Facilitating Reuse on the Web of Data

Benjamin Moreau

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

opendatasoft

UNIVERSITÉ DE NANTES

Thèse de doctorat

Benjamin Moreau

Université de Nantes
Comue Université Bretagne Loire
École Doctorale N°601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Facilitating Reuse on the Web of Data

Thèse présentée et soutenue à Nantes, le 6 Novembre 2020
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes

Composition du Jury

Président :	**M Pascal Molli**	Professeur, Université de Nantes
Rapporteurs :	**M^me^ Serena Villata**	Chargée de recherche CNRS HDR, Sophia Antipolis
	M Olivier Curé	Maître de conférences HDR, Université Marne-la-Vallée
Examinateurs :	**M Bernd Amann**	Professeur, Sorbonne Université
	M Serge Garlatti	Professeur, IMT Atlantique de Brest
	M Philippe Pucheral	Professeur, Université de Versailles Saint Quentin
Directeur de thèse :	**M^me^ Patricia Serrano Alvarado**	Maître de conférences HDR, Université de Nantes
Co-encadrant de thèse :	**M Emmanuel Desmontils**	Maître de conférences, Université de Nantes
Invités :	**M David Thoumas**	Co-Founder & CTO, Opendatasoft
	M Nicolas Terpolilli	Director of Engineering, Opendatasoft

Acknowledgements

I would like to thanks people that have contributed to this thesis through their ideas, knowledge, advice, and general support.

First, I want to thank Serena VILLATA and Olivier CURÉ for accepting reading my Ph.D. thesis and writing up a report. I would also like to thank Bernd AMANN, Serge GARLATTI, Philippe PUCHERAL and, Pascal MOLLI for accepting to be part of the Ph.D. jury. I also thank Sylvie CAZALENS and Serge GARLATTI for following the progress of my thesis each year.

I want to thank my supervisor Patricia SERRANO ALVARADO for offering and encouraging me to do this thesis. I also want to thank my second supervisor Emmanuel DESMONTILS for introducing me to teaching. That was an enriching experience. Thank you both for supporting me with your knowledge and experience through these three years.

I am also thankful to all the GDD team and the LS2N laboratory that provided a excellent environment, and that helped me throughout my thesis: Thomas MINIER, Arnaud GRALL, Matthieu PERRIN, Achour MOSTEFAOUI, Hala SKAF-MOLLI, Pascal MOLLI, Pauline FOLZ, and Brice NÉDELEC.

I also thank Margo BERNELIN and Sonia DESMOULIN-CANSELIER from the laboratory of Droit et Changement Social for our helpful discussions about my work. I would also want to thank all students from the University of Nantes that participate in this work: Chanez AMRI, Alan BARON, Marion HUNAULT, Fatim TOURE, and Manoé KIEFFER.

Last but not least, I am very thankful to the Opendatasoft team for having welcomed me so well in the company and entrusting me during these three years. They all taught me a lot and gave me the opportunity to have interesting discussions with their clients. Finally, I want to thank David THOUMAS and Nicolas TERPOLILLI for taking the time to have followed and actively participated in this thesis.

Thank you all for taking part in this work. It was a pleasure to work with you.

Contents

Chapter 1

Introduction

Contents

1.1 Reuse on the Web of Data

The *Web of Data*, also known as the *Semantic Web* [2], is an extension of the World Wide Web based on standards set by the *World Wide Web Consortium* (W3C)[1]. It provides a normalized way to find, share, reuse and combine information. Semantics is encoded in the data with ontologies that are sets of concepts and categories that formally represent an area or domain. Semantic data is machine-readable and machine-understandable offering significant advantages such as reasoning over

[1] https://www.w3.org/

data and facilitating interoperability among heterogeneous data sources in terms of format or schema.

The W3C recommendation to represent data on the Web of Data is the *Resource Description Framework* (RDF). RDF allows writing directed labelled graphs (RDF graphs) using triples $\langle subject, predicate, object \rangle$ where the elements may be *Internationalized Resource Identifiers* (IRIs), blank nodes, or datatyped literals. A triple represents a statement about a resource. For example, Listing 1 shows two RDF triples stating that Opendatasoft is a company located in the city of Paris.

```
1 ex:Opendatasoft a dbo:Company .
2 ex:Opendatasoft dbo:locationCity dbpedia:Paris .
```

Listing 1: Two triples about *Opendatasoft*.

OWL (Web Ontology Language)[2] and *RDFS* (RDF Schema)[3] are the two recommended languages to describe RDF ontologies. These languages are based on description logics, making automatic processing of data possible such as inferring implicit triples or checking the consistency of an RDF dataset[4].

The main idea behind the Web of Data is to break data silos by creating a global graph-based web of interlinked datasets [4]. This web of interlinked datasets is often called the *Linked Data*[5]. To make such a web a reality, Tim Berners-Lee proposed the *Linked Data principles*[6] that are best practices to publish RDF datasets on the Web. These principles are about:

1. the use of IRIs as names for resources;
2. the ability to get further information and navigate among resources using the Hypertext Transfer Protocol (HTTP);
3. and the use of existing IRIs to refer to a resource the same way other datasets do.

These principles encourage the publication of interlinked and interoperable data. The last decade has seen significant growth in the Web of Data. Over 2.5 billion web pages describe resources using the schema.org[7] ontology [3]. Since 2007, the

[2] https://www.w3.org/TR/owl2-overview/
[3] https://www.w3.org/TR/rdf-schema/
[4] https://www.w3.org/standards/semanticweb/inference
[5] https://www.w3.org/standards/semanticweb/data
[6] https://en.wikipedia.org/wiki/Linked_data#Principles
[7] https://schema.org/

Linked Open Data cloud (LOD) grew from 12 RDF datasets to more than 1239 in May 2020[8]. Tim Berners-Lee suggested the 5-star deployment scheme[9] that promotes the publication of RDF Open Data according to Linked Data principles.

The *SPARQL* Query Language is the W3C recommended language to retrieve and manipulate information on the Web of Data. *Query engines* can execute SPARQL queries on live HTTP interfaces. To take advantage of the interoperability of the Linked Data, SPARQL allows expressing queries across several RDF graphs published by autonomous data providers[10]. They are called *federated queries* and can be evaluated on a set of RDF data sources by a *federated SPARQL query engine*.

An example of a federated query would be to asks for the population of each city where the Opendatasoft company is located. Listing 2 shows this federated query. Such a query can be evaluated on RDF graphs of Listing 1 and DBpedia[11] dataset that describes among others statements about the city of Paris. Listing 3 shows two RDF triples from DBpedia about the city of Paris. The result of the federated query contains data from both RDF graphs.

```
1 PREFIX ex: <https://example.org/> .
2 PREFIX dbo: <http://dbpedia.org/ontology/> .
3
4 SELECT *
5 WHERE
6 {
7  ex:Opendatasoft dbo:locationCity ?city .
8  ?city dbo:populationTotal ?population .
9 }
```

Listing 2: A SPARQL query that retrieves population of each city where Opendatasoft company is located.

[8]The Linked Open Data Cloud is a diagram that depicts publicly available linked datasets https://lod-cloud.net/#about

[9]https://5stardata.info/

[10]https://www.w3.org/TR/sparql11-federated-query/

[11]DBpedia is an RDF dataset generated by extracting structured information from the infoboxes in Wikipedia https://wiki.dbpedia.org/

```
dbpedia:Paris a dbo:City .
dbpedia:Paris dbo:populationTotal "2229621"^^xsd:integer .
```

Listing 3: Two triples about the city of Paris.

Linked Data principles and federated queries encourage the reuse of datasets on the Web of Data to create new ones. Thus, in the Web of Data, reuse is common and occurs, for example, when an RDF dataset uses IRIs from existing ontologies or another dataset, or when the result of a federated query is used.

To facilitate reuse, resource producers should systematically protect their resources with licenses before sharing or publishing them[12] [64]. A *license* specifies precisely the conditions of reuse of a resource, i.e., what actions are *permitted*, *obliged*, and *prohibited* when using it. To protect a new resource, resource producers can create a new license or reuse an existing one. For instance, very well known and reusable licenses are the set of Creative Commons RDF licenses[13] that are described using the CC Rights Expression Language (CC REL)[14]. Datasets published on the web are usually distributed along with licenses. At the time of writing, publicly available catalogs of datasets, such as Opendatasoft's Data Network[15] or DataHub.io[16] respectively have 80% and 86% of their datasets protected by licenses.

When several licensed datasets are reused to create a new one, the resulting dataset must be protected with a license such that all licenses of reused datasets are compatible with it. For instance, the result of the federated query in Listing 2 should be protected by a license such that the license of the RDF graph in Listing 1 and the license of DBpedia are both compatible with.

1.2 Research Issues

To facilitate reuse of licensed datasets, web applications that combine them should preserve license compliance. More precisely, we focus on license compliance during federated query processing. But, to make this possible, applications need to know the compatibility among licenses. In this thesis, we focus on two problems, (1) how to position licenses in terms of compatibility and (2) how to create a license-aware federated query engine.

[12] https://en.wikipedia.org/wiki/FAIR_data
[13] https://creativecommons.org/licenses/
[14] https://wiki.creativecommons.org/wiki/CC_REL
[15] https://data.opendatasoft.com/
[16] https://old.datahub.io/

1.2.1 Ordering Licenses in Terms of Compatibility

Choosing the appropriate license for a combined dataset or choosing appropriate licensed datasets for a combination is a difficult process. It involves choosing a license compliant with all the licenses of combined datasets as well as analyzing the reusability of the resulting dataset through the compatibility of its license. Finding the right trade-off between protection and reusability is delicate. The risk is either, to choose a license too restrictive making the dataset difficult to reuse, or to choose a not enough restrictive license that will not sufficiently protect the dataset.

Relations of compatibility, compliance and restrictiveness on licenses could be very useful in a wide range of applications. We consider simplified definitions of compliance and compatibility inspired by works like [19, 23, 34, 70]: *a license l_j is compliant with a license l_i if a resource licensed under l_i can be licensed under l_j without violating l_i.* If a license l_j is compliant with l_i then we consider that l_i is compatible with l_j and that resources licensed under l_i are reusable with resources licensed under l_j. In general, if l_i is compatible with l_j then l_j is more (or equally) restrictive than l_i.

In this thesis, we are interested in facilitating users to choose a license for a combined dataset or selecting licensed datasets for a combination. We think that an automatic ordering over licenses would facilitate this task.

There exist some works[34, 71] proposing compatibility graphs of well-known licenses. However, these graphs are built from a manual interpretation of each license, making it impossible to insert a new license automatically.

Thus, the first research problem we focus on is: *given a license l_i, how to automatically position l_i over a set of licenses in terms of compatibility and compliance?*

The challenge is to generalize the definition of the ordering relations among licenses while taking into account the semantics of the actions that influence the compatibility and compliance relation.

1.2.2 Ensuring License Compliance During Federated Query Processing

Another problem occurs when multiple licensed datasets across the Web of Data participate in the evaluation of a federated query. The query result must be protected by a license that is compliant with each license of involved datasets. However, such a license does not always exist, and this leads to a query result that cannot be reused.

A solution to the compatibility of licenses is to negotiate licenses with data providers. But this negotiation takes time and is not always possible. Another solution is to discard data sources of conflicting licenses during the source selection process. But removing data sources before query evaluation can lead to an empty

query result. This problem can be tackled by using query relaxation techniques [15, 16, 31, 32]. That is, to relax the constraints of a query to match triples from other datasets. But the number of possible relaxed queries is exponential with the size of the query and the relaxation possibilities of it. Moreover, checking if each relaxed query returns a non-empty result is costly in terms of execution time in a federated context.

In this thesis, we are interested in facilitating the combination of licensed datasets using a license compliant federated query engine. Existing federated query engines do not ensure license compliance with all licenses involved in query execution.

Thus, the second problem we focus on is: *given a SPARQL query and a federation of licensed datasets, how to guarantee a relevant and non-empty query result whose license is compliant with each license of involved datasets?*

In a distributed environment, the challenge is to limit the overhead on the query execution time when the relaxation process is necessary.

1.3 Contributions

To answer our research problems, we propose two contributions: (1) *CaLi*, a model that can partially order licenses in terms of compatibility, and (2) *FLiQue*, a license aware query processing strategy for federated query engines. Sections 1.3.1 and 1.3.2 introduce these contributions.

This thesis is done in collaboration with the Opendatasoft[17] company. Opendatasoft is a data-sharing platform that can be used to easily access, reuse, and share data across an organization or publicly on the web. Within the scope of this thesis, we also proposed several demonstrators to facilitate the integration of datasets to the Web of Data. Section 1.3.3 introduces these contributions.

1.3.1 CaLi, a Model that Orders Licenses in Terms of Compatibility

To automatically position a license over a set of licenses in terms of compatibility and compliance, we propose CaLi. CaLi is a model to order licenses that uses restrictiveness relations and constraints among licenses to define compatibility and compliance. We validate CaLi experimentally with an algorithm that can add any RDF license in a set of ordered licenses in terms of compatibility and compliance.

[17] https://www.opendatasoft.com/

Our approach enables the development of license compliant applications. We show the usability of CaLi with a license-based search engine that can find resources reusable under a specific license.

This contribution led to the following publications:

1. B. Moreau, P. Serrano-Alvarado, and E. Desmontils. “CaLi: A Lattice-Based Model for License Classifications”. In: *Gestion de Données – Principes, Technologies et Applications (BDA)*. Oct 2018, Bucharest, Romania. https://hal.archives-ouvertes.fr/hal-01934596

2. B. Moreau, P. Serrano-Alvarado, M. Perrin, and E. Desmontils. “Modelling the Compatibility of Licenses”. In: *Extended Semantic Web Conference (ESWC)*. Jun 2019, Portorož, Slovenia. https://hal.archives-ouvertes.fr/hal-02069076

3. B. Moreau, P. Serrano-Alvarado, M. Perrin, and E. Desmontils. “A License-Based Search Engine”. In: *Extended Semantic Web Conference (ESWC), Demo session*. Jun 2019, Portorož, Slovenia. https://hal.archives-ouvertes.fr/hal-02097027

4. B. Moreau, P. Serrano-Alvarado, M. Perrin, and E. Desmontils. "Modéliser la Compatibilité Entre les Licences". In: *Journées Francophones d'Ingénierie des Connaissances (IC)*, Jun 2020, Angers, France. https://hal.archives-ouvertes.fr/hal-02877913

1.3.2 FLiQue, a License Aware Federated Query Processing Strategy

To guarantee a relevant, license compliant, and non-empty query result for any federated SPARQL query, we propose FLiQue, a federated license-aware query processing strategy. FLiQue is designed to detect and prevent license conflicts and gives informed feedback with licenses able to protect a result set of a federated query. If necessary, it uses query relaxation to propose a set of relevant relaxed queries whose result set can be licensed. Finally, to reduce the overhead induced by the query relaxation process in a federated context, FLiQue uses pre-calculated dataset summaries instead of communicating with the federation.

Our approach enables the creation of license-aware federated query engines facilitating reuse and creation of license compliant datasets on the Web of Data. We show the usability of our approach by integrating FLiQue in a state-of-the-art federated query engine. We experimentally validate our approach with a federated query benchmark to show the overhead produced by our approach compared to the original federated query engine.

The contribution led to the following publication:

5. B. Moreau, and P. Serrano-Alvarado. "Ensuring License Compliance in Federated Query Processing". In: *Gestion de Données – Principes, Technologies et Applications (BDA)*. Oct 2020, Online.
https://hal.archives-ouvertes.fr/hal-02904076

1.3.3 Integrating Data into the Web of Data

Opendatasoft's Data Network[18] allows to search, find and reuse datasets among publicly available datasets published with the Opendatasoft platform. However, despite the benefits of the Web of Data, many data publishers like Opendatasoft are sharing data in documents, or column-oriented formats and not in RDF. One of the reasons is that the integration of data in the Web of Data needs an investment in terms of time, storage, and maintainability.

To facilitate the integration of structured datasets in the Web of Data, we propose three solutions

- *ODMTP*, an interface to execute SPARQL queries on non-RDF datasets with high availability using RDF mappings. An RDF mapping allows describing the transformation of a structured dataset into RDF. The advantage of this approach is that the RDF dataset is not materialized and thus it does not increase storage costs.

- *EvaMap*, a framework that can evaluate the quality of an RDF mapping. EvaMap can evaluate the quality of the resulting RDF dataset at the beginning of the integration process on the RDF mapping and, thus, saves time.

- The *SemanticBot*, a conversational interface to semi-automatically generate RDF mappings for structured datasets. It allows users that are not familiar with RDF concepts to integrate their dataset in the Web of Data easily and quickly.

These contributions led to the the following demonstration papers:

6. B. Moreau, P. Serrano-Alvarado, E. Desmontils, and D. Thoumas. "Querying non-RDF Datasets Using Triple Patterns". In: *International Semantic Web Conference (ISWC)*, Demo session. Oct 2017, Vienna, Austria.
https://hal.archives-ouvertes.fr/hal-01583518

[18] https://data.opendatasoft.com/

7. B. Moreau, E. Desmontils, and P. Serrano-Alvarado. "Enrichissement de Données RDF Intégrées à la Volée". In: *Atelier Web des Données (AWD) in EGC*. Jan 2019, Metz, France.
https://hal.archives-ouvertes.fr/hal-01990875

8. B. Moreau and P. Serrano-Alvarado. "Assessing the Quality of RDF Mappings with EvaMap". In: *Extended Semantic Web Conference (ESWC)*, Demo session. Jun 2020, Heraklion, Greece.
https://hal.archives-ouvertes.fr/hal-02612705

9. B. Moreau, N. Terpolilli, and P. Serrano-Alvarado. "A Semi-Automatic Tool for Linked Data Integration". In: *International Semantic Web Conference (ISWC)*, Demo session. Oct 2019, Auckland, New Zealand.
https://hal.archives-ouvertes.fr/hal-02194315

10. B. Moreau, N. Terpolilli, and P. Serrano-Alvarado. "SemanticBot: Intégration Semi-Automatique de Données au Web des Données". In: *Atelier Web des Données (AWD) in EGC*. Jan 2020, Brussels, Belgium.
https://hal.archives-ouvertes.fr/hal-02454592

1.4 Organization

In the following, we introduce the organization of this document.

Chapter 2 explains the main concepts concerning the representation and the manipulation of information on the Web of Data. It is dedicated to readers that are not familiar with the Web of Data and contains the main concepts required for a good understanding of this thesis.

Chapter 3 gives a more in-depth explanation of RDF mappings and presents our demonstrators concerning the integration of data into the Web of Data. First, we present **ODMTP**, an approach to execute SPARQL queries on non-RDF datasets with high availability using RDF mappings. Then, we present **EvaMap**, a framework that can evaluate the quality of an RDF dataset through its RDF mapping. Finally, we present the **SemanticBot**, a conversational interface, to semi-automatically generate RDF mappings for structured datasets.

Chapter 4 presents **CaLi**, our model, that partially orders licenses in terms of compatibility. We also show how it can be used to create license-aware applications through a demonstrator of a license-aware search engine for datasets and source codes.

Chapter 5 presents **FLiQue**, our license aware query processing strategy for federated query engines. We show how it can be integrated into a federated query engine to ensure that compliant licenses protect the results of federated queries.

Chapter 6 outlines conclusions and presents new challenges that are highlighted by this thesis.

Chapter 2

Preliminaries

Contents

This chapter is intended to provide readers that are not familiar with the Web of Data with a more in-depth understanding of the context of this thesis. First section explains the main concepts of the Web of Data, the Resource Description Framework, inference, and the principles of Linked Data. Second section reminds the fundamentals of SPARQL and federated queries to retrieve and manipulate the Web of Data.

2.1 The Web of Data

This section shows standardized semantic web technologies that make the Web of Data possible. Section 2.1.1 presents the standard model for data representation on the Web of Data, ontologies to encodes the semantic in the data are introduced in Section 2.1.2, Section 2.1.3 presents the inference that enables discovering knowledge, and the Linked Data principles to create a web of interlinked data is presented in Section 2.1.4.

2.1.1 The Resource Description Framework

The *Resource Description Framework*[1] (RDF) is the standard model for data representation on the Web of Data. The RDF model provides three *RDF terms* to represent information:

1. *Internationalized Resource Identifiers* (IRIs) that extends URL with various alphabets used in different languages. It identifies web resources uniquely. I denotes the infinite set of IRIs.

2. *Literals* that are values. Literals can be tagged with a data type (e.g., string, integer, date, etc.) or a language (e.g., @fr, @en, etc.). L denotes the infinite set of Literals.

3. *Blank Nodes* that are identifiers locally scoped to a dataset. B denotes the infinite set of blank nodes.

An RDF graph is a set of *RDF triples*. An RDF triple $\langle subject, predicate, object \rangle$ describes a statement about a resource where the *subject*, is either an IRI or a blank node, the *predicate* is an IRI and, the *object* is either an IRI, a literal or a blank node. That is $\langle subject, predicate, object \rangle \in \langle (I \cup B) \times I \times (I \cup L \cup B) \rangle$.

Several serialization formats for storing and exchanging RDF graphs are available such as *Turtle*, *JSON-LD*, *RDF-XML*, *RDFa*, etc.

For example, Listing 4 shows an RDF graph serialized in the Turtle format. It contains five triples. These triples are statements about a resource identified by the IRI:

```
<https://example.org/Opendatasoft>.
```

They respectively state that it is a *company*, founded in *2011*, and located in three cities identified by the following IRIs:

```
<http://dbpedia.org/resource/Paris>
<http://dbpedia.org/resource/Boston>
<http://dbpedia.org/resource/Nantes>
```

Listings 9, 10, and 11 in the Appendix A show the same RDF graph serialized in other formats.

[1] https://www.w3.org/TR/rdf11-concepts/

```
@prefix ex: <https://example.org/> .
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix dbpedia: <http://dbpedia.org/resource/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:Opendatasoft a dbo:Company .
ex:Opendatasoft dbo:formationYear "2011"^^xsd:gYear .
ex:Opendatasoft dbo:locationCity dbpedia:Paris .
ex:Opendatasoft dbo:locationCity dbpedia:Boston .
ex:Opendatasoft dbo:locationCity dbpedia:Nantes .
```

Listing 4: An RDF graph serialized in Turtle that describes five statements about *Opendatasoft*.

Notice the usage of prefixes to reduce the size of the IRIs. A prefixed name is a prefix label and a local part separated by a colon ":". RDF turns a prefixed name into an IRI by concatenating the IRI corresponding to the prefix label with the local part (e.g., `ex:Opendatasoft` is converted into the IRI:

`<https://example.org/Opendatasoft>`).

The "a" in the predicate position of the first triple is an abbreviation for the IRI:

`<http://www.w3.org/1999/02/22-rdf-syntax-ns##type>`.

An RDF graph can also be visualized as a labelled directed multigraph where nodes represent subjects and objects, and arcs represent predicates. Figure 2.1 shows the RDF graph of Listing 4 as a labelled directed multigraph.

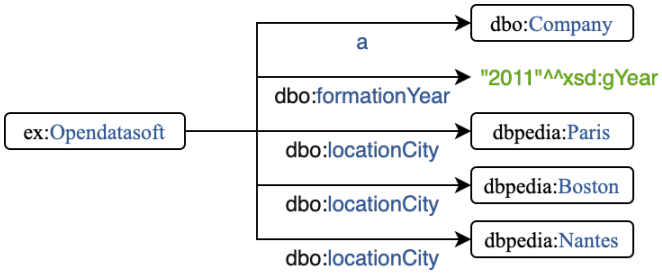

Figure 2.1: An rdf graph vizualized as a labelled directed multigraph.

2.1.2 Ontologies

The main advantage of RDF is its capability to encode the semantics with the data for both machines and humans using ontologies.

An ontology, also called vocabulary, defines concepts and relationships used to formally describe an area of concern.

Languages such as *RDF Schema* (RDFS) and *Web Ontology Langage* (OWL) enable the description of RDF ontologies.

An *RDF ontology* provides a set of *classes* that can be understood as types of objects, *properties* that are relations between classes and, *constraints* that restrict the usage of classes and properties.

Ontologies for a variety of domains are published on the Web of Data, for instance, on the Linked Open Vocabularies (LOV) [68]. Figure 2.2 shows an excerpt of the DBpedia ontology[2] used in Figure 2.1. DBpedia is a widely used cross-domain ontology which has been manually created based on the most commonly used infoboxes within Wikipedia.

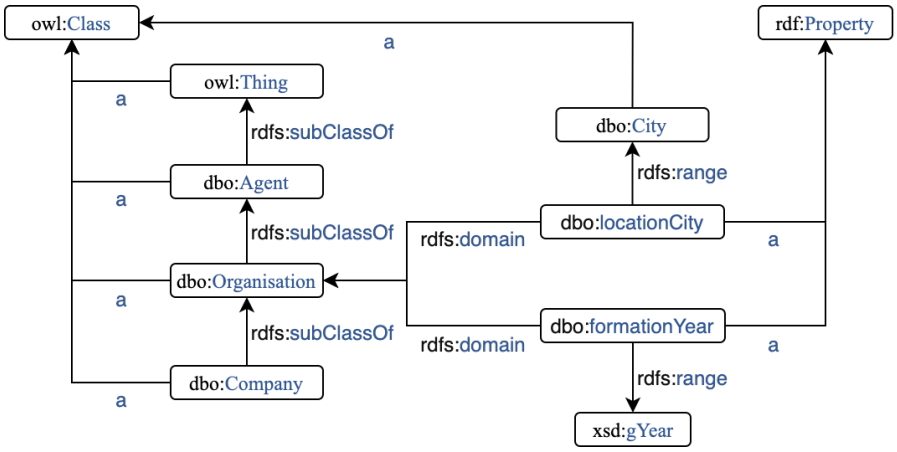

Figure 2.2: An extract of the DBpedia ontology.

In ontologies, classes are organized in a class hierarchy, also called a taxonomy, using the property `rdfs:subClassOf`. It is also possible to define a property hierarchy using `rdfs:subPropertyOf`. In the DBpedia ontology, the class `dbo:Company` is defined as a sub-class of `dbo:Organisation`, itself defined as a sub-class of `dbo:Agent` that is a sub-class of `owl:Thing`. It means that resources of type `dbo:Company` are also resources of types `dbo:Organisation`, `dbo:Agent`, and `owl:Thing`. A property may be seen as a function that associates a set of objects (i.e., a class), called *domain*, to another set of objects,

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

called *range*. That is *Property* : *domain* → *range*. RDFS provides properties `rdfs:domain` and `rdfs:range` to restrict domain and range of a property to a set of objects that belong to a specific class. In the DBpedia ontology, property `dbo:locationCity` associates objects of type `dbo:Organisation` to objects of type `dbo:City`, `dbo:formationYear` associates objects of type `dbo:Organisation` to literals of datatype `xsd:gYear`.

2.1.3 Inference

One of the main advantages of the RDF representation model is that it enables deducing implicit statements in RDF graphs through ontologies. This approach, called *inference*, is based on entailment rules defined in both RDFS[3] and OWL[4]. A *saturated* RDF graph is a graph where there are no longer statements to deduce. Figure 2.3 shows the saturated RDF graph of Figure 2.1. Inferred information is represented with red dashed lines. Resources `dbpedia:Paris`, `dbpedia:Boston` and, `dbpedia:Nantes` are inferred as resources of type `dbo:City` because the class `dbo:City` is the range of the property `dbo:locationCity`. Moreover, `ex:Opendatasoft` is inferred as a resource of type `dbo:Organisation`, `dbo:Agent` and `owl:Thing` because these are super-classes of `dbo:Company`.

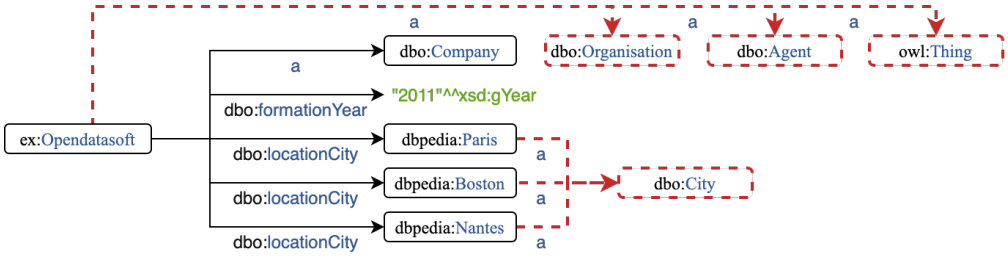

Figure 2.3: The saturated RDF graph of Figure 2.1.

2.1.4 Linked Data Principles

To take full advantage of the Web of Data, data producers must respect Linked Data principles as well as possible. *Linked Data* is a set of best practices to publish data on the Web of Data [4].

1. Use *Internationalized Resource Identifier* (IRIs) to identify any object or concept in the data.

[3] https://www.w3.org/TR/rdf-mt/#RDFSRules
[4] https://www.w3.org/TR/owl-ref/#Property

2. Use dereferenceable HTTP IRIs. That is, returning information about the object or concept when someone looks up for its IRI.

3. Use already existing IRIs as much as possible to link RDF graphs.

Figure 2.4 shows RDF graphs D1 and D2 that respect the third principle of Linked Data. Dotted lines represent the links between the two graphs. Notice that using the same IRIs in several RDF graphs creates links among datasets. That is, describing statements about the same resource or using the same ontology to describe a resource.

Publishing RDF data according to Linked Data principles, makes it readable and understandable by both machines and humans. Thus, it has many advantages, such as:

- interoperability, because several datasets may describe the same resources and use equivalent ontologies;
- discoverability, because search engines understand ontologies;
- and maintainability, because ontologies remove ambiguities that may exist on the name of attributes used in datasets;

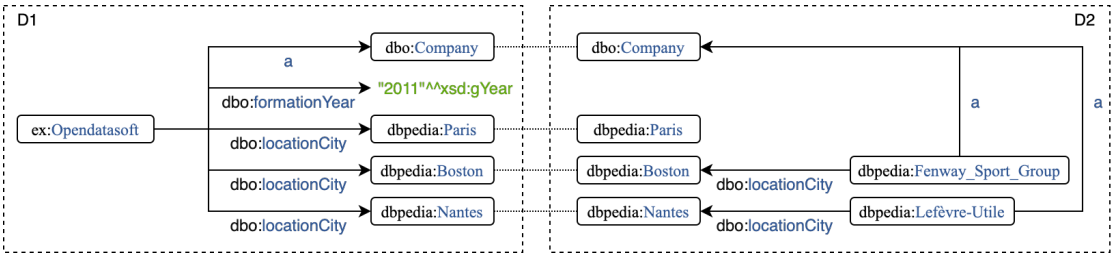

Figure 2.4: Links between two RDF graphs that share the same IRIs.

When published as open data, RDF datasets that respect the principles of Linked Data are interlinked and form a *Linked Open Data Cloud*[5] (LOD). Some of these RDF datasets are more and more used such as DBpedia[6], Wikidata[7], Geonames[8], etc.

Since the 2012 announcement by Google[9], *knowledge graph* (KG) [30] has become a non-formally defined term for a graph that uses an ontology to describe entities

[5] https://lod-cloud.net/
[6] https://wiki.dbpedia.org/
[7] https://www.wikidata.org/
[8] https://www.geonames.org/
[9] https://www.blog.google/products/search/introducing-knowledge-graph-things-not/

such as real-world objects, events, situations or abstract concepts. Knowledge graphs are now used in a variety of domains such as: query engines (e.g., Google, Yahoo![10], Bing[11]), social media (e.g., Facebook [51], LinkedIn[12]), commerce (e.g., Amazon[13], Ebay[14]), and media (e.g., BBC[15], New York Times[16]).

2.2 Querying the Web of Data

This section presents the RDF query language (SPARQL) (cf. Section 2.2.1) and query engines (cf. Section 2.2.2) that are used to retrieve and manipulate RDF graphs on the Web of Data.

2.2.1 The RDF Query Language

SPARQL Protocol and RDF Query Language (SPARQL)[17] is a query language to retrieve and manipulate data available in RDF format.

A SPARQL query, denoted by Q, is a set of *basic graph patterns* (BGP) that is a set of triple patterns. A *triple pattern* (tp) is a triple where subject, predicate and object may be a *variable*, denoted by V. That is $\langle subject, predicate, object \rangle \in \langle (I \cup B \cup V) \times (I \cup V) \times (I \cup L \cup B \cup V) \rangle$.

Listing 5 shows an example of SPARQL query that retrieves companies, associated to their city, that are located in the same city as an office of the Opendatasoft company. This query contains one BGP which consists of three triple patterns $tp1$, $tp2$, and $tp3$.

[10] https://www.researchgate.net/publication/322899161_The_Yahoo_Knowledge_Graph
[11] https://blogs.bing.com/search-quality-insights/2017-07/bring-rich-knowledge-of-people-places-things-and-local-businesses-to-your-apps
[12] https://engineering.linkedin.com/blog/2016/10/building-the-linkedin-knowledge-graph
[13] https://blog.aboutamazon.com/innovation/making-search-easier
[14] https://www.ebayinc.com/stories/news/cracking-the-code-on-conversational-commerce/
[15] https://www.bbc.co.uk/blogs/internet/entries/78d4a720-8796-30bd-830d-648de6fc9508
[16] https://open.nytimes.com/build-your-own-nyt-linked-data-application-8b91fb71fd23
[17] https://www.w3.org/TR/rdf-sparql-query/

```
1  PREFIX ex: <https://example.org/> .
2  PREFIX dbo: <http://dbpedia.org/ontology/> .
3
4  SELECT ?city ?company
5  WHERE
6   { #BGP
7    ex:Opendatasoft dbo:locationCity ?city . #tp1
8    ?company dbo:locationCity ?city .        #tp2
9    ?company a dbo:Company .                 #tp3
10  }
```

Listing 5: A SPARQL query that retrieves companies, associated to their city, that are located in the same city as an office of the Opendatasoft company.

2.2.2 SPARQL Query Engines

A *SPARQL query engine* can evaluate a SPARQL query on an RDF graph. That is to *match* each BGP of the query to sub-graphs of the RDF graph. A BGP matches an RDF graph when RDF terms from that graph may be substituted for the variables, and the result are two equivalent RDF graphs.

Figure 2.5 shows the BGP of the SPARQL query in Listing 5 matching a sub-graph of an RDF graph. The RDF sub-graph that matches the BGP is highlighted. Notice that sub-graph containing the resource `dbpedia:Paris` does not match the BGP because, in our example, no other company is located in this city.

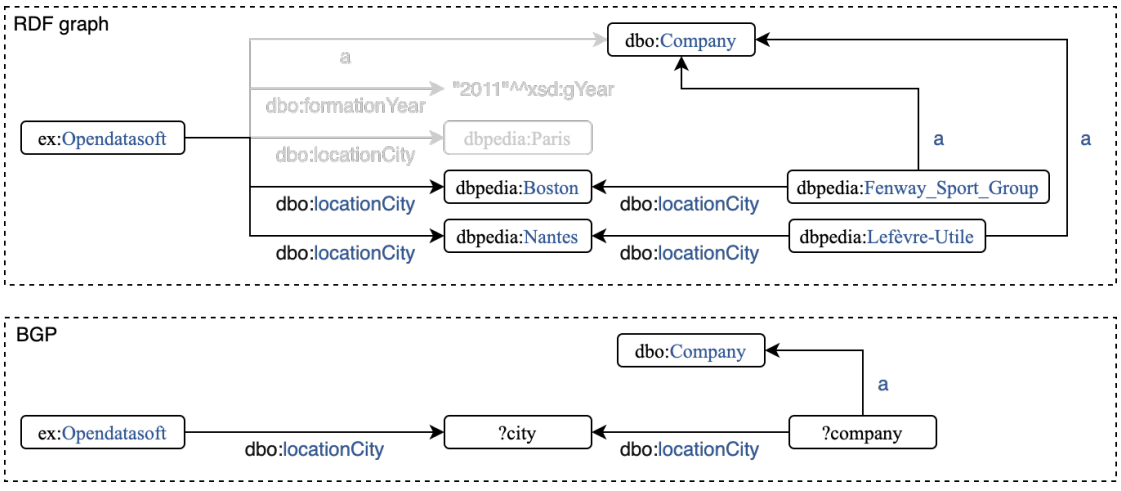

Figure 2.5: A BGP matching a subgraph of an RDF graph.

The `SELECT` clause of the SPARQL query identifies the variables to appear in the result set of the query. RDF terms that match these variables are retrieved from the RDF graph. In the query of Listing 5, the `SELECT` clause identifies the variables `?city` and `?company`. If the query is evaluated on the RDF graph of Figure 2.5, the resources that match the variables `?city` and `?company` in the matching subgraph are respectively `dbpedia:Boston` with `dbpedia:Fenway_Sport_group` and `dbpedia:Nantes` with `dbpedia:Lefèvre-Utile`.

To take advantage of the interoperability provided by the Linked Data, SPARQL allows expressing queries across several RDF graphs published by autonomous data providers[18]. A *federated SPARQL query engine* [54] can evaluate a federated SPARQL query on a set of RDF data sources, known as *a federation*. An *RDF data source* is any dataset accessible in RDF format (e.g., RDF file, triplestore, etc.)

As an example, consider that dataset $D1$ and dataset $D2$ in Figure 2.4 are two autonomous RDF data sources on the web. Then, a federated query engine can evaluate the federated query in Listing 5 on $D1$ and $D2$ as if they were a single RDF graph.

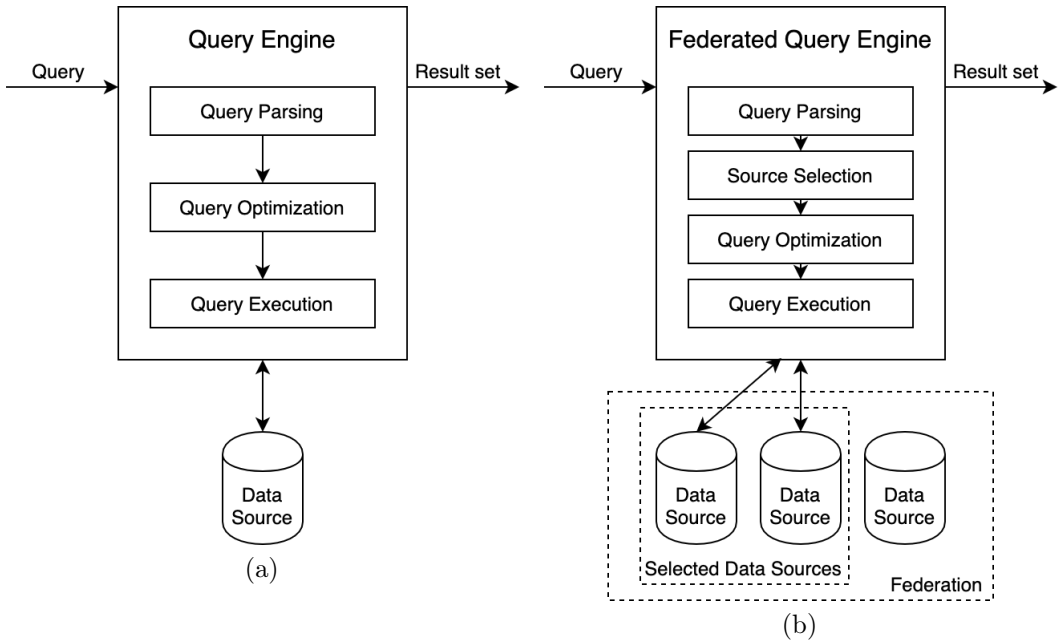

Figure 2.6: A query processed by a query engine (a) and a federated query engine (b).

[18] https://www.w3.org/TR/sparql11-federated-query/

Figure 2.6 shows how a query is processed on a query engine (a) and a federated query engine (b). During federated query processing, not all data sources participate in the evaluation of a federated query. Thus, the *source selection* process allows the identification of the data sources in the federation that may contain relevant data. During the processing of query in Listing 5, the source selection process can identify data source $D1$ as relevant to evaluate triple pattern $tp1$, $tp2$ and $tp3$, and $D2$ as relevant to evaluate $tp2$ and $tp3$. It is because $D1$ contains one triple that match $tp1$ and $tp3$, and three triples that matches $tp2$, and $D2$ contains two triples that matches $tp2$ and two triples that matches $tp3$.

Data published on the web in RDF format and queryable through a live SPARQL interface are integrated into the Web of Data. Once integrated, data has the advantage of being highly reusable. The following chapter presents approaches that facilitate the integration of non-RDF data into the Web of Data.

Chapter 3

Integrating Data into the Web of Data

Contents

The majority of datasets on the web are not published as RDF. Yet, transforming structured datasets into RDF datasets is possible thanks to RDF Mappings. During the process of transformation and publication of non-RDF datasets as RDF, data producers face several challenges, which can be obstacles to integrate datasets to the Web of Data. We think that facilitating this process can help some data-producers to publish RDF data and, thus, will foster the Web of Data growth.

This chapter is based on articles [41, 42, 45, 49, 50] and presents three approaches that aim to facilitate the publication of RDF datasets through RDF mappings. Section 3.2 presents ODMTP. It allows executing SPARQL queries on non-RDF datasets through RDF mappings with high availability. Then, Section 3.3 presents EvaMap, a framework to evaluate the quality of RDF mappings. Finally, Section 3.4 presents SemanticBot, a semi-automatic tool to generate RDF mappings for structured datasets.

3.1 Introduction and Motivation

Despite the growing number of information on the Web of Data and the benefits of Linked Data, a lot of datasets are not available in RDF or do not always respect Linked Data principles. Publishing data sources as RDF datasets that respect Linked Data principles requires methods to translate structured datasets into an RDF representation. This is possible through the implicit description of a mapping, for instance, in the source code of a software, that describes the transformation of the data into RDF. Another solution is through the explicit description of an *RDF Mapping* such that the mapping is an RDF dataset that can be shared and reused. Many *RDF mapping languages* where proposed to transform relational databases in RDF [66]. More recent works like RDF Mapping Language (RML) [13] or SPARQLGenerate [37] propose generic languages to integrate data from heterogeneous formats into RDF.

An *RDF mapping* is an RDF graph that allows describing the transformation of a structured dataset into RDF. It consists of a set of triples that are *transformation rules* containing references to the initial dataset, and represents the semantics of the resulting RDF dataset. Consider Table 3.1 that shows an excerpt of a structured dataset describing Roman Emperors and Figure 3.1 that represents an RDF mapping allowing to transform such dataset in RDF. In this mapping, Bold text starting with $ are references to a column in the dataset.

A *mapper* generates an RDF dataset by evaluating an RDF mapping on a dataset. Figure 3.1 shows the RDF dataset resulting from the transformation of the dataset in Table 3.1 with the mapping in Figure 3.1. In this example, the mapper generates the RDF dataset by replacing references in the RDF mapping

string	date	string	string	float	float
Name	**Birth**	**Birth City**	**Birth Province**	**Lat**	**Long**
Augustus	0062-09-23	Rome			
Caligula	0012-08-31	Antitum			
Claudius	0009-08-01	Lugdunum	Gallia Lugdunensis	47.932559	0.191854
...	...	...	...	...	...

Table 3.1: An excerpt from a structured and typed dataset describing Roman emperors.

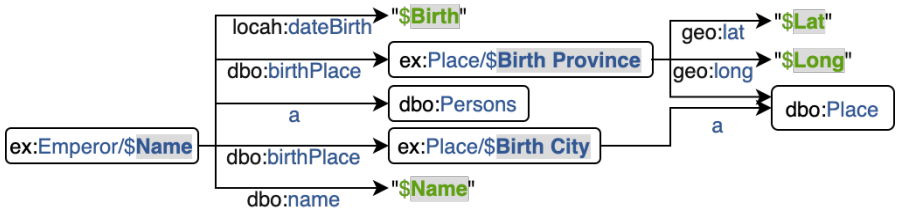

Figure 3.1: An RDF mapping for Roman emperors dataset.

by values of the referenced columns for each row in the initial dataset.

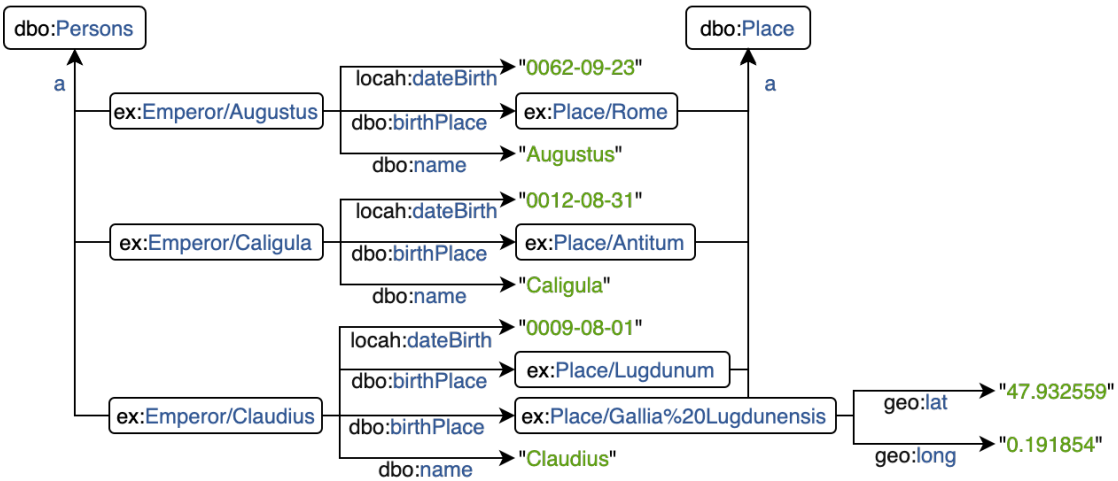

Figure 3.2: An RDF dataset describing Roman emperors.

However, publishing an existing dataset as an RDF dataset that respects Linked Data principles is not easy. In this chapter, we focus on three problems that data producers face during transformation and publication process.

3.1.1 Virtually Integrate non-RDF Datasets with Triple Pattern Fragments Interface

After the transformation of a dataset into RDF, storing the result in a triple-store is an important investment in terms of storage and maintainability. Existing works propose to use RDF mappings to integrate datasets as RDF virtually. That is to enable querying non-RDF datasets with SPARQL as if they were stored as RDF. These works focus on the SPARQL endpoint interface to expose non-RDF data. But, this interface suffers from availability issues [5]. To tackle this problem, Triple Pattern Fragments (TPF) [69] has been proposed. However, no work focus on the TPF interface to expose non-RDF data.

Thus, the first problem we focus on is: *how to virtually integrate non-RDF datasets as RDF simply and efficiently using RDF mappings and Triple Pattern Fragments interface?*

The challenge is to limit the overhead produced by the virtual integration on the global query execution time.

3.1.2 Assessing the Quality of RDF Mappings

Making an RDF mapping for a dataset is a crucial step in the integration of a dataset into the Web of Data. The quality of the resulting RDF dataset highly depends on the quality of its RDF mapping. Making a relevant RDF mapping for a dataset while respecting the Linked Data principles is a challenging task. Moreover, in addition to possible errors a user can make, different RDF mappings are possible for the same dataset, for example, depending on the ontology chosen to describe the dataset.

Thus another problem we focus on is: *How to automatically help users to create RDF mappings without errors and how to choose the best mapping from a set of RDF mappings?*

The challenge is to evaluate the quality of the RDF mapping instead of the resulting RDF dataset to identify errors at the beginning of the transformation process and saves time.

3.1.3 Facilitating the Creation of RDF Mappings

Writing a relevant mapping for a dataset is not a simple task. It requires answering several questions, for instance: what are the concepts described in the dataset? Which existing RDF ontologies are relevant to describe these concepts? But, answering these questions requires both to know the dataset perfectly and to be familiar with RDF. Unfortunately, many data producers are not familiar with RDF and are not yet ready to invest time in writing RDF mappings.

The third problem we focus on is: *how to simplify as much as possible the creation of RDF mappings for existing structured datasets?*

The challenge we face is to automate the part of the integration process that requires getting familiar with RDF.

In the following, we present three contributions that answer our three problems.

3.2 Querying non-RDF Datasets using Triple Patterns

Transforming existing datasets as RDF and storing them in triple-stores requires to store data as RDF and non-RDF in two separate databases and to synchronize them during updates. Existing works [17, 39] propose to query non-RDF datasets on-demand with SPARQL. That is to virtually integrate non-RDF data sources through RDF mappings to make run-time evaluations of SPARQL queries.

Several HTTP client-server interfaces have been proposed to publish and access datasets on the Web of Data [26, 40, 69], but the SPARQL endpoint interface remains the most popular despite a study [5] that shows that this interface suffers from availability issues.

Each of these client-server SPARQL interfaces is characterized by the type of requests the server can evaluate. The type of requests has a significant impact on the effort made by the server during the evaluation of a SPARQL query and, thus, on its availability. Figure 3.3 shows several HTTP client-server SPARQL interfaces ordered according to the effort made by the server during the evaluation of a SPARQL query.

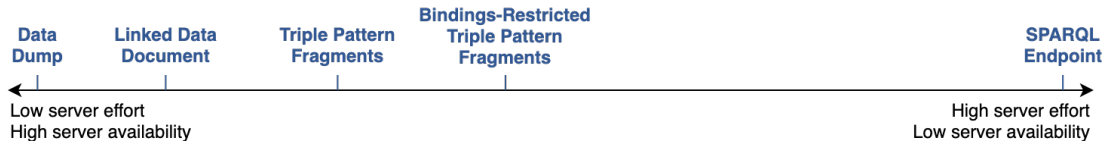

Figure 3.3: HTTP client-server SPARQL interfaces.

Triple Pattern Fragments interface [69] is one of the interfaces proposed to query RDF datasets with high availability. One of the reasons for server availability is the simplicity of the server interface. An important part of the query execution is on the client-side.

Differences between SPARQL endpoints and TPF interfaces are respectively shown in Figures 3.4 and 3.5. With TPF, the client receives a SPARQL query and decomposes it into *Triple Pattern Queries* (TPQ). TPQs are sent to TPF servers.

A server matches triple pattern and page to an RDF dataset and sends results back to the client along with metadata and controls (e.g., the total number of triples matching the triple pattern, the number of triples per page, the link to the next page, etc.). RDF graph returned by TPF servers are called fragments and are used by the TPF client to build the complete answer of the SPARQL query. Listing 12 in the Appendix A shows a fragment returned by a TPF server.

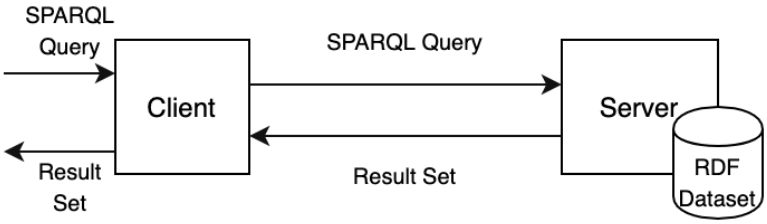

Figure 3.4: The SPARQL Endpoint interface.

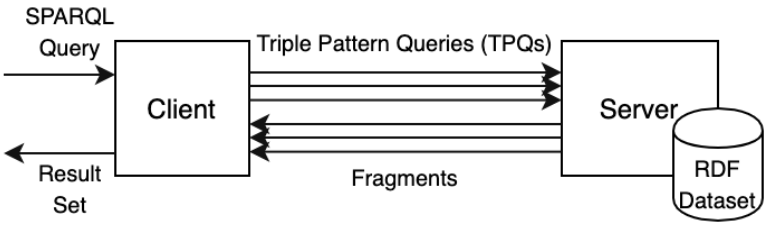

Figure 3.5: The Triple Pattern Fragments interface.

Existing works focus on exposing non-RDF datasets through SPARQL endpoints but not through Triple Pattern Fragments interface. Figure 3.6 shows the execution of a SPARQL query on a virtually integrated non-RDF dataset.

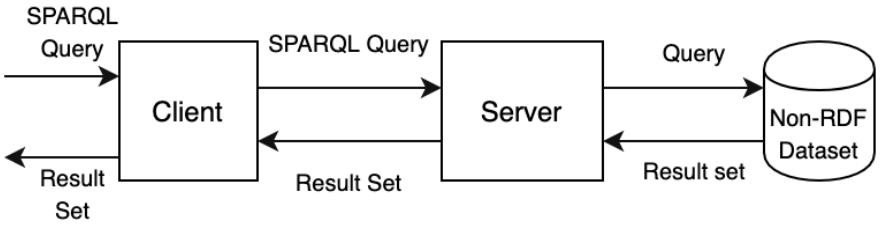

Figure 3.6: The execution of a SPARQL query on a virtually integrated non-RDF dataset.

As explained in section 2.1.3, the inference is the execution of entailment rules on RDF datasets through ontologies to discover implicit triples. Supporting inference

in a query engine is very important to return complete results. But supporting inference on virtually integrated non-RDF datasets is not straightforward because explicit triples are required to deduce implicit triples.

In this section, the problem we focus on is *how to integrate non-RDF datasets on-demand as Linked Data simply and efficiently using TPF*.

The challenge is to support inference and limit the overhead produced by the virtual integration and the inference on the global query execution time.

We propose **ODMTP**, an **O**n-**D**emand **M**apping using **T**riple **P**atterns over non-RDF datasets [45] that supports inference [41]. To illustrate our approach, we implemented ODMTP to query Twitter, GitHub, and LinkedIn with SPARQL queries.

3.2.1 ODMTP: On-Demand Mapping using Triple Patterns

We propose to modify the TPF server such that, instead of evaluating TPQs over an RDF store, it sends TPQs to ODMTP.

Figure 3.7 presents the global architecture of ODMTP. TPF clients receive SPARQL queries and decompose them into several TPQs. TPQs are sent to ODMTP through a TPF server.

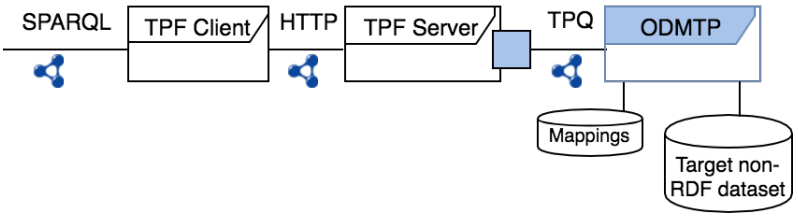

Figure 3.7: The global architecture of ODMTP.

When ODMTP receives a TPQ, it translates it into the target query language and sends it to the target non-RDF dataset. The query engine of the non-RDF dataset evaluates this query and returns the result set to ODMTP. Then, ODMTP translates this answer in triples, it constructs fragments and sends them to the TPF server.

ODMTP translations are possible thanks to the RDF mapping of the target dataset. Several mappings can be defined for one target dataset depending on the needs of semantic applications.

Figure 3.8 shows more details about the ODMTP module. The trimmer component receives a TPQ and from a mapping file it extracts the mappings

pertinent to the triple pattern. Then, the TPQ and its corresponding mapping is received by TP2Query that uses the mapping to translate the triple pattern into a query in the query language of the non-RDF dataset. TP2Query communicates with the query engine of the non-RDF dataset and obtains the result set corresponding to TPQ. It also estimates the total number of triples matching the triple pattern. The implementation of this component depends on the non-RDF dataset. Finally, the Mapper component uses the mapping to translate the resultset in triples. And it produces the fragment that is sent to the TPF server.

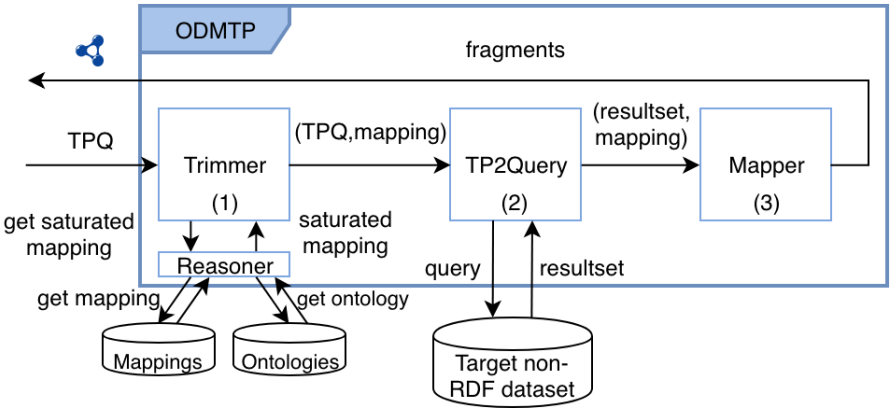

Figure 3.8: The components of the ODMTP module.

3.2.2 Supporting Inference with ODMTP

Supporting inference in a query engine is very important to return complete results. Implicit triples can be materialized before query execution or returned during query execution.

Existing works [20, 67] propose to materialize implicit triples. But, this approach increases the size of RDF datasets. Other works [53, 58] use query rewriting to evaluate queries on implicit triples without materializing them. The size of the RDF dataset remains constant but the query execution time increases.

Supporting inference on virtually integrated non-RDF datasets is not straightforward because explicit triples are required to deduce implicit triples. Moreover, query rewriting approaches increase the query execution time that is already increased by virtual integration. To limit the overhead produced by inference, we propose a simple approach that consists of applying entailment rules on RDF mapping to allow the execution of new SPARQL queries on non-RDF datasets.

As an example, consider the RDF mapping for the Roman emperors dataset in Figure 3.1. With this mapping, ODMTP can execute query of Listing 6 that asks for resources of type `dbo:Place` but returns an empty result when the query of

Listing 7 asking for resources of type `dbo:Location` is executed. Figure 3.9 shows an extract of the DBpedia ontology that defines super classes and equivalent classes of the `dbo:Place` class. The query of Listing 7 should not return an empty result set because `dbo:Place` is equivalent to the class `dbo:Location`. To tackle this problem, we use the ontology to deduce and materialize implicit transformation rules in the RDF mapping. Figure 3.10 shows the saturated RDF mapping of Figure 3.1. Inferred transformation rules are represented with red dashed lines. The saturated mapping allows ODMTP to execute queries on implicit triples such as the query in Listing 7.

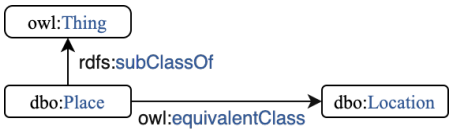

Figure 3.9: Super classes and equivalent classes of the `dbo:Place` class.

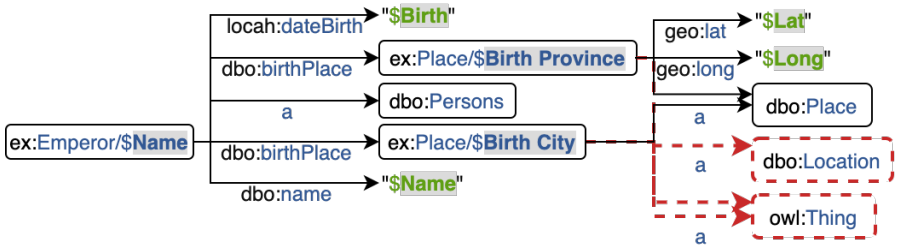

Figure 3.10: The saturated RDF mapping of Figure 3.1 according to the ontology of Figure 3.9.

```
PREFIX dbo: <http://dbpedia.org/ontology/>.

SELECT ?place
WHERE
 {
  ?place a dbo:Place .
 }
```

Listing 6: A query that retrieves resources of type `dbo:Place`.

```
PREFIX dbo: <http://dbpedia.org/ontology/>.

SELECT ?loc
WHERE
 {
  ?loc a dbo:Location .
 }
```

Listing 7: A query that retrieves resources of type `dbo:Location`.

ODMTP supports inference through its reasoner module. Figure 3.11 shows the reasoner module integrated into the ODMTP approach. It can infer implicit

triples from the RDF mapping using RDFS and OWL entailment rules and the ontology used in the mapping.

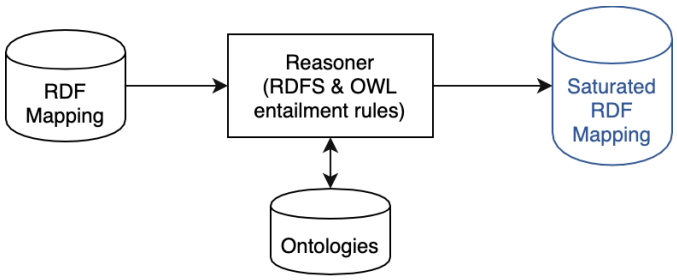

Figure 3.11: The reasoner module integrated to ODMTP.

The saturated RDF mapping is computed before query execution during the deployment of ODMTP. Thus, the execution time remains constant. Moreover, the overhead in terms of storage is limited because the number of implicit triples in an RDF mapping is small compared to implicit triples in the resulting RDF dataset.

A limitation of this approach is that only the rules concerning properties and classes in the RDF mapping are taken into account in our approach. Rules that apply to instances are not supported by ODMTP because instances are not materialized in RDF mappings. The list of supported rules is available on the repository of ODMTP[1].

3.2.3 Implementation

To show the usability of our approach, we implemented ODMTP for Twitter, GitHub, and LinkedIn APIs. This implementation is available on GitHub under the MIT license[2]. ODMTP is a Django[3] application that can receive requests from any TPF client[4]. Our implementation allows users to query tweets, GitHub repositories, or LinkedIn profiles using SPARQL. RDF mappings are also available on the repository[5]. Moreover an online video demonstration is available[6].

[1] https://github.com/benjimor/odmtp-tpf#supported-rules
[2] https://github.com/benjimor/odmtp-tpf
[3] https://www.djangoproject.com/
[4] http://query.linkeddatafragments.org/
[5] https://github.com/benjimor/odmtp-tpf/tree/master/mapping
[6] https://youtu.be/wruH8teK9tU

S	**P**	**O**
S	**P**	?o
S	?p	**O**
S	?p	?o
?s	**P**	**O**
?s	**P**	?o
?s	?p	**O**
?s	?p	?o

Table 3.2: The eight types of triple patterns.

3.2.4 Experimental Evaluation

The goal of our experimental evaluation is to measure the overhead produced by the ODMTP TPF server compared to the original TPF server in terms of execution time of a triple pattern query. This execution time is tightly related to the performance and the capabilities of the data source of the TPF server. The ODMTP TPF server is a Python implementation connected to several datasets stored in JSON and queried through Elasticsearch. The cardinality of datasets goes from 100,000 to 500,000 triples. Experiments were executed locally[7].

We measured the time to evaluate the eight possible types of triple patterns to see if the place of the variable in a triple pattern has an impact on the execution time. Table 3.2 shows the eight types of triple patterns. Variables are preceded with a question mark. Constants are in bold capitals and can be IRIs, Literals, or blank nodes. We executed each query five times and get the average execution time.

We also measured the time to evaluate the last page (i.e., the potentially most expensive result page) to take into account the pagination capabilities of the TPF server data source.

We compared these performances with two traditional TPF servers. One is connected to RDF datasets materialized in HDT [14] files that support pagination. The other is connected to RDF datasets materialized in Turtle files (TTL) that do not support pagination.

Figure 3.12 shows the average execution time of the last page for each type of triple pattern, depending on the size of the dataset. The execution time increases linearly with the size of the dataset when the TPF server query a data source that does not support pagination. That is because, to compute a page, it needs to compute all previous pages. For the sources that support pagination, we see that execution time does not increase with dataset size and that the type of triple pattern

[7] on a computer with a processor intel i7 2,5 GHz with 16 GB of RAM.

does have an impact on the execution time but does not affect the complexity. Thus, we can compute the average overhead of ODMTP comparing to a TPF server connected to an HDT data source.

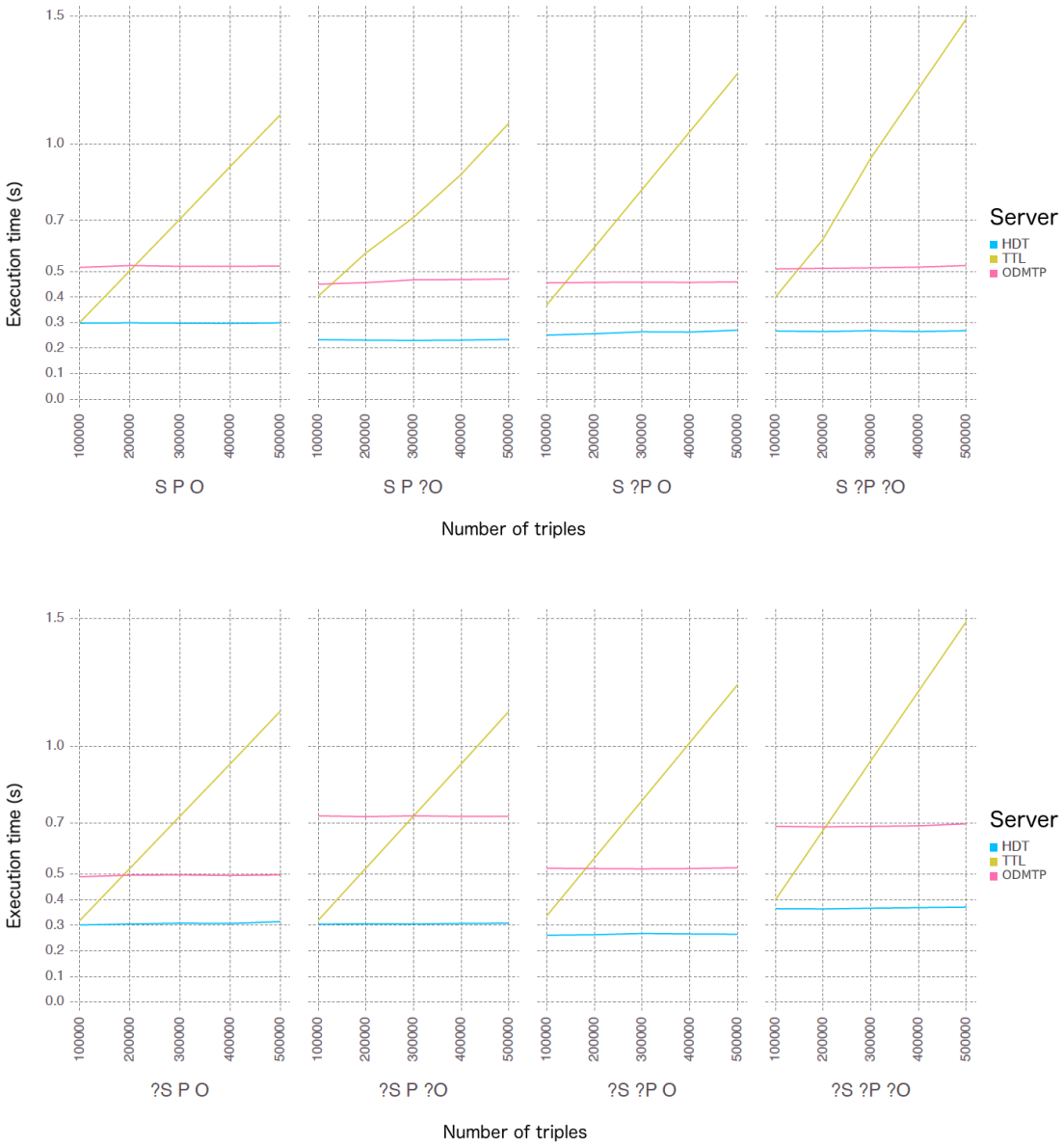

Figure 3.12: The average execution time to retrieve the last page for each type of triple pattern.

Figure 3.13 shows the average execution time of the last page for all types of triple patterns. On average, ODMTP produced a constant overhead (on average

+0,250 seconds). The reason for this limited overhead is the set of efficient indexes produced by Lucene[8] that is used by Elasticsearch.

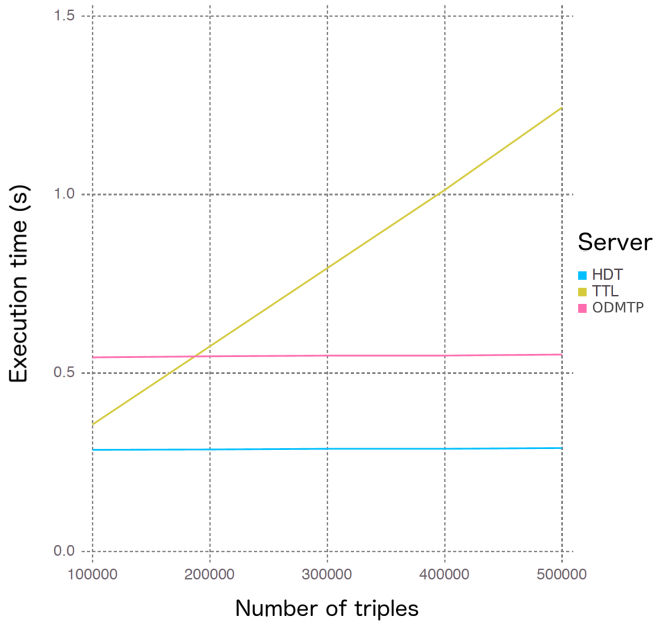

Figure 3.13: The average execution time to retrieve the last page of a triple triple pattern.

Finally, we also measured the average execution time of a complete SPARQL query that retrieves all triples of the dataset (i.e., a query with a BGP containing the triple pattern ?s ?p ?o). We compared the execution time of the complete set of TPQs with our ODMTP approach versus the TPF server connected to the HDT file. The execution time is measured on the client side.

Figure 3.14 shows the average execution time of the SPARQL query depending on the size of the RDF dataset. The execution time of this SPARQL query grows linearly with both TPF servers. It grows faster with ODMTP compared to the TPF server with an HDT file. That is because of the overhead produced by ODMTP for each TPQ to execute in order to answer the complete SPARQL query.

[8] https://lucene.apache.org/

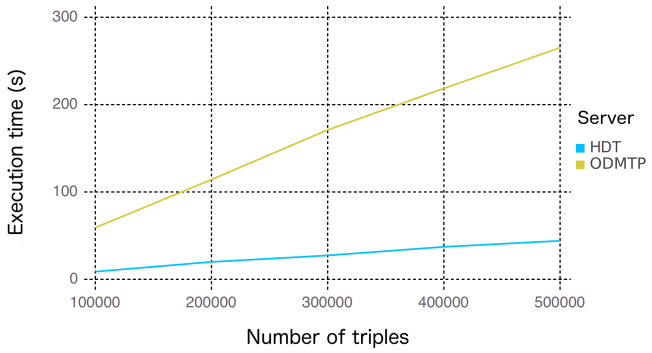

Figure 3.14: The average time to execute a SPARQL query.

This experimental evaluation confirms that ODMTP can evaluate any triple pattern query in constant time despite the size of the dataset. Moreover, the complexity of the ODMTP approach is similar to a TPF server connected to an HDT file.

3.3 Assessing the Quality of RDF Mappings with EvaMap

Making a relevant RDF mapping for a dataset is a challenging task because it requires to answer several questions:

1. What are the different resources described in the dataset (e.g., cars, persons, cities, places, etc.)?
2. What are the attributes of these resources (e.g., price, age, etc.)?
3. How should the IRI of resources be defined?
4. What are the possible relations between the different resources (e.g., the city is the birthplace of the person)?
5. Which ontology, classes, and properties should be used?

As well as possible errors by the user, different answers are possible for some of these questions and, thus, different RDF mappings are possible for the same dataset.

In addition to the mapping in Figure 3.1, Figure 3.15 presents two possible mappings for the dataset in Table 3.1. Unlike mapping 3.15(a), mapping 3.15(b) does not include a class description in resource IRIs and does not reference the *Birth Province* column.

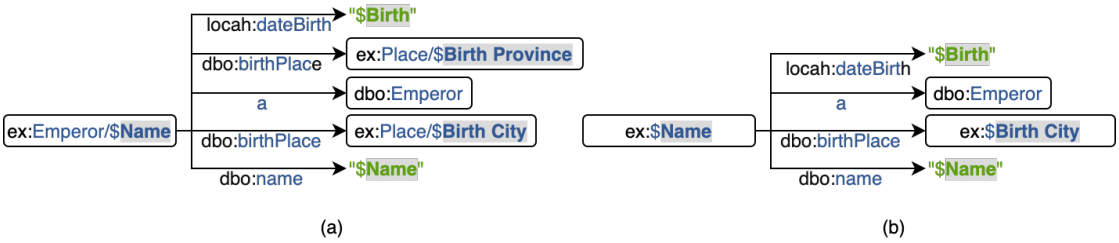

Figure 3.15: Two other RDF mappings for the Roman emperors dataset.

We think that assessing the quality of mappings is important because it directly impacts the quality of the resulting RDF dataset. A triple in an RDF mapping can generate multiple triples in the resulting RDF dataset. Thus, correcting an error in the mapping is more effective than correcting the generated errors in the RDF dataset. Moreover, to save time during the publishing process, the earlier quality is assessed, the better [12].

We believe that a tool capable of evaluating the quality of an RDF mapping and returning feedback would make the creation and the choice of RDF mappings easier.

In this work, we answer the following question: *Given a structured dataset, how to help users to create RDF mappings without errors automatically, and how to choose the best mapping from a set of RDF mappings?*

In the state-of-the-art, [12] proposes a framework that assesses and refines RML mappings. However, authors focus on logical errors due to incorrect usage of ontologies (e.g., violation of domain, range, disjoin classes, etc.). [72] proposes a framework to assess the quality of RDF datasets through metrics. Metrics are organized in dimensions evaluating different aspects of a dataset (e.g., availability, interlinking, etc.). But, [72] does not propose to assess the quality of an RDF mapping. In our work, like in [12], we evaluate metrics on the RDF mapping instead of on the resulting RDF dataset. This choice allows us to identify errors at the beginning of the publishing process and saves time.

However, not all metrics can be applied to mapping. Thus, the challenge is to evaluate as many metrics as possible on the RDF mapping.

Based on the framework proposed in [72], we propose **EvaMap** [42]. EvaMap is a framework to **Eva**luate RDF **Map**pings. We propose an extension of the set of metrics proposed in [72]. The goal is to control the quality of the resulting dataset through its mapping without having to generate the RDF dataset. It is also capable of returning feedback to the user to improve its mapping.

Dimension	Description
Availability	Checks if IRIs are dereferenceable
Clarity	Checks human-readability of the mapping and the resulting dataset
Conciseness	Checks if the mapping and the resulting dataset is minimal while being complete
Consistency	Checks if the mapping is free from logical errors
Coverability	Checks if the RDF mapping is exhaustive compared to the initial dataset
Connectability	Checks if links exist between local and external resources
Reusability	Checks if metadata enables reuse

Table 3.3: The set of dimensions used in EvaMap.

3.3.1 EvaMap: A Framework to Evaluate RDF Mappings

EvaMap uses a set of metrics organized in 7 dimensions. When it is possible, metrics are evaluated on the RDF mapping. Otherwise, they are evaluated on an extract of the resulting RDF dataset. For example, the *available resource IRIs* metric needs RDF dataset to check if generated IRIs are dereferenceable. In this case, EvaMap generates a sample such that applying each mapping rule to the entire input dataset is not necessary. Table 3.3 describes each dimension of EvaMap. These dimensions are based on [72]. From these dimensions, we propose the *Coverability* one that detects the lose of data between the input dataset and the resulting RDF dataset. Table 3.4 shows the set of metrics used by EvaMap. New metrics proposed in EvaMap are highlighted in blue. The last column indicates if the metric is evaluated on the RDF mapping or if the initial dataset is also needed.

In order to compute the quality of a mapping, M_i applied on a raw dataset D, we propose a function $q(M_i, D) \in [0, 1]$ that is the weighted mean of the quality of each metric $m_j(M_i, D)$:

$$q(M_i, D) = \frac{\sum_{j=1}^{n} w_j.m_j(M_i, D)}{\sum_{j=1}^{n} w_j}$$

EvaMap also computes the score for each dimension. To do that, it only considers the subset of metrics for the corresponding dimension.

Weights w_j associated with metrics can be used to give more or less importance to each metric. For example, the user does not always want to generate RDF triples for all data in the input dataset. Thus, weights associated with *coverability* metrics can be lowered or set to zero.

Figure 3.16 shows the global architecture of EvaMap. In addition to the quality of the RDF mapping, it returns feedback to improve the mapping.

Dimension	Metric	Description	Evaluation
Availability	Ontology availability	Checks if IRIs of classes and properties return a 2xx success HTTP code.	Mapping
	Dataset availability	Checks if IRIs of instances return a 2xx success HTTP code.	Dataset
Clarity	Entities human-readability	Checks if entities have a label or a description.	Mapping
	IRIs human-readability	Checks if IRIs are human-readable.	Mapping
Conciseness	Rules conciseness	Checks that there are no several transformation rules generating the same set of triples.	Mapping
	IRIs conciseness	Checks if IRIs are not too long.	Mapping
Consistency	Properties consistency	Checks if domain and range of property are respected.	Mapping
	Class hierarchy consistency	Checks super-classes and equivalent classes.	Mapping
	Property hierarchy consistency	Checks super-properties and equivalent properties.	Mapping
	Disjoint class consistency	Checks that there are no instances belonging to disjoint classes.	Mapping
	Datatype consistency	Checks if datatypes are consistent with datatypes in the input dataset.	Dataset
Coverability	Vertical coverability	Checks if all attributes of the input dataset are considered.	Dataset
Connectability	Entities connectability	Checks if entities are linked to other entities using the sameAs property.	Mapping
	External connectability	Checks if IRIs from external graphs are referenced.	Mapping
	Local connectability	Checks if IRIs from the local graph are linked to each other.	Mapping
	Ontology connectability	Checks if already existing ontologies are used.	Mapping
Reusability	License availability	Checks if the mapping is protected with a license.	Mapping
	License compatibility	Checks if licenses of the dataset and the mapping are compatible.	Dataset
	Expiration	Checks if the last update of the mapping is newer than the last update of the dataset.	Dataset

Table 3.4: The set of metrics used in EvaMap.

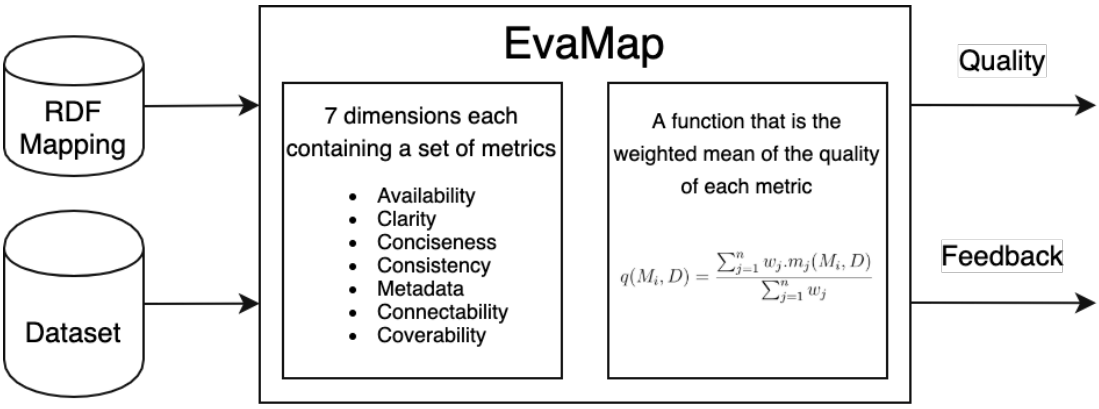

Figure 3.16: The global architecture of EvaMap.

3.3.2 Implementation and Demonstration Tool

We implemented EvaMap to evaluate YARRRML [28] mappings for datasets of the Opendatasoft's Data Network[9]. The source code of our tool[10] and web service[11] are available on GitHub under the MIT license.

Our tool is available as a web service[12]. Users are able to select different mappings and use EvaMap to compare them. For each mapping, the global quality score is computed as well as the quality score for each dimension. Our tool also gives feedback that can be used to refine the RDF mapping. For instance, users can assess two mappings for the dataset *football-ligue*. They can see that the mapping *football-ligue* obtains a worse global score than the mapping *football-ligue-fixed*. In the detailed report, users can analyze by dimension why these scores are different.

Figures 3.17 and 3.18 respectively show the quality score and feedback on the clarity dimension returned by our implementation of EvaMap.

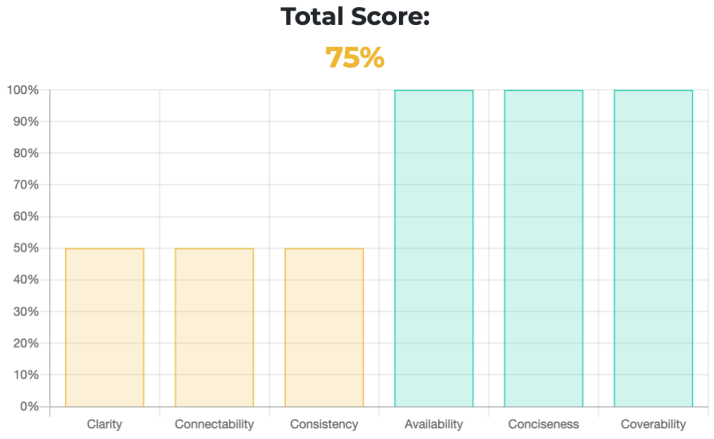

Figure 3.17: The quality score for an RDF mapping returned by EvaMap.

[9] https://data.opendatasoft.com
[10] https://github.com/benjimor/EvaMap
[11] https://github.com/benjimor/EvaMap-Web
[12] https://evamap.herokuapp.com/

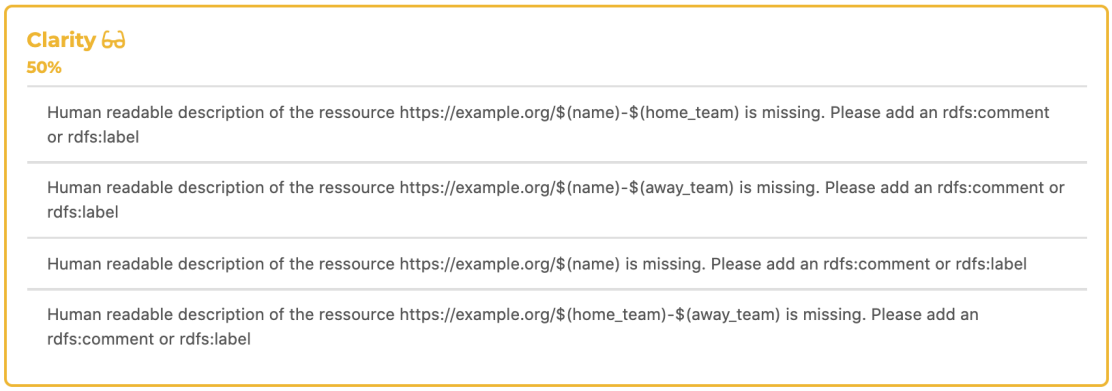

Figure 3.18: The feedback on the clarity dimension for an RDF mapping returned by EvaMap.

3.4 Generating RDF Mappings with a Semi-Automatic Tool

As we have seen previously, writing RDF mappings is not easy. Consider the Roman Emperor dataset in Table 3.1 and Figure 3.1 that represents an RDF mapping for this dataset. Writing this mapping requires to answer several questions, for instance: (i) what concepts contain the *Name* and *Birth city* columns? In this case, *Name* contains entities that are Persons (emperors) and *Birth city* contains entities that are Places (cities). (ii) What are the relationships between these two concepts? Here, Places are birth places of Persons. (iii) Which existing ontologies are relevant to describe these concepts? In this example, DBpedia, GeoNames, etc.

Answering these questions requires two types of skills. It requires, to know the dataset perfectly (i.e., its structure, context, meaning, etc.) and to be familiar with RDF concepts such as RDFS, OWL, and RDF mapping languages. Unfortunately, many data producers are not familiar with RDF and are not yet ready to invest time to integrate their data. In this work, we focus on *how to simplify as much as possible the integration of existing structured datasets as Linked Data*. The challenge we face is to automate part of the integration process that requires getting familiar with RDF.

Even if there exist simplified and human-readable syntaxes of mapping languages like YARRRML [28], writing a mapping requires to be familiar with RDF. Recently, interesting tools have been proposed to assist users during the creation of an RDF mapping. These tools use different representations of the mapping to hide the complexity of RDF mapping languages. KARMA [24] and RMLeditor [29] represent the mapping as a graph while Juma [33] uses a puzzle block metaphor to avoid misuse of RDF mapping languages terms. KARMA also uses machine learning to

automatically generate parts the RDF mappings that are common to previously integrated datasets. Figures 3.19, 3.20, and 3.21 respectively show the user interface of KARMA, RMLeditor, and Juma.

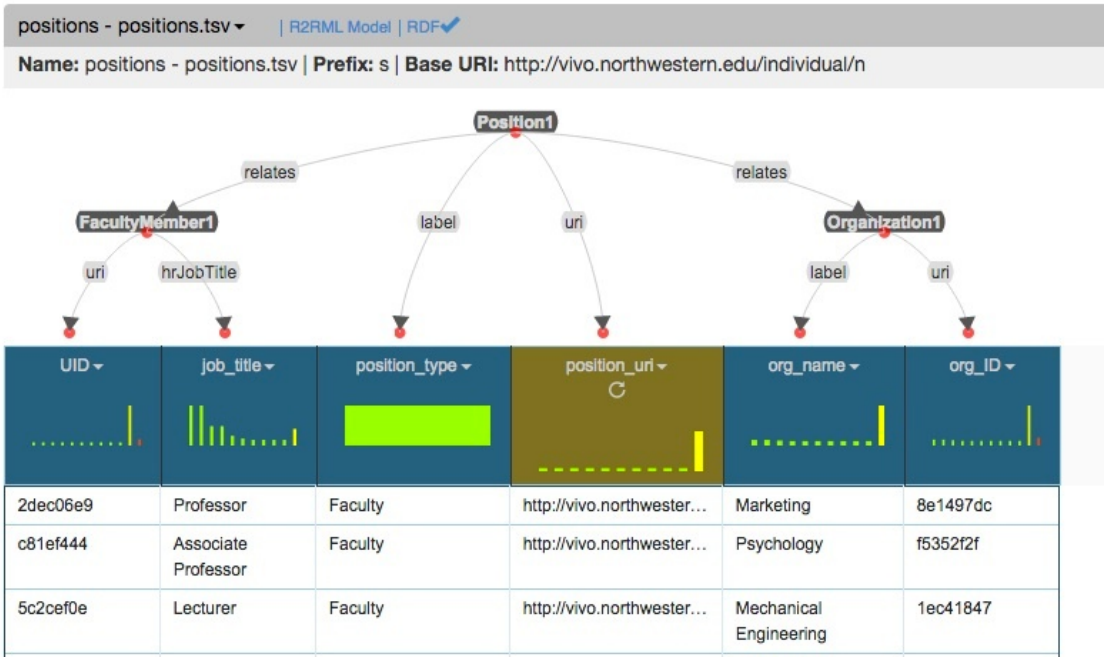

Figure 3.19: The user interface of KARMA.

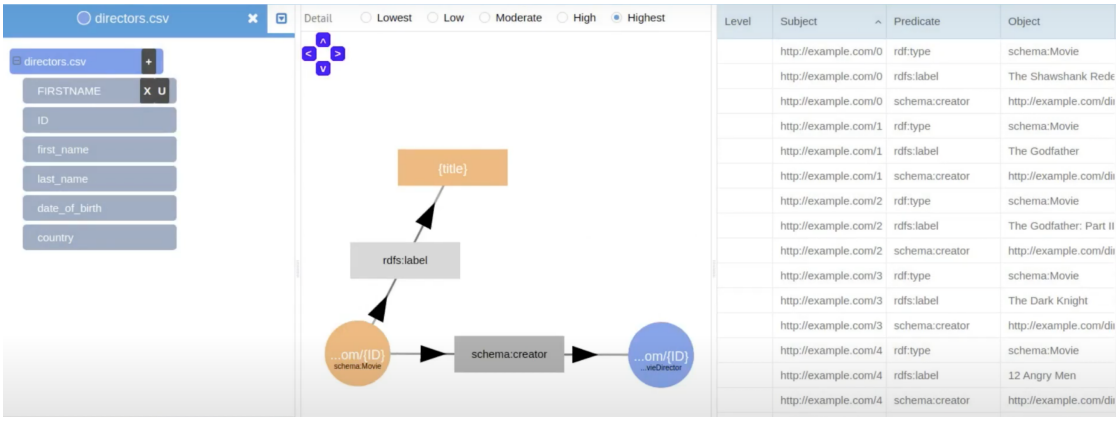

Figure 3.20: The user interface of RMLEditor.

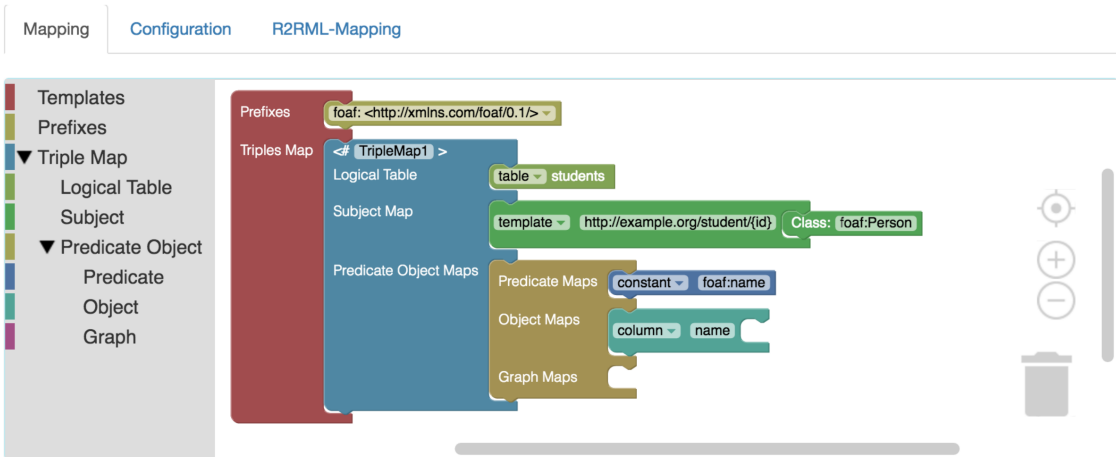

Figure 3.21: The user interface of Juma.

However, these tools are not easy to use for users that are not familiar with RDF concepts because they require to search for ontologies and understand RDFS and OWL rules.

We propose **SemanticBot** [49], a conversational interface that can generate an RDF mapping from a structured dataset by only asking simple questions to users about their dataset. Our tool can simply and quickly integrate datasets as Linked Data and encourages new users to make their first steps into Linked Data.

3.4.1 SemanticBot: A Conversational Interface to Generate RDF Mappings

To generate an RDF mapping from a structured dataset, our tool uses two knowledge graphs, DBpedia and YAGO, the ontologies of LOV[13], and the semantic web languages OWL and RDFS.

Roughly speaking, from a set of instances of each column, our tool searches corresponding entities in DBpedia and YAGO. The goal is to find a class corresponding to each column. Then, similarly, LOV is used to find the most relevant properties that may correspond to column names, such that instances of a column correspond to the object of a property. To confirm and complete these correspondences, the tool asks simple questions to the user. User confirmed correspondences allow to generate a first RDF mapping. Finally, this mapping is saturated with entailment rules of OWL and RDFS.[14] Figure 3.22 shows the architecture of the SemanticBot.

[13] https://lov.linkeddata.es/dataset/lov/

[14] We only consider rules 2, 3, 5, 7, 9 and 11 from RDFS: https://www.w3.org/TR/rdf11-mt/#rdfs-entailment and rules based on `owl:equivalentClass` and `owl:equivalentProperty` from OWL: https://www.w3.org/TR/owl-ref

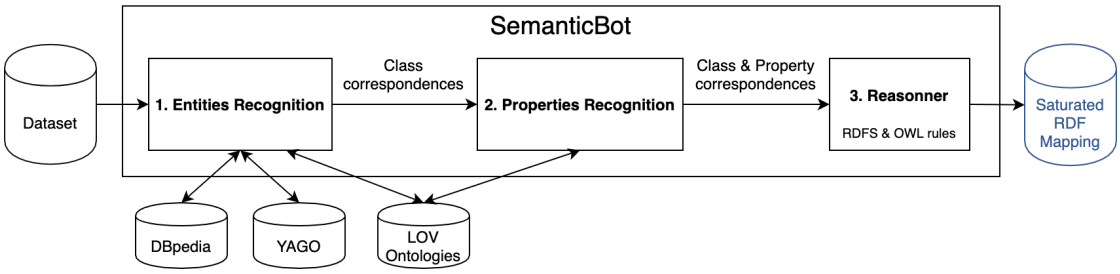

Figure 3.22: The global architecture of Semanticbot.

Using the Roman emperor dataset of Table 3.1, in the *class recognition* step, the *Augustus* value of the *Name* column corresponds to the entity `dbpedia:Augustus` of the class `dbo:Person` in DBpedia. Thus, *Augustus* is identified as an entity of the class `dbo:Person`. In this example, we obtain two class correspondences suggesting that columns *Name* and *Birth City* contain respectively entities of the classes `dbo:Person` and `dbo:Place`. These correspondences are suggested to the user with simple yes or no questions: "Does the column Name in your dataset contain Persons?". In order to hide IRIs, questions are built using the `rdfs:label` property of classes.

In the *property recognition* step, our tool obtains 5 property correspondences suggesting that columns *Name*, *Birth*, *Birth City*, *Lat* and *Long* are respectively objects of properties `dbo:name`, `locah:dateBirth`, `dbo:birthPlace`, `geo:lat` and `geo:long`. Again, these correspondences are suggested to the user with simple yes or no questions: "It seems that the column Lat is the latitude of a Spatial thing. Is it true?". These questions are built using the `rdfs:label` and the `rdfs:domain` of the property.

To complete confirmed correspondences, the tool asks the user to select the column of the dataset that will correspond to the subject of the property. If the user confirms the `geo:lat` correspondence with column *Lat*, the tool asks "latitude is a characteristic of a Spatial Thing. Select the column that contains Spatial Thing.". In our example, if the user answers correctly, the column *Birth Province* is used as the subject of the `geo:lat` property.

Our tool uses heuristics to reduce the number of questions. It suggests at most one class and one property for each column. Only the class that corresponds to the most instances of a column is suggested. The property that is suggested for a column is the property that has the best popularity score in the LOV answer. Our tool does not suggest a property if its LOV score is lower than a fixed lower bound. Moreover, to improve the pertinence of suggested properties, the type of a column can also be added in the text search. In our example, it searches for the property *Birth date* instead of *Birth*.

From user confirmed correspondences, our tool generates a first RDF map-

ping. In a final step, our tool saturates this mapping by applying RDFS and OWL entailment rules. Using the range and domain of all properties (`rdfs:range` and `rdfs:domain`), new classes are infered. This is possible because the domain of a property represents the class of the subject and the range represents the type (i.e., a class or the literal datatype) of the object. In our example, for instance, the user defined *Birth Province* as the subject of the *latitude* property. In the GeoNames ontology, the `rdfs:domain` of this property is `geo:SpatialThing`. Thus, our tool infers that the column *Birth Province* contains entities of type `geo:SpatialThing`. Our tool also takes into account `owl:equivalentClass`, `owl:equivalentProperty`, `rdfs:subClassOf` and `rdfs:subPropertyOf` properties of concerned ontologies. For example, column *Birth City* containing entities of the class `dbo:Place` are also considered as entities of the classes `dbo:Location`, `schema:Place` and `geo:SpatialThing`.

3.4.2 Implementation and Demonstration Tool

We implemented a chatbot-like tool that is able to generate YARRRML mappings for datasets of the Opendatasoft's Data Network[15]. We chose YARRRML because, at our knowledge, it is the most human readable and understandable RDF mapping syntax for users that are not familiar with LD. Listing 13 in Appendix A shows an RDF mapping for the Roman emperors dataset serialized in YARRRML. Source code of out tool is available at GitHub[16] under the MIT license. Our tool is also available as a web service[17]. Figure 3.23 shows the user interface of SemanticBot.

[15] https://data.opendatasoft.com
[16] https://github.com/opendatasoft/ontology-mapping-chatbot
[17] https://semanticbot.opendatasoft.com/

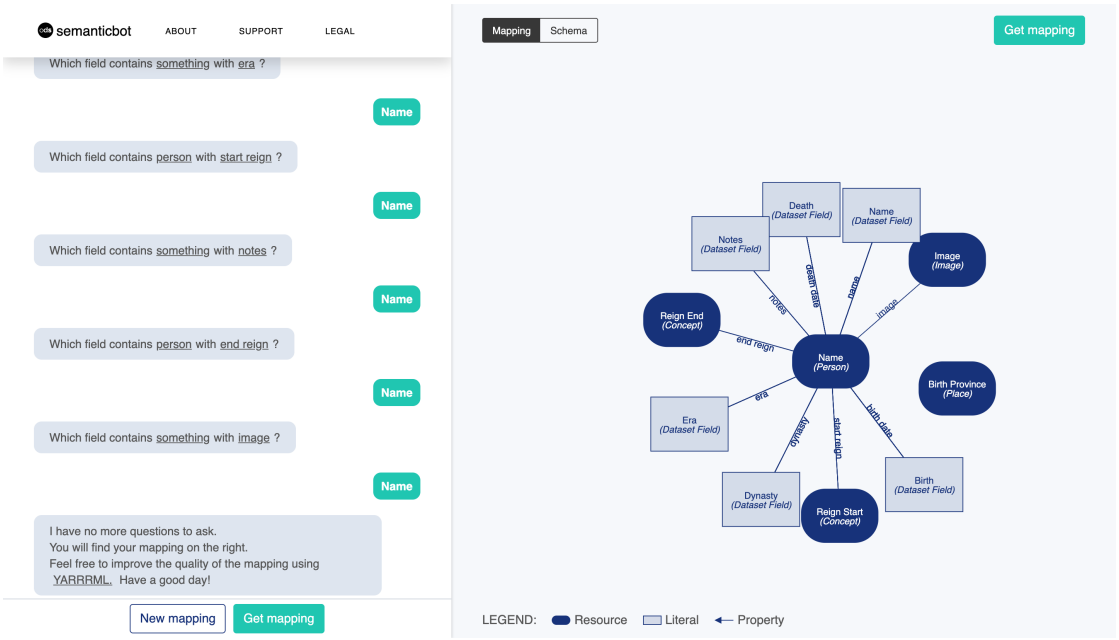

Figure 3.23: The user interface of SemanticBot.

A limitation of this tool is that the quality of the resulting RDF mapping is highly dependant on:

- the quality of the initial dataset and, in particular its column names, data types, and entities;
- the instances described in DBpedia and YAGO datasets;
- and the ontologies available on Linked Open Vocabularies.

SemanticBot does not aim to generate the best mapping for a dataset but, the first mapping for a user that wants to try semantic web technologies and see how Linked Data can enrich its dataset.

3.5 Conclusion

This chapter proposed solutions for difficulties that data producers face during the publication of non-RDF datasets on the Web of Data. In particular, we presented three approaches: ODMTP to query non-RDF datasets with SPARQL using mappings, EvaMap to evaluate the quality and refine mappings, and SemanticBot to semi-automatically generate RDF mappings for dataset.

Through these three tools, we proposed a complete workflow to publish non-RDF datasets on the Web of Data. Figure 3.24 shows how SemanticBot, EvaMap, and ODMTP can be used together to publish RDF datasets.

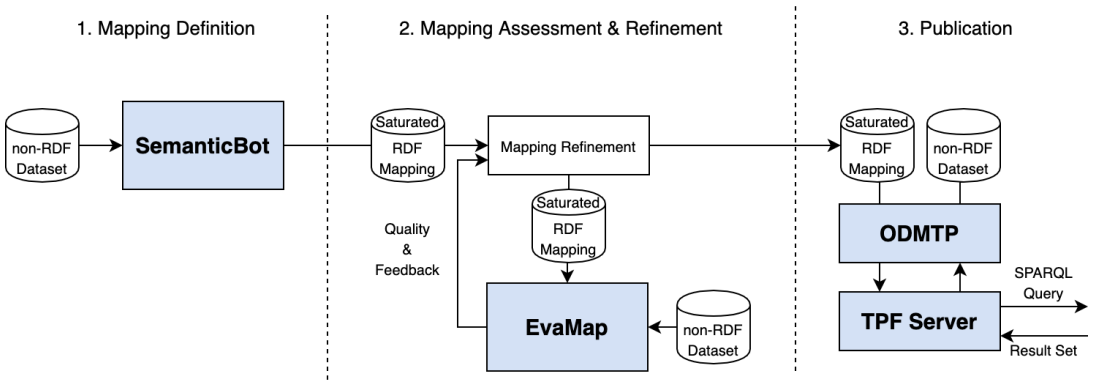

Figure 3.24: The worflow to publish non-RDF datasets on the Web of Data.

SemanticBot can be used to generate a first RDF mapping for the dataset. But, this mapping is not always complete. Thus, mapping quality and feedback returned by EvaMap can be used to refine the mapping. RDF mapping can be improved iteratively during the assessment and refinement process. Finally, the dataset can be published on the Web of Data using the ODMTP approach.

In this worflow, each tool can be replaced by other existing tools. This interoperability among tools is possible thanks to the explicit description of mappings using a state of the art RDF mapping language. For example, the mapping refinement process can be made with RDF mapping editors presented before (i.e., RMLeditor, Karma, or Juma). These approaches are used on the Opendatasoft platform to integrate hundreds of open datasets into the Web of Data.

Chapter 4

Modelling the Compatibility of Licenses

Contents

Once a dataset is ready for integration on the Web of Data, it has to be associated with a license[64]. The license protects and facilitates the reuse of the dataset by specifying the conditions of reuse, i.e., what actions are *permitted*, *obliged* and *prohibited* when using the dataset.

The Web of Data facilitate combining datasets to create new ones. For a dataset producer, choosing the appropriate license for a combined dataset is not easy. It involves choosing a license compliant with all the licenses of combined datasets and analyzing the reusability of the resulting dataset through the compatibility

of its license. The risk is either, to choose a license too restrictive making the dataset difficult to reuse, or to choose a not enough restrictive license that will not sufficiently protect the dataset. Finding the right trade-off between compliance and compatibility is a difficult process. An automatic ordering over licenses would facilitate this task. In this chapter, we present CaLi, a model that partially orders licenses. It answers our research question: *given a license l_i, how to automatically position l_i over a set of licenses in terms of compatibility and compliance?*

This chapter is based on articles [44, 46, 47, 48] and is organized as follows. Section 4.1 illustrates and motivates our research question. Section 4.2 discuses related works, Section 4.3 introduces the CaLi model, Section 4.4 illustrates the usability of our model, Section 4.5 shows experiments of the implemented algorithm as well as the prototype of a license-based search engine, and Section 4.6 concludes.

4.1 Introduction and Motivation

Relations of compatibility, compliance and restrictiveness on licenses could be very useful in a wide range of applications. Imagine license-based search engines for services such as GitHub[1], APISearch[2], LODAtlas[3], DataHub[4], Google Dataset Search[5] or OpenDataSoft[6] that could find resources licensed under licenses compatible or compliant with a specific license. Answers could be partially ordered from the least to the most restrictive license. We argue that a model for license orderings would allow the development of such applications.

We consider simplified definitions of compliance and compatibility inspired by works like [19, 23, 34, 70]: *a license l_j is compliant with a license l_i if a resource licensed under l_i can be licensed under l_j without violating l_i.* If a license l_j is compliant with l_i then we consider that l_i is compatible with l_j and that resources licensed under l_i are reusable with resources licensed under l_j. In general, if l_i is compatible with l_j then l_j is more (or equally) restrictive than l_i. We also consider that *a license l_j is more (or equally) restrictive than a license l_i if l_j allows at most the same permissions and has at least the same prohibitions/obligations than l_i.*

Usually but not always, when l_i is less restrictive than l_j then l_i is compatible with l_j. For instance, see Figure 4.1 that shows an excerpt of three Creative Commons (CC)[7] licenses described in RDF and using the ODRL vocabulary[8].

[1] https://github.com/
[2] http://apis.io/
[3] http://lodatlas.lri.fr/
[4] https://datahub.io/
[5] https://toolbox.google.com/datasetsearch
[6] https://data.opendatasoft.com/
[7] https://creativecommons.org/
[8] The term *duty* is used for obligations https://www.w3.org/TR/odrl-model/

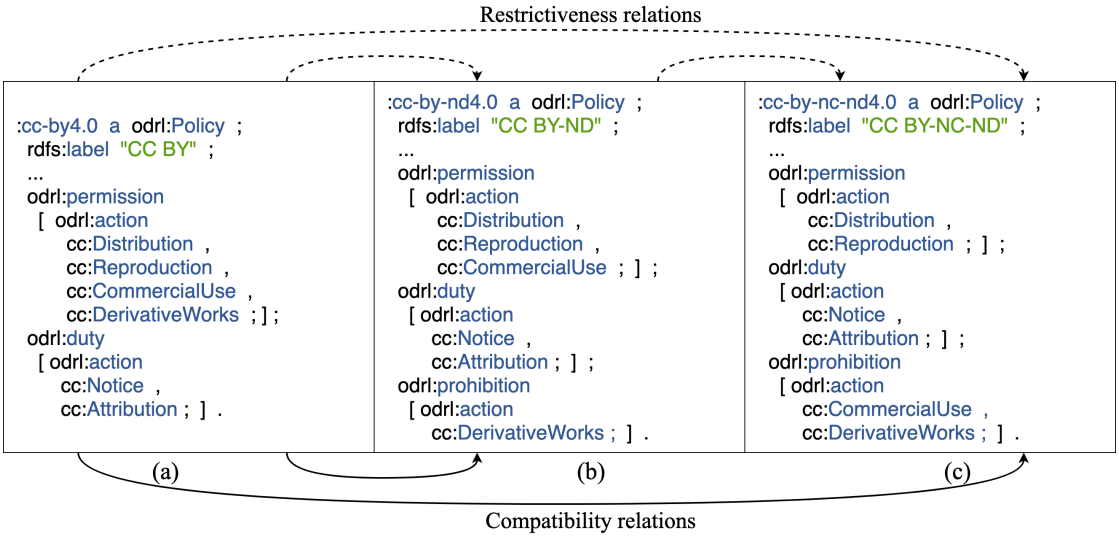

Figure 4.1: A set of three RDF Creative Commons licenses with their compatibility and restrictiveness relationships.

Notice that there exists a restrictiveness order among these licenses, (a) is less restrictive than (b) and (b) is less restrictive than (c). By transitivity (a) is less restrictive than (c). Notice also that (a) is compatible with (b) and (c), but (b) is not compatible with (c). This is due to the semantics of the prohibited action *DerivativeWorks* that forbids the distribution of a derivation (remix, transform or build upon) of the protected resource under a different license. Thus, depending on the semantics of their actions, a restrictiveness relation between two licenses does not imply a compatibility relation.

Our research question is: *given a license l_i, how to automatically position l_i over a set of licenses in terms of compatibility and compliance?* The challenge we face is how to generalise the automatic definition of the ordering relations among licenses while taking into account the influence of the semantics of actions.

Inspired by lattice-based access control models [11, 63], we propose **CaLi** (ClAssification of LIcenses) [48], a model for license orderings that uses restrictiveness relations and constraints among licenses to define compatibility and compliance. We validate experimentally CaLi with a quadratic algorithm and show its usability through a prototype of a license-based search engine [46]. Our work is a step towards facilitating and encouraging the publication and reuse of licensed resources in the Web of Data. However, it is not intended to provide legal advice.

4.2 Related Work

Machine-readable licenses Automatic license classification requires machine-readable licenses. License expression languages such as CC REL[9], ODRL, or L4LOD[10] enable fine-grained RDF description of licenses. Works like [59] and [6] use natural language processing to automatically generate RDF licenses from licenses described in natural language. Other works such as [8, 27, 57] propose a set of well-known licenses in RDF described in CC REL and ODRL. Thereby, in this work, we suppose that there exist consistent licenses described in RDF.

Tools for license compliant resources There exist some tools to facilitate the creation of license compliant resources. TLDRLegal[11], CC Choose[12] and ChooseALicense[13] help users to choose actions to form a license for their resources. CC search[14] allows users to find images licensed under Creative Commons licenses that can be commercialized, modified, adapted, or built upon. DALICC [27] allows to compose arbitrary licenses and provides information about equivalence, similarity and compatibility of licenses. Finally, Licentia[15], based on deontic logic to reason over the licenses, proposes a web service to find licenses compatible with a set of permissions, obligations and prohibitions chosen by the user. From these tools, only Licentia and DALICC use machine-readable licenses[16,17] in RDF. But unfortunately, these works do not order licenses in terms of compatibility or compliance.

License compatibility and license combination The easiest way to choose a license for a combined resource is to create a new one by combining all resource licenses to combine. Several works address the problem of license compatibility and license combination. In web services, [19] proposes a framework that analyses compatibility of licenses to verify if two services are compatible and then generates the composite service license. [38] addresses the problem of license preservation during the combination of digital resources (music, data, picture, etc.) in a collaborative environment. Licenses of combined resources are combined into a new one. In the Web of Data, [70] proposes a framework to check compatibility among CC REL licenses. If licenses are compatible, a new license compliant with

[9] https://creativecommons.org/ns
[10] https://ns.inria.fr/l4lod/
[11] https://tldrlegal.com/
[12] https://creativecommons.org/choose/
[13] https://choosealicense.com/
[14] https://ccsearch.creativecommons.org/
[15] http://licentia.inria.fr/
[16] http://rdflicense.appspot.com/rdflicense
[17] https://www.dalicc.net/license-library

combined ones is generated. [23] formally defines the combination of licenses using deontic logic. [65] proposes PrODUCE, an approach to combine usage policies taking into account the usage context. These works focus on combining operators for automatic license combination but do not propose to position a license over a set of licenses.

License ordering Concerning the problem of license ordering to facilitate the selection of a license, [9] uses Formal Concept Analysis (FCA) to generate a lattice of actions. Once pruned and annotated, this lattice can be used to classify licenses in terms of features. This classification reduces the selection of a license to an average of three to five questions. However, this work does not address the problem of license compatibility. Moreover, FCA is not suitable to generate compatibility or restrictiveness relations among licenses. FCA defines a derivation operator on objects that returns a set of attributes shared by the objects. We consider that the set of actions in common of two licenses is not enough to infer these relations. If applied to our introductory example, FCA can only work with permissions but not with obligations and prohibitions. That is because l_i is less restrictive than l_j if permissions of l_i are a superset of permissions of l_j, but regarding obligations and prohibitions, l_i is less restrictive than l_j if they are a subset of those of l_j. In the context of Free Open Source Software (FOSS), [34] proposes an approach, based on a directed acyclic graph, to detect license violations in existing software packages. It considers that license l_i is compatible with l_j if the graph contains a path from l_i to l_j. However, as such a graph is built from a manual interpretation of each license, its generalisation and automation is not possible.

Lattice-based access control In the domain of access control, [11] proposes a lattice model of secure information flow. This model classifies security classes with associated resources. Like in the compatibility graph of [34], security class sc_i is compatible with sc_j if the lattice contains a path from sc_i to sc_j. Thus, this path represents the authorized flow of resources (e.g., resource r_i protected with sc_i can flow to a resource protected by sc_j without violating sc_i.). The lattice can be generated automatically through a pairwise combination of all security classes if sc_i combined with sc_k gives sc_j where sc_i and sc_k are both compatible with sc_j. [63] describes several models based on this approach but none focuses on classifying licenses.

None of these works answers our research question. They do not allow to automatically position a license over a set of licenses in terms of compatibility or compliance. In our work we propose a lattice-based model inspired by [11]. This

model is independent of any license description language, application context and licensed resource so that it can be used in a wide variety of domains.

4.3 CaLi: a Lattice-based License Model

We propose to partially order licenses in terms of compatibility and compliance through the restrictiveness relation. In a license, actions can be distributed in what we call *status*, e.g., permissions, obligations and prohibitions. To decide if a license l_i is less restrictive than l_j, it is necessary to know if an action in a status is considered as less restrictive than the same action in another status. In the introductory example (Figure 4.1), we consider that permissions are less restrictive than obligations, which are less restrictive than prohibitions, i.e., $Permission \leqslant Duty \leqslant Prohibition$. This relation can be seen in Fig 4.2b.

We remark that if two licenses have a restrictiveness relation then it is possible that they have a compatibility relation too. The restrictiveness relation between the licenses can be automatically obtained according to the status of actions without taking into account the semantics of the actions. Thus, based on lattice-ordered sets [10], we define a restrictiveness relation among licenses.

To identify the compatibility among licenses, we refine the restrictiveness relation with constraints. The goal is to take into account the semantics of actions. Constraints also distinguish valid licenses from non-valid ones. We consider a license l_i as non-valid if a resource can not be licensed under l_i, e.g., a license that simultaneously permits the *Derive* action[18] and prohibits *DerivativeWorks*[19].

This approach is based on:

1. a set of *actions* (e.g., *read, modify, distribute*, etc.);

2. a *restrictiveness lattice of status* that defines (i) all possible status of an action in a license (i.e., permission, obligation, prohibition, recommendation, undefined, etc.) and (ii) the restrictiveness relation among status; a *restrictiveness lattice of licenses* is obtained from a combination of 1 and 2;

3. a set of *compatibility constraints* to identify if a restrictiveness relation between two licenses is also a compatibility relation; and

4. a set of *license constraints* to identify non-valid licenses.

Next section introduces formally the CaLi model and Section 4.3.2 introduces a simple example of a CaLi ordering.

[18] https://www.w3.org/TR/odrl-vocab/#term-derive
[19] https://www.w3.org/TR/odrl-vocab/#term-DerivativeWorks

4.3.1 Formal model description

Before introducing our definitions, we recall some basis related to lattices. A *partially ordered set* (or poset) is a pair $(X, \leqslant)$, where X is a set and $\leqslant$ is a reflexive, transitive and antisymmetric binary relation on X. For a subset $Y \subset X$ and an element x of X, x is *an upper bound* of Y if it is greater than all elements of Y, that is if $\forall y \in Y, y \leqslant x$. Moreover, x is *the least upper bound* (or *supremum*) of Y if it is smaller than all upper bounds of Y, that is $\forall z \in X, (\forall y \in Y, y \leqslant z) \Rightarrow x \leqslant z$. Symmetrically, a *lower bound* of Y is smaller than all elements in Y, and the greatest of them, if it exists, is called *the greatest lower bound* (or *infimum*) of Y. If they exist, the greatest lower bound and the least upper bound are necessarily unique. Let $(X, \leqslant_X)$ and $(Y, \leqslant_Y)$ be two posets. Their product $(X, \leqslant_X) \times (Y, \leqslant_Y)$ is defined as the pair $(X \times Y, \leqslant)$, where $\leqslant$ designates the *coordinatewise order* on $X \times Y$ defined as follows. For all $(x_i, y_i), (x_j, y_j) \in X \times Y$, $(x_i, y_i) \leqslant (x_j, y_j)$ if $x_i \leqslant_X x_j$ and $y_i \leqslant_Y y_j$. We denote $\prod_{i=1}^{n}(X_i, \leqslant_i) = (X_1, \leqslant_1) \times \cdots \times (X_n, \leqslant_n)$. A poset $(X, \leqslant)$ is a *lattice* [10] if any pair $\{x, y\}$ of elements of X has a least upper bound, called the *join* of x and y and denoted $x \vee y$, and a greatest lower bound, called the *meet* of x and y and denoted $x \wedge y$. If $(X, \leqslant_X)$ and $(Y, \leqslant_Y)$ are two lattices, then $(X, \leqslant_X) \times (Y, \leqslant_Y)$ is also a lattice.

We now formally introduce the CaLi model. We first define a *restrictiveness lattice of status*. We use a lattice structure because it is necessary, for every pair of status, to know which status is less (or more) restrictive than both.

Definition 1 (Restrictiveness lattice of status $\mathcal{LS}$)**.**
A restrictiveness lattice of status *is a lattice* $\mathcal{LS} = (S, \leqslant_S)$ *that defines all possible* status S *for a license and the relation* $\leqslant_S$ *as the* restrictiveness relation *over* S*. For two status* s_i*,* s_j*, if* $s_i \leqslant_S s_j$ *then* s_i *is less restrictive than* s_j*.*

Different $\mathcal{LS}$s can be defined according to the application domain. Figure 4.2 shows examples of $\mathcal{LS}$s. Dashed arrows represent restrictiveness, for example, in 4.2a, Permission is less restrictive than Prohibition (i.e., *Permission* $\leqslant_S$ *Prohibition*). Figure 4.2a shows the diagram of a $\mathcal{LS}$ inspired by file systems where actions can be either prohibited or permitted. With this lattice, prohibiting to read a file is more restrictive than permitting to read it. Figure 4.2b illustrates a $\mathcal{LS}$ for CC licenses where actions are either permitted, required (Duty) or prohibited. Figure 4.2c shows a $\mathcal{LS}$ inspired by the ODRL vocabulary. In ODRL, actions can be either permitted, obliged, prohibited or not specified (i.e., undefined). In this lattice, the undefined status is the least restrictive and the prohibited one the most restrictive. Figure 4.2d shows a $\mathcal{LS}$ where a recommended or permitted action is less restrictive than the same action when it is permitted and recommended.

Now we formally define a license based on the status of its actions.

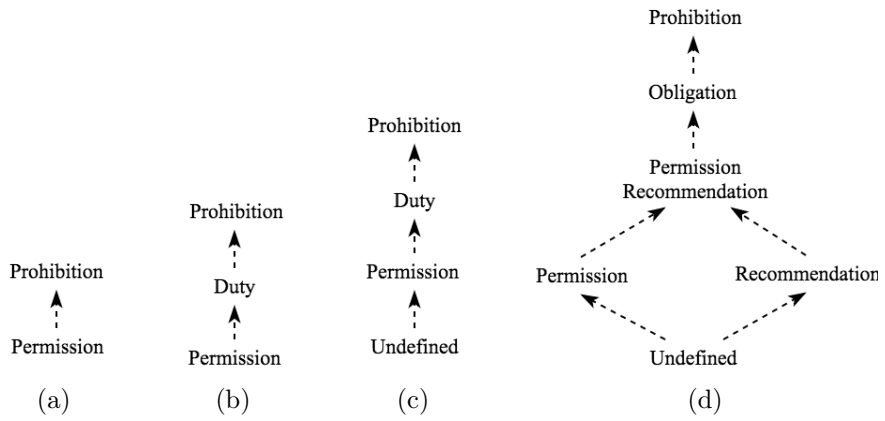

Figure 4.2: Examples of restrictiveness lattices of status ($\mathcal{LS}$).

Definition 2 (License)**.**
Let $\mathcal{A}$ be a set of actions and $\mathcal{LS} = (S, \leqslant_S)$ be a restrictiveness lattice of status. A license *is a function $l : \mathcal{A} \to S$. We denote by $\mathcal{L}_{\mathcal{A},\mathcal{LS}}$ the set of all licenses.*

For example, consider $\mathcal{A} = \{read, modify, distribute\}$, $\mathcal{LS}$ the lattice of Figure 4.2c and two licenses: l_i which permits *read* and *distribute* but where *modify* is undefined and l_j where *modify* is also undefined but which permits *read* and prohibits *distribute*. We define l_i and l_j as follows:

$\forall a \in \mathcal{A}$:

$$l_i(a) = \begin{cases} \text{Undefined} & \text{if } a \in \{modify\}; \\ \text{Permission} & \text{if } a \in \{read, distribute\}. \end{cases}$$

$$l_j(a) = \begin{cases} \text{Undefined} & \text{if } a \in \{modify\}; \\ \text{Permission} & \text{if } a \in \{read\}; \\ \text{Prohibition} & \text{if } a \in \{distribute\}. \end{cases}$$

A restrictiveness lattice of status and a set of licenses make possible to partially order licenses in a restrictiveness lattice of licenses.

Definition 3 (Restrictiveness relation over licenses)**.**
Let A be a set of actions and $\mathcal{LS} = (S, \leqslant_S)$ be a restrictiveness lattice of status associated to the join and meet operators $\vee_S$ and $\wedge_S$, and $l_i, l_j \in \mathcal{L}_{\mathcal{A},\mathcal{LS}}$ be two licenses. We say that l_i is less restrictive than l_j, denoted $l_i \leqslant_{\mathcal{R}} l_j$, if for all actions $a \in \mathcal{A}$, the status of a in l_i is less restrictive than the status of a in l_j. That is, $l_i \leqslant_{\mathcal{R}} l_j$ if $\forall a \in \mathcal{A}, l_i(a) \leqslant_S l_j(a)$.

Moreover, we define the two operators $\vee$ and $\wedge$ as follows. For all actions $a \in \mathcal{A}$, the status of a in $l_i \vee l_j$ (resp. $l_i \wedge l_j$) is the join (resp. meet) of the

status of a in l_i and the status of a in l_j. That is, $(l_i \vee l_j)(a) = l_i(a) \vee_S l_j(a)$ and $(l_i \wedge l_j)(a) = l_i(a) \wedge_S l_j(a)$.

For example, consider $\mathcal{LS}$ the lattice of Figure 4.2c, and licenses l_i and l_j defined previously; $l_i \leqslant_{\mathcal{R}} l_j$ because $l_i(read) \leqslant_S l_j(read), l_i(modify) \leqslant_S l_j(modify)$ and $l_i(distribute) \leqslant_S l_j(distribute)$. In this example, $l_i \vee l_j = l_j$ because $\forall a \in \mathcal{A}$, $(l_i \vee l_j)(a) = l_j(a)$, e.g., $(l_i \vee l_j)(distribute) = l_j(distribute) = Prohibition$. If for an action, it is not possible to say which license is the most restrictive then the compared licenses are not comparable by the restrictiveness relation.

Remark 1. *The pair $(\mathcal{L}_{\mathcal{A},\mathcal{LS}}, \leqslant_{\mathcal{R}})$ is a restrictiveness lattice of licenses, whose $\vee$ and $\wedge$ are respectively the join and meet operators.*

In other words, for two licenses l_i and l_j, $l_i \vee l_j$ (resp. $l_i \wedge l_j$) is the least (resp. most) restrictive license that is more (resp. less) restrictive than both l_i and l_j.

Remark 2. *For an action $a \in \mathcal{A}$, we call $(\mathcal{L}_{\{a\},\mathcal{LS}}, \leqslant_{\mathcal{R}})$ the* action lattice *of a. Remark that $(\mathcal{L}_{\mathcal{A},\mathcal{LS}}, \leqslant_{\mathcal{R}})$ and $\prod_{a\in\mathcal{A}}(\mathcal{L}_{\{a\},\mathcal{LS}}, \leqslant_{\mathcal{R}})$ are isomorphic. That is, a restrictiveness lattice of licenses can be generated through the coordinatewise product [10] of all its* action lattices. *The total number of licenses in this lattice is $|\mathcal{LS}|^{|\mathcal{A}|}$.*

For example, consider $\mathcal{A} = \{read, modify\}$, $\mathcal{LS}$ the lattice of Figure 4.2a, $(\mathcal{L}_{\mathcal{A},\mathcal{LS}}, \leqslant_{\mathcal{R}})$ is isomorphic to $(\mathcal{L}_{\{read\},\mathcal{LS}}, \leqslant_{\mathcal{R}}) \times (\mathcal{L}_{\{modify\},\mathcal{LS}}, \leqslant_{\mathcal{R}})$. Figure 4.3a,b,c illustrates the product of these action lattices and the produced restrictiveness lattice of licenses.

To identify the compatibility relation among licenses and to distinguish valid licenses from non-valid ones it is necessary to take into account the semantics of actions. Thus, we apply two types of constraints to the restrictiveness lattice of licenses: license constraints and compatibility constraints.

Definition 4 (License constraint)**.**
Let $\mathcal{L}_{\mathcal{A},\mathcal{LS}}$ be a set of licenses. A license constraint is a function $\omega_{\mathcal{L}} : \mathcal{L}_{\mathcal{A},\mathcal{LS}} \rightarrow Boolean$ which identifies if a license is valid or not.

For example, the license constraint $\omega_{\mathcal{L}_1}$ considers a license $l_i \in \mathcal{L}_{\mathcal{A},\mathcal{LS}}$ non-valid if *read* is prohibited but modification is permitted (i.e., a *modify* action implies a *read* action):

$$\omega_{\mathcal{L}_1}(l_i) = \begin{cases} False & \text{if } l_i(read) = \text{Prohibition and } l_i(modify) = \text{Permission;} \\ True & \text{otherwise.} \end{cases}$$

Definition 5 (Compatibility constraint)**.**
Let $(\mathcal{L}_{\mathcal{A},\mathcal{LS}}, \leqslant_{\mathcal{R}})$ be a restrictiveness lattice of licenses. A compatibility constraint is a function $\omega_{\rightarrow} : \mathcal{L}_{\mathcal{A},\mathcal{LS}} \times \mathcal{L}_{\mathcal{A},\mathcal{LS}} \rightarrow Boolean$ which constraints the restrictiveness relation $\leqslant_{\mathcal{R}}$ to identify compatibility relations among licenses.

For example, consider that a license prohibits the action *modify*. In the spirit of *DerivativeWork*, we consider that the distribution of the modified resource under a different license is prohibited. Thus, the compatibility constraint $\omega_{\rightarrow_1}$, considers that a restrictiveness relation $l_i \leqslant_{\mathcal{R}} l_j$ can be also a compatibility relation if l_i does not prohibit *modify*. This constraint is described as:

For $l_i, l_j \in \mathcal{L}_{\mathcal{A},\mathcal{LS}}$,

$$\omega_{\rightarrow_1}(l_i, l_j) = \begin{cases} False & \text{if } l_i(modify) = \text{Prohibition;} \\ True & \text{otherwise.} \end{cases}$$

Now we are able to define a CaLi ordering from a restrictiveness lattice of licenses and constraints defined before.

Definition 6 (CaLi ordering)**.**
A CaLi ordering is a tuple $\langle \mathcal{A}, \mathcal{LS}, C_{\mathcal{L}}, C_{\rightarrow} \rangle$ *such that* $\mathcal{A}$ *and* $\mathcal{LS}$ *form a restrictiveness lattice of licenses* $(\mathcal{L}_{\mathcal{A},\mathcal{LS}}, \leqslant_{\mathcal{R}})$*,* $C_{\mathcal{L}}$ *is a set of license constraints and* $C_{\rightarrow}$ *is a set of compatibility constraints. For two licenses* $l_i \leqslant_{\mathcal{R}} l_j \in \mathcal{L}_{\mathcal{A},\mathcal{LS}}$*, we say that* l_i is compatible with l_j*, denoted by* $l_i \rightarrow l_j$*, if* $\forall \omega_{\mathcal{L}} \in C_{\mathcal{L}}, \omega_{\mathcal{L}}(l_i) = \omega_{\mathcal{L}}(l_j) = True$ *and* $\forall \omega_{\rightarrow} \in C_{\rightarrow}, \omega_{\rightarrow}(l_i, l_j) = True$*.*

Remark 3. *We define the* compliance relation *as the opposite of the* compatibility relation*. For two licenses* l_i, l_j*, if* $l_i \rightarrow l_j$ *then* l_j is compliant with l_i*.*

A CaLi ordering is able to answer our research question, *given a license* l_i*, how to automatically position* l_i *over a set of licenses in terms of compatibility and compliance?* It allows to evaluate the potential reuse of a resource depending on its license. Knowing the compatibility of a license allows to know to which extent the protected resource is reusable. On the other hand, knowing the compliance of a license allows to know to which extent other licensed resources can be reused. Next section shows an example of CaLi ordering.

4.3.2 Example

Consider a CaLi ordering $\langle \mathcal{A}, \mathcal{LS}, \{\omega_{\mathcal{L}_1}\}, \{\omega_{\rightarrow_1}\} \rangle$ such that:

- $\mathcal{A}$ is the set of actions $\{read, modify\}$,
- $\mathcal{LS}$ is a restrictiveness lattice of status where an action can be either permitted or prohibited, and $Permission \leqslant_S Prohibition$ (cf Figure 4.2a),
- $\omega_{\mathcal{L}_1}$ is the license constraint introduced in the example of Def. 4, and
- $\omega_{\rightarrow_1}$ is the compatibility constraint introduced in the example of Def. 5.

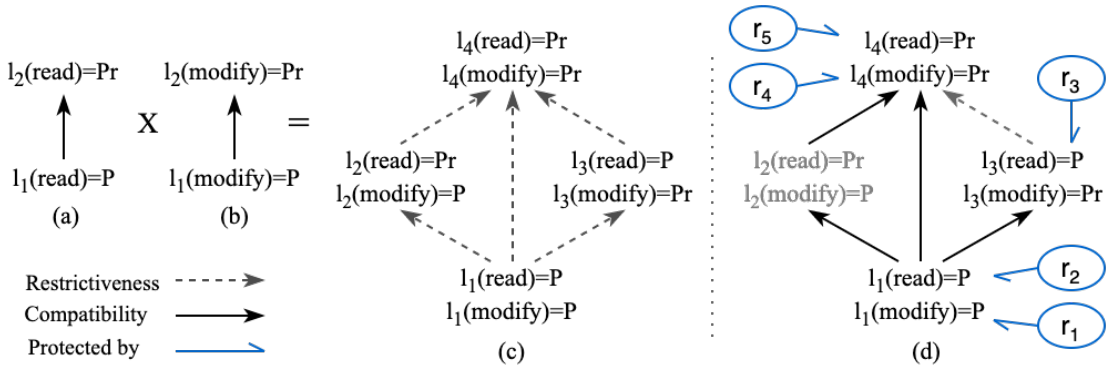

Figure 4.3: An example of CaLi ordering.

Figure 4.3 shows how this CaLi ordering can be obtained. Figure 4.3a and Figure 4.3b are the action latices $(\mathcal{L}_{\{read\},\mathcal{LS}}, \leqslant_{\mathcal{R}})$ and $(\mathcal{L}_{\{modify\},\mathcal{LS}}, \leqslant_{\mathcal{R}})$, where $\mathcal{A} = \{read, modify\}$ and $\mathcal{LS}$ is the lattice of Figure 4.2a (Pr=Prohibition and P=Permission). The product of these action lattices gives the restrictiveness lattice of licenses Figure 4.3c $(\mathcal{L}_{\mathcal{A},\mathcal{LS}}, \leqslant_{\mathcal{R}})$ (reflexive relations are not represented). Figure 4.3d is the CaLi ordering $\langle \mathcal{A}, \mathcal{LS}, \{\omega_{\mathcal{L}_1}\}, \{\omega_{\rightarrow_1}\}\rangle$.

Figure 4.3d shows a visual representation of this CaLi ordering. Licenses in grey are identified as non-valid by $\omega_{\mathcal{L}_1}$. They are part of the ordering but cannot protect resources. Dashed arrows represent restrictiveness relations $\leqslant_{\mathcal{R}}$. Black arrows represent restrictiveness relations that are also compatibility relations.

Consider a set of resources $R = \{r_1, r_2, r_3, r_4, r_5\}$. $\rightharpoonup$ is the *has license* relation such that $\{r_1, r_2\} \rightharpoonup l_1; r_3 \rightharpoonup l_3; \{r_4, r_5\} \rightharpoonup l_4$. Thanks to our CaLi ordering, next questions can be answered.

- *Which licensed resources can be reused in a resource that has as license l_3?* Those resource whose licenses are compatible with l_3: r_1 and r_2 that have license l_1 which precedes l_3, as well as r_3 that has the license l_3 itself.

- *Which licensed resources can reuse a resource that has as license l_1?* Those resource whose licenses are compliant with l_1: r_3, r_4 and r_5 that have licenses l_3 and l_4 which follow l_1, as well as r_1 and r_2 that have the license l_3 itself.

Resulting licenses can be returned ordered in a graph of compatibility.

We illustrated CaLi with a simple restrictiveness lattice of status, next section introduces a more realistic CaLi ordering inspired by licenses of Creative Commons.

4.4 A CaLi Ordering for Creative Commons Licenses

Creative Commons proposes 7 licenses that are legally verified, free of charge, easy-to-understand and widely used when publishing resources on the Web. These licenses use 7 actions that can be permitted, required or prohibited. In this CaLi example, we search to model a complete compatibility ordering of all possible valid licenses using these 7 actions.

4.4.1 Description of a CC ordering based on CaLi

Consider CC_CaLi, a CaLi ordering $\langle \mathcal{A}, \mathcal{LS}, C_{\mathcal{L}}, C_{\rightarrow} \rangle$ such that:

- $\mathcal{A}$ is the set of actions *{cc:Distribution, cc:Reproduction, cc:DerivativeWorks, cc:CommercialUse, cc:Notice, cc:Attribution, cc:ShareAlike}*,
- $\mathcal{LS}$ is the restrictiveness lattice of status depicted in 4.2b[20], and
- $C_{\mathcal{L}}, C_{\rightarrow}$ are the sets of constraints defined next.

$C_{\mathcal{L}} = \{\omega_{\mathcal{L}_2}, \omega_{\mathcal{L}_3}\}$ allows to invalidate a license (1) when *cc:CommercialUse* is required and (2) when *cc:ShareAlike* is prohibited:

$$\omega_{\mathcal{L}_2}(l_i) = \begin{cases} False & \text{if } l_i(\textit{cc:CommercialUse}) = \text{Duty;} \\ True & \text{otherwise.} \end{cases}$$

$$\omega_{\mathcal{L}_3}(l_i) = \begin{cases} False & \text{if } l_i(\textit{cc:ShareAlike}) = \text{Prohibition;} \\ True & \text{otherwise.} \end{cases}$$

$C_{\rightarrow} = \{\omega_{\rightarrow_2}, \omega_{\rightarrow_3}\}$ allows to identify (1) when *cc:ShareAlike* is required and (2) when *cc:DerivativeWorks* is prohibited. That is because *cc:ShareAlike* requires that the distribution of derivative works be under the same license only, and *cc:DerivativeWorks*, when prohibited, does not allow the distribution of a derivative resource, regardless of the license.

$$\omega_{\rightarrow_2}(l_i, l_j) = \begin{cases} False & \text{if } l_i(\textit{cc:ShareAlike}) = \text{Duty;} \\ True & \text{otherwise.} \end{cases}$$

$$\omega_{\rightarrow_3}(l_i, l_j) = \begin{cases} False & \text{if } l_i(\textit{cc:DerivativeWorks}) = \text{Prohibition;} \\ True & \text{otherwise.} \end{cases}$$

Other constraints could be defined to be closer to the CC schema[21] but for the purposes of this compatibility ordering these constraints are enough.

[20] To simplify, we consider that a requirement is a duty.

[21] https://creativecommons.org/ns

4.4.2 Analysis of CC_CaLi

The size of the restrictiveness lattice of licenses is 3^7 but the number of valid licenses of CC_CaLi is 972 due to $C_{\mathcal{L}}$. That is, 5 actions in whatever status and 2 actions (*cc:CommercialUse* and *cc:ShareAlike*) in only 2 status: $3^5 * 2^2$.

The following CC_CaLi licenses are like the official CC licenses.

$$CCBY(a) = \begin{cases} \text{Permission} & \text{if } a \in \{\textit{cc:Distribution, cc:Reproduction} \\ & \quad \textit{cc:DerivativeWorks, cc:CommercialUse} \\ & \quad \textit{cc:ShareAlike}\}; \\ \text{Duty} & \text{if } a \in \{\textit{cc:Notice, cc:Attribution}\}. \end{cases}$$

$$CCBYNC(a) = \begin{cases} \text{Permission} & \text{if } a \in \{\textit{cc:Distribution, cc:Reproduction} \\ & \quad \textit{cc:DerivativeWorks, cc:ShareAlike}\}; \\ \text{Duty} & \text{if } a \in \{\textit{cc:Notice, cc:Attribution}\}; \\ \text{Prohibition} & \text{if } a \in \{\textit{cc:CommercialUse}\}. \end{cases}$$

The following CC_CaLi licenses are not part of the official CC licenses. License $CC\ l_1$ is like CC BY-NC but without the obligation to give credit to the copyright holder/author of the resource. $CC\ l_2$ is like CC BY but with the prohibition of making multiple copies of the resource. License $CC\ l_3$ allows only exact copies of the original resource to be distributed. $CC\ l_4$ is like $CC\ l_3$ with the prohibition of commercial use.

$$CC\ l_1(a) = \begin{cases} \text{Permission} & \text{if } a \in \{\textit{cc:Distribution, cc:Reproduction,} \\ & \quad \textit{cc:DerivativeWorks, cc:ShareAlike,} \\ & \quad \textit{cc:Notice, cc:Attribution}\}; \\ \text{Prohibition} & \text{if } a \in \{\textit{cc:CommercialUse}\}. \end{cases}$$

$$CC\ l_2(a) = \begin{cases} \text{Permission} & \text{if } a \in \{\textit{cc:Distribution, cc:DerivativeWorks,} \\ & \quad \textit{cc:CommercialUse, cc:ShareAlike}\}; \\ \text{Duty} & \text{if } a \in \{\textit{cc:Notice, cc:Attribution}\}; \\ \text{Prohibition} & \text{if } a \in \{\textit{cc:Reproduction}\}. \end{cases}$$

$$CC\ l_3(a) = \begin{cases} \text{Permission} & \text{if } a \in \{\textit{cc:Distribution, cc:ShareAlike, cc:CommercialUse}\}; \\ \text{Duty} & \text{if } a \in \{\textit{cc:Notice, cc:Attribution, cc:Reproduction}\}; \\ \text{Prohibition} & \text{if } a \in \{\textit{cc:DerivativeWorks}\}. \end{cases}$$

$$CC\ l_4(a) = \begin{cases} \text{Permission} & \text{if } a \in \{\textit{cc:Distribution, cc:ShareAlike}\}; \\ \text{Duty} & \text{if } a \in \{\textit{cc:Notice, cc:Attribution, cc:Reproduction}\}; \\ \text{Prohibition} & \text{if } a \in \{\textit{cc:DerivativeWorks, cc:CommercialUse}\}. \end{cases}$$

In CC_CaLi, the minimum is the license where all actions are permitted (i.e., CC Zero) and the maximum is the license where all actions are prohibited.

Figure 4.4 shows two subgraphs of CC_CaLi with their compatibility relations. Figure 4.4a shows only the 7 official CC licenses and Figure 4.4b includes also

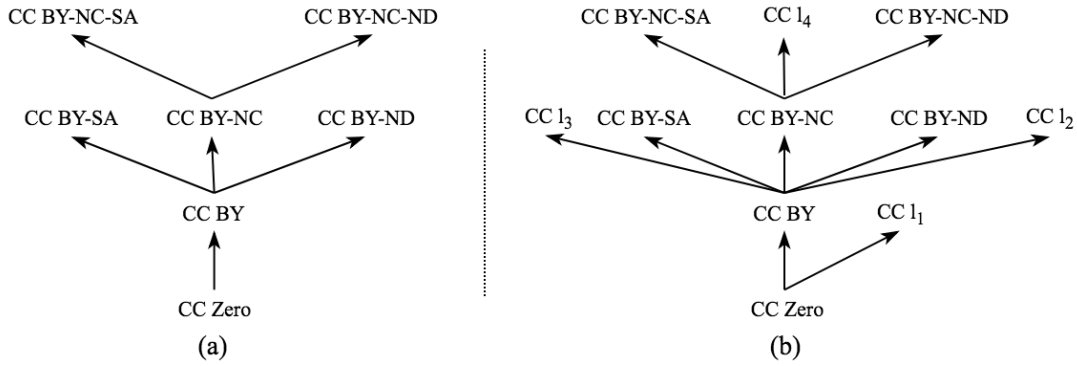

Figure 4.4: Compatibility subgraphs of CC_CaLi.

$CC\ l_1$ to $CC\ l_4$. These graphs can be generated using the CaLi implementation (cf Section 4.5). Thanks to $\omega_{\rightarrow_2}$, the restrictiveness relation between CC BY-SA and CC BY-NC-SA is not identified as a compatibility relation and thanks to $\omega_{\rightarrow_3}$, the restrictiveness relation between CC BY-ND and CC BY-NC-ND is not identified as a compatibility relation. We recall that a license that prohibits *cc:DerivativeWorks* is not compatible even with itself.

The compatibility relations of Figure 4.4a are conform to the ones obtained from the CC license compatibility chart[22]. This example shows the usability of CaLi with a real set of licenses.

4.5 Implementation of CaLi Orderings

The goal of this section is threefold, to present the algorithm we implemented to produce CaLi orderings, to evaluate this algorithm and, to illustrate the usability of CaLi through a prototype of a license-based search engine.

4.5.1 Insertion sort algorithm

The size growth of CaLi orderings is exponential, i.e., $|\mathcal{LS}|^{|\mathcal{A}|}$. Nevertheless, it is not necessary to explicitly build a CaLi ordering to use it. Sorting algorithms like insertion sort can be used to produce subgraphs of a CaLi ordering.

We implemented an algorithm that can sort any set of licenses. The goal is to be able to insert a license in a graph in linear time $O(n)$ without sorting again the graph. We use a heuristic, based on the restrictiveness of the new license and the restrictiveness of the licenses in the graph, to chose between two strategies,

[22] https://wiki.creativecommons.org/wiki/Wiki/cc_license_compatibility

1) to insert a license traversing the graph from the bottom or *2)* from the top of the graph. The goal is to minimize the number of comparison needed to find the position of the new license in the graph. To do this, our algorithm calculates the relative position of the new license (i.e., the level of the node) from the number of actions that it obliges and prohibits. The median depth (i.e., number of levels) of the existing graph is calculated from the median of the number of prohibited and obliged actions of existing licenses. Depending on these numbers, a strategy is chosen to find the place of the new license in the graph. Algorithm 1 choose a strategy to insert a new license in a CaLi ordering graph.

Algorithm 1: The choice of a strategy to insert a new license in a CaLi ordering graph.

```
1 Function insert(l_i, G):
      Data: l_i: License,
      G: Graph,
      level_median_G: Median of the number of levels of G
      level_{l_i}: Level of l_i
      Result: Returns G with l_i classified
2     if valid(l_i, C_L) then
          // l_i respects all C_L constraints
3         if (level_{l_i} < level_median_G) then
              // l_i will be classified in the bottom half of G
4             G = insertFromBottom(l_i, G)
5         else
              // l_i will be classified in the top half of G
6             G = insertFromTop(l_i, G)
7     return G
```

Algorithm 3 in Appendix A resursively insert a new license in a CaLi ordering graph starting from the bottom of the graph. Notice the usage of supremum S and infimum I to access respectively the top and the bottom of the graph. The supremum is the most restrictive license of the CaLi ordering and the infimum is the least restrictive license.

The insertion sort algorithm to add a new license in the Graph starting from the top (i.e., *insertFromTop*) can be obtained by inverting restrictiveness relation tests in Algorithm 3.

4.5.2 Experimental validation

We experimentally validate that our algorithm can sort any set of licenses in $\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$ comparisons of restrictiveness (approx. $n^2/2$), n being the number of licenses to sort, i.e., $O(n^2)$.

Results shown in Figure 4.5 demonstrate that our algorithm sorts a set of licenses with at most $n^2/2$ comparisons. We used 20 subsets of licenses of different sizes from the CC_CaLi ordering. Size of subsets was incremented by 100 up to 2187 licenses. Each subset was created and sorted 3 times randomly. The curve was produced with the average of the number of comparisons to sort each subset.

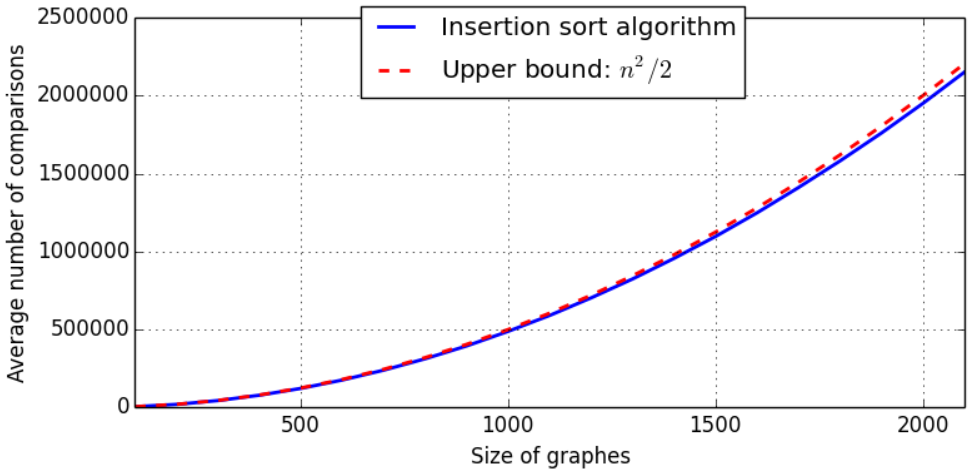

Figure 4.5: The performance of the CaLi insertion sort algorithm in number of comparisons with incremental size of subsets of licenses.

A comparison of restrictiveness takes on average 6 milliseconds[23], thus to insert a license in a 2000 licenses graph takes an average of 12 seconds.

The implementation in Python of our algorithm and details of our experiments are available on GitHub[24]. For interoperability purposes, constraints can be described in RDF using OWL or Shapes Constraint Language (SHACL)[25]. Moreover, to represent restrictiveness lattices of status in RDF format, we extended the ODRL ontology with the `cali:Status` class and the `cali:lessRestrictiveThan` property[26]. Figure 14 in Appendix A shows the restrictiveness lattice of status of Figure 4.2c in RDF format.

[23] With a 160xIntel(R) Xeon(R) CPU E7-8870 v4 2.10GHz 1,5 Tb RAM.
[24] https://github.com/benjimor/CaLi-Search-Engine
[25] https://www.w3.org/TR/shacl/
[26] http://cali.priloo.univ-nantes.fr/ontology

4.5.3 A search engine based on an ODRL CaLi ordering

Imagine a license-based search engine that can answer questions such as "*find all resources that can be reused under the CC BY-NC license*". The answer must contain resources licensed under licenses such as CC BY and CC BY-NC itself that are less or as restrictive as CC BY-NC and compatible with it.

There exist search engines that can find resources licensed under a particular license. However they can not find resources whose licenses are compatible or compliant with a particular license.

We illustrate the usability of CaLi by implementing a prototype of a search engine[27] that allows to find linked data[28] and source code repositories[29] based on the compatibility or the compliance of their licenses. The goal is to be able find resources whose licenses are compatible or compliant with a particular license. Our prototype can answer questions such as: *"find licensed resources that can be reused under a given license"* or *"find licensed resources that can reuse a resource that has a particular license"*.

In our search engine, resources (i.e., linked data and source code) are associated to licenses. Licenses are described in RDF with the ODRL vocabulary[30]. In addition to indexing licenses, the titles, descriptions and IRI of each licensed resources are also indexed to enable full-text search. We use the ODRL vocabulary because it is the most complete vocabulary for licenses and it is well accepted by the community. Our search engine is based on the $ODRL\_CaLi$ ordering. It is a CaLi ordering $\langle \mathcal{A}, \mathcal{LS}, C_{\mathcal{L}}, C_{\rightarrow} \rangle$ such that:

- $\mathcal{A}$ is the set of 72 actions considered by ODRL[31];
- $\mathcal{LS}$ is the restrictiveness lattice of status depicted in 4.2c; and
- $C_{\mathcal{L}}, C_{\rightarrow}$ are the sets of constraints, inspired from the ODRL information model.

It use license constraints and compatibility constraints of CC_CaLi ordering but with a new license constraint that invalidates a license when the semantics of a permitted or obliged action is included in a prohibited action (e.g. if *CommercialUse* is permitted then *use* should not be prohibited because *CommercialUse* implies

[27] http://cali.priloo.univ-nantes.fr/
[28] http://cali.priloo.univ-nantes.fr/ld/
[29] http://cali.priloo.univ-nantes.fr/rep/
[30] https://www.w3.org/TR/odrl-model/
[31] https://www.w3.org/TR/odrl-vocab/#actionConcepts

use). This constraint $\omega_{\mathcal{L}_4}$ is defined below.

$$\omega_{\mathcal{L}_3}(l_i) = \begin{cases} False & \text{if } a_i \text{ odrl:includedIn } a_j \\ & \text{AND } (l_i(a_i) = \text{Permitted OR } l_i(a_i) = \text{Obliged}) \\ & \text{AND } l_i(a_j) = \text{Prohibited;} \\ True & \text{otherwise.} \end{cases}$$

Other constraints could be defined to be closer to the ODRL information model but for the purposes of this demonstration these constraints are enough.

The size of this ordering is 4^{72} and it is not possible to build it. This search engine illustrates the usability of $ODRL\_CaLi$ through two subgraphs. The first one is a subgraph with the most used licenses in DataHub[32] and Opendatasoft. Licenses in this graph are linked to some RDF datasets such that it is possible to find datasets whose licenses are compatible or compliant with a particular license. The second one is a subgraph with the most used licenses in GitHub. Here, licenses are linked to some GitHub repositories and it is possible to find repositories whose licenses are compatible or compliant with a particular license.

Both compatibility graphs of licences are visually available. Figure 4.6 shows the compatibility graph of the CaLi ordering for some licensed RDF datasets. Blue nodes are licenses, grey arrows are compatibility relations among licenses and orange nodes are RDF datasets associated to licenses. Licenses that have the same actions in the same status are represented in the same node. In the graph, licenses that are compatible with a particular license l_i are below l_i and licenses that are compliant with l_i are above l_i. We recall that the ordering relations of compatibility and compliance that we define are reflexive, transitive and asymmetric.

Figure 4.7 shows the search bar of our search engine. It enables full-text and license-compliant searches over each graph, for RDF datasets[33] or repositories[34]. For example, users can search for *datasets about 'bikes' whose licenses are compatible with the CC BY-NC license* (i.e. datasets about 'bikes' that can be reused under the CC BY-NC license). The result contains all RDF datasets indexed in the search engine where title or description contains the word 'bikes' and whose license is compatible with CC BY-NC (e.g. CC BY, MIT, CC-Ze, etc.).

Both compatibility graphs of licences are available through a documented API[35]. Finally, these graphs are also accessible through a TPF server[36,37] or can be exported in RDF (turtle, xml, n3 and json-ld).

[32] https://old.datahub.io/
[33] http://cali.priloo.univ-nantes.fr/ld/
[34] http://cali.priloo.univ-nantes.fr/rep/
[35] http://cali.priloo.univ-nantes.fr/api
[36] http://cali.priloo.univ-nantes.fr/api/ld/tpf
[37] http://cali.priloo.univ-nantes.fr/api/rep/tpf

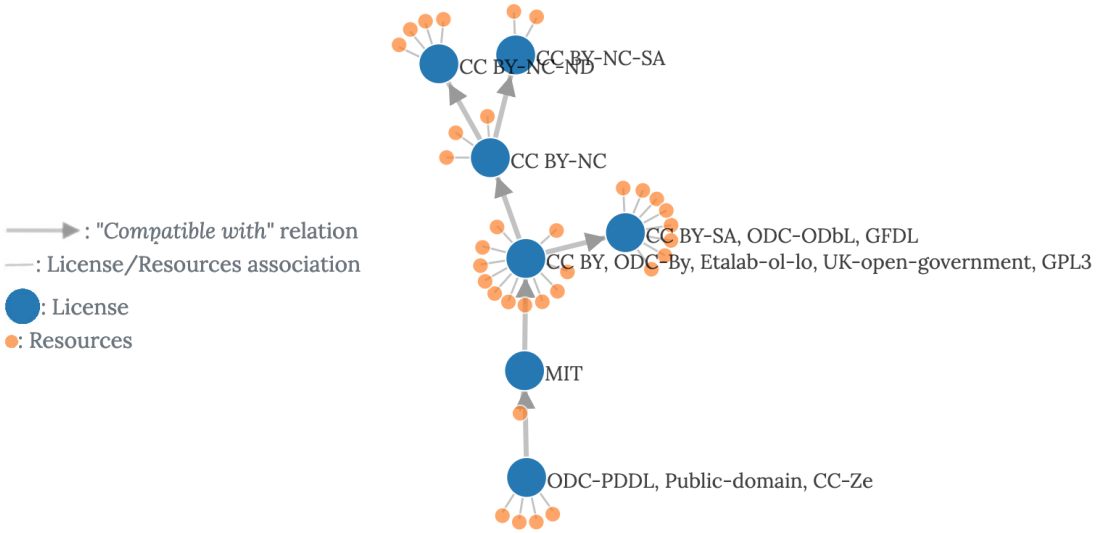

Figure 4.6: A graph of the $ODRL\_CaLi$ ordering for some licensed RDF datasets.

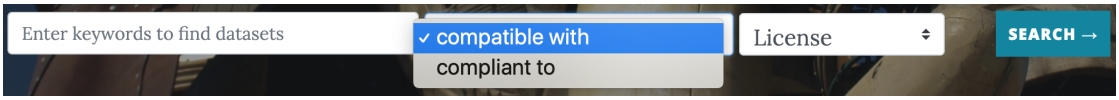

Figure 4.7: The search bar of the license-based search engine.

A possible extension of our search engine is to allow the collaborative addition of licenses and licensed resources. That is, to allow users to add new licenses and resources to increase the size and therefore the interest of these two graphs.

We also proposed a framework that implements the $CaLi$ model on Github[38]. It allows creating any CaLi ordering by defining a set of actions, a restrictiveness lattice of status, license constraints, compatibility constraints, and a set of licenses.

4.6 Conclusion

We proposed a lattice-based model to define compatibility and compliance relations among licenses. Our approach is based on a restrictiveness relation that is refined with constraints to take into account the semantics of actions existing in licenses. We have shown the feasibility of our approach through two CaLi orderings, one using the Creative Commons vocabulary and the second using ODRL. We experimented the production of CaLi orderings with the implementation of an insertion sort algorithm whose cost is $n^2/2$. We implemented a prototype of a license-based

[38] https://github.com/benjimor/CaLi

search engine that highlights the feasibility and usefulness of our approach. Our compatibility model does not intent to provide a legal advice but it allows to exclude those licenses that would contravene a particular license.

The model we propose uses restrictiveness as the basis to define compatibility and compliance among licenses. This strategy works most of the time, as we have shown in this work, but it has certain limitations. In particular, CaLi is not designed to define the compatibility of two licences if it is not coherent with their restrictiveness relation. As an example consider two versions of MPL licenses. Version 2.0 relaxes some obligations compared to version 1.1. Thus, MPL-2.0 is less restrictive than MPL-1.1. With CaLi constraints, it can only be possible to say that MPL-2.0 is compatible with MPL-1.1. But in the legal texts it is said the opposite, i.e., MPL-1.1 is compatible with MPL-2.0.

Thereby, particularities in the usage of compatibility of licenses, the granularity of the semantisation of licenses and the understanding of some actions (like *Share-Alike*) are the main reasons of the difference between CaLi orderings and other classifications. This is the case, for instance, of our compatibility graph devoted to licenses of GitHub and the graph presented in [34].

A perspective of this work is to take into account other aspects of licenses related to usage contexts like jurisdiction, dates of reuse, etc. Another perspective is to analyse how two compatibility orderings can be compared. That is, given two CaLi orderings, if there is an alignment between their vocabularies and their restrictiveness lattices of status are homomorphic then find a function to pass from a CaLi ordering to another.

Chapter 5

Ensuring License Compliance in Federated Query Processing

Contents

In the previous chapter, we showed the usefulness of the CaLi model for the creation of license-aware applications. We now focus on using our model to create a license-aware federated query engine.

When several licensed data sources participate in the evaluation of a federated query, the query result must be protected by a license that is compliant with each license of the involved datasets. However, such a license does not always exist, and this leads to a query result that cannot be reused. One solution is to discard datasets of conflicting licenses during the federated query processing. But, a query with an empty result set can be obtained. In this chapter, we present FLiQue, a license-aware query processing strategy for federated query engines. It answers our

research question: *given a SPARQL query and a federation of licensed datasets, how to guarantee a relevant and non-empty query result whose license is compliant with each license of involved datasets?*

This chapter is based on [43] and is organized as follows. Section 5.1 illustrates and motivates our research question. Section 5.2 discuses related works, Section 5.3 introduces FLiQue, Section 5.4 shows experimental results, and Section 5.5 concludes.

5.1 Introduction and Motivation

A *federated SPARQL query* can retrieve information from several RDF data sources distributed across the Linked Data. When two or more licensed data sources participate in the evaluation of a federated query, the query result must be protected by a license that is compliant with each license of involved datasets.

Unfortunately, it is not always possible to find a license compliant with each license of datasets involved in a federated query [48]. If such a license does not exist, the query result cannot be licensed and, thus, should not be reused nor published. We consider that a query whose result set cannot be licensed should not be executed. Notice that having the rights to query several datasets individually does not mean having the rights to execute a federated query involving these datasets.

Consider datasets of LargeRDFBench [60], a benchmark for federated query processing. Figure 5.1 shows the compatibility graph of Creative Commons licenses[1] that protect LargeRDFBench datasets. License CC BY is compatible with itself, with CC BY-SA, CC BY-NC, and CC BY-NC-SA. Thus, datasets protected by CC BY can be queried along with other datasets protected by these licenses. But the whole set of datasets of Figure 5.1 cannot be queried together because there is no license compliant with the fourth licenses. For instance, there is no license with which CC BY-SA and CC BY-NC-SA are both compatible.

In May 2020, Linked Data catalogs like Datahub[2] or LODAtlas[3] show that the problem of license compliance when combining datasets can be frequent. For instance, Datahub has 382 datasets with the license CC BY-SA and 292 with license CC BY-NC. LODAtlas similarly has 118 datasets with the license CC-BY SA and 90 with license CC NC.

One solution to the incompatibility of licenses is to negotiate with data providers to change a conflicting license, e.g., to ask DBpedia to change their license to CC BY or CC BY-NC. But negotiation takes time and is not always possible. A second

[1] This compatibility graph conforms to the license compatibility chart shown in https://wiki.creativecommons.org/wiki/Wiki/cc_license_compatibility.

[2] https://old.datahub.io

[3] http://lodatlas.lri.fr

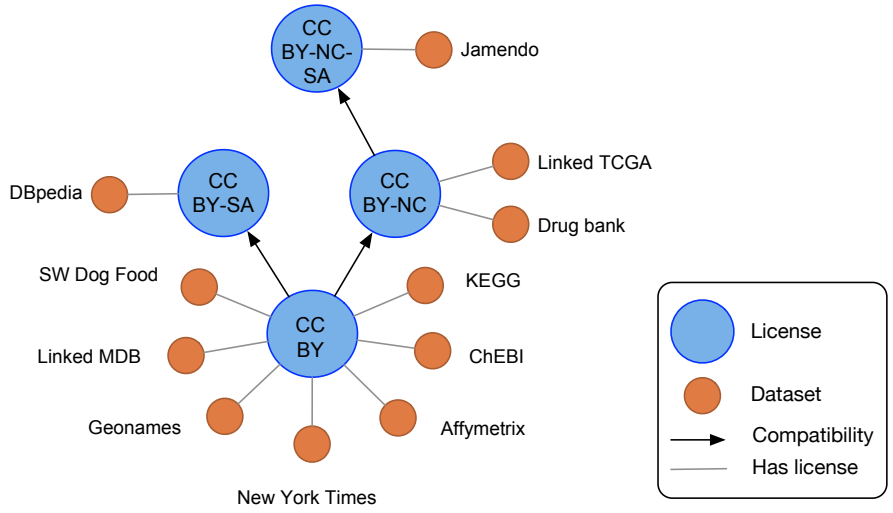

Figure 5.1: The compatibility graph of licenses for datasets of LargeRDFBench.

solution is to discard datasets that are protected by conflicting licenses. However, this solution can lead to a query with an empty result set. To face this problem, we use query relaxation techniques. That is, we use *relaxation rules* to relax triple patterns to match triples of other datasets.

Consider the query Q of Listing 8 showing a SPARQL query annotated with the datasets over which each triple pattern can be evaluated. It asks for students enrolled in a course held at the University of Nantes taught by `ex:Ben`. $D2$ and $D3$ should not be queried together because their licenses are respectively CC BY-SA and CC BY-NC, and there is no license compliant with both as shown in Figure 5.1. Thus, the result set of the query cannot be licensed. Creating a sub-federation without $D2$ makes the result set licensable with CC BY-NC or with another license compliant with CC BY-NC (e.g., CC BY-NC-SA). The problem is that the query gives no result because there is no more dataset to evaluate $tp4$. The case is similar if $D3$ is discarded because there is no more dataset to evaluate $tp1$ and $tp2$.

As there is no sub-federation able to produce a licensable and non-empty result set for Q, we use query relaxation techniques. In such relaxation, for instance, instead of asking for students in $tp1$, the query could ask for persons, or instead of asking for courses taught by `ex:Ben` in $tp4$, the query could ask for courses taught by anybody.

```
1 SELECT ?student  WHERE {
2   ?student  rdf:type  ex:Student .              #tp1@{D3} CC BY-NC
3   ?student  ex:enrolledIn  ?course .            #tp2@{D3} CC BY-NC
4   ?course   ex:heldAt  ex:UniversityOfNantes .  #tp3@{D1} CC BY
5   ex:Ben   ex:teaches  ?course .               #tp4@{D2} CC BY-SA
6 }
```

Listing 8: A SPARQL query Q annotated with the sources for each triple pattern and dataset licenses.

The number of possible relaxed queries can be huge. To find the most relevant relaxed queries efficiently, we use approaches that compute relaxed queries from the most to the least *similar* to the original query [15, 16, 31, 32]. But the most similar queries may produce no results. In a distributed environment, verifying each relaxed query is not feasible. So the challenge is to find the most similar relaxed queries that return a non-empty result while limiting communication costs.

Our research question is, *given a SPARQL query and a federation of licensed datasets, how to guarantee a relevant and non-empty query result whose license is compliant with each license of involved datasets?* The challenge is to limit the communication cost when the relaxation process is necessary.

We propose **FLiQue**[4] [43], a Federated License-aware Query processing strategy. FLiQue is designed to detect and prevent license conflicts and gives informed feedback with licenses able to protect a result set of a federated query. If necessary, it applies distributed query relaxation to propose a set of most similar relaxed queries whose result set can be licensed. Our contributions are:

- a license-aware query processing strategy,
- an implementation of a license-aware federated query engine, and
- an experimental evaluation of our approach.

5.2 Related Work

To our knowledge, there is no federated query engine that ensures license compliance with all licenses involved in query execution.

Many works focus on access control over linked data using policy-based [7, 35, 36, 56], view-based [18], or query-rewriting [52] approaches. In these works, datasets are protected by *access control rules* that prevent non-authorized users

[4] In French, FLiQue is a homophone of *flic*, which means *cop*.

from querying data of each dataset. These approaches do not resolve our problem statement because having the right to query datasets individually does not mean that it should be possible to execute a federated query involving these datasets.

Compatibility graph of licenses. To know if a result set can be licensed, we need to know the license(s) with whom all licenses of datasets involved in a federated query are compatible. A compatibility graph of licenses contains a set of licenses partially ordered by compatibility. It can be defined by hand using, for instance, the license compatibility chart of Creative Commons. But licenses used in the Linked Data are not limited to Creative Commons licenses. Works like [70] address the problem of license compatibility and license combination. If licenses are compatible, a new license compliant with combined ones is generated. This approach allows defining the compatibility graph of licenses progressively. However, it does not allow us to know all the compliant licenses that can be used to protect a result set. CaLi [46, 48], is a lattice-based model for license orderings. It automatically positions a license over a set of licenses in terms of compatibility and compliance. It uses restrictiveness relations and constraints among licenses to define compatibility. CaLi defines all the licenses that can be expressed with a set of actions. For instance, the CaLi ordering for the set of 7 actions used by Creative Commons has 972 licenses. CaLi can provide all the licenses than can protect a result set ordered by restrictiveness. It can also identify which licenses are in conflict. In this work, we use CaLi to verify license compliance.

When the result set of a federated query cannot be licensed, we propose to define sub-federations that avoid license conflicts. If there is no sub-federation able to produce a licensable and non-empty result set for the user query, we propose alternative queries through query relaxation.

Query relaxation. Query relaxation techniques are used to provide an alternative for queries producing no result. [32] proposes query relaxation using RDFS entailment and RDFS ontologies. The idea consists of relaxation rules that use information from the ontology; these include relaxing a class to its super-class, relaxing a property to its super-property, etc. Other relaxations include dropping triple patterns, replacing constants with variables, and suppressing join dependencies. All possible relaxed queries are organized in a lattice called *relaxation graph*. The size of the relaxation graph grows combinatorially with the number of relaxation rules, the richness of the ontology, and the relaxation possibilities of each triple pattern in the original query.

[16, 31] focus on obtaining a certain number of alternative results (top-k) by relaxing a query that produces no results. Their challenge is to execute as less as possible relaxed queries to obtain the top k results. Relaxed queries are executed in

a similarity-based rank order to avoid executing all relaxed queries in the relaxation graph. The *information content* is used to measure the *similarity* between a relaxed query and the original query. That is, statistical information about the concerned dataset, like the number of entities per class and the number of triples per property. But, the number of failing relaxed queries executed before obtaining the top-k results can be considerable. Thus, it is necessary to identify unnecessary relaxations that do not generate new answers. Relaxed queries containing unnecessary relaxation should not be executed.

[31] proposes OBFS (Optimized Best First Search algorithm) to identify unnecessary relaxations in a similarity-based relaxation graph. It is based on the selectivity of relaxations using the number of entities per class or the number of triples per property. If the selectivity is the same before and after the relaxation, the relaxation is considered unnecessary. That is, if the number of entities of a class is equal to the number of entities of its super-class, then the class relaxation does not generate new answers. The same idea is used for property relaxation.

[16] proposes OMBS (Optimized Minimal-failing-sub-queries-Based Search algorithm) as an improvement to OBFS. The contribution of OMBS is to identify the minimal sets of triple patterns in failing queries that fail to return answers. These failing sets of triple patterns are called Minimal Failing Sub-queries (MFS). MFS existing in a query must be relaxed, otherwise, the query fails in producing results. Relaxed queries where the MFS are not relaxed are considered unnecessary. OMBS defines optimal similarity-based relaxation graphs where relaxed queries producing no results (based on MFS), or not new results (based on selectivity) are not executed.

We use OMBS to find relaxed queries with non-empty result sets. To limit the communication overhead, during the *distributed query relaxation* process, we use *data summaries*.

Data summaries. Some federated query engines, use statistics to reduce the number of requests sent to data sources during query processing, in particular in the source selection and query optimization steps. For instance, SPLENDID [21], uses VOID descriptions of datasets to speed-up query processing. VOID descriptions contain basic statistical information about datasets, such as the number of entities per class and the number of triples per property. HIBISCuS [61], a join-aware source selection algorithm, discards dataset that are relevant for a triple pattern, but that do not contribute to a query result. It proposes data summaries, called *dataset capabilities*, containing all the distinct properties with all the URI authorities of their subjects and objects. CostFed [62], an index-assisted federated engine for SPARQL endpoints, extends the join-aware source selection of HIBISCuS by considering URI prefixes instead of URI authorities. The dataset capabilities calculated by CostFed

are more precise, its source selection chooses, in general, more pertinently the data sources for each query.

We use the licenses of the data sources identified by a source selection process to know if the result set of a federated query (or a relaxed federated query) would be licensable. We use the join-aware source selection of CostFed.

5.3 A Federated License-Aware Query Processing Strategy

We propose FLiQue, a federated license-aware query processing strategy to detect and prevent license conflicts. Our approach gives informed feedback with licenses that can protect a result set of a federated query. When the result set of a federated query cannot be licensed, we define sub-federations that avoid license conflicts. If there is no sub-federation able to produce a licensable and non-empty result set, we propose alternative relaxed federated queries.

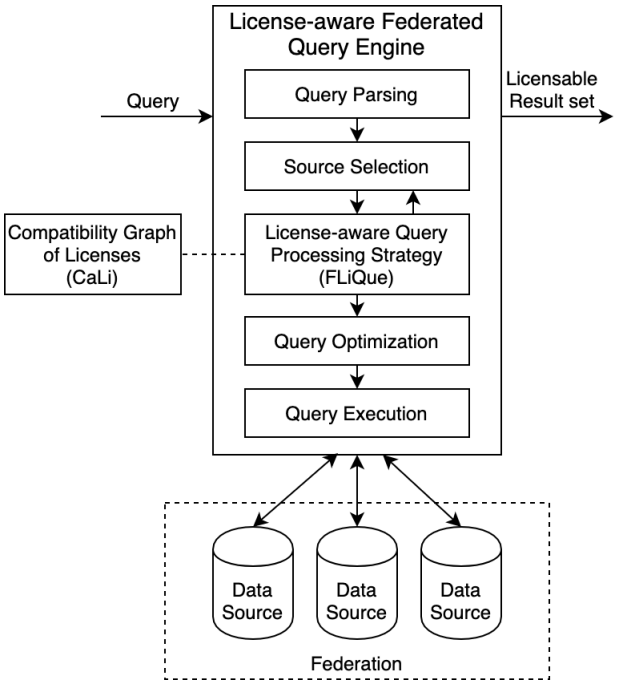

Figure 5.2: A federated license-aware query engine using FLiQue.

Figure 5.2 shows a license-aware federated query engine that uses FLiQue. FLiQue is located between the query parsing and the query optimization functions of a federated query engine. A join-aware source selection [62], selects the *capable datasets* for each triple pattern of a query. The Algorithm 2 shows the global

approach of FLiQue. Using a compatibility graph of licenses, we search for the set of licenses compliant with each license of the chosen pertinent endpoints (line 3). If it is not empty, the original query can be executed and is returned with the licenses that can protect its result set (line 18). If no compliant license exists, we identify the license conflicts and define sub-federations that avoid these conflicts (line 5). If one sub-federation can produce a licensable and non-empty result, the query can be executed (line 8). Otherwise, for each sub-federation (line 12), we propose to the query issuer a relaxed query whose result set is licensable and non-empty (line 13). Notice that when the result set of a relaxed query is computed during query relaxation (line 13), the query is only executed if the source selection finds at least one dataset capable of executing the query. The result of the algorithm is a set of ordered triplets representing candidate queries. A triplet $(Q, E, \mathcal{L})$ is a query Q that returns a non-empty result set when executed on a set of pertinent endpoints E that can be protected by a set of licenses $\mathcal{L}$.

Several sub-federations may produce a licensable and non-empty result. In that case, we choose the sub-federation that produces a result set licensable by the least restrictive license.[5]

Consider the query Q of Listing 8, and the federation containing datasets $D1$, $D2$, and $D3$ shown in Tables 5.1, 5.2, and 5.3. As there is no license compliant with the licenses of $D2$ and $D3$, the result set of Q cannot be licensed. Thus, our strategy defines the sub-federations $F1$=$\{D1, D2\}$ and $F2$=$\{D1, D3\}$ that avoid license conflicts. The source selection for Q over $F1$ and $F2$ fails to obtain a data source for each triple pattern. This launches a process of federated query relaxation for each sub-federation. To avoid verifying that the result set of a large number of relaxed queries is not empty, our strategy defines, by sub-federation, an optimal similarity-based relaxation graph. When we find one licensable and non-empty relaxed query, we stop the relaxation process. OMBS guarantees that a candidate query is the most similar to Q for a sub-federation.

Figure 5.3 shows Q and three relaxed queries. Figure 5.4 shows the ontology used in our example. $\preceq_{sc}$ is rdfs:subClassOf, $\preceq_{sp}$ is rdfs:subPropertyOf, $\hookleftarrow_{d}$ is rdfs:domain, and $\hookrightarrow_{r}$ is rdfs:range.

As we explain next, $Q'4b$ and $Q'4d$ are the most similar licensable, and non-empty relaxed query for $F1$ and $F2$ respectively.

In the next, Section 5.3.1 shows the relaxation techniques we use. Section 5.3.2 presents the information content measures that allow us to rank relaxed queries. Section 5.3.3 shows the data summaries that allow limiting communication costs. Finally, Section 5.3.4 explains how we define the similarity-based relaxation graph.

[5]Other options could be defined, for example, based on the cardinality estimations of result sets or based on the number of involved data sources.

Algorithm 2: The global approach of FLiQue.

```
1  Function FLiQue(Q, F, E, S, C):
       Data: Q: RDF Query,
       F: Federation of endpoints,
       E ⊆ F: Set of pertinent endpoints for Q,
       S: Dataset summaries,
       C: Compatibility graph of licenses.
       Precondition : resultSet(Q, E) ≠ ∅.
       Result: A set of tuples ⟨Q, E, ℒ⟩ representing candidate queries with
               corresponding pertinent endpoints and compliant licenses.
2      Candidates = ∅
3      ℒ = compliantLicenses(E,C)
4      if ℒ == ∅ then
5          Fs = {F' ⊆ F | compliantLicenses(F', C) ≠ ∅}
6          for F' ∈ Fs do
7              E' = sourceSelection(Q,F',S)
8              if E' can evaluate Q then
                   // The original query can be executed on E'.
9                  ℒ = compliantLicenses(E',C)
10                 Candidates = Candidates ∪ ⟨Q, E', ℒ⟩
11         if Candidates == ∅ then
               // Compute most similar queries.
12             for F' ∈ Fs do
13                 Q' = queryRelaxation(Q, F', S)
14                 E' = sourceSelection(Q',F',S)
                   // The relaxed query Q' can be executed on E'.
15                 ℒ = compliantLicenses(E',C)
16                 Candidates = Candidates ∪ ⟨Q', E', ℒ⟩
17     else
           // The original query can be executed on E.
18         Candidates = {⟨Q, E, ℒ⟩}
19     return Candidates
```

5.3.1 Query relaxation techniques

In this work, we use query relaxation using RDFS entailment and RDFS ontologies. We consider that ontologies of datasets are accessible and that SPARQL endpoints expose saturated RDF data (or support on-the-fly entailment) according to the

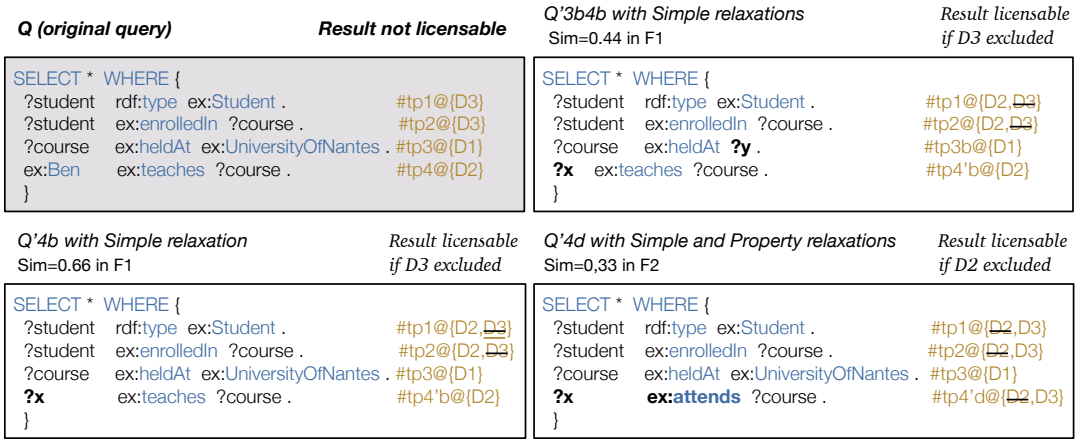

Figure 5.3: Example of SPARQL query Q and some relaxed queries Q'.

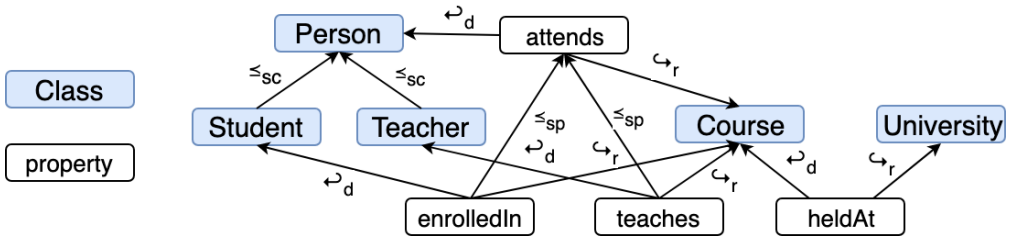

Figure 5.4: Ontology representing courses in a university.

RDFS entailment rules rdfs7 and rdfs9. We use the relaxations of triple patterns and queries as proposed in [31].

Triple Pattern Relaxation. Given two triple patterns tp and tp', tp' is a relaxed triple pattern obtained from tp, denoted $tp \prec tp'$, by applying one or more triple pattern relaxations. We use the three following triple pattern relaxations:

- *Simple relaxation* replaces a constant of a triple pattern by a variable. For example, $tp_4 = \langle$`ex:Ben`, `ex:teaches`, `?course`$\rangle$, can be relaxed to $tp'_4 = \langle$`?x`, `ex:teaches`, `?course`$\rangle$, thus $tp_4 \prec_s tp'_4$.
- *Type relaxation* replaces a class C of a triple pattern with its super-class C'. It is based on the rdfs9 rule (rdfs:subClassOf). For example, $tp_1 = \langle$`?student`, `rdf:type`, `ex:Student`$\rangle$, can be relaxed to $tp'_1 = \langle$`?student`, `rdf:type`, `ex:Person`$\rangle$, thus $tp_1 \prec_{sc} tp'_1$.
- *Property relaxation* replaces a property P of a triple pattern with its super-property P'. It is based on the rdfs7 rule (rdfs:subPropertyOf). For ex-

Subject	Predicate	Object
ex:UniversityOfNantes	rdf:type	ex:University
ex:SemanticWeb	rdf:type	ex:Course
ex:SemanticWeb	ex:heldAt	ex:UniversityOfNantes
ex:Databases	rdf:type	ex:Course
ex:Databases	ex:heldAt	ex:UniversityOfNantes

Table 5.1: Dataset D1 containing courses. D1 has licence CC BY.

Subject	Predicate	Object
ex:Ben	rdf:type	ex:Teacher
ex:Ben	rdf:type	ex:Person
ex:Ben	ex:attends	ex:SemanticWeb
ex:Ben	ex:teaches	ex:SemanticWeb
ex:Mary	rdf:type	ex:Teacher
ex:Mary	rdf:type	ex:Person
ex:Mary	ex:attends	ex:Databases
ex:Mary	ex:teaches	ex:Databases
my:William	rdf:type	ex:Student
my:William	rdf:type	ex:Person
my:William	ex:attends	ex:Databases
my:William	ex:enrolledIn	ex:Databases

Table 5.2: Dataset D2 containing teachers and students. D2 has licence CC BY-SA.

Subject	Predicate	Object
ex:Elsa	rdf:type	ex:Student
ex:Elsa	rdf:type	ex:Person
ex:Elsa	ex:attends	ex:SemanticWeb
ex:Elsa	ex:enrolledIn	ex:SemanticWeb

Table 5.3: Dataset D3 containing students. D3 has licence CC BY-NC.

ample, $tp_1 = \langle$?student, ex:enrolledIn, ?course$\rangle$, can be relaxed to $tp'_1 = \langle$?student, ex:attends, ?course$\rangle$, thus $tp_1 \prec_{sp} tp'_1$.

The set of all possible relaxed triple patterns of tp can be represented as a lattice called a *relaxation lattice of a triple pattern*. Figure 5.5 shows this lattice for triple pattern $tp4$ of Q. $tp4'b$, $tp4'c$ and $tp4'e$ show simple relaxations. $tp4'a$ shows a property relaxation. This lattice has three levels of relaxation.

Query Relaxation. Given two queries Q and Q', Q' is a relaxed query obtained from Q, denoted $Q \prec Q'$, by applying one or more triple pattern relaxations to triple patterns of Q. $\prec$ is a partial order over the set of all possible relaxed queries of Q. This order can be represented as a lattice, called a *relaxation lattice of a query* (or relaxation graph) that is the product of the relaxation lattices of all triple patterns of the query. Figure 5.3 shows the query Q and three relaxed queries of its relaxation graph where, $Q \prec Q'4b \prec Q'4d$ and $Q \prec Q'4b \prec Q'3b4b$.

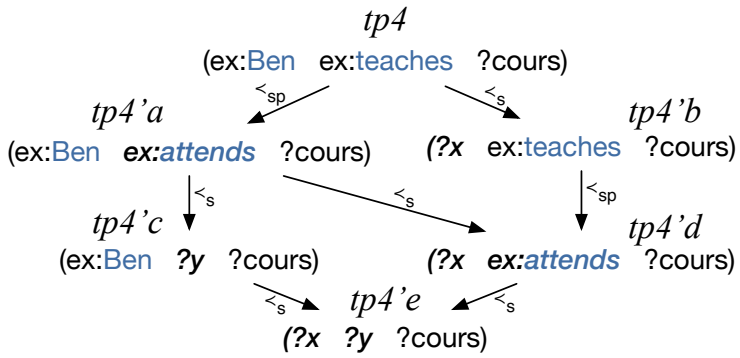

Figure 5.5: Relaxation lattice of triple pattern $tp4$ of query Q.

5.3.2 Information content measures

Analyzing all relaxed queries is time-consuming and unnecessary. We use *information content measures* to compute the similarity of relaxed queries to the original query. To avoid the analysis of an important number relaxed queries, our approach generates and executes relaxed queries from the most to the least similar. This execution allows to verify that the result set is not empty. It is stopped when the first result is returned. We use the *similarity mesures* proposed in [31], and explained in the following.

Similarity between terms. FLiQue uses three similarity measures for terms in a triple pattern. They correspond to the three relaxations described in Section 5.3.1.

- *Similarity between classes* is $Sim(C, C') = \frac{IC(C')}{IC(C)}$ where $IC(C)$ is the information content of C: $-logPr(C)$, where $Pr(C) = \frac{|Instances(C)|}{|Instances|}$ is the probability of finding an instance of class C in the RDF dataset. For example, if the subject or object of a triple pattern is a class c_1 and is relaxed to its super class c_2 using type relaxation, the similarity between c_1 and c_2 is $Sim(c_1, c_2)$.

- *Similarity between properties* is $Sim(P, P') = \frac{IC(P')}{IC(P)}$ where $IC(P)$ is the information content of P: $-logPr(P)$, where $Pr(P) = \frac{|Triples(P)|}{|Triples|}$ is the probability of finding a property of P in triples of the RDF dataset. For example, if the predicate of a triple pattern is a property p_1 and is relaxed to its super property p_2 using property relaxation, the similarity between p_1 and p_2 is $Sim(p_1, p_2)$.

- *Similarity between constants and variables* is $Sim(T_{const}, T_{var}) = 0$. For example, if the object of a triple pattern t_{const} is a class and is relaxed to a variable t_{var} using simple relaxation, the similarity between t_{const} and t_{var} is 0.

Similarity between triple patterns. Given two triple patterns tp and tp', such that $tp \prec tp'$, the similarity of the triple pattern tp' to the original triple pattern tp, denoted $Sim(tp, tp')$, is the average of the similarities between the terms of the triple patterns:

$$Sim(tp, tp') = \frac{1}{3}.Sim(s, s') + \frac{1}{3}.Sim(p, p') + \frac{1}{3}.Sim(o, o')$$

where s, p, o, s', p' and o' are respectively the subject, predicate and object of the triple pattern tp and the relaxed triple pattern tp'. If tp' and tp'' are two relaxations obtained from tp and $tp' \prec tp''$ then $Sim(tp, tp') \geq Sim(tp, tp'')$.

Similarity between queries. Given two queries Q and Q', such that $Q \prec Q'$, the similarity of the original query Q' to the original query Q, denoted $Sim(Q, Q')$, is the product of the similarity between triple patterns of the query:

$$Sim(Q, Q') = \prod_{i=1}^{n} w_i.Sim(tp_i, tp'_i)$$

Where tp_i is a triple pattern of Q, tp'_i a triple pattern of Q', $tp_i \preceq tp'_i$ and, $w_i \in [0, 1]$ is the weight of triple patterns tp_i. Weight can be specified by the user to take into account the importance of a triple pattern tp_i in query Q. Thus $Sim(Q, Q') \in [0, 1]$ is a function that defines a total order among relaxed queries.

This similarity function is monotone, i.e., given two relaxed queries $Q'(tp'_1, ..., tp'_n)$ and $Q''(tp''_1, ..., tp''_n)$ of the user query Q, if $Q' \prec Q''$ then $Sim(Q, Q') \geq Sim(Q, Q'')$.

Considering the query Q and datasets D1 and D2, $Sim(Q, Q'4b) = 0.66$ is greater than $Sim(Q, Q'3b4b) = 0.44$. This verifies the ordering of these relaxed queries, $Q'4b \prec Q'3b4b$, where $Q'4b$ is analyzed first.

5.3.3 Data summaries

A data summary is a compact structure that represents an RDF dataset. Using dataset statistics and dataset capabilities as in [62], allow us to limit communication cost in the similarity calculation and the source selection process.

Property	Number of triples	
	F1 = {D1, D2}	F2 = {D1, D3}
ex:enrolledIn	1	1
ex:teaches	2	0
ex:heldAt	2	2
ex:attends	3	1
rdf:type	9	5
Total	17	9

Table 5.4: Statistics of properties in federations F1 and F2.

Class	Number of entities	
	F1 = {D1, D2}	F2 = {D1, D3}
ex:University	1	1
ex:Student	1	1
ex:Teacher	2	0
ex:Course	2	2
ex:Person	3	1
Total	6	4

Table 5.5: Statistics of classes in federations F1 and F2.

Property	F1 = {D1, D2}		F2 = {D1, D3}	
	subjPrefixes	objPrefixes	subjPrefixes	objPrefixes
rdf:type	ex: my:William	ex:Person ex:Student ex:Teacher	ex:	ex:University ex:Course ex:Student ex:Person
ex:heldAt	ex:	ex:UniversityOfNantes	ex:	ex:UniversityOfNantes
ex:attends	ex: my:William	ex:	ex:Elsa	ex:SemanticWeb
ex:teaches	ex:	ex:		
ex:enrolledIn	my:William	ex:Database	ex:Elsa	ex:SemanticWeb

Table 5.6: Capabilities of federations F1 and F2.

Dataset statistics. Statistics contain VOID descriptions [1], such as the number of entities per class and the number of triples per property. Having dataset statistics is twofold; they allow computing similarities, and they help during the source selection process. Tables 5.4 and 5.5 show respectively statistics about properties and classes for F1 and F2. In Table 5.4, the property `ex:teaches` has no triples in F2. So there is no data source for $tp4$. That allows us to identify $Q'4b$ as a failing query in F2.

Dataset capabilities. Capabilities contain the properties of a dataset with the common prefixes of their subjects and objects. The `rdf:type` property, is treated differently. The prefixes of its objects are replaced by all the classes used in the dataset. Dataset capabilities are used in the source selection process. The goal is to discard datasets that individually return results for a triple pattern, but that fail to perform joins with other triple patterns of the query. For multiple triple patterns of a query sharing a variable, the dataset capabilities allow identifying data sources that do not share the same URIs prefixes and thus whose joins yield empty results. This information allows performing an optimal source selection by limiting the communication with the data sources. FLiQue performs source selection for the original query but also during query relaxation for relaxed queries.

Table 5.6 shows the capabilities of F1 and F2. Consider $tp4'a$ of $Q'4a$ that asks for ⟨`ex:Ben ex:attends ?cours`⟩. Table 5.4 shows one triple for `ex:attends` but capabilities of this property in F2 show only one subject prefix that is `ex:Elsa`, not `ex:Ben`. Consider the join $tp3$. $tp4'c$ of $Q'4c$:

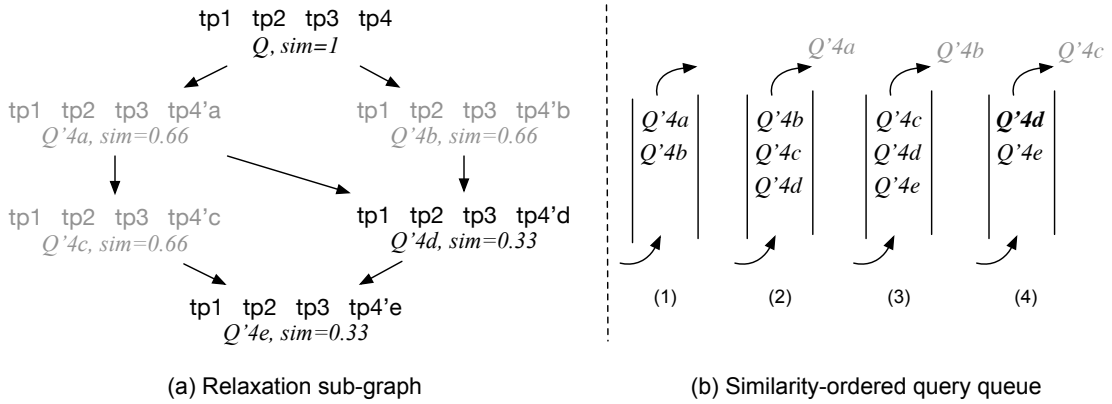

Figure 5.6: Relaxation sub-graph of Q over $F2$ with relaxations of $tp4$.

$$\{\langle \texttt{?course ex:heldAt ex:UniversityOfNantes}\rangle, \langle \texttt{ex:Ben ?y ?cours}\rangle\}$$

Table 5.4 shows two triples for `ex:heldAt`. Capabilities of `ex:heldAt` do not discard this join. But, analyzing the subject and object capabilities of whatever property (the property of $tp4'c$ is a variable), we notice that when there exists `ex:SemanticWeb` in the object, the subject contains `ex:Elsa`, not `ex:Ben`, so the join dependency on `?cours` cannot be satisfied. Thus, thanks to the dataset capabilities of F2, we identify $Q'4a$ and $Q'4c$ as failing queries.

5.3.4 Algorithm for the similarity-based relaxation graph

When the distributed query relaxation is necessary, we define an optimal similarity-based relaxation graph by sub-federation. The goal is to avoid verifying that the result set of an important number of relaxed queries is not empty. Relaxed queries are generated and executed from the most to the least similar. When we find one licensable and non-empty relaxed query that we call *candidate query*, we stop the relaxation process.

In the following, we explain how FLiQue finds the candidate query for the sub-federation F2. First, the algorithm computes the MFS of the original query, MFS(Q)=$\{\langle$`ex:Ben ex:teaches ?course`$\rangle\}$. It contains only $tp4$ because $F2$ does not contain a data source to evaluate $tp4$. Using the MFS, the algorithm considers only relaxed queries that contain a relaxation of $tp4$.

The relaxation algorithm uses a query queue ordered by similarity. This query queue gives the analysis order of relaxed queries. Figure 5.6 shows (a) a relaxation sub-graph where $tp4$ is relaxed, and (b) the analysis process of relaxed queries with the query queue (failing relaxed queries are in gray).

Relaxed queries of the first level, $Q'4a$ and $Q'4b$, are generated and inserted in the queue (1). The most similar relaxed query $Q'4a$ is analyzed. It is identified as a failing query. It is not executed, but it is relaxed, so $Q'4c$, and $Q'4d$ are generated and inserted in the query queue (2). Then, the first relaxed query in the queue, now $Q'4b$, is analyzed. It is also identified as a failing query so it is relaxed in $Q'4d$, which is already in the queue (3). Then, the first relaxed query in the queue, now $Q'4c$, is analyzed and identified as a failing query, so it is relaxed into $Q'4e$, which is inserted in the queue (4). Then, the first relaxed query in the queue, now $Q'4d$, is analyzed. It is executed returning a non-empty result set. Thus, $Q'4d$ (in bold) is the candidate query of query Q for the federation $F2$, and the relaxation process stops.

The MFS and the failing relaxed queries of this example are identified thanks to data summaries without making requests to data sources (cf. Section 5.3.3).

In this example, we found a candidate query only with the relaxation of $tp4$. But the relaxation may continue until all triple patterns are composed of variables. A threshold of similarity can be used to avoid such a case.

Figure 5.3 shows the candidate query $Q'4d$, whose similarity with Q is 0.33. This query asks for students attending a course held at the University of Nantes.

The candidate query for federation $F1$ is $Q'4b$, whose similarity with Q is 0.66. Figure 5.3 shows $Q'4b$, this query asks for students enrolled in a course held at the University of Nantes and taught by someone.

CC BY-SA can protect relaxed queries for $F1$. Relaxed queries for $F2$ can be protected by CC BY-NC but also by CC BY-NC-SA because both licenses are compliant with licenses of $D1$ and $D3$. Table 5.7 shows the feedback returned to the query issuer so that she can choose which query to execute.

Sub-federation	Query	Similarity	Compliant licenses
F1={D1, D2}	Q'4b	0.66	CC BY-SA
F2={D1, D3}	Q'4d	0.33	CC BY-NC, CC BY-NC-SA

Table 5.7: Feedback with candidate queries for the user query Q.

5.4 Experimental Evaluation

The goal of our experimental evaluation is to measure the overhead produced by the implementation of our proposal. In particular, (a) when the result set of the original query is licensable, and (b) when the original query is relaxed.

5.4.1 Setup and implementation

FLiQue is implemented over CostFed, a state-of-the-art federated query engine, which relies on a join-aware triple-wise source selection. Recent studies show that the source selection of CostFed least overestimates the set of capable data sources, with a small number of ASK requests [60, 62]. These performances make CostFed a good choice for our license-aware query processing strategy. The join ordering of CostFed is based on left-deep join trees. It implements bind and symmetric hash joins.

Our test environment uses LargeRDFBench[60], a benchmark for federated query engines. It contains real linked datasets and real queries that require several data sources to be evaluated. This benchmark contains 32 queries that are executed over a federation of 11 data sources. We identified the license of each dataset (cf. Figure 5.1). We use a Creative Commons CaLi ordering [48] to verify compatibility and compliance among licenses.

Our experiment runs on a single machine with a 160xIntel(R) Xeon(R) CPU E7-8870 v4 2.10GHz 1,5 Tb RAM. Each dataset of LargeRDFBench is saturated and made available using a single-threaded Virtuoso endpoint in a docker container with 4 Gb RAM. Between each query execution, caches are reset.

5.4.2 Performance of FLiQue vs CostFed

To measure the overhead produced by FLiQue, we compare two different federated query engines: CostFed and CostFed+FLiQue (that we call FLiQue to simplify). They correspond to the original implementation of CostFed[6] and our extension of CostFed that includes FLiQue[7]. CostFed executes a query without considering licenses while FLiQue ensures license compliance of the result set. We executed all queries 5 times with each federated query engine. We measured the time in milliseconds to return the first result of each query.

Using the capable data sources by query, and the compatibility graph of licenses, we identified 16 queries whose result set cannot be licensed. Table 5.8 shows these queries, their conflicting capable data sources and conflicting licenses. 10 need to be relaxed, they are shown in bold. We recall that the DBpedia license (CC BY-SA) is not compliant with the licenses of Jamendo (CC BY-NC-SA), Linked TCGA and Drug bank (CC BY-NC). The average time to check license conflicts is 296 milliseconds which is negligible.

Evaluation of queries that do not need relaxation. Figure 5.7 presents the execution of 22 queries of LargeRDFBench. For 16 queries, {S2, S3, S4, S5, S7,

[6] https://github.com/dice-group/CostFed
[7] https://github.com/benjimor/FLiQuE

Conflicting sources	Conflicting licenses	Queries
DBP, DB	CC BY-SA, CC BY-NC	S1, **S10, C9**
DBP, TCGA	CC BY-SA, CC BY-NC	**L7**
DBP, JA	CC BY-SA, CC BY-NC-SA	**L6**
DBP, DB, TCGA	CC BY-SA, CC BY-NC	**C10**
DBP, DB, TCGA, JA	CC BY-SA, CC BY-NC, CC BY-NC-SA	S6, **S8**, S9, C3, **C5, C8**, L1, L3, **L5, L8**

Table 5.8: The 16 queries of LargeRDFBench whose result set cannot be licensed. DBP (DBpedia), DB (Drug bank), TCGA (Linked TCGA), JA (Jamendo).

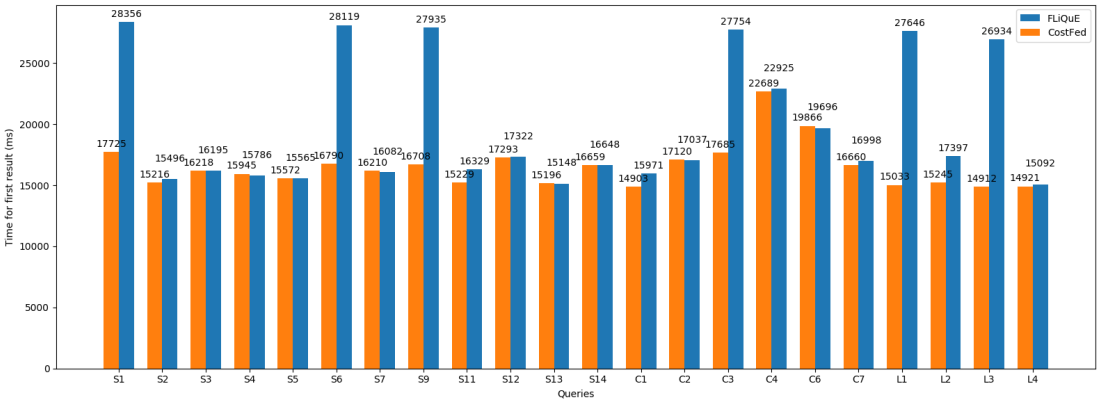

Figure 5.7: Average time to get the first result of the 22 queries of LargeRDFBench that can produce a licensable result set without relaxation.

S11, S12, S13, S14, C1, C2, C4, C6, C7, L2, L4}, FLiQue finds a license that can protect the result set when the query is executed on the complete federation. For these queries, the overhead of FLiQue is negligible and corresponds to the time to check license conflicts among the capable datasets. This overhead depends on the number of distinct licenses that protect the capable datasets.

For 6 queries, {S1, S6, S9, C3, L1, L3}, FLiQue does not find a license that can protect the result set when the query is executed over the complete federation. However, it finds a sub-federation such that the original query returns a non-empty result set that is licensable. In this case, the overhead of FLiQue corresponds to the time to check license conflicts, to compute sub-federations, and to execute the original query on these sub-federations until the first result is returned. This overhead depends on the number of tested sub-federations. The number of sub-federations depends on the number of distinct conflicting licenses by query. In our test environment, this number is always 2. For instance, conflicting licenses CC BY-SA, CC BY-NC, and CC BY-NC-SA can be separated into two non-conflicting sets {CC BY-SA} and {CC BY-NC, and CC BY-NC-SA}. These sub-federations are ordered by the number of datasets in the federation. In the benchmark, the average time to generate the sub-federations and find a non-empty result set is

11020 milliseconds. For these 6 queries, we remark that this overhead is almost constant. That is because, a non-empty result set is found when FLiQue executes the original query on the second sub-federation.

Evaluation of queries that are relaxed. Figure 5.8 represents the execution of 10 queries of LargeRDFBench that need relaxation to return a non-empty result set that can be protected by a license. For each query, we compare the time to get the first result of the original query for CostFed, and the time to get the first result of the first candidate query found by FLiQue.

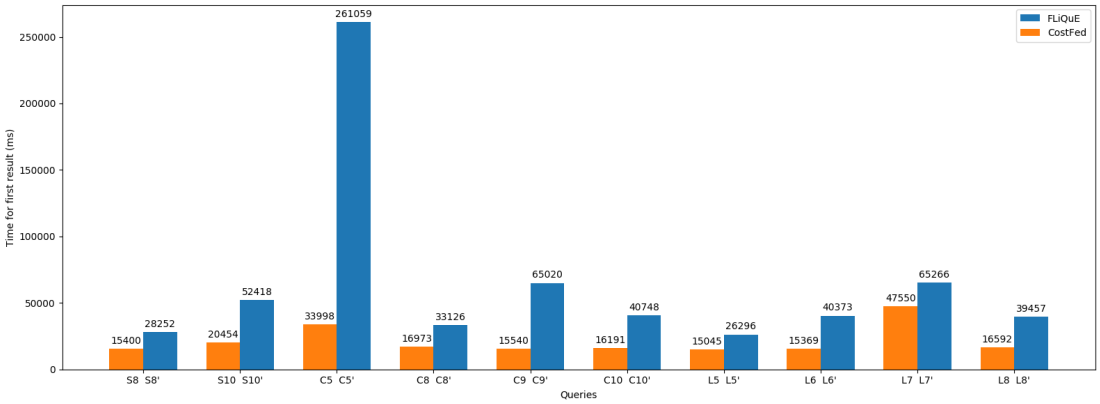

Figure 5.8: Average time to get the first result of the 10 queries of LargeRDFBench that need relaxation to produce a licensable result set.

The FLiQue overhead corresponds to the time to check license conflicts, to compute sub-federations, and to find the first candidate query. We remark that the execution time of an original query and a candidate query is not comparable. They are not the same query, and they are not executed on the same number of data sources. To have an idea (non-representative) of the similarities, the maximum is $0,811$ (L5'), the minimum is $0,077$ (C8'), the average is $0,487$, and the median is 0.603. Figures 15-24 in Appendix A show candidate queries and their similarities.

Overhead varies a lot depending on the queries. It depends on the number of generated and executed failing relaxed queries, before finding the first candidate query. Figure 5.9 shows the number of failing relaxed queries, (1) generated, and (2) executed before finding each candidate query. We recall that an important number of generated relaxed queries are identified as failing thanks to data summaries. The candidate query $C5'$, is found after generating 69 failing relaxed queries, but only 3 were executed. In contrast, candidate query $L5'$ is found after generating 3 failing relaxed queries but executing only one. For 6 out of 10 relaxed queries, FLiQue does not need to execute any generated relaxed query to identify them as failing. With

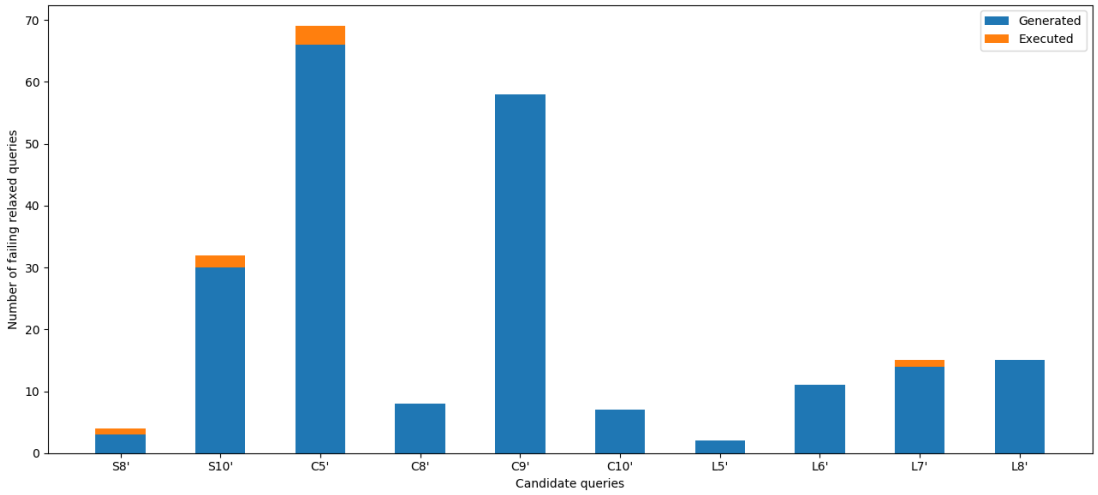

Figure 5.9: Number of generated and executed failing relaxed queries until finding each candidate query.

this benchmark, on average FLiQue generates 21.4 failing relaxed queries, and executes 1.75 failing relaxed queries. Thus, we consider that FLiQue succeeds in limiting communication costs during the relaxation of queries whose result set cannot be licensed.

We use SPARQL 1.0. We think that, with SPARQL 1.1, the SERVICE clause can restrict the evaluation of a BGP to a subset of data sources. Thus, the source selection and the source discovering processes would generate less licensing conflicts.

5.5 Conclusion

In this chapter, we propose FLiQue, a federated license-aware query processing strategy. It ensures that a license protects the result set of any SPARQL query. To our knowledge, this is the first work that uses query relaxation in a distributed environment. Our implementation extends an existing federated query engine with our license-aware query processing strategy. Our prototype demonstrates the usability of our approach. Experimental evaluation shows that FLiQue ensures license compliance with a limited overhead in terms of execution time. FLiQue is a step towards facilitating and encouraging the publication and reuse of licensed resources in the Web of Data. FLiQue is not a data access control strategy. It empowers well-intentioned data users in respecting the licenses of datasets involved in a federated query.

Chapter 6

Conclusion

Contents

6.1 Summary

In this thesis, we focused on license compliance during reuse on the Web of Data. In particular, we focused on two research problems, (1) how to position licenses in terms of compatibility and (2) how to ensure license compliance in federated query engines that we answered with two contributions. Within the scope of this thesis, we also proposed three demonstrators that address the problem of data integration in the Web of Data.

6.1.1 Ordering Licenses in Terms of Compatibility

The first research problem we answered is: *given a license, how to automatically position it over a set of licenses in terms of compatibility and compliance?*

1. **CaLi** is the first model to partially order any set of RDF licenses in terms of compatibility. Restrictiveness relation among RDF licenses is automatically deduced and refined with constraints to identify compatibility relations. We validate CaLi experimentally with a quadratic algorithm that can add an RDF license in a set of n ordered licenses in at most $n^2/2$ comparisons. To show its usability, we implemented a prototype of a license-based search engine that can find RDF datasets or source codes that can be reused under a specific license. CaLi is a step towards facilitating and encouraging the publication and reuse of licensed resources on the Web. A limitation of our approach is that it does not take into account some aspects of licenses that can impact compatibility such as, jurisdiction, date of reuse or explicit definition of compatibility relations that are not coherent with the restrictiveness relation.

6.1.2 Ensuring License Compliance During Federated Query Processing

The second research problem we answered is: *given a SPARQL query and a federation of licensed datasets, how to guarantee a relevant and non-empty query result whose license is compliant with each license of involved datasets?*

2. **FLiQue** is the first license aware query processing strategy for federated query engines. For any federated query, it guarantees that a non-empty result set that can be protected by a license compliant with each license of involved datasets is returned. It uses CaLi to check license compliance. If such a license does not exist, FLiQue discards datasets with conflicting licenses and uses query rewriting to find the most similar query that returns a non-empty result set. It also uses data summaries to limit communication costs during query evaluation and query relaxation process. To demonstrate the usability of FLiQue, we implemented our strategy on a state-of-the-art federated query engine, enriched an existing federated querying benchmark with licenses and, executed the benchmark with our implementation. Experimental evaluation shows that the overhead produced by FLiQue is negligible when queries do not need relaxation. In the other case, it depends on the number of executed relaxed queries before finding the most similar one. To our knowledge, it is the first attempt to query relaxation in a distributed environment.

6.1.3 Integrating Data into the Web of Data

In chapter 3 we proposed three demonstrators addressing the problem of data integration in the Web of Data. With Opendatasoft, we aimed at facilitating as much as possible the transformation and publication of structured datasets on the Web of Data. Our approaches are based on RDF mappings. Explicit description of mappings using RDF mapping languages improves interoperability among data integration tools and allows mappings to be shared and reused across the Web of Data.

3. **ODMTP** is the first interface that allows executing SPARQL queries on non-RDF datasets using the Triple Pattern Fragments interface and RDF mappings. The advantage of our approach is that RDF is not materialized and thus reduces storage costs. Another advantage is the high availability guaranteed by the simplicity of the Triple Pattern Fragments server interface. We also show how ODMTP supports RDFS and OWL inference rules. Experimental results show that the ODMTP approach produces a limited overhead compared to the classical Triple Pattern Fragments approach. To demonstrate its usability, we implemented ODMTP to query Twitter, GitHub, and LinkedIn APIs. An implementation of this approach is used on the Opendatasoft platform. A limitation of this approach is that inference rules that need instances are not supported because they are not materialized in RDF mappings.

4. **EvaMap** is a framework that can evaluate the quality of RDF mappings. We proposed new dimensions and metrics to state-of-the-art to evaluate different aspects of an RDF mapping. The advantage of EvaMap is that metrics are evaluated at the beginning of the publishing process on the RDF mapping instead of the resulting RDF dataset and, thus, saves time. EvaMap is highly configurable and allows giving more or less importance to each metric. To shows its usability, we implemented EvaMap to evaluate mappings for datasets of the Opendatasoft platform.

5. The **SemanticBot** is a conversational interface that can generate RDF mappings for structured datasets. It only asks simple questions to users about their datasets. The advantage of SemanticBot is that it allows users that are not familiar with RDF concepts to quickly integrate their datasets as Linked Data, allowing them to make their first steps into the Web of Data. To demonstrate its usability, we implemented SemanticBot to generate RDF mappings for datasets of Opendatasoft's data network. A limitation of our approach is that the quality of the generated RDF mapping is highly

dependant on the quality of the structured dataset, the instances described in DBpedia and YAGO and, ontologies available on Linked Open Vocabularies.

In the next section, we discuss about perspectives related to our contributions.

6.2 Perspectives

Our contributions highlight several challenges that need to be addressed. Here, we discuss these perspectives, in particular: (1) extending the SemanticBot to improve the quality of generated mappings; (2) deducing queries from RDF mappings to know how an RDF dataset can benefit from other datasets of the Web of Data; (3) extending our definition of licenses to improve the reliability of CaLi orderings; (4) defining relations among CaLi orderings to exchange information among license-aware applications based on CaLi orderings; and finally, (5) ranking candidate queries returned by FLiQue to take into account other aspects than reuse.

6.2.1 Extending the SemanticBot

A limitation of the SemanticBot is that the coverability dimension quality of the generated mapping is not always satisfying. It means that all attributes of an initial dataset are not always mapped. That is because the quality of the generated mapping highly depends on (1) the quality of the initial dataset because the SemanticBot uses the metadata of the dataset (i.e., column name, data types) to find relevant ontologies; (2) the ontologies available on Linked Open Vocabularies that is the dataset used to find ontologies; and (3) the resources described on DBpedia and YAGO that are used in the entities recognition step.

To improve the average quality of the generated mappings, we propose to extend the SemanticBot. DBpedia, YAGO, and Linked Open Vocabularies are cross-domain RDF datasets. Thus, our approach fails to describe domain-specific datasets with their ontologies. A solution is to consider other domain-specific datasets and especially to allow users to choose ontologies and datasets. Another problem is that the name and the data type of a column are not always enough to find an ontology to describe it. To tackle this problem, we are working on new interactions with the user, such as asking him to describe the meaning of a column in natural language.

6.2.2 Deducing Federated Queries from RDF Mappings

Publishing data as RDF is an expensive investment in terms of storage and time. Before publishing data as RDF, a data provider needs to know how and which

datasets might benefit from the Web of Data. That is, with which datasets of the Web of Data its dataset can be queried. We think that having examples of federated queries and a list of ranked datasets from the Web of Data according to their degree of conjunction or disjunction with a specific dataset would be very valuable.

Existing works propose to generate SPARQL queries based on the dataset [22, 55] or logs [25] of query execution. But these approaches cannot be used with newly virtually integrated non-RDF datasets because the dataset is not materialized in RDF, and a log of SPARQL queries does not yet exist. Moreover, they do not propose to rank RDF datasets according to their degree of conjunction or disjunction with a specific dataset.

An interesting approach is to generate federated queries from a set of RDF mappings instead of an RDF datasets or query logs. This approach has the advantage of working with newly integrated datasets and does not require an important investment in terms of time from the data provider.

6.2.3 Extending the Definition of Licenses

Creating applications that preserve license compliance requires to take into account compatibility among licenses. We proposed CaLi, a model that can automatically order licenses in terms of compatibility and compliance.

CaLi allows to guarantee incompatibilities among licenses but fails to guarantee that a license is legally compatible with another license. Thus, applications using CaLi cannot provide legal advice and need to be confirmed by legal experts.

To improve the reliability of the compatibility relation in CaLi, a solution is to extend the definition of licenses such that it takes into account other aspects. Among the most important (1) the jurisdiction that can impact the distribution of actions in status according to the country in which the license is used; (2) the explicit definition of licenses with which the license is compatible with; and (3) the temporal validity of the license.

6.2.4 Defining Relations Among CaLi Orderings

The CaLi model is based on a set of actions, a restrictiveness lattice of status, and constraints. Composing these three elements allows users to define their own CaLi orderings according to the application domain.

Imagine two applications using two different CaLi orderings. In this context, reusing resources of these two applications while preserving license compliance is difficult. That is because adding a license from one CaLi order to the other is not always possible.

A perspective is to analyze how two CaLi orderings can be compared. That is (1) to define conditions allowing two CaLi orderings to be compared (e.g., a function to pass from a set of actions to another, An inclusion relation between both sets of actions, A lattice homomorphism between the restrictiveness lattice of status, etc.) and (2) to define the function that passes from a CaLi ordering to another.

6.2.5 Ranking Candidate Queries

When several licensed data sources (i.e., federation) participate in the evaluation of a federated query, the result must be protected by a license that preserves license compliance.

In our implementation of FLiQue, candidate queries are ranked from the least to the most restrictive license that can protect the result set of the query. This choice encourages producing highly reusable result sets. However, other ranking orders could be considered.

Even if each candidate query is the most similar to the original query for its sub-federation, this ranking does not consider the similarity to the original query across sub-federations. A different ranking could be a trade-off between similarity and reuse. Another problem occurs when the execution time of the first candidate query is longer than the second one. In this case, the time to retrieve the first result could be improved if the second candidate query is executed first. To tackle this problem, we can imagine an order based on an estimation of the time to execute each query.

Appendix A

Supplemental Materials

```
{
  "@context": {
    "dbpedia": "http://dbpedia.org/resource/",
    "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#"
  },
  "@id": "https://example.org/Opendatasoft",
  "@type": "http://dbpedia.org/ontology/Company",
  "http://dbpedia.org/ontology/formationYear": {
    "@type": "xsd:gYear",
    "@value": "2011"
  },
  "http://dbpedia.org/ontology/locationCity": [
    {
      "@id": "dbpedia:Paris"
    },
    {
      "@id": "dbpedia:Nantes"
    },
    {
      "@id": "dbpedia:Boston"
    }
  ]
}
```

Listing 9: An RDF graph serialized in JSON-LD.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dbo="http://dbpedia.org/ontology/">
  <rdf:Description rdf:about="https://example.org/Opendatasoft">
    <rdf:type rdf:resource="http://dbpedia.org/ontology/Company"/>
    <dbo:formationYear rdf:datatype="http://www.w3.org/2001/XMLSchema#gYear">
            2011
    </dbo:formationYear>
    <dbo:locationCity rdf:resource="http://dbpedia.org/resource/Paris"/>
    <dbo:locationCity rdf:resource="http://dbpedia.org/resource/Boston"/>
    <dbo:locationCity rdf:resource="http://dbpedia.org/resource/Nantes"/>
  </rdf:Description>
</rdf:RDF>
```

Listing 10: An RDF graph serialized in RDF-XML.

```xml
<div xmlns="http://www.w3.org/1999/xhtml"
  prefix="
    dbo: http://dbpedia.org/ontology/
    rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
    xsd: http://www.w3.org/2001/XMLSchema#
    rdfs: http://www.w3.org/2000/01/rdf-schema#"
  >
  <div typeof="dbo:Company" about="https://example.org/Opendatasoft">
    <div rel="dbo:locationCity" resource="http://dbpedia.org/resource/Boston">
    </div>
    <div rel="dbo:locationCity" resource="http://dbpedia.org/resource/Paris">
    </div>
    <div rel="dbo:locationCity" resource="http://dbpedia.org/resource/Nantes">
    </div>
    <div property="dbo:formationYear" datatype="xsd:gYear" content="2011">
    </div>
  </div>
</div>
```

Listing 11: An RDF graph serialized in RDFa.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix hydra: <http://www.w3.org/ns/hydra/core#> .
@prefix ns1: <http://purl.org/dc/terms/> .
@prefix ns2: <urn:x-rdflib:> .
@prefix ods: <https://public.opendatasoft.com/ld/ontologies/roman-emperors/> .
@prefix ods_tpf: <https://public.opendatasoft.com/api/tpf/roman-emperors/#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix void: <http://rdfs.org/ns/void#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ods_tpf:metadata { # Metadata and Controls
    <https://public.opendatasoft.com/api/tpf/#dataset/#dataset>
            hydra:member ods_tpf:dataset .

    ods_tpf:metadata
            foaf:primaryTopic <https://public.opendatasoft.com/api/tpf/roman-emperors/> .

    <https://public.opendatasoft.com/api/tpf/roman-emperors/?predicate=https%3A%2F%2Fpublic....>
        a hydra:PartialCollectionView ;
        ns1:description "Triples of the roman-emperors dataset matching the pattern {?s=..." ;
        ns1:source ods_tpf:dataset ;
        ns1:title "roman-emperors Opendatasoft dataset of public domain" ;
        void:triples "9"^^xsd:int ;
        hydra:first <https://public.opendatasoft.com/api/tpf/roman-emperors/?predicate=...> ;
        hydra:itemsPerPage "40"^^xsd:int ;
        hydra:totalItems "9"^^xsd:int .

    <https://public.opendatasoft.com/api/tpf/roman-emperors/>
            void:subset <https://public.opendatasoft.com/api/tpf/roman-emperors/...> .

    ods_tpf:dataset a void:Dataset,
            hydra:Collection ;
        void:subset <https://public.opendatasoft.com/api/tpf/roman-emperors/> ;
        void:uriLookupEndpoint "https://public.opendatasoft.com/api/tpf/roman-emperors/..." ;
        hydra:search [ hydra:mapping [ hydra:property rdf:object ;
                           hydra:variable "object" ],
                       [ hydra:property rdf:predicate ;
                           hydra:variable "predicate" ],
                       [ hydra:property rdf:subject ;
                           hydra:variable "subject" ] ;
                   hydra:template "https://public.opendatasoft.com/api/tpf/roman-emperors/..." ;
                   hydra:variableRepresentation hydra:ExplicitRepresentation ] .
}

{ # Results
    <https://public.opendatasoft.com/ld/resources/roman-emperors/roman-emperors-record/5057...>
    ods:birth_cty "Rome"^^xsd:string .

    <https://public.opendatasoft.com/ld/resources/roman-emperors/roman-emperors-record/6971...>
    ods:birth_cty "Rome"^^xsd:string .
    ...
}
```

Listing 12: An extract of a fragment returned by a Triple Pattern Fragments server.

```
prefixes:
  ex: "http://example.org/"
  dbo: "http://dbpedia.org/ontology/"
  locah: "http://data.archiveshub.ac.uk/def/"
  geo: "http://www.w3.org/2003/01/geo/wgs84_pos#"
  foaf: "http://xmlns.com/foaf/0.1/"
  schema: "http://schema.org/"
  wikidata: "http://www.wikidata.org/entity/"
  dul: "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#"
sources:
  dataset-source: [people.csv~csv]
mappings:
  person:
    source: dataset-source
    s: ex:Person/$(Name)
    po:
      - [a, dbo:Person]
      - [a, dbo:Agent]
      - [a, foaf:Person]
      - [a, dul:NaturalPerson]
      - [a, wikidata:Q215627]
      - [a, wikidata:Q5]
      - [a, schema:Person]
      - [dbo:name, $(Name)]
      - [locah:dateBirth, $(Birth)]
      - [dbo:birthPlace, ex:Place/$(Birth City)~iri]
      - [dbo:birthPlace, ex:Place/$(Birth Province)~iri]
      - [rdfs:label, $(Name)]
  place:
    source: dataset-source
    s: ex:Place/$(Birth City)
    po:
      - [a, dbo:Place]
      - [a, dbo:Location]
      - [a, schema:Place]
      - [a, geo:SpatialThing]
      - [rdfs:label, $(Birth City)]
  spatialthing:
    source: dataset-source
    s: ex:Place/$(Birth Province)
    po:
      - [a, geo:SpatialThing]
      - [geo:lat, $(Lat)]
      - [geo:long, $(Long)]
      - [rdfs:label, $(Birth Province)]
```

Listing 13: An RDF mapping serialized in YARRRML.

```
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix odrl:  <http://www.w3.org/ns/odrl/2/> .
@prefix cali:  <http://cali.priloo.univ-nantes.fr/ontology#> .

cali:Undefined  a  cali:Status  ;
          rdfs:label         "Undefined status" ;
          rdfs:comment       "Stating that an action is in an Undefined state" ;
          cali:lessRestrictiveThan       odrl:Permission .

odrl:Permission  a  cali:Status  ;
          rdfs:label         "Permission" ;
          rdfs:comment       "The ability to perform an Action over an Asset." ;
          cali:lessRestrictiveThan       odrl:Duty .

odrl:Duty  a  cali:Status  ;
          rdfs:label         "Duty" ;
          rdfs:comment       "The obligation to perform an Action" ;
          cali:lessRestrictiveThan       odrl:Prohibition .

odrl:Prohibition  a  cali:Status  ;
          rdfs:label         "Prohibition" ;
          rdfs:comment       "The inability to perform an Action over an Asset." .
```

Listing 14: The restrictiveness lattice of status of Figure 4.2c in RDF format.

```
PREFIX tcga: <http://tcga.deri.ie/schema/>
PREFIX kegg: <http://bio2rdf.org/ns/kegg#>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX purl: <http://purl.org/dc/terms/>

SELECT  DISTINCT ?patient  ?gender ?country ?popDensity ?drugName ?indication ?formula ?compound
WHERE
{
 ?uri tcga:bcr_patient_barcode ?patient .
 ?patient tcga:gender ?gender .
 ?patient dbpedia:country ?country .
 ?country dbpedia:populationDensity ?popDensity.
 ?patient tcga:bcr_drug_barcode ?drugbcr.
 ?drugbcr tcga:drug_name ?drugName.
 ?drgBnkDrg drugbank:genericName ?drugName.
 ?drgBnkDrg drugbank:indication ?indication.
 ?drgBnkDrg drugbank:chemicalFormula ?formula.
 ?drgBnkDrg drugbank:keggCompoundId ?compound .
}


SELECT  DISTINCT ?patient  ?gender ?country ?popDensity ?drugName ?indication ?formula ?compound
WHERE
{
 ?uri tcga:bcr_patient_barcode ?patient .
 ?patient tcga:gender ?gender.
 ?patient dbpedia:country ?country.
 ?country ?7sqC60 ?popDensity.
 ?patient tcga:bcr_drug_barcode ?drugbcr.
 ?drugbcr tcga:drug_name ?drugName.
 ?drgBnkDrg drugbank:genericName ?drugName.
 ?drgBnkDrg drugbank:indication ?indication.
 ?drgBnkDrg drugbank:chemicalFormula ?formula.
 ?drgBnkDrg drugbank:keggCompoundId ?compound .
}

# Similarity: 0.66666666666
```

Listing 15: The query C10 followed by the candidate query C10'.

```
PREFIX linkedmdb: <http://data.linkedmdb.org/resource/movie/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?actor ?movie ?movieTitle ?movieDate ?birthDate ?spouseName
{
 ?actor rdfs:label ?actor_name_en .
 ?actor dbpedia:birthDate ?birthDate .
 ?actor dbpedia:spouse ?spouseURI .
 ?spouseURI rdfs:label ?spouseName .
 ?imdbactor linkedmdb:actor_name ?actor_name.
 ?movie linkedmdb:actor ?imdbactor .
 ?movie dcterms:title ?movieTitle .
 ?movie dcterms:date ?movieDate .
 FILTER(STR(?actor_name_en ) = STR(?actor_name))
}

SELECT ?actor ?movie ?movieTitle ?movieDate ?birthDate ?spouseName
{
 ?actor rdfs:label ?actor_name_en .
 ?actor ?SxzR4W ?birthDate .
 ?actor ?1OCiE4 ?spouseURI .
 ?spouseURI rdfs:label ?spouseName .
 ?imdbactor linkedmdb:actor_name ?actor_name .
 ?movie linkedmdb:actor ?imdbactor .
 ?movie dcterms:title ?movieTitle .
 ?movie dcterms:date ?movieDate .
 FILTER(STR(?actor_name_en ) = STR(?actor_name))
}
# Similarity: 0.4444444444
```

Listing 16: The query C5 followed by the candidate query C5'.

```
PREFIX swc: <http://data.semanticweb.org/ns/swc/ontology#>
PREFIX swrc: <http://swrc.ontoware.org/ontology#>
PREFIX eswc: <http://data.semanticweb.org/conference/eswc/>
PREFIX iswc:  <http://data.semanticweb.org/conference/iswc/2009/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX purl: <http://purl.org/ontology/bibo/>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT * WHERE
{
 ?paper swc:isPartOf iswc:proceedings .
 iswc:proceedings swrc:address ?proceedingAddress.
 ?paper swrc:author ?author .
 ?author swrc:affiliation ?affiliation ;
 ?author rdfs:label ?fullnames ;
 ?author foaf:based_near ?place.
 ?place dbpedia:capital ?capital .
 ?place dbpedia:populationDensity ?populationDensity .
 ?place dbpedia:governmentType ?governmentType .
 ?place dbpedia:language ?language .
 ?place dbpedia:leaderTitle ?leaderTitle .
}


SELECT DISTINCT * WHERE
{
 ?paper swc:isPartOf iswc:proceedings .
 iswc:proceedings swrc:address ?proceedingAddress.
 ?paper swrc:author ?author .
 ?author swrc:affiliation ?affiliation .
 ?author rdfs:label ?fullnames .
 ?author foaf:based_near ?place.
 ?place ?mM9RIT ?capital .
 ?place ?cZP8iP ?populationDensity .
 ?place ?Pp7c1t ?governmentType .
 ?place ?z2uYJB ?language .
 ?place ?de7OQZ ?leaderTitle .
}

# Similarity: 0.07777777777
```

Listing 17: The query C8 followed by the candidate query C8'.

```
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>

SELECT * WHERE
{
 ?Drug rdf:type dbpedia:Drug .
 ?drugbankDrug owl:sameAs ?Drug .
 ?InteractionName drugbank:interactionDrug1 ?drugbankDrug .
 ?InteractionName drugbank:interactionDrug2 ?drugbankDrug2 .
 ?InteractionName drugbank:text ?IntEffect .
 OPTIONAL
 {
  ?drugbankDrug  drugbank:affectedOrganism 'Humans and other mammals' .
  ?drugbankDrug drugbank:description ?description .
  ?drugbankDrug drugbank:structure ?structure .
  ?drugbankDrug drugbank:casRegistryNumber ?casRegistryNumber .
 }
}
ORDER BY (?drugbankDrug)


SELECT * WHERE
{
 ?Drug ?OOgzMk ?Cvqg5H .
 ?drugbankDrug ?c5DqMr ?Drug .
 ?InteractionName drugbank:interactionDrug1 ?drugbankDrug .
 ?InteractionName drugbank:interactionDrug2 ?drugbankDrug2 .
 ?InteractionName drugbank:text ?IntEffect .
 OPTIONAL
 {
  ?drugbankDrug  drugbank:affectedOrganism 'Humans and other mammals' .
  drugbank:description ?description .
  drugbank:structure ?structure .
  drugbank:casRegistryNumber ?casRegistryNumber .
 }
}
ORDER BY (?drugbankDrug)

# Similarity: 0.222222222222
```

Listing 18: The query C9 followed by the candidate query C9'.

```
1  PREFIX dbpedia: <http://dbpedia.org/resource/>
2  PREFIX dbprop: <http://dbpedia.org/property/>
3  PREFIX dbowl: <http://dbpedia.org/ontology/>
4  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
5  PREFIX owl: <http://www.w3.org/2002/07/owl#>
6  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
7  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
8  PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
9  PREFIX factbook: <http://www4.wiwiss.fu-berlin.de/factbook/ns#>
10 PREFIX mo: <http://purl.org/ontology/mo/>
11 PREFIX dc: <http://purl.org/dc/elements/1.1/>
12 PREFIX fb: <http://rdf.freebase.com/ns/>
13
14 SELECT * WHERE {
15  ?a dbowl:artist dbpedia:Michael_Jackson .
16  ?a rdf:type dbowl:Album .
17  ?a foaf:name ?n .
18 }
19
20
21 SELECT * WHERE {
22  ?a dbowl:artist dbpedia:Michael_Jackson .
23  ?a rdf:type dbowl:Album .
24  ?a rdfs:label ?n .
25 }
26
27 # Similarity: 0.8115399282213058
```

Listing 19: The query L5 followed by the candidate query L5'.

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbowl: <http://dbpedia.org/ontology/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX linkedMDB: <http://data.linkedmdb.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX geo: <http://www.geonames.org/ontology#>

SELECT * WHERE {
 ?director dbowl:nationality dbpedia:Italy .
 ?film dbowl:director ?director.
 ?x owl:sameAs ?film .
 ?x foaf:based_near ?y .
 ?y geo:officialName ?n .
}


SELECT * WHERE {
 ?director dbowl:nationality dbpedia:Italy .
 ?film dbowl:director ?director.
 ?x owl:sameAs ?film .
 ?x ?6GKJwd ?y .
 ?y geo:officialName ?n .
}

# Similarity: 0.66666666666
```

Listing 20: The query L6 followed by the candidate query L6'.

```
PREFIX tcga: <http://tcga.deri.ie/schema/>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?patient ?p ?o
WHERE
{
 ?uri tcga:bcr_patient_barcode ?patient .
 ?patient dbpedia:country ?country.
 ?country dbpedia:populationDensity ?popDensity.
 ?patient tcga:bcr_aliquot_barcode ?aliquot.
 ?aliquot ?p ?o.
}


SELECT DISTINCT ?patient ?p ?o
WHERE
{
 ?uri tcga:bcr_patient_barcode ?patient .
 ?patient dbpedia:country ?country .
 ?country ?cG4icP ?popDensity.
 ?patient tcga:bcr_aliquot_barcode ?aliquot .
 ?aliquot ?p ?o .
}

# Similarity: 0.66666666666
```

Listing 21: The query L7 followed by the candidate query L7'.

```
PREFIX kegg: <http://bio2rdf.org/ns/kegg#>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>

SELECT * WHERE {
        ?drug drugbank:drugCategory drugbank:micronutrient .
        ?drug drugbank:casRegistryNumber ?id .
        ?drug owl:sameAs ?s .
        ?s foaf:name ?o .
        ?s skos:subject ?sub .
}


SELECT * WHERE {
 ?drug drugbank:drugCategory drugbank:micronutrient .
 ?drug drugbank:casRegistryNumber ?id .
 ?drug ?XKeC36 ?s .
 ?s rdfs:label ?o .
 ?s skos:subject ?sub .
}

# Similarity: 0.5410266188142039
```

Listing 22: The query L8 followed by the candidate query L8'.

```
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?Drug ?IntDrug ?IntEffect WHERE {
 ?Drug a dbo:Drug .
 ?y owl:sameAs ?Drug .
 ?Int drugbank:interactionDrug1 ?y .
 ?Int drugbank:interactionDrug2 ?IntDrug .
 ?Int drugbank:text ?IntEffect .
}


SELECT ?Drug ?IntDrug ?IntEffect WHERE {
 ?Drug ?zkB8o2 ?OKS9kY .
 ?y ?Y2df3t ?Drug .
 ?Int drugbank:interactionDrug1 ?y .
 ?Int ?Q6kLIS ?IntDrug .
 ?Int drugbank:text ?IntEffect .
}

# Similarity: 0.11111111111
```

Listing 23: The query S10 followed by the candidate query S10'.

```
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?drug ?melt WHERE {
 { ?drug drugbank:meltingPoint ?melt. }
 UNION
 { ?drug dbo:meltingPoint ?melt . }
}


SELECT ?drug ?melt WHERE {
 { ?drug drugbank:meltingPoint ?melt. }
 UNION
 { ?drug ?1HmoLC ?melt . }
}

# Similarity: 0.666666666666666
```

Listing 24: The query S8 followed by the candidate query S8'.

Algorithm 3: The insertion of a new license in a CaLi ordering graph starting from the bottom.

```
1  Function insertFromBottomRec(l_i, l_j, G):
       Data: l_i, l_j, l_compl_j: License,
       G: Graph,
       S: Supremum,
       I: Infimum,
       C_→: Compatibility constraints,
       LRCompl_{l_j}: Least restrictive licenses compliant with l_j in G,
       LRCompl_{l_i}: Least restrictive licenses compliant with l_i in G
2      if l_j is not already visited then
3          for l_compl_j ∈ LRCompl_{l_j} do
4              if l_i ∨ l_compl_j = l_compl_j and is_compatible(l_i →_L l_compl_j, C_→) then
                   // l_i is compatible with l_compl_j
5                  for l_compl_i ∈ LRCompl_{l_i} − l_compl_j do
6                      if l_compl_i ∨ l_compl_j = l_compl_i and
                        is_compatible(l_compl_j →_L l_compl_i, C_→) then
                           // l_compl_j is compatible with l_compl_i
7                          delete l_i → l_compl_i from G
                           // l_i is now compatible with l_compl_i through
                               l_compl_j
8                  if not compatible(l_i →_L l_compl_j, G) then
9                      add l_i → l_compl_j to G
10                 if l_i ∨ l_j = l_i and is_compatible(l_j →_L l_i, C_→) then
                       // l_i is between l_j and l_compl_j
11                     add l_j → l_i to G
12                     delete l_j → l_compl_j from G
13             else
                   // Classifies l_i in licenses compliant with l_compl_j
14                 return insertFromBottomRec(l_i, l_compl_j, G)
15     l_j is marked as visited if LRCompl_{l_i} == ∅ then
           // l_i is compatible with no license
16         add l_i → S to G
17     if MRComp_{l_i} == ∅ then
           // l_i is compliant with no license
18         add I → l_i to G
19     return G
```

List of Figures

List of Algorithms

List of Tables

List of Listings

Bibliography

[1] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. "Describing Linked Datasets". In: *Linked Data on the Web (LDOW) collocated with WWWC*. 2009.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. "The Semantic Web". In: *Scientific American* (2001).

[3] A. Bernstein, J. Hendler, and N. Noy. "A New Look at the Semantic Web". In: *Communications of the ACM* (2016).

[4] C. Bizer, T. Heath, and T. Berners-Lee. "Linked Data: The Story so Far". In: *Semantic Services, Interoperability and Web Applications: Emerging Concepts*. 2011.

[5] C. Buil-Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche. "Sparql Web-Querying Infrastructure: Ready for Action?" In: *International Semantic Web Conference (ISWC)*. 2013.

[6] E. Cabrio, A. P. Aprosio, and S. Villata. "These Are Your Rights". In: *Extended Semantic Web Conference (ESWC)*. 2014.

[7] L. Costabello, S. Villata, and F. Gandon. "Context-Aware Access Control for RDF Graph Stores". In: *European Conference on Artificial Intelligence (ECAI)*. 2012.

[8] *Creative Commons licenses in RDF*. https://github.com/creativecommons/cc.licenserdf.

[9] E. Daga, M. d'Aquin, E. Motta, and A. Gangemi. "A Bottom-up Approach for Licences Classification and Selection". In: *Workshop on Legal Domain and Semantic Web Applications (LeDA-SWAn) collocated with ESWC*. 2015.

[10] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. 2002.

[11] D. E. Denning. "A Lattice Model of Secure Information Flow". In: *Communications of the ACM* (1976).

[12] A. Dimou, D. Kontokostas, M. Freudenberg, R. Verborgh, J. Lehmann, E. Mannens, S. Hellmann, and R. Van de Walle. "Assessing and Refining Mappings to RDF to Improve Dataset Quality". In: *International Semantic Web Conference (ISWC)*. 2015.

[13] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data". In: *Linked Data on the Web (LDOW) collocated with WWWC*. 2014.

[14] J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. "Binary RDF Representation for Publication and Exchange (HDT)". In: *Journal of Web Semantics* (2013).

[15] S. Ferré. "Answers Partitioning and Lazy Joins for Efficient Query Relaxation and Application to Similarity Search". In: *Extended Semantic Web Conference (ESWC)*. 2018.

[16] G. Fokou, S. Jean, A. Hadjali, and M. Baron. "RDF Query Relaxation Strategies Based on Failure Causes". In: *Extended Semantic Web Conference (ESWC)*. 2016.

[17] P. Folz, G. Montoya, H. Skaf-Molli, P. Molli, and M.-E. Vidal. "Semlav: Querying Deep Web and Linked Open Data with SPARQL". In: *Extended Semantic Web Conference (ESWC)*. 2014.

[18] A. Gabillon and L. Letouzey. "A View Based Access Control Model for SPARQL". In: *International Conference on Network and System Security (NSS)*. 2010.

[19] G. Gangadharan, M. Weiss, V. D'Andrea, and R. Iannella. "Service License Composition and Compatibility Analysis". In: *International Conference on Service-Oriented Computing (ICSOC)*. 2007.

[20] B. Glimm, A. Hogan, M. Krötzsch, and A. Polleres. "OWL: Yet to Arrive on the Web of Data?" In: *Linked Data on the Web (LDOW) collocated with WWWC*. 2012.

[21] O. Görlitz and S. Staab. "SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions". In: *Workshop Consuming Linked Data (COLD) collocated with ISWC*. 2011.

[22] O. Görlitz, M. Thimm, and S. Staab. "SPLODGE: Systematic Generation of SPARQL Benchmark Queries for Linked Open Data". In: *International Semantic Web Conference (ISWC)*. 2012.

[23] G. Governatori, A. Rotolo, S. Villata, and F. Gandon. "One License to Compose Them All. A Deontic Logic Approach to Data Licensing on the Web of Data". In: *International Semantic Web Conference (ISWC)*. 2013.

[24] S. Gupta, P. Szekely, C. A. Knoblock, A. Goel, M. Taheriyan, and M. Muslea. "Karma: A System for Mapping Structured Sources Into the Semantic Web". In: *Extended Semantic Web Conference (ESWC), Poster&Demo*. 2012.

[25] F. Hacques, H. Skaf-Molli, P. Molli, and S. E. Hassad. "PFed: Recommending Plausible Federated SPARQL Queries". In: *International Conference on Database and Expert Systems Applications (DEXA)*. 2019.

[26] O. Hartig and C. Buil-Aranda. "Bindings-restricted Triple Pattern Fragments". In: *On the Move to Meaningful Internet Systems Confederated International Conferences (OTM)*. 2016.

[27] G. Havur, S. Steyskal, O. Panasiuk, A. Fensel, V. Mireles, T. Pellegrini, T. Thurner, A. Polleres, and S. Kirrane. "DALICC: A Framework for Publishing and Consuming Data Assets Legally". In: *International Conference on Semantic Systems (SEMANTICS), Poster&Demo*. 2018.

[28] P. Heyvaert, B. De Meester, A. Dimou, and R. Verborgh. "Declarative Rules for Linked Data Generation at Your Fingertips!" In: *Extended Semantic Web Conference (ESWC), Poster&Demo*. 2018.

[29] P. Heyvaert, A. Dimou, A.-L. Herregodts, R. Verborgh, D. Schuurman, E. Mannens, and R. Van de Walle. "RMLEditor: a Graph-Based Mapping Editor for Linked Data Mappings". In: *Extended Semantic Web Conference (ESWC)*. 2016.

[30] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, J. E. L. Gayo, S. Kirrane, S. Neumaier, A. Polleres, et al. "Knowledge Graphs". In: *arXiv Preprint arXiv:2003.02320* (2020).

[31] H. Huang, C. Liu, and X. Zhou. "Approximating Query Answering on RDF Databases". In: *Journal of World Wide Web* (2012).

[32] C. A. Hurtado, A. Poulovassilis, and P. T. Wood. "Query Relaxation in RDF". In: *Journal on Data Semantics X* (2008).

[33] A. C. Junior, C. Debruyne, and D. O'Sullivan. "An Editor that Uses a Block Metaphor for Representing Semantic Mappings in Linked Data". In: *Extended Semantic Web Conference (ESWC), Poster&Demo*. 2018.

[34] G. M. Kapitsaki, F. Kramer, and N. D. Tselikas. "Automating the License Compatibility Process in Open Source Software With SPDX". In: *Journal of Systems and Software* (2017).

[35] Y. Khan, M. Saleem, A. Iqbal, M. Mehdi, A. Hogan, A.-C. N. Ngomo, S. Decker, and R. Sahay. "SAFE: Policy Aware SPARQL Query Federation Over RDF Data Cubes". In: *Semantic Web Applications and Tools for Life Sciences (SWAT4LS)*. 2014.

[36] S. Kirrane, A. Abdelrahman, A. Mileo, and S. Decker. "Secure Manipulation of Linked Data". In: *International Semantic Web Conference (ISWC)*. 2013.

[37] M. Lefrançois, A. Zimmermann, and N. Bakerally. "A SPARQL Extension for Generating RDF from Heterogeneous Formats". In: *Extended Semantic Web Conference (ESWC)*. 2017.

[38] M. Mesiti, P. Perlasca, and S. Valtolina. "On the Composition of Digital Licenses in Collaborative Environments". In: *Conference on Database and Expert Systems Applications (DEXA)*. 2013.

[39] F. Michel, C. Faron-Zucker, and J. Montagnat. "A Mapping-based Method to Query MongoDB Documents with SPARQL". In: *Database and Expert Systems Applications (DEXA)*. 2016.

[40] T. Minier, H. Skaf-Molli, and P. Molli. "SaGe: Web Preemption for Public SPARQL Query Services". In: *The World Wide Web Conference (WWW)*. 2019.

[41] B. Moreau, E. Desmontils, and P. Serrano-Alvarado. "Enrichissement de Données RDF Integrées à la Volée". In: *Atelier Web des Données (AWD) collocated with EGC*. 2019.

[42] B. Moreau and P. Serrano-Alvarado. "Assessing the Quality of RDF Mappings with EvaMap". In: *Extended Semantic Web Conference (ESWC), Poster&Demo*. 2020.

[43] B. Moreau and P. Serrano-Alvarado. "Ensuring License Compliance in Federated Query Processing". In: *Gestion de Données – Principes, Technologies et Applications (BDA)*. 2020.

[44] B. Moreau, P. Serrano-Alvarado, and E. Desmontils. "CaLi: A Lattice-Based Model for License Classifications". In: *Gestion de Données – Principes, Technologies et Applications (BDA)*. 2018.

[45] B. Moreau, P. Serrano-Alvarado, E. Desmontils, and D. Thoumas. "Querying non-RDF Datasets Using Triple Patterns". In: *International Semantic Web Conf (ISWC), Poster&Demo*. 2017.

[46] B. Moreau, P. Serrano-Alvarado, M. Perrin, and E. Desmontils. "A License-Based Search Engine". In: *Extended Semantic Web Conference (ESWC), Poster&Demo*. 2019.

[47] B. Moreau, P. Serrano-Alvarado, M. Perrin, and E. Desmontils. "Modéliser la Compatibilité Entre les Licences". In: *Journées francophones d'Ingénierie des Connaissances (IC)*. 2020.

[48] B. Moreau, P. Serrano-Alvarado, M. Perrin, and E. Desmontils. "Modelling the Compatibility of Licenses". In: *Extended Semantic Web Conference (ESWC)*. 2019.

[49] B. Moreau, N. Terpolilli, and P. Serrano-Alvarado. "A Semi-Automatic Tool for Linked Data Integration". In: *International Semantic Web Conference (ISWC), Poster&Demo*. 2019.

[50] B. Moreau, N. Terpolilli, and P. Serrano-Alvarado. "SemanticBot: Intégration Semi-Automatique de Données au Web des Données". In: *Atelier Web des Données (AWD) collocated with EGC*. 2020.

[51] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor. "Industry-scale Knowledge Graphs: Lessons and Challenges". In: *Queue* (2019).

[52] S. Oulmakhzoune, N. Cuppens-Boulahia, F. Cuppens, S. Morucci, M. Barhamgi, and D. Benslimane. "Privacy Query Rewriting Algorithm Instrumented by a Privacy-Aware Access Control Model". In: *Annals of Telecommunications* (2014).

[53] H. Pérez-Urbina, I. Horrocks, and B. Motik. "Efficient Query Answering for OWL 2". In: *International Semantic Web Conference (ISWC)*. 2009.

[54] B. Quilitz and U. Leser. "Querying Distributed RDF Data Sources with SPARQL". In: *Extended Semantic Web Conference (ESWC)*. 2008.

[55] N. A. Rakhmawati, M. Saleem, S. Lalithsena, and S. Decker. "QFed: Query Set for Federated SPARQL Query Benchmark". In: *International Conference on Information Integration and Web-based Applications & Services (iiWAS)*. 2014.

[56] P. Reddivari, T. Finin, A. Joshi, et al. "Policy-Based Access Control for an RDF Store". In: *Workshop Semantic Web for Collaborative Knowledge Acquisition (SWeCKa) collocated with IJCAI*. 2007.

[57] V. Rodríguez Doncel, A Gómez-Pérez, and S. Villata. "A Dataset of RDF Licenses". In: *Legal Knowledge and Information Systems Conference (ICLKIS)*. 2014.

[58] R. Rosati and A. Almatelli. "Improving Query Answering over DL-Lite Ontologies". In: *Principles of Knowledge Representation and Reasoning (KR)*. 2010.

[59] N. Sadeh, A. Acquisti, T. D. Breaux, L. F. Cranor, and et.al. "Towards Usable Privacy Policies: Semi-Automatically Extracting Data Practices from Websites' Privacy Policies". In: *Symposium on Usable Privacy and Security (SOUPS), Poster&Demo*. 2014.

[60] M. Saleem, A. Hasnain, and A.-C. N. Ngomo. "LargeRDFBench: a Billion Triples Benchmark for Sparql Endpoint Federation". In: *Journal of Semantic Web* (2018).

[61] M. Saleem and A.-C. N. Ngomo. "HIBISCuS: Hypergraph-Based Source Selection For SPARQL Endpoint Federation". In: *Extended Semantic Web Conference (ESWC)*. 2014.

[62] M. Saleem, A. Potocki, T. Soru, O. Hartig, and A. N. Ngomo. "CostFed: Cost-Based Query Optimization for SPARQL Endpoint Federation". In: *International Conference on Semantic Systems (SEMANTICS)*. 2018.

[63] R. S. Sandhu. "Lattice-Based Access Control Models". In: *Computer* (1993).

[64] O. Seneviratne, L. Kagal, and T. Berners-Lee. "Policy-Aware Content Reuse on the Web". In: *International Semantic Web Conference (ISWC)*. 2009.

[65] V. Soto-Mendoza, P. Serrano-Alvarado, E. Desmontils, and J. A. Garcia-Macias. "Policies Composition Based on Data Usage Context". In: *Workshop Consuming Linked Data (COLD) collocated with ISWC*. 2015.

[66] D.-E. Spanos, P. Stavrou, and N. Mitrou. "Bringing Relational Databases Into the Semantic Web: A Survey". In: *Journal of Semantic Web* (2012).

[67] J. Subercaze, C. Gravier, J. Chevalier, and F. Laforest. "Inferray: Fast In-memory RDF Inference". In: *VLDB Endowment* (2016).

[68] P.-Y. Vandenbussche, G. A. Atemezing, M. Poveda-Villalón, and B. Vatant. "Linked Open Vocabularies (LOV): a Gateway to Reusable Semantic Vocabularies on the Web". In: *Semantic Web* (2017).

[69] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, and P. Colpaert. "Triple Pattern Fragments: A Low-cost Knowledge Graph Interface for the Web". In: *Journal of Web Semantics* (2016).

[70] S. Villata and F. Gandon. "Licenses Compatibility and Composition in the Web of Data". In: *Workshop Consuming Linked Data (COLD) collocated with ISWC*. 2012.

[71] D. A. Wheeler. *The Free-Libre/Open Source Software (FLOSS) License Slide*. https://www.dwheeler.com/essays/floss-license-slide.pdf. 2007.

[72] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. "Quality Assessment for Linked Data: a Survey". In: *Journal of Semantic Web* (2016).

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

opendatasoft
Leading solution for data sharing

UNIVERSITÉ DE NANTES

Facilitating Reuse on the Web of Data

Keywords : Web of Data, Linked Data, RDF, SPARQL, licenses, usage control, federated query processing, query relaxation, data integration, RDF mappings

Abstract: The Web of Data is a web of interlinked datasets that can be queried and reused through federated query engines. To protect their datasets, data producers use licenses to specify their condition of reuse. But, choosing a compliant license is not easy. Licensing reuse of several licensed datasets must consider compatibility among licenses. To facilitate reuse, federated query engines should preserve license compliance. To do so, we focus on two problems (1) how to compute compatibility relations among licenses, and (2) how to ensure license compliance during federated query processing. To the first problem, we propose **CaLi**, a model that partially orders any set of licenses in terms of compatibility. To the second problem, we propose **FLiQue**, a license-aware federated query processing strategy. FLiQue uses CaLi to detect license compatibility conflicts and ensures that the result of a federated query preserves license compliance. Within the scope of this thesis, we also propose three approaches **ODMTP**, **EvaMap**, and the **SemanticBot** that aim to facilitate the integration of datasets to the Web of Data.

Faciliter la Réutilisation sur le Web des Données

Mots-clés : Web des données, données liées, RDF, SPARQL, licences, contrôle d'usage, traitement des requêtes fédérées, relâchement de requête, intégration de données, mappings RDF

Résumé : Le Web des données est un ensemble de données liées qui peuvent être interrogées et réutilisées à l'aide de moteurs de requêtes fédérées. Pour protéger les jeux de données, les licences renseignent leurs conditions d'utilisation. Cependant, choisir une licence conforme n'est pas toujours aisé. En effet, pour protéger la réutilisation de plusieurs jeux de données, il est nécessaire de prendre en considération la compatibilité entre leurs licences. Pour faciliter la réutilisation, les moteurs de requêtes fédérées devraient respecter les licences. Dans ce contexte, nous nous intéressons à deux problèmes (1) comment calculer la relation de compatibilité entre des licences, et (2) comment respecter les licences pendant le traitement de requêtes fédérées. Pour le premier problème, nous proposons **CaLi**, un modèle capable d'ordonner partiellement n'importe quel ensemble de licences selon leur compatibilité. Pour le second problème, nous proposons **FLiQue**, un moteur de requête fédéré respectant les licences. FLiQue utilise CaLi pour détecter les conflits de compatibilité entre licences et assure que le résultat d'une requête fédérée respecte les licences. Dans le cadre de cette thèse, nous proposons également trois approches **ODMTP**, **EvaMap** et le **SemanticBot** ayant pour objectif de faciliter l'intégration de données au web des données.