Hernandez. “Demo: Swip, a Semantic Web Interface using Patterns”. In: *undefined* (2013). URL: https://www.semanticscholar.org/paper/Demo%3A-Swip%2C-a-Semantic-Web-Interface-using-Patterns-Pradel-Haemmerl%C3%A9/03154b67d3a7f5f13bb40a5d474678ebe0ec8a3f.

[79] Majid Latifi, Horacio rodriguez, and Miquel Sànchez-Marrè. “ScoQAS: A Semantic-based Closed and Open Domain Question Answering System”. In: *Procesamiento de Lenguaje Natural* 59 (2017), pp. 73–80. ISSN: 1989-7553. URL: https://www.researchgate.net/publication/319702899_ScoQAS_A_Semantic-based_Closed_and_Open_Domain_Question_Answering_System.

[80] Svitlana Vakulenko et al. “Message passing for complex question answering over knowledge graphs”. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019, pp. 1431–1440.

[81] Chenhao Zhu et al. “A Graph Traversal Based Approach to Answer Non-Aggregation Questions over DBpedia”. In: *Semantic technology*. Ed. by Guilin Qi. Lecture notes in computer science Information systems and applications, incl. Internet/Web, and HCI. Cham and Heidelberg: Springer, 2016, pp. 219–234. ISBN: 978-3-319-31676-5.

[82] Marti A. Hearst. “UIs for Faceted Navigation Recent Advances and Remaining Open Problems”. In: (2008). URL: https://www.semanticscholar.org/paper/UIs-for-Faceted-Navigation-Recent-Advances-and-Open-Hearst/a2a03cb956b7c69103cc6f8c972546582c049b62.

[83] Jennifer English et al. *Flexible Search and Navigation using Faceted Metadata*. 2002. URL: https://www.researchgate.net/publication/228751407_Flexible_Search_and_Navigation_using_Faceted_Metadata.

[84] Ivan Heibi, Silvio Peroni, and David Shotton. “Enabling text search on SPARQL endpoints through OSCAR”. In: *Data Science* 2.1-2 (2019), pp. 205–227. ISSN: 24518484. DOI: 10.3233/DS-190016.

[85] Marcelo Arenas et al. “Faceted search over RDF-based knowledge graphs”. In: *Journal of Web Semantics* 37-38 (2016), pp. 55–74. ISSN: 15708268. DOI: 10.1016/j.websem.2015.12.002.

[86] Stéphane Campinas. “Graph summarisation of web data: data-driven generation of structured representations”. In: (2016). URL: https://www.semanticscholar.org/paper/Graph-summarisation-of-web-data%3A-data-driven-of-Campinas/6a6fec6cb64c9f9aa7e3e73fd301eb7c80565ef0.

[87] Yuangui Lei, Victoria Uren, and Enrico Motta. “SemSearch: A Search Engine for the Semantic Web”. In: *Managing knowledge in a world of networks*. Ed. by Steffen Staab and Vojtěch Svátek. Vol. 4248. Lecture notes in computer science Lecture notes in artificial intelligence. Berlin: Springer, 2006, pp. 238–245. ISBN: 978-3-540-46363-4. DOI: 10.1007/11891451_22.

[88] Saeedeh Shekarpour et al. *SINA: Semantic Interpretation of User Queries for Question Answering on Interlinked Data*. 2015. DOI: 10.2139/ssrn.3199174.

[89] Jens Lehmann and Lorenz Bühmann. “AutoSPARQL: Let Users Query Your Knowledge Base”. In: *The semantic web: research and applications*. Ed. by Grigoris Antoniou et al. Lecture Notes in Computer Science. Berlin: Springer, 2011, pp. 63–79. ISBN: 978-3-642-21034-1.

[90] *Linked swissbib - swissbib*. URL: http://www.swissbib.org/wiki/index.php?title=Linked_swissbib (visited on 08/31/2020).

[91] Markus Mandalka. *Open Semantic Search: Your own search engine for documents, images, tables, files, intranet & news*. URL: https://www.opensemanticsearch.org// (visited on 08/31/2020).

[92] Lei Zhang et al. "Semplore: An IR Approach to Scalable Hybrid Query of Semantic Web Data". In: *The semantic web*. Ed. by Karl Aberer. Lecture Notes in Computer Science. Berlin: Springer, 2007, pp. 652–665. ISBN: 978-3-540-76298-0.

[93] Cristiano Rocha, Daniel Schwabe, and Marcus Poggi Aragao. "A hybrid approach for searching in the semantic web". In: *13th International Conference on World Wide Web Conference*. Ed. by Stuart Feldman et al. New York, N.Y.: ACM Press, 2005, p. 374. ISBN: 158113844X. DOI: 10.1145/988672.988723.

[94] *Handle.Net Registry*. URL: https://www.handle.net/ (visited on 08/24/2020).

[95] *The Organization Ontology*. URL: https://www.w3.org/TR/vocab-org/ (visited on 08/13/2020).

[96] *GitHub - zazuko/rdf-vocabularies: Zazuko's Default Ontologies & Prefixes*. URL: https://github.com/zazuko/rdf-vocabularies (visited on 08/24/2020).

[97] *VocabularyMarket - W3C Wiki*. URL: https://www.w3.org/wiki/VocabularyMarket (visited on 08/24/2020).

[98] *CSMD: the Core Scientific Metadata Model*. URL: http://icatproject-contrib.github.io/CSMD/csmd-4.0.html (visited on 08/24/2020).

[99] *SPDX License List | Software Package Data Exchange (SPDX)*. URL: https://spdx.org/licenses/ (visited on 08/24/2020).

[100] *DCMI: DCMI Metadata Terms*. URL: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/#http://purl.org/dc/terms/DCMIType (visited on 08/24/2020).

[101] DFG. *DFG - Deutsche Forschungsgemeinschaft -*. URL: https://www.dfg.de/dfg_profil/gremien/fachkollegien/faecher (visited on 08/24/2020).

[102] *EngMeta - Beschreibung von Forschungsdaten | Informations- und Kommunikationszentrum | Universität Stuttgart*. German. URL: https://www.izus.uni-stuttgart.de/fokus/engmeta/ (visited on 08/24/2020).

[103] *FDM-Projekt DIPL-ING | IZUS / Universitätsbibliothek Stuttgart | Universität Stuttgart*. URL: http://www.ub.uni-stuttgart.de/dipling# (visited on 08/24/2020).

[104] *Form Generation using SHACL and DASH*. URL: http://datashapes.org/forms.html (visited on 08/24/2020).

[105] *vos.openlinksw.com/owiki/wiki/VOS/VOSTriple*. URL: http://vos.openlinksw.com/owiki/wiki/VOS/VOSTriple (visited on 08/24/2020).

[106] Sarah Bensberg. “Evaluation verschiedener Triple-Stores zum Speichern von Metadaten im Kontext des Forschungsdatenmanagements”. Seminararbeit. FH Aachen, 2017.

[107] *Projekt „Applying Interoperable Metadata Standards“ wurde bewilligt « Forschungsdaten – Aktuelles und Wissenswertes*. URL: https://blog.rwth-aachen.de/forschungsdaten/2020/06/03/projekt-applying-interoperable-metadata-standards-wurde-bewilligt/ (visited on 09/17/2020).

[108] International Organization for Standardization. *ISO 9000:2015(en), Quality management systems — Fundamentals and vocabulary*. URL: https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-4:v1:en (visited on 08/14/2020).

[109] *ISO 9241-11:2018(en), Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*. URL: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en (visited on 09/29/2020).

[110] Jakob Nielsen. *Usability engineering*. Boston: Academic Press, 1993. ISBN: 0125184050.

[111] *SHACL Advanced Features*. URL: https://www.w3.org/TR/shacl-af/#rules (visited on 09/01/2020).

[112] *16.3.1. Using Full Text Search in SPARQL*. URL: http://docs.openlinksw.com/virtuoso/rdfsparqlrulefulltext/ (visited on 09/03/2020).

[113] Pavel Grafkin et al. “SPARQL Query Builders : Overview and Comparison”. In: CEUR-WS, 2016. URL: http://hj.diva-portal.org/smash/record.jsf?pid=diva2%3A1070495&dswid=6004 (visited on 02/01/2017).

[114] Adam Styperek et al. “Evaluation of SPARQL-compliant semantic search user interfaces”. In: *Vietnam Journal of Computer Science* 2.3 (2015), pp. 191–199. ISSN: 2196-8896. DOI: 10.1007/s40595-015-0044-y.

[115] *What is Elasticsearch? | Elasticsearch Reference [master] | Elastic*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/master/elasticsearch-intro.html (visited on 09/21/2020).

[116] *Removal of mapping types | Elasticsearch Reference [7.9] | Elastic*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/removal-of-types.html (visited on 09/03/2020).

[117] *Meta data enrichment or annotator from Resource Description Framework (RDF) to Solr or Elasticsearch | Open Semantic Search*. URL: https://www.opensemanticsearch.org/enhancer/rdf (visited on 09/01/2020).

[118] *DBpedia*. URL: https://wiki.dbpedia.org/ (visited on 08/16/2020).

[119] *DCMI: DCMI Metadata Terms*. (Accessed on 09/05/2020). URL: https://dublincore.org/specifications/dublin-core/dcmi-terms/#section-7 (visited on 09/05/2020).

[120] *Media Types*. URL: https://www.iana.org/assignments/media-types/media-types.xhtml (visited on 09/05/2020).

[121] *dotNetRDF*. URL: https://www.dotnetrdf.org/ (visited on 09/14/2020).

[122] J. B. Lovins. “Development of a stemming algorithm”. In: *undefined* (1968). URL: https://www.semanticscholar.org/paper/Development-of-a-stemming-algorithm-Lovins/6b3853f08c482fe1bfbe39d656d50a8c73976f3c.

[123] Thomas Kurz. *Including Filters into SPARQL Optimization Process – Mico Project*. 2016. URL: https://www.mico-project.eu/including-filters-into-sparql-optimization-process/ (visited on 09/08/2020).

[124] Don Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[125] Max L. Wilson. *Search user interface design*. Vol. 20. Synthesis lectures on information concepts, retrieval, and services. San Rafael, Calif.: Morgan & Claypool, 2012. ISBN: 9781608456901.

[126] Shiroq Al-Megren, Joharah Khabti, and Hend S. Al-Khalifa. “A Systematic Review of Modifications and Validation Methods for the Extension of the Keystroke-Level Model”. In: *Advances in Human-Computer Interaction* 2018 (2018), pp. 1–26. ISSN: 1687-5893. DOI: 10.1155/2018/7528278. URL: https://www.hindawi.com/journals/ahci/2018/7528278/.

[127] David Kieras. *Using the Keystroke-Level Model to Estimate Execution Times*. 2003. URL: https://www.researchgate.net/publication/2848715_Using_the_Keystroke-Level_Model_to_Estimate_Execution_Times.

[128] Marcelo Arenas et al. “SemFacet: semantic faceted search over yago”. In: 2014, pp. 123–126. DOI: 10.1145/2567948.2577011. URL: https://www.researchgate.net/publication/261959905_SemFacet_semantic_faceted_search_over_yago.

[129] Jeff Sauro. “Estimating Productivity: Composite Operators for Keystroke Level Modeling”. In: *Human-computer interaction*. Ed. by Julie A. Jacko. Lecture Notes in Computer Science. Berlin: Springer, 2009, pp. 352–361. ISBN: 978-3-642-02574-7.

[130] *Query string query | Elasticsearch Reference [7.9] | Elastic*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html (visited on 09/28/2020).

[131] *Similarity module | Elasticsearch Reference [7.9] | Elastic*. (Accessed on 10/05/2020). 2020. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-similarity.html (visited on 10/06/2020).

[132] André Freitas, Seán O’Riáin, and Edward Curry. “Querying and Searching Heterogeneous Knowledge Graphs in Real-time Linked Dataspaces”. In: *Real-time linked dataspaces*. Ed. by Edward Curry. Cham, Switzerland: SpringerOpen, 2020, pp. 105–124. ISBN: 978-3-030-29664-3. DOI: 10.1007/978-3-030-29665-0_7. URL: https://www.researchgate.net/publication/337362086_Querying_and_Searching_Heterogeneous_Knowledge_Graphs_in_Real-time_Linked_Dataspaces.

[133] *Libraries for developing modern search experiences | Elastic*. URL: https://www.elastic.co/de/enterprise-search/search-ui (visited on 09/29/2020).

[134] *Suggesters | Elasticsearch Reference [7.9] | Elastic*. 2020. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-suggesters.html (visited on 10/06/2020).

[135] *Nested field type | Elasticsearch Reference [7.9] | Elastic*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/nested.html (visited on 09/29/2020).

G Appendix

All turtle and SPARQL excerpts listed in the master thesis are truncated and contain only the relevant information without prefix definition and header. Table G.1 contains the prefixes to be assumed.

Table G.1: Prefixes for turtle and SPARQL excerpts

Prefix	Url
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
xsd	http://www.w3.org/2001/XMLSchema#
dcterms	http://purl.org/dc/terms/
engmeta	http://www.ub.uni-stuttgart.de/dipling#
prov	http://www.w3.org/ns/prov#
premis3	http://www.loc.gov/premis/rdf/v3/
dctype	http://purl.org/dc/dcmitype/
mode	https://purl.org/coscine/terms/mode#
license	http://spdx.org/licenses/
format	https://www.iana.org/assignments/media-types/application/
csmd	http://www.purl.org/net/CSMD/4.0#
coscineprojectstructure	https://purl.org/coscine/terms/projectstructure#
coscinevariable	https://purl.org/coscine/terms/variable#
coscinesearch	ttps://purl.org/coscine/terms/search#
coscineresource	https://purl.org/coscine/vocabularies/resource#

G.1 Guideline SPARQL Queries for Search Intentions

In the following, the SPARQL queries for each search intention (see section 5.2) are listed, whereby the language-specific supplement and the triple pattern identification of metadata records are omitted.

Data records which were published at the IT Center:

```
SELECT ?s WHERE {
  {
    SELECT ?s {
      ?s dcterms:publisher <https://www.rwth-aachen.de/22000> .
    }
  }
  UNION
  {
    SELECT ?s {
      ?s dcterms:publisher ?person .
```

```
      ?membership a org:Membership.
      ?membership org:member ?person.
      ?membership org:organization <https://www.rwth-aachen.de/22000>.
    }
  }
}
```

Data records with version number 10:

```
SELECT ?s WHERE {
  ?s engmeta:version 10 .
}
```

Data records which were created in 2007:

```
SELECT ?s WHERE {
  ?s dcterms:created ?date .
  FILTER (?date >= xsd:date("2007-01-01") && ?date < xsd:date("2008-01-01"))
}
```

Data records about design patterns and with version number less than 5:

```
SELECT ?s WHERE {
  {
    SELECT ?s {
      ?s dcterms:title ?title .
      ?title bif:contains "'design patterns'" .
    }
  }
  UNION
  {
    SELECT ?s {
      ?s dcterms:abstract ?abstract .
      ?abstract bif:contains "'design patterns'" .
    }
  }
  ?s engmeta:version ?version .
  FILTER (?version < 5) .
}
```

Data records about computer science:

```
SELECT ?s WHERE {
  {
    SELECT ?s {
      ?s dcterms:abstract ?abstract .
      ?abstract bif:contains "'computer science'" .
    }
  }
  UNION
  {
    SELECT ?s {
      ?s dcterms:subject ?subject .
      ?class rdfs:subClassOf* ?superclass .
      ?subject a ?class .
```

```
      FILTER (STR(?superclass) =
        "http://www.dfg.de/.../liste/index.jsp?id=409") .
    }
  }
}
```

The newest data record of resource 2:

```
SELECT ?s WHERE {
  ?s engmeta:version ?version .
  ?s coscineprojectstructure:isFileOf
    <https://purl.org/coscine/vocabularies/resource#resource2> .
  {
    SELECT ?newestVersion WHERE {
      ?file engmeta:version ?newestVersion .
      ?file coscineprojectstructure:isFileOf
        <https://purl.org/coscine/vocabularies/resource#resource2> .
    } ORDER BY DESC(?newestVersion) LIMIT 1
  } .
  FILTER( ?version = ?newestVersion) .
}
```

Data records which are experiments or simulations and not analysis:

```
SELECT ?s WHERE {
  ?s engmeta:mode ?mode .
  FILTER ((?mode = <https://purl.org/coscine/terms/mode#experiment> ||
    ?mode = <https://purl.org/coscine/terms/mode#simulation>) && ?mode != 
    <https://purl.org/coscine/terms/mode#analysis>) .
}
```

Data records about non computability of the human consciousness:

```
SELECT ?s WHERE {
  ?s dcterms:abstract ?abstract .
  ?abstract bif:contains "'human consciousness' AND non-algorithmic" .
}
```

Data records which are available since 03.07.2020:

```
SELECT ?s WHERE {
  ?s dcterms:available "2020-07-03+02:00"^^xsd:date .
}
```

Data records about political left:

```
SELECT ?s WHERE {
  ?s dcterms:abstract ?abstract .
  ?abstract bif:contains "left" .
}
```

Data records which contain a variable with the value 2 meters:

```
SELECT ?s WHERE {
  {
    ?s engmeta:controlledVariable
        <https://purl.org/coscine/terms/variable#variable1>.
  }
  UNION
  {
    ?s engmeta:measuredVariable
        <https://purl.org/coscine/terms/variable#variable1>.
  }
}
```

Data records about object-oriented software which are published before 2015

```
SELECT ?s WHERE {
  ?s dcterms:issued ?date .
  ?s dcterms:abstract ?abstract .
  ?abstract bif:contains "'object-oriented software'" .
  FILTER (?date <= xsd:date("2015-01-01")) .
}
```

Data records which are published by Thomas Pynchon, created in 2016 and about object-oriented software:

```
SELECT ?s WHERE {
  ?s dcterms:publisher <http://dbpedia.org/resource/Thomas_Pynchon> .
  ?s dcterms:created ?date .
  ?s dcterms:abstract ?abstract .
  ?abstract bif:contains "'object-oriented software'" .
  FILTER (?date >= xsd:date("2016-01-01") && ?date <
     xsd:date("2017-01-01")).
}
```

G.2 Search Inputs and Confusion Matrices for Evaluating Effectiveness

In the following, the user input or the translated query (search input) and the corresponding confusion matrices are listed for each search intention and approach.

Search Inputs of Full-Text Search in RDF Literals

Table G.2: Search inputs for *regex* whereby words separated by spaces (that are not in quotation marks) are automatically linked with the logical operation `OR`

Search Intention	**Search Input**
Data records which were published at the IT Center	'IT Center'
Data records with version number 10	^10$
Data records which were created in 2007	2007
Data records about design patterns and with version number less than 5	'design patterns' ^[0-4]$
Data records about computer science	'computer science'
The newest data record of resource 2	'resource 2'
Data records which are experiments or simulations and not analysis	experiment simulation
Data records about non computability of the human consciousness	'human consciousness' non-algorithmic
Data records which are available since 03.07.2020	2020-07-03
Data records about political left	left
Data records which contain a variable with the value 2 meters	'2 meter'
Data records about object-oriented software which are published before 2015	'object-oriented software'
Data records which are published by Thomas Pynchon, created in 2016 and about object-oriented software	'Thomas Pynchon' 2020 'object-oriented software'

Table G.3: Search inputs for *bif:contains*

Search Intention	**Search Input**
Data records which were published at the IT Center	'IT Center'
Data records with version number 10	'10'
Data records which were created in 2007	'2007'
Data records about design patterns and with version number less than 5	'design patterns'
Data records about computer science	'computer science'
The newest data record of resource 2	'resource 2'
Data records which are experiments or simulations and not analysis	(experiment OR simulation) AND NOT analysis
Data records about non computability of the human consciousness	'human consciousness' AND non-algorithmic
Data records which are available since 03.07.2020	'2020-07-03'
Data records about political left	left
Data records which contain a variable with the value 2 meters	'2 meter'
Data records about object-oriented software which are published before 2015	'object-oriented software'

Table G.3: Search inputs for *bif:contains* (continued)

Search Intention	Search Input
Data records which are published by Thomas Pynchon, created in 2016 and about object-oriented software	'Thomas Pynchon' OR '2016' OR 'object-oriented software'

Search Inputs of SPARQL Query Builder

The generated SPARQL queries for the SPARQL query builder approach are listed in the following. Only the queries that differ from the quideline are listed.

The newest data record of resource 2:

```
SELECT ?s WHERE {
  ?s coscineprojectstructure:isFileOf
     <https://purl.org/coscine/vocabularies/resource#resource2> .
  ?file engmeta:version ?version .
} ORDER BY DESC(?version) LIMIT 1
```

Data records which are experiments or simulations and not analysis:

```
SELECT DISTINCT ?s WHERE {
  {
    SELECT ?s {
      ?s engmeta:mode <https://purl.org/coscine/terms/mode#experiment>
    }
  }
  UNION
  {
    SELECT ?s {
    ?s engmeta:mode <https://purl.org/coscine/terms/mode#simulation>
  }
  ?s engmeta:mode ?mode .
  FILTER (?mode != <https://purl.org/coscine/terms/mode#analysis>) .
}
```

Data records which contain a variable with the value 2 meters:

```
SELECT DISTINCT ?s WHERE {
  {
    SELECT ?s {
      ?s engmeta:controlledVariable ?var .
      ?var qudt:unit unit:M .
      ?var qudt:value "2" .
    }
  }
  UNION
  {
    SELECT ?s {
      ?s engmeta:measuredVariable ?var .
      ?var qudt:unit unit:M .
      ?var qudt:value "2" .
    }
  }
}
```

Search Inputs of Faceted Search

The respective SPARQL addition to the *regex* approach for the faceted search can be viewed in the following. Only those search intentions are listed for which an addition could be made. The addition is automatically generated by the user selecting values in the facets.

Data records which were published at the IT Center:

```
{
  ?s dcterms:publisher <https://www.rwth-aachen.de/22000>
}
UNION
{
  ?s dcterms:publisher ?publisher .
  ?membership a org:Membership .
  ?membership org:member ?publisher .
  ?membership org:organization <https://www.rwth-aachen.de/22000>
}
```

Data records with version number 10:

```
?s engmeta:version 10 .
```

Data records which were created in 2007:

```
?s dcterms:created ?date .
FILTER (?date >= xsd:date("2007-01-01") && ?date < xsd:date("2008-01-01"))
```

Data records about design patterns and with version number less than 5:

```
?s engmeta:version 3
```

The newest data record of resource 2:

```
?s engmeta:version 10
```

Data records which are experiments or simulations and not analysis:

```
{
  ?s engmeta:mode <https://purl.org/coscine/terms/mode#experiment>
}
UNION
{
  ?s engmeta:mode <https://purl.org/coscine/terms/mode#simulation>
}
```

Data records which are available since 03.07.2020:

```
?s dcterms:available ?date .
FILTER (?date >= xsd:date("2020-01-01") && ?date < xsd:date("2021-01-01"))
```

Data records which contain a variable with the value 2 meters:

```
?s engmeta:controlledVariable ?var .
?var qudt:unit unit:M .
?var qudt:value "2" .
?s engmeta:measured ?var .
?var qudt:unit unit:M .
?var qudt:value "2"
```

Data records which are published by Thomas Pynchon, created in 2016 and about object-oriented software:

```
?s dcterms:publisher <http://dbpedia.org/resource/Thomas_Pynchon> .
?s dcterms:created ?date .
FILTER (?date >= xsd:date("2016-01-01") && ?date < xsd:date("2017-01-01"))
```

Search Inputs of Using Elasticsearch as a Search Engine

Table G.4: Search inputs for ES simple

Search Intention	Search Input
Data records which were published at the IT Center	"IT Center"
Data records with version number 10	10
Data records which were created in 2007	2007
Data records about design patterns and with version number less than 5	"design patterns"
Data records about computer science	"computer science"
The newest data record of resource 2	+"newest version true" +"resource 2"
Data records which are experiments or simulations and not analysis	experiment simulation +-analysis
Data records about non computability of the human consciousness	+"human consciousness" +non-algorithmic
Data records which are available since 03.07.2020	2020-07-03
Data records about political left	left
Data records which contain a variable with the value 2 meters	"2 meter"
Data records about object-oriented software which are published before 2015	"object-oriented software"
Data records which are published by Thomas Pynchon, created in 2016 and about object-oriented software	+"Thomas Pynchon" +2016 +"object-oriented software"

Table G.5: Search inputs for ES advanced

Search Intention	**Search Input**
Data records which were published at the IT Center	publisher:"IT Center"
Data records with version number 10	version:10
Data records which were created in 2007	date_created:[2007-01-01 TO 2007-12-31]
Data records about design patterns and with version number less than 5	"design patterns" AND version:<5
Data records about computer science	"computer science"
The newest data record of resource 2	is_file_of:"resource 2" AND is_newest_version:true
Data records which are experiments or simulations and not analysis	mode:((experiment OR simulation) AND NOT analysis)
Data records about non computability of the human consciousness	abstract:("human consciousness" AND non-algorithmic)
Data records which are available since 03.07.2020	date_available:2020-07-03
Data records about political left	left
Data records which contain a variable with the value 2 meters	controlled_variables:"2 meter" measured_variables:"2 meter"
Data records about object-oriented software which are published before 2015	abstract:"object-oriented software" AND date_issued:* TO 2015-01-01
Data records which are published by Thomas Pynchon, created in 2016 and about object-oriented software	publisher:"Thomas Pynchon" AND date_created_year:2016 AND abstract:"object-oriented software"

Table G.6: Confusion matrices for *regex*

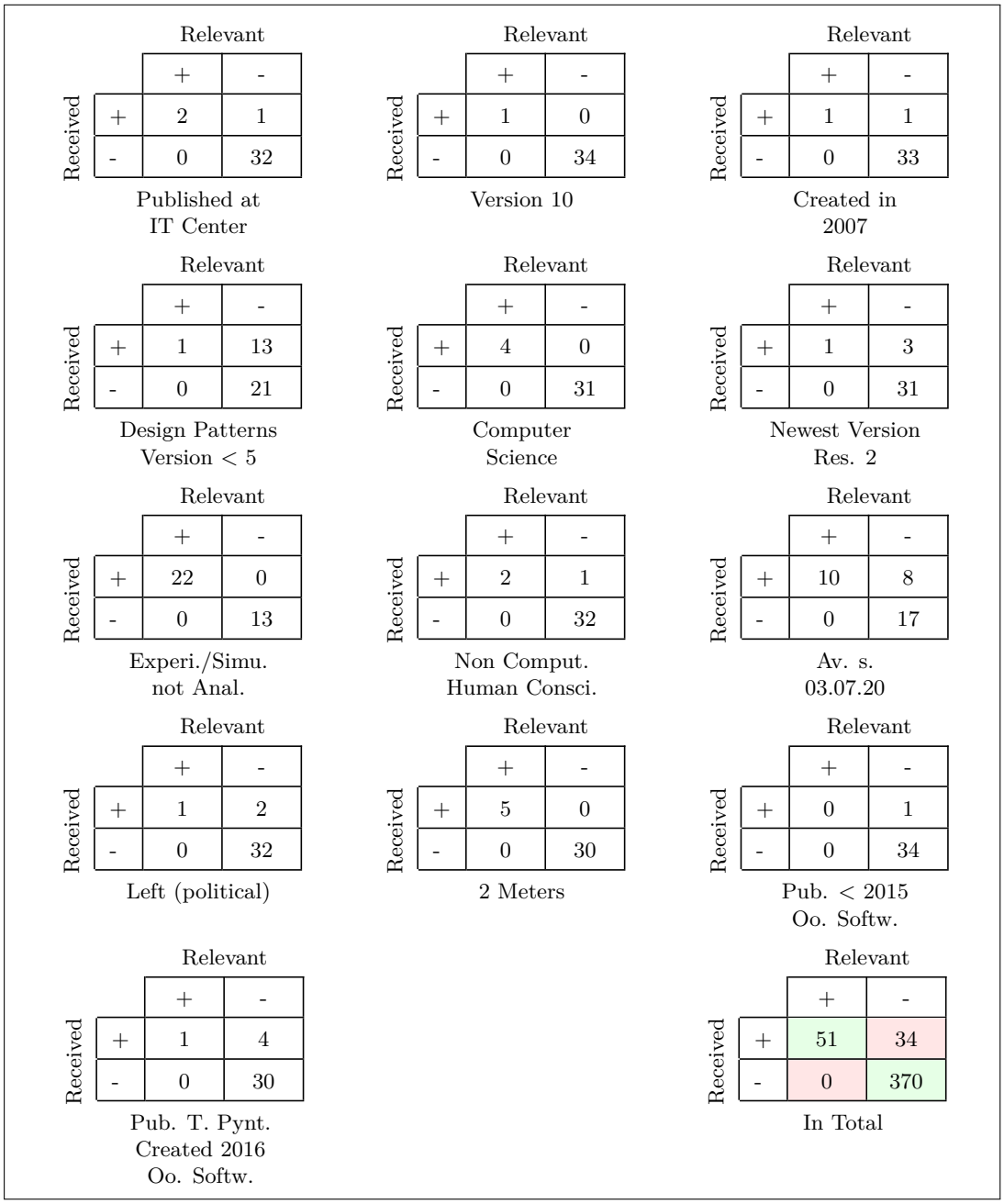


Table G.7: Confusion matrices of *bif:contains*

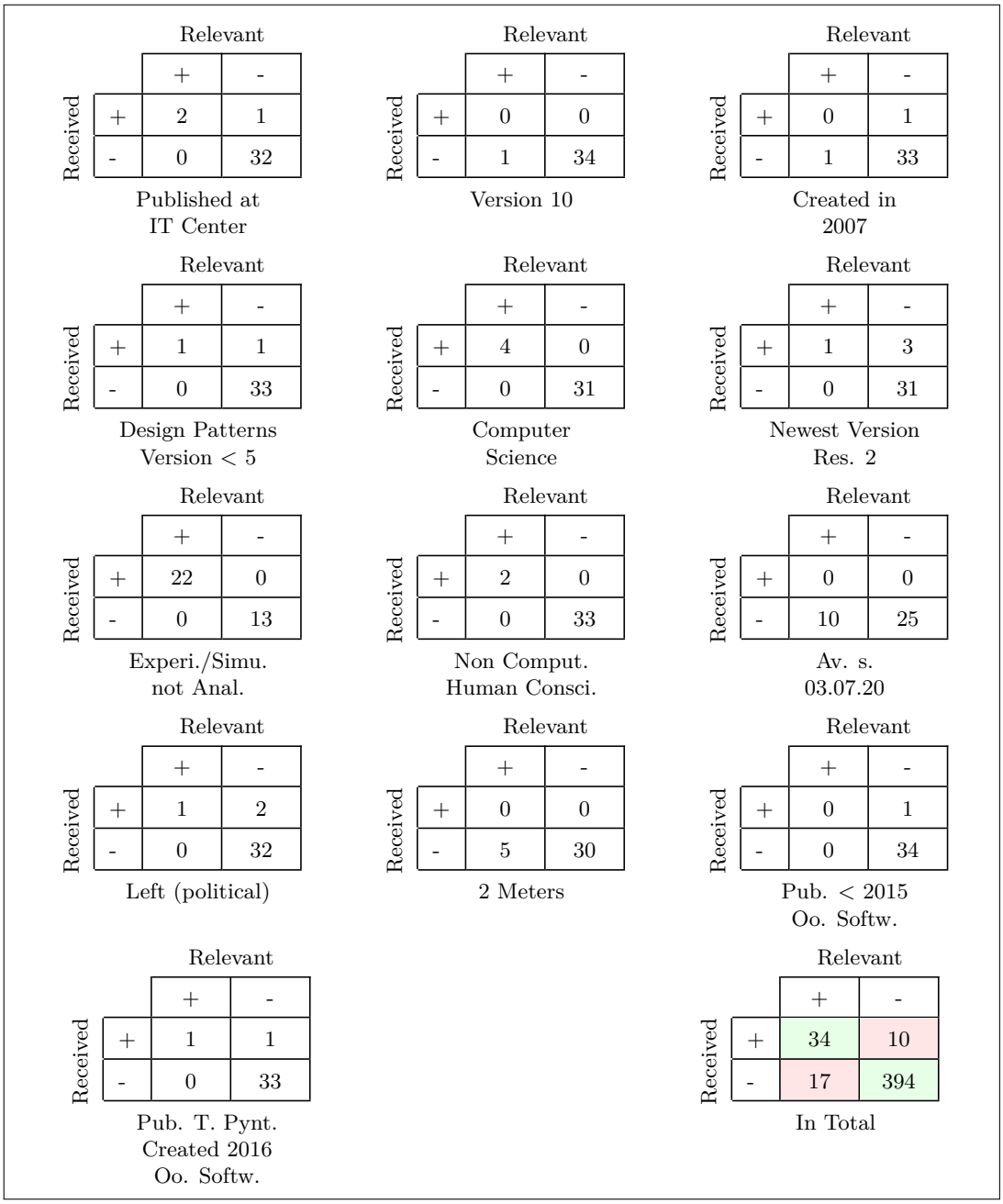
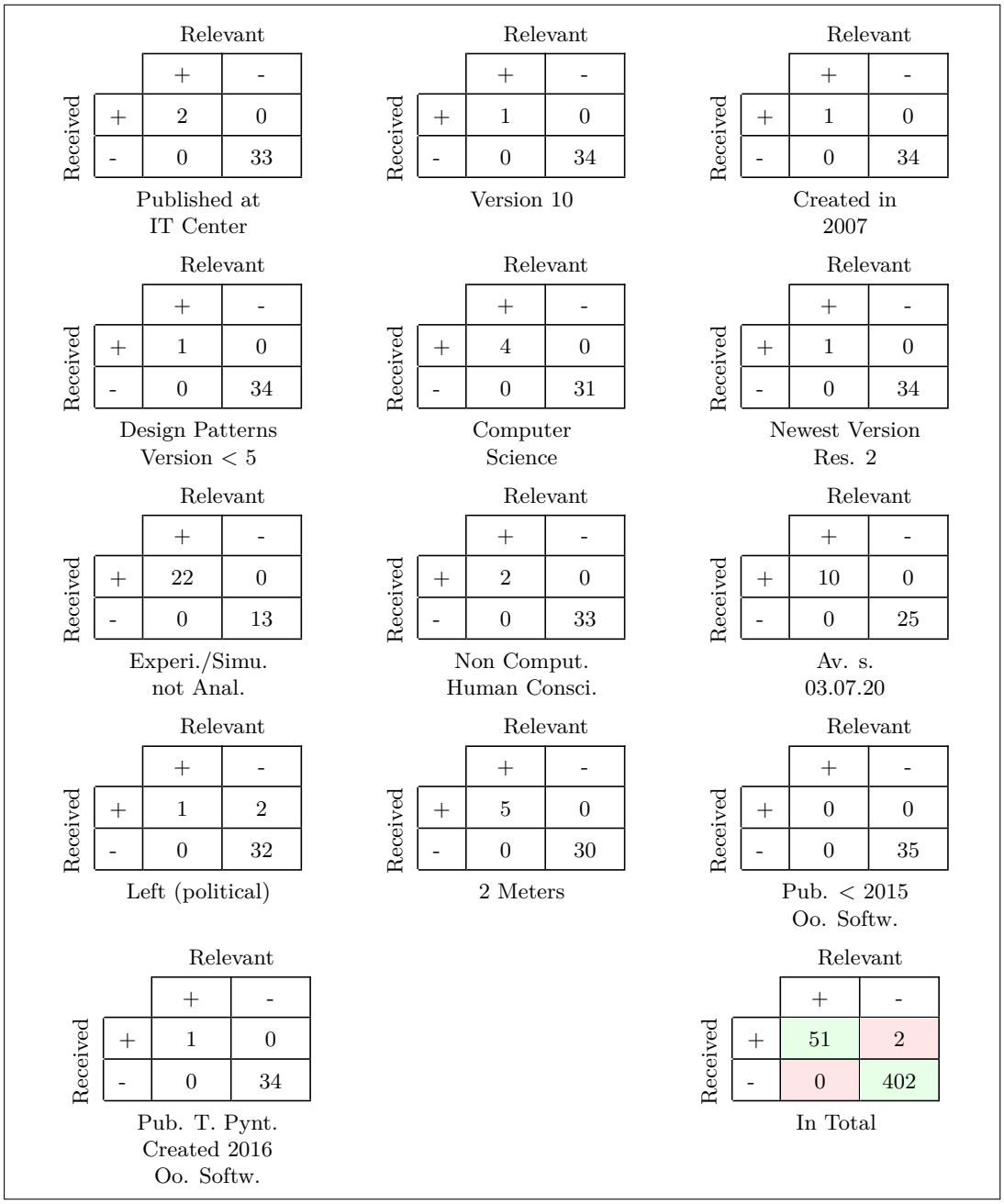


Table G.8: Confusion matrices of SPARQL query builder



Received \ Relevant	+	-
+	2	0
-	0	33

Published at IT Center

Received \ Relevant	+	-
+	1	0
-	0	34

Version 10

Received \ Relevant	+	-
+	1	0
-	0	34

Created in 2007

Received \ Relevant	+	-
+	1	0
-	0	34

Design Patterns Version < 5

Received \ Relevant	+	-
+	4	0
-	0	31

Computer Science

Received \ Relevant	+	-
+	1	0
-	0	34

Newest Version Res. 2

Received \ Relevant	+	-
+	22	0
-	0	13

Experi./Simu. not Anal.

Received \ Relevant	+	-
+	2	0
-	0	33

Non Comput. Human Consci.

Received \ Relevant	+	-
+	10	0
-	0	25

Av. s. 03.07.20

Received \ Relevant	+	-
+	1	2
-	0	32

Left (political)

Received \ Relevant	+	-
+	5	0
-	0	30

2 Meters

Received \ Relevant	+	-
+	0	0
-	0	35

Pub. < 2015 Oo. Softw.

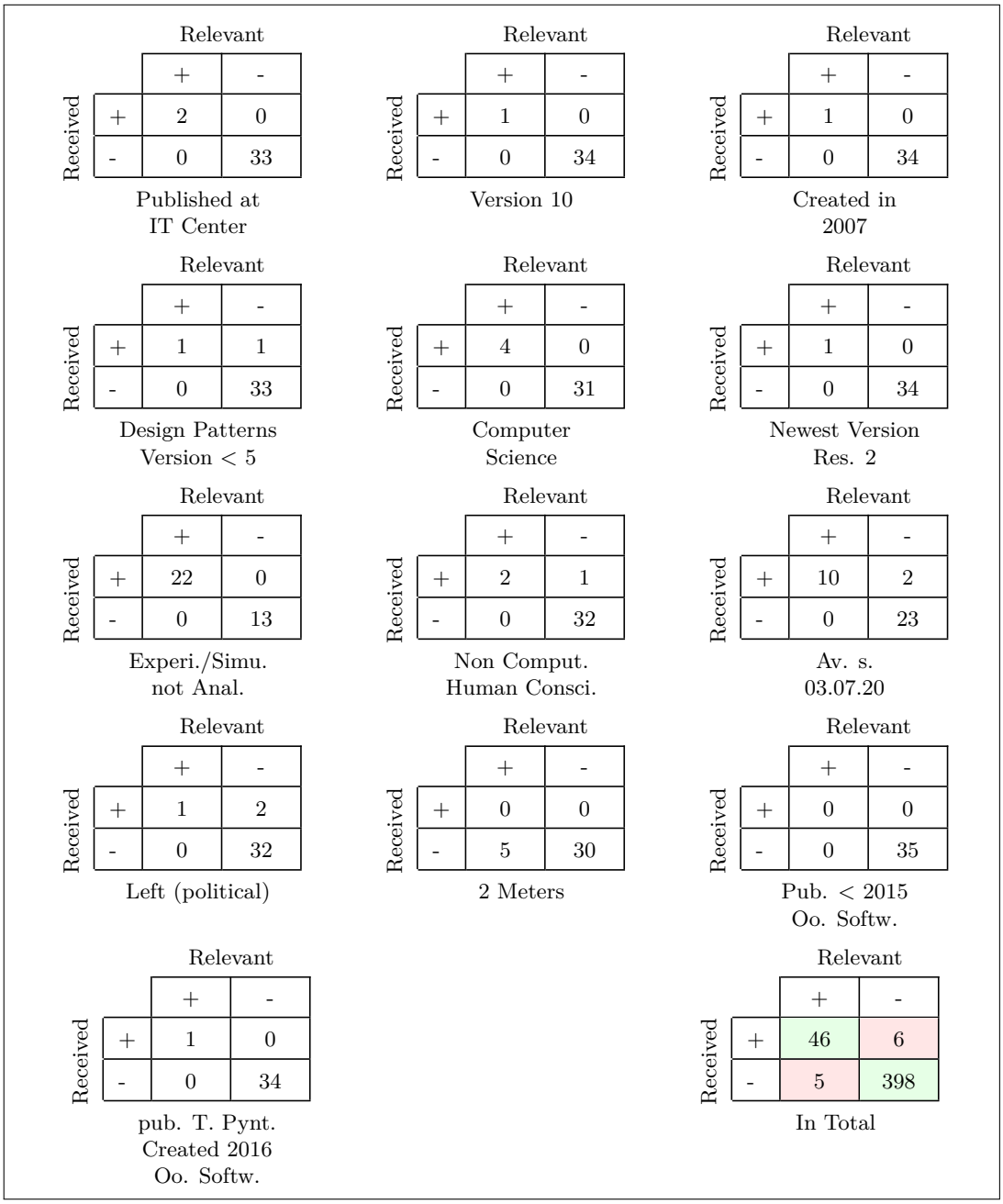
Received \ Relevant	+	-
+	1	0
-	0	34

Pub. T. Pynt. Created 2016 Oo. Softw.

Received \ Relevant	+	-
+	51	2
-	0	402

In Total

Table G.9: Confusion matrices of faceted search



Received \ Relevant	+	-
+	2	0
-	0	33

Published at IT Center

Received \ Relevant	+	-
+	1	0
-	0	34

Version 10

Received \ Relevant	+	-
+	1	0
-	0	34

Created in 2007

Received \ Relevant	+	-
+	1	1
-	0	33

Design Patterns Version < 5

Received \ Relevant	+	-
+	4	0
-	0	31

Computer Science

Received \ Relevant	+	-
+	1	0
-	0	34

Newest Version Res. 2

Received \ Relevant	+	-
+	22	0
-	0	13

Experi./Simu. not Anal.

Received \ Relevant	+	-
+	2	1
-	0	32

Non Comput. Human Consci.

Received \ Relevant	+	-
+	10	2
-	0	23

Av. s. 03.07.20

Received \ Relevant	+	-
+	1	2
-	0	32

Left (political)

Received \ Relevant	+	-
+	0	0
-	5	30

2 Meters

Received \ Relevant	+	-
+	0	0
-	0	35

Pub. < 2015 Oo. Softw.

Received \ Relevant	+	-
+	1	0
-	0	34

pub. T. Pynt. Created 2016 Oo. Softw.

Received \ Relevant	+	-
+	46	6
-	5	398

In Total

Table G.10: Confusion matrices of the simple variant of ES

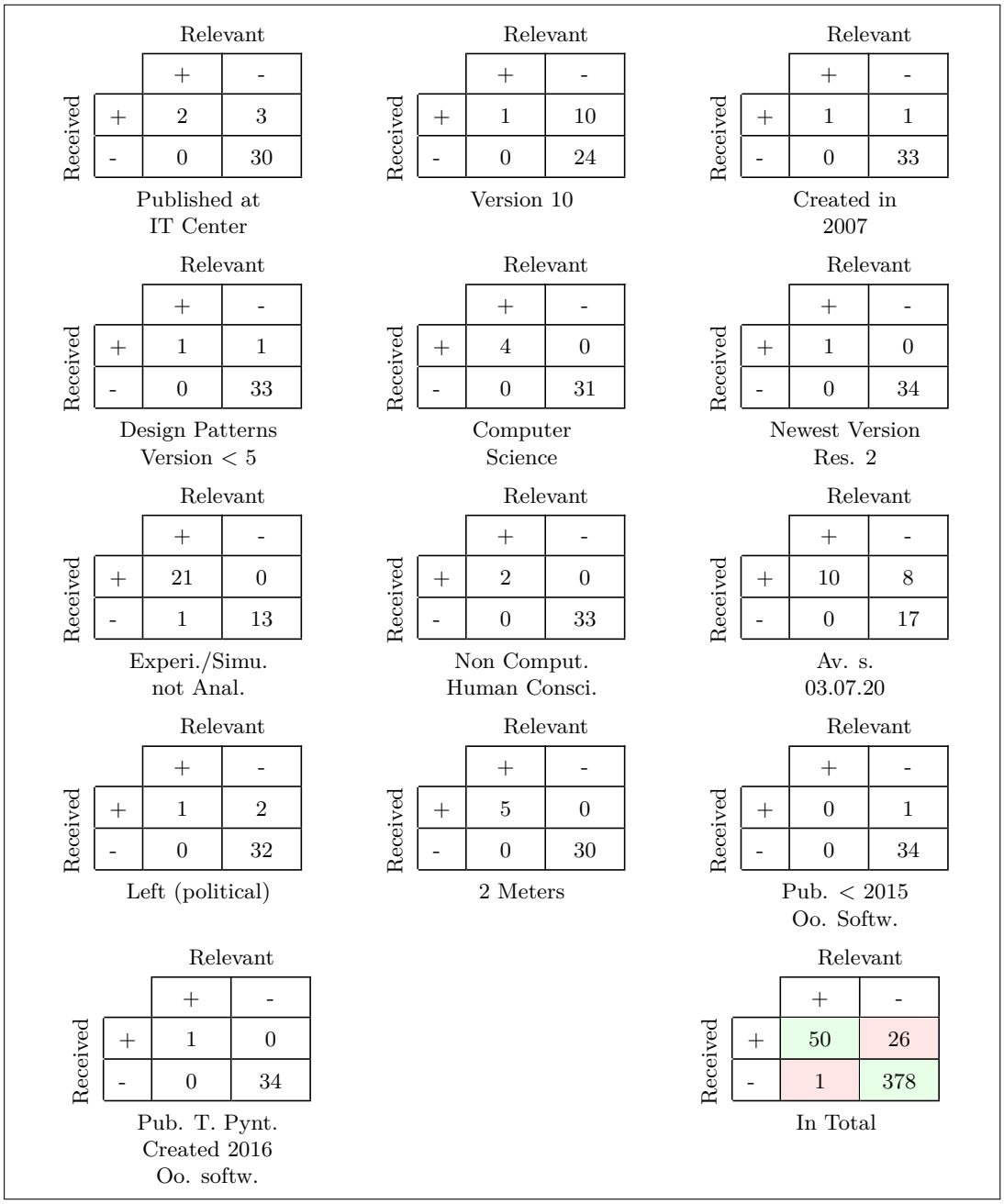
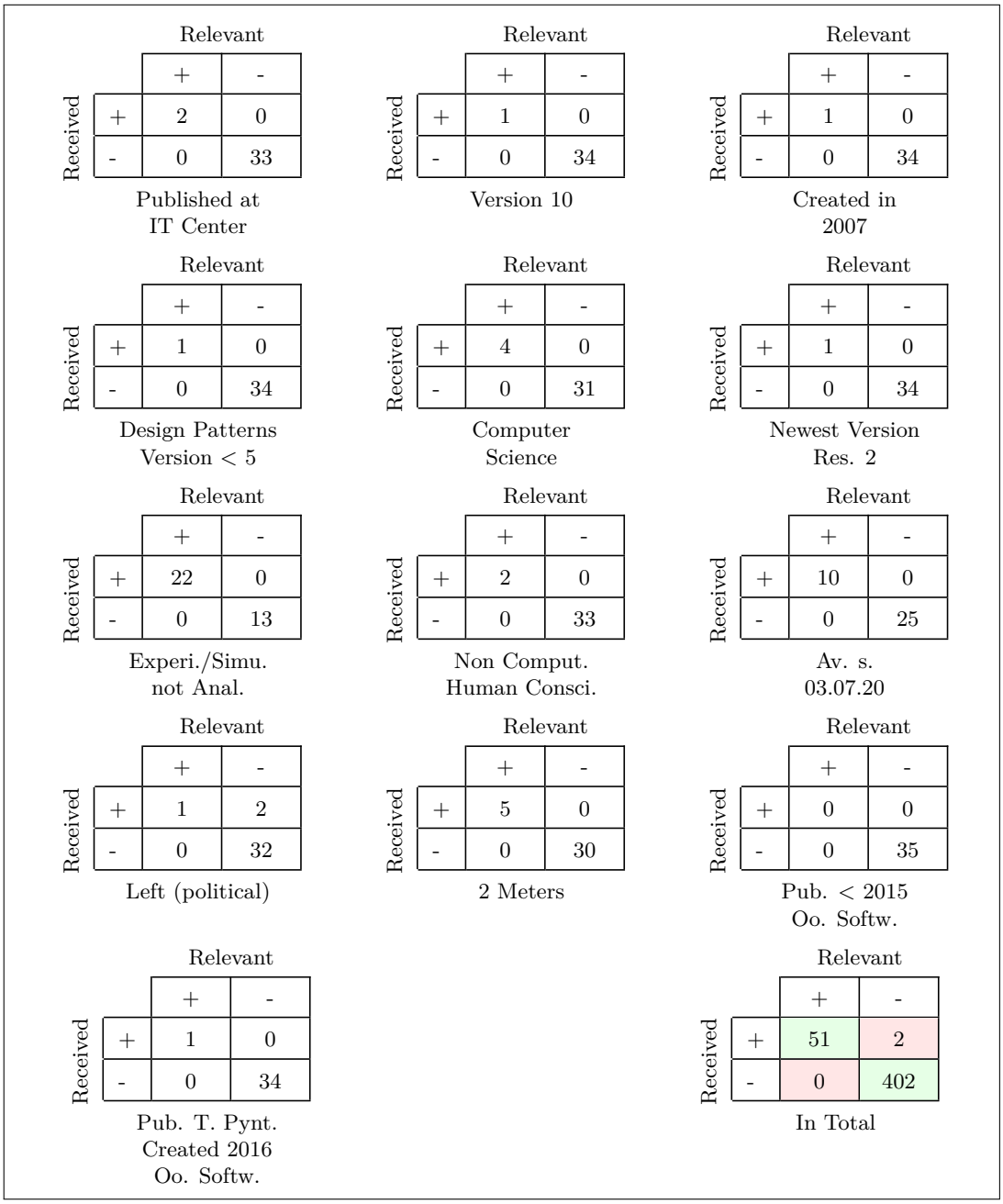


Table G.11: Confusion matrices of the advanced variant of ES



G.3 Application of the KLM for Estimating the Task Execution Time

Table G.12 shows the necessary operators of the KLM per search intention and approach, which are the basis for the calculations in Table 5.6. They are derived as described in section 5.4.3. For the faceted search only the operators of the second step, i.e., the selection of facets, are listed. For the first step, the same operators are valid as for the respective search intention in the regex approach.

Table G.12: Application of the KLM

Intention	Regex	Bif:contains	SPARQL Query Builder	Faceted Search	ES Simple	ES Advanced
Published at IT Center	H-P-BB-H-11K-K	H-P-BB-H-11K-K	H-P-PDL-P-PDL-P-CP-P-PDL-P-CP-P-PDL-P-PDL-P-BB	H-P-BB-P-BB	H-P-BB-H-11K-K	H-P-PDL-P-BB-H-21K-K
Version 10	H-P-BB-H-4K-K	H-P-BB-H-4K-K	H-P-PDL-P-BB-H-2K-H-P-BB	H-P-BB	H-P-BB-H-2K-K	H-P-PDL-P-BB-H-10K-K
Created in 2007	H-P-BB-H-4K-K	H-P-BB-H-6K-K	H-P-PDL-P-CP-P-BB-H-2K-H-P-BB-H-10K-H-P-BB-H-K-H-P-BB-H-10K-H-P-BB	H-P-BB	H-P-BB-H-4K-K	H-P-PDL-P-BB-H-39K-K
Design Patterns and Version < 5	H-P-BB-H-25K-K	H-P-BB-H-17K-K	H-P-PDL-P-CP-P-BB-H-17K-H-P-CP-P-PDL-P-CP-P-BB-H-17K-H-P-PDL-P-CP-P-BB-H-2K-H-P-BB	H-P-BB	H-P-BB-H-17K-K	H-P-PDL-P-BB-H-32K-K
Computer Science	H-P-BB-H-18K-K	H-P-BB-H-18K-K	H-P-PDL-P-CP-P-BB-H-18K-H-P-CP-P-PDL-P-CP-P-PDL-H-P-BB	/	H-P-BB-H-18K-K	H-P-BB-H-18K-K
Newest Version Res. 2	H-P-BB-H-12K-K	H-P-BB-H-12K-K	H-P-PDL-P-PDL-P-PDL-P-CP-P-BB-H-8K-H-P-CP-P-BB-H-K-H-P-BB	H-P-BB	H-P-PDL-PDL-P-BB-H-36K-K	H-P-BB-H-50K-K
Experi./Simu. not Anal.	H-P-BB-H-43K-K	H-P-BB-H-6K-K	H-P-PDL-P-PDL-P-CP-P-PDL-P-PDL-P-PDL-P-CP-P-BB-H-48K-H-P-BB	H-P-BB-P-BB	H-P-BB-H-32K-K	H-P-PDL-P-BB-H-50K-K
Non Comput. Human Con.	H-P-BB-H-37K-K	H-P-BB-H-41K-K	H-P-PDL-P-CP-P-BB-H-41K-H-P-BB	/	H-P-BB-H-39K-K	H-P-PDL-P-BB-H-52K-K
Av. s. 03.07.20	H-P-BB-H-10K-K	H-P-BB-H-12K-K	H-P-PDL-P-BB-H-28K-H-P-BB	H-P-BB	H-P-BB-H-10K-K	H-P-PDL-P-BB-H-25K-K
Left (political)	H-P-BB-H-4K-K	H-P-BB-H-4K-K	H-P-PDL-P-CP-P-BB-H-4K-H-P-BB	/	H-P-BB-H-4K-K	H-P-BB-H-4K-K
2 Meters	H-P-BB-H-9K-K	H-P-BB-H-9K-K	H-P-PDL-P-CP-P-PDL-P-PDL-P-PDL-P-BB-H-K-H-P-CP-P-PDL-P-CP-P-PDL-P-PDL-P-PDL-P-BB-H-K-H-P-BB	H-P-BB-P-BB-P-BB-P-BB-P-BB-P-BB	H-P-BB-H-9K-K	H-P-PDL-PDL-P-BB-H-59K-K
Pub. < 2015 Oo. Softw.	H-P-BB-H-26K-K	H-P-BB-H-26K-K	H-P-PDL-P-CP-P-BB-H-26K-H-P-PDL-P-CP-P-BB-H-K-H-P-BB-H-10K-H-P-BB	/	H-P-BB-H-26K-K	H-P-PDL-PDL-P-BB-H-67K-K
Pub. T. Pyn. Created 2016 Oo. Softw.	H-P-BB-H-48K-K	H-P-BB-H-54K-K	H-P-PDL-P-PDL-P-PDL-P-CP-P-BB-H-26K-H-P-PDL-P-CP-P-BB-H-2K-H-P-BB-H-10K-H-P-BB-H-K-H-P-BB-H-10K-H-P-BB	H-P-BB-P-BB	H-P-BB-H-51K-K	H-P-PDL-PDL-PDL-P-BB-H-93K-K

G.4 Calculation of Efficiency

Table G.13: Calculation of efficiency (rounded to two digits) using effectiveness, usability, and complexity

Regex	Bif:contains	SPARQL Query Builder	Faceted Search	ES Simple	ES Advanced
$\frac{3}{1+1} = 1.5$	$\frac{2}{1+1} = 1$	$\frac{6}{3+2} = 1.2$	$\frac{5}{2+1} = 1.67$	$\frac{4}{1+1} = 2$	$\frac{6}{2+1} = 2$

G.5 Calculation of Final Ranking

Table G.14: Calculation of the ranking based on the min-max normalization of the different categories

Regex	Bif:contains	Query Builder	Faceted Search	ES Simple	ES Advanced
$\frac{1}{4}+1+1+\frac{1}{2}+\frac{3}{5}+\frac{1}{5}+\frac{1}{2} = 4.05$	$0+1+1+0+\frac{4}{5}+\frac{3}{5}+\frac{1}{2} = 3.9$	$1+0+0+\frac{1}{4}+1+\frac{2}{5}+1 = 3.65$	$\frac{3}{4}+\frac{1}{2}+1+\frac{3}{4}+0+0+1 = 4$	$\frac{1}{2}+1+1+1+\frac{3}{5}+\frac{4}{5}+0 = 4.9$	$1+\frac{1}{2}+1+1+\frac{1}{5}+1+0 = 4.7$

G.6 Published Research Data

In the context of the master thesis the following research data are published:

- Sarah Bensberg. "Search Engine Evaluation for Research Data in an RDF-based Knowledge Graph". 2020. DOI: 10.18154/RTH-2020-09885.
- Sarah Bensberg. "Sample Dataset for Search Engine Evaluation for Research Data in an RDF-based Knowledge Graph". 2020. DOI: 10.18154/RWTH-2020-09886.
- Sarah Bensberg. "RDF-based Knowledge Graph Mapping for Elastic Search". 2020. DOI: 10.18154/RWTH-2020-09884.

The master thesis itself is published under:

- Sarah Bensberg. "An Efficient Semantic Search Engine for Research Data in an RDF-based Knowledge Graph". 2020. DOI: 10.18154/RWTH-2020-09883.

Index