LDBC

Cooperative Project

FP7 – 317548

D4.4.1 Use Case Analysis and Classification of Choke Points

Coordinator: [Irini Fundulaki]

With contributions from: [Eva Daskalaki, Giorgos Flouris, Vassilis Papakonstantinou, Nikos Minadakis]

1st Quality Reviewer: Norbert Martinez (UPC)
2nd Quality Reviewer: Venelin Kotsev (ONTO)

Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	M12
Actual delivery date:	M12
Version:	1.0
Total number of pages:	69
Keywords:	Linked Open Data, RDF, RDFS, OWL reasoning, instance matching, ETL

Abstract

The purpose of this deliverable is to identify and describe the main challenges, named *choke points* faced by existing RDF engines in the tasks of *reasoning*, *instance matching* and *ETL processing*. The identification of a choke point involves understanding the most important challenges that current state-of-the-art systems face in their respective tasks, in order to be included as (hidden) challenges in benchmarks; the ultimate goal is to encourage systems to address these challenges, thus stimulating and encouraging technological progress. For our analysis, we used real-world datasets from the semantic publishing and the life sciences domains, as well as state-of-the-art tools in the respective areas of RDF reasoning, instance matching and ETL processing.

Executive Summary

This is the first deliverable for WP4 *Semantic Choke Point Analysis* of the LDBC project, which focuses on choke points related to three specific semantical tasks namely *reasoning*, *instance matching* and *ETL processing*. The purpose of this deliverable is to identify and discuss the main challenges, named *choke points*, faced by existing state-of-the-art systems for the aforementioned tasks; these choke points will be used for the upcoming benchmark designs due on M24 of the project (to be described in Deliverables D4.4.2, D4.4.3 and D4.4.4).

To identify the choke points for each task, we used the available state-of-the-art systems, as well as appropriate datasets from the Semantic Publishing Domain (as provided by the respective Task Force) and the life sciences domain. The identification of a choke point involves understanding the most important challenges that current systems face in their respective tasks, in order to be included as (hidden) challenges in the benchmarks that we will design; the ultimate goal is to encourage systems to address these challenges, thus stimulating and encouraging technological progress.

Chapter 4 discusses a set of choke points that are related to *reasoning*, which, in our context, refers to the identification and use of information that is not explicit in the dataset but can be inferred by the semantics of the used RDFS/OWL constructs.

The dataset used for our study on the reasoning choke points was the BBC dataset, which is a dataset from the Semantic Publishing Domain (provided by the respective Task Force) and is described in Section 2.1. The BBC ontologies were relatively simple (used mainly the RDFS `rdfs:subClassOf`, `rdfs:subPropertyOf` and `rdf:type` RDFS built-in properties), so in order to identify reasonably hard cases (choke points) we enriched them with additional OWL constructs from the OWL 2 RL [20] fragment. In addition, the data level of the BBC dataset was enriched by generating additional data using the generator described in Deliverable D2.2.2 [9]. The RDF engines used for running our reasoning choke points analysis was Virtuoso [40] and OWLIM [36], which are presented in Sections 3.1.1, 3.1.2 respectively.

The reasoning choke points are essentially queries which involve (either in their formulation, or in the accessed data) reasoning-intensive constructs, essentially forcing the query engine to perform RDFS/OWL reasoning in order to answer correctly the query. This is different from the objective of deliverable D2.2.1 [10], where reasoning was not considered and the emphasis was on choke points related mostly to query optimization of queries that did not involve reasoning (e.g., complex large join queries). In our analysis we measured whether the RDF engines correctly implemented the reasoning semantics of the various constructs (*conformance choke points*).

Chapter 5 discusses the choke points for the *instance matching* task. Instance matching (also referred to as record linkage [16], duplicate detection [7], entity resolution [1], and object identification in the context of databases [22]) refers to identifying instances that correspond to the same real world object.

For the instance matching choke points analysis, we used the OpenPHACTS datasets and golden standards (called linksets in the OpenPHACTS documentation) provided by the OpenPHACTS [35] FP7 European Project; a detailed description of this dataset and golden standards is provided in Section 2.2. Golden standards were put together by experts (curators) in the domain of pharmacology; the results of the tested systems were compared against such golden standards to evaluate their performance, i.e., how well they managed to match the input datasets. The systems we chose to evaluate in this deliverable were LIMES [21] and SILK [14, 15, 44], both of which are open source and publicly available. The systems are presented in Sections 3.2.1 and 3.2.2.

Our analysis of the instance matching choke points targeted on identifying cases where the tested systems perform poorly in the matching task (with respect to the golden standards). We ran tests under various parameterizations of the tested tools and measured the efficiency and effectiveness of these tools, as well as the quality of the results with respect to the golden standards. The metrics used for measuring the quality of the results were the standard metrics of *precision*, *recall* and *F-measure*.

In Chapter 6 we discuss the choke points for the *extract, transform* and *load* (abbreviated as *ETL*) processes. ETL techniques involve reshaping the data in various ways, e.g., by merging data that refer to the same resource together, or by making value-based changes, or by making structural changes.

As in the case of the instance matching task, we used the OpenPHACTS dataset for testing the different systems that perform the *extraction* and *transformation* of *relational data* to RDF. Loading of the result to the RDF database was made using the Virtuoso RDF engine. The tested systems for the ETL choke points were D2R [3], Triplify [38] (presented in Sections 3.3.2 and 3.3.1 respectively) and Virtuoso Views [43] .

Our choke points analysis consisted in identifying transformations that would stress the tested systems. In particular, the input to the process was the OpenPHACTS data, which were stored in a relational MySQL database. For the case of D2R and Triplify, the extract process consisted in reading the relational database, expressing the relational information using RDF triples and dumping the result to RDF files in the disk. The extraction process involved also some transformations, which were performed using appropriate mappings or mapping queries, depending on the system. The Virtuoso RDF engine was subsequently used to load the result (RDF data) to an RDF database (load). For the case of VirtuosoViews, the extract and transform process was performed within the Virtuoso database engine using adequate mapping files, and essentially consisted in generating non-materialized views in the Virtuoso database.

Document Information

IST Project Number	FP7 – 317548	Acronym	LDBC
Full Title	LDBC		
Project URL	http://www.ldbc.eu/		
Document URL	`http://www.ldbc.eu:8090/display/PROJECT/Deliverable+summary`		
EU Project Officer	Carola Carstens		

Deliverable	Number	D4.4.1	Title	Use Case Analysis and Classification of Choke Points
Work Package	Number	WP4	Title	Semantic Choke Point Analysis

Date of Delivery	Contractual	M12	Actual	M12
Status	version 1.0		final ☐	
Nature	Report (R) ☒ Prototype (P) ☐ Demonstrator (D) ☐ Other (O) ☐			
Dissemination Level	Public (PU) ☒ Restricted to group (RE) ☐ Restricted to programme (PP) ☐ Consortium (CO) ☐			

Authors (Partner)	Irini Fundulaki, Eva Daskalaki, Giorgos Flouris, Vassilis Papakonstantinou, Nikos Minadakis (FORTH)			
Responsible Author	Name	Irini Fundulaki	E-mail	fundul@ics.forth.gr
	Partner	FORTH	Phone	+302810391725

Abstract (for dissemination)	The purpose of this deliverable is to identify and describe the main challenges, named *choke points* faced by existing RDF engines in the tasks of *reasoning*, *instance matching* and *ETL processing*. The identification of a choke point involves understanding the most important challenges that current state-of-the-art systems face in their respective tasks, in order to be included as (hidden) challenges in benchmarks; the ultimate goal is to encourage systems to address these challenges, thus stimulating and encouraging technological progress. For our analysis, we used real-world datasets from the semantic publishing and the life sciences domains, as well as state-of-the-art tools in the respective areas of RDF reasoning, instance matching and ETL processing.
Keywords	Linked Open Data, RDF, RDFS, OWL reasoning, instance matching, ETL

Version Log			
Issue Date	Rev. No.	Author	Change
18/09/2013	0.1	Irini Fundulaki, Eva Daskalaki, Giorgos Flouris, Vassilis Papakonstantinou, Nikos Minadakis	First version
30/09/2013	1.0	Irini Fundulaki, Eva Daskalaki, Giorgos Flouris, Vassilis Papakonstantinou, Nikos Minadakis	Final version

Table of Contents

List of Figures

List of Tables

1 Introduction

This deliverable focuses on three semantical tasks, namely *reasoning*, *instance matching* and *ETL processing*. Reasoning refers to the identification and use of information that is not explicit in the dataset but can be inferred from the semantics of the used RDFS/OWL constructs. Instance matching (also referred to as record linkage [16], duplicate detection [7], entity resolution [1], and object identification in the context of databases [22]) refers to identifying instances that correspond to the same real world object. The *extract, transform* and *load* (abbreviated as *ETL*) processes involve reshaping the data in various ways, e.g., by merging data that refer to the same resource together, or by making value-based changes, or by making structural changes.

The purpose of this deliverable is to identify and discuss the main challenges, named *choke points*, faced by existing state-of-the-art systems for these tasks in order to be used for the upcoming benchmark designs. To identify these choke points, we used the available state-of-the-art systems, as well as appropriate datasets from the Semantic Publishing Domain (as provided by the respective Task Force in the context of LDBC) and the life sciences domain presented in Chapter 2. The identification of a choke point involves understanding the most important challenges that current systems face in their respective tasks, in order to be included as (hidden) challenges in the benchmarks that we will design; the ultimate goal is to encourage systems to address these challenges, thus stimulating and encouraging technological progress. We present the systems we used for our experiments in Chapter 3.

Reasoning choke points are discussed in Chapter 4; towards this purpose we used the BBC dataset, which was taken from the Semantic Publishing Domain described in Chapter 2, Section 2.1. The BBC ontologies were relatively simple (used mainly the RDFS `rdfs:subClassOf` `rdfs:subPropertyOf` and `rdf:type` RDFS built-in properties), so in order to identify reasonably hard cases (choke points) we enriched them with additional OWL constructs from the OWL 2 RL [20] fragment. The additional constructs are described in Section 4.2. In addition, the data level of the BBC dataset was enriched by generating additional data using the generator described in Deliverable D2.2.2 [9]. The BBC workload (also described briefly in Deliverable D2.2.2 [9]) was used as a guide for identifying choke points that are related to the actual, real-world usage patterns of the Semantic Publishing domain.

The RDF engines we used for running our reasoning choke points analysis was **Virtuoso** [40] and **OWLIM** [36], which are presented in Chapter 3, Sections 3.1.1, 3.1.2 respectively. Both engines represent the current state-of-the-art in RDFS/OWL reasoning. Another interesting characteristic is that Virtuoso uses backward reasoning, whereas OWLIM uses forward reasoning, which allowed us to evaluate our choke points against those two different approaches in implementing RDF engines capable of executing reasoning tasks.

The reasoning choke points are essentially queries which involve (either in their formulation, or in the accessed data) reasoning-intensive constructs, essentially forcing the query engine to perform RDFS/OWL reasoning in order to answer correctly the query. This is different from the objective of deliverable D2.2.1 [10], where reasoning was not considered and the emphasis was on choke points related mostly to query optimization of queries that did not involve reasoning (e.g., complex large join queries). In our analysis we measured, whether the RDF engines correctly implement the reasoning semantics of the various constructs (*conformance choke points*).

For the instance matching choke points analysis discussed in Chapter 5, we used the datasets provided by the **Open PHACTS** [35] FP7 European Project. **Open PHACTS** also provided a set of golden standards (called *linksets*) which were put together by experts (curators) in the domain of pharmacology. The results of the tested systems were compared against these golden standards to evaluate their performance, i.e., how well they managed to match the input datasets. A description of **Open PHACTS** and the related golden standards is provided in Chapter 2, Section 2.2.

For our tests we used two open source, publicly available instance matching systems, namely Limes [21] and Silk [14, 15, 44] both presented in Chapter 3, Sections 3.2.1 and 3.2.2. We also tried to obtain the systems that were evaluated using the Ontology Alignment Evaluation Initiative (OAEI) over the last three years. However, the versions available by OAEI were benchmark-specific, and thus unusable for our analysis.

Our analysis of the instance matching choke points targeted on identifying cases where the tested systems perform poorly in the matching task (with respect to the golden standards) under a set of evaluation criteria. In particular, we run tests under various parameterizations of the tested systems and measured their efficiency, as well as the quality of the results with respect to the golden standards.

The ETL choke point analysis task is discussed in Chapter 6; for this analysis we used the **Open PHACTS** dataset for the instance matching task. However, only two of the ontologies that comprise **Open PHACTS** were used for ETL, namely **ChEMBL** and **ChEBI**. The tested systems for the ETL choke points were **D2R** [3], **Triplify** [38] and **Virtuoso Views** [43] of **Virtuoso** discussed in Chapter 3, Sections 3.3.2 and 3.3.1. These systems were tested for the *extraction* and *transformation* of *relational data* to RDF. Loading of the result to the RDF database was made using the Virtuoso RDF engine (Section 3.1.1). Typically, ETL produces derived data (such as mapping triples) and also involves some reasoning.

Our ETL choke point analysis consisted in identifying transformations that would stress the tested systems. In particular, the input to the process was the **Open PHACTS** data, that was stored in a relational MySQL database. The extraction process involved also some transformations that consisted in expressing *exact match* links in terms of `skos:exactMatch` triples between **ChEMBL** and **ChEBI**, as well as aggregating the information found in each dataset by merging the triples that refer to the same URI. **Virtuoso** was subsequently used to store the obtained RDF triples (*loading phase*). In our analysis we measured the performance of these tools, as well as the richness of possible transformations that can be accomplished with each tool.

To follow this deliverable, some understanding of the RDF language [17] is required (a short introduction appears in Deliverable 1.1.1 [8]). Also, some understanding of the SPARQL query language [23] will be necessary (again, a short introduction can be found at Deliverable 1.1.1 [8]). Understanding RDFS and OWL semantics is also a prerequisite to follow the reasoning part, but this semantics will be briefly presented in Chapter 4. However, this presentation is not intended as a complete presentation of said languages; instead, we focus only on the semantics of the relevant constructs, to be used as a point of reference for the choke point analysis.

This deliverable is not intended to provide a full benchmark for the corresponding tasks. Instead, our aim is to identify the related choke points, upon which the subsequent benchmark definitions will be based. The benchmarks for the reasoning, instance matching and ETL tasks are planned to be reported in M24 of this project, as part of future Deliverables D4.4.2, D4.4.3 and D4.4.4 respectively.

2 Description of the Datasets

In this chapter, we describe the two datasets that we used for identifying the *choke points* for the tasks of *reasoning*, *instance matching* and *ETL*. In particular, for the *reasoning choke point analysis* we used the *ontologies* provided by BBC [27] enriched with manually added constructs from the OWL 2 RL fragment [20] as well as data automatically produced by the data generator described in Deliverable D.2.2.2 [9]. The OpenPHACTS datasets from the life science domain were used for the *data integration* tasks, namely *instance matching* and *ETL*.

We used the BBC dataset for the reasoning benchmark since (a) ontologies were made available to us, and (b) there is a real need for reasoning intensive tasks in the Semantic Publishing use case scenario. On the other hand, **Open PHACTS** datasets were more suitable for the instance matching and ETL tasks since this data is already used in a data integration context. More specifically, in the first task necessary input such as golden standards are available and can be used to evaluate the existing systems based on established metrics. **Open PHACTS** data were available as relational data, making this collection of datasets a very good use case for the ETL task, whereas this is not the case for the BBC use case scenario.

2.1 Semantic Publishing: The BBC Use Case

In this Section we present the *BBC ontologies* that we used for the *reasoning choke point analysis*. The ontologies provided by the *Semantic Publishing Task Force* were relatively simple, in terms of the reasoning constructs used (employed mainly the RDFS `rdfs:subClassOf`, `rdfs:subPropertyOf` and `rdf:type` RDFS built-in properties) and in order to test the reasoning choke points we enriched them with additional OWL constructs from the OWL 2 RL [20] fragment. Data was generated using the enriched BBC ontologies using the generator described in Deliverable D2.2.2 [9].

2.1.1 BBC Ontologies

In the figures of this Section, ontologies are represented as *node* and *edge labeled directed* graphs where *classes* are depicted by an *oval*, *class instances* by a *rhombus* and *properties* as *edges* between classes and instances, where the name of the property is the label of the edge. User defined and RDF properties (`rdf:type`, `rdfs:subClassOf`, `rdfs:subPropertyOf`) are depicted in the same manner. Cardinality constraints for properties are also recorded on the edges at the target class. The BBC ontologies that we used for the choke point analysis are discussed below.

CreativeWork 0.9: this ontology defines the *classes* and *properties* for *creative works*. Figure 2.1 shows the classes and properties for the creative works ontology. A creative work (also called a *journalistic asset*) is something created by the publisher's editorial team. It is not a representation of the item itself (which could be text, a photo, a video, an audio recording, etc), rather the *metadata* that describes it and its location (in an appropriate content management system). A creative work has a *title*, *shortTitle*, exactly one description, modification and creation date (properties `cwork:description`, `cwork:dateModified` and `cwork:dateCreated` respectively). Property `cwork:liveCoverage` indicates that the creative work is the live coverage of an event. It has zero or more audiences (property `cwork:audience`), instances of class `cwork:Audience`, a single format (property `cwork:primaryFormat`), instance of class `cwork:Format`. Creative works can be tagged (property *tag*) by *anything* (instance of class `core:Thing`), and is associated with exactly one category (property `cwork:category`) that can be any URI. Properties `cwork:about` and `cwork:mentions` are subproperties of property `cwork:tag`. Class `cwork:Audience` describes the kinds of audience for the story presented by a creative work; instance of this class are `cwork:NationalAudience`, `cwork:InternationalAudience`. Class `cwork:Format` collects all different kinds of formats of a creative work. These are `cwork:PictureGalleryFormat`, `cwork:AudioFormat`, `cwork:InteractiveFormat`, `cwork:VideoFormat`, `cwork:TextualFormat`.

A creative work has exactly one thumbnail (property `cwork:thumbnail`). Thumbnails have at most one type (`cwork:thumbnailType`), instance of class `cwork:ThumbnailType`: `cwork:StandardThumbnail`,

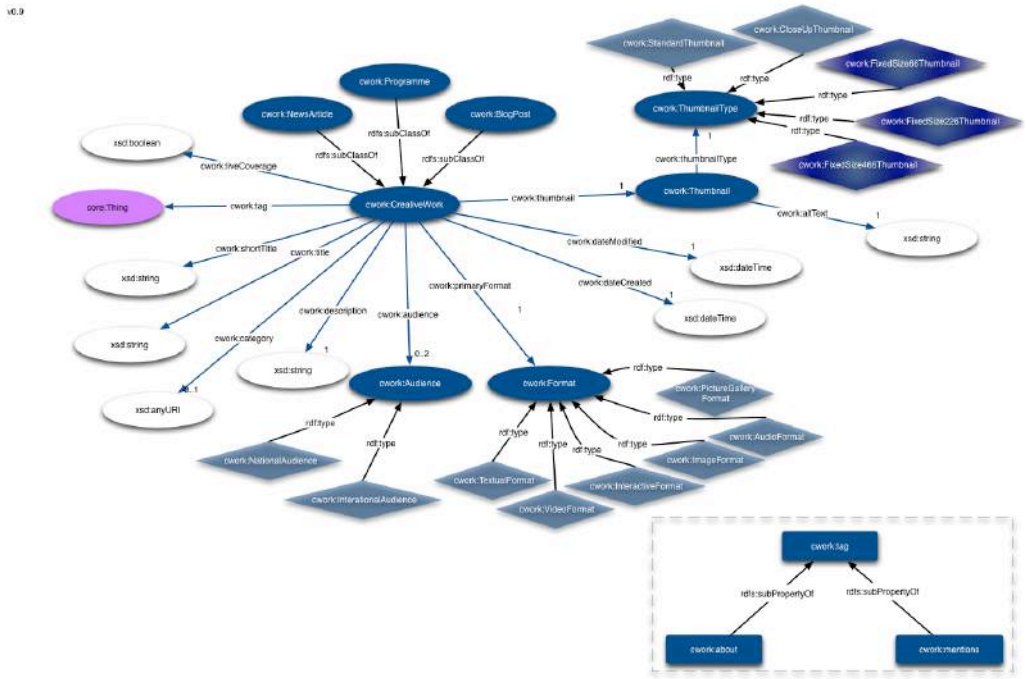

Figure 2.1: Creative Work 0.9

`cwork:CloseUpThumbnail`, `cwork:FixedSize66Thumbnail`, `cwork:FixedSize228Thumbnail` and `cwork:FixedSize466Thumbnail` and a text description (property `cwork:altText`).

There are different types of creative works: *news article* (class `cwork:NewsArticle`), a *programme* (class `cwork:Programme`) and a *blog post* (class `cwork:BlogPost`); these types are represented using the `rdfs:subClassOf` RDFS property and are subclasses of `cwork:CreativeWork` class.

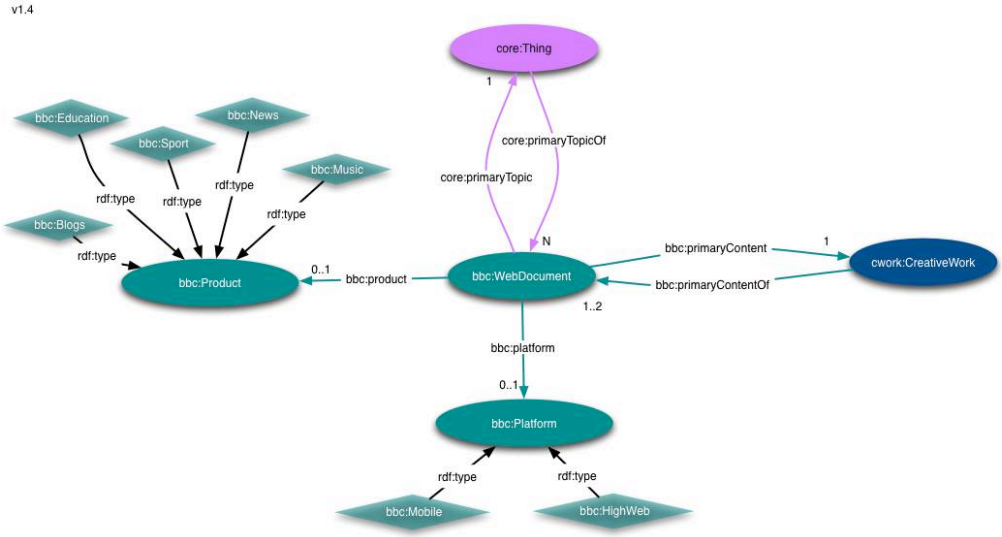

Figure 2.2: Company Ontology 1.4

Company 1.4: This ontology describes the relationship between the *web documents* produced by a content management system (class bbc:WebDocument), *BBC products* (class bbc:Product) and *creative works* (class cwork:CreativeWork). A BBC product can be a *blog* (bbc:Blogs), *education* (bbc:Education), *news* (bbc:News), *music* (bbc:Music) or *sport* (bbc:Sport), all instances of bbc:Product. A web document (instance of class bbc:WebDocument) has an associated *product* (property bbc:product). These documents have exactly one *primary topic* (property core:primaryTopic) that can be anything (instance of core:Thing). Such documents are presented in at most one platform such as bbc:Mobile, bbc:HighWeb, instances of class bbc:Platform. A creative work (instance of cwork:CreativeWork) can be the primary content (bbc:primaryContentOf)[1] at least one and at most two web documents. Finally, a BBC web document has an associated such product, the former being the *primary content* of a *creative work*. Figure 2.2 presents the classes and properties of the company ontology.

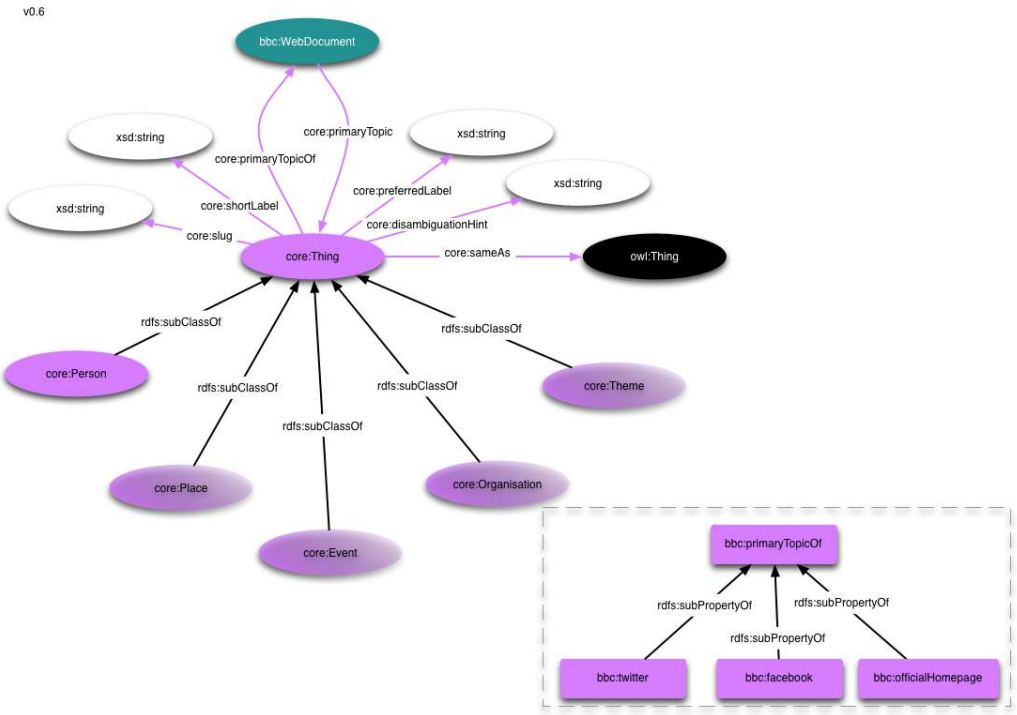

Figure 2.3: Core Concepts Ontology 0.6

Core Concepts 0.6 defines the main classes used in BBC datasets such as core:Person , core:Place, core:Event, core:Organization and core:Theme. These are all subclasses of the class core:Thing, defined as equivalent of class (using the owl:sameAs property) of owl:Thing, that is the "class of all individuals in the OWL world"[2]. An instance of class core:Thing has *short*, *preferred* labels (properties core:shortLabel core:preferredLabel), *disambiguation hint* (property core : disambiguationHint). Finally, each instance of class core:Thing has an associated URI slug (property core:slug) that is the fragment of a URI that uniquely identifies a resource within a domain. For instance, in the the case of Wikipedia the URI slug for the entry Stoat: http://en.wikipedia.org/wiki/Stoat is "Stoat" [24]. core:primaryTopicOf property is the *inverse* of core:primaryTopic. An instance of core:Thing can be the primary topic of more than one web documents. Properties bbc:twitter, bbc:facebook and bbc:officialHomepage are *subproperties* (modeled using the RDFS built-in rdfs:subPropertyOf) relationship) of core:primaryTopicOf that are

[1] bbc:primaryContent is the *inverse* of bbc:primaryContentOf: a web document can have as primary content exactly one a creative work.

[2] http://www.w3.org/TR/owl-guide/

used to indicate the different kind of web documents that have a "thing" as primary content. The main classes and properties of the core concepts ontology are shown in Figure 2.3.

CMS 1.2. is the ontology is used for interpreting locators into various specialized content management systems. A creative work and a BBC "thing" can have multiple such locators, that can be *sport-stats* (class `cms:Sports − Stats`), *music bootstrap* (class `cms:MusicBootstrap`), *iscript* (class `cms:iScript`) and *content api* (class `cms:ContentApi`). The CMS classes are all subclasses of `cms:Locator`. The CMS ontology used in BBC is shown in Figure 2.4.

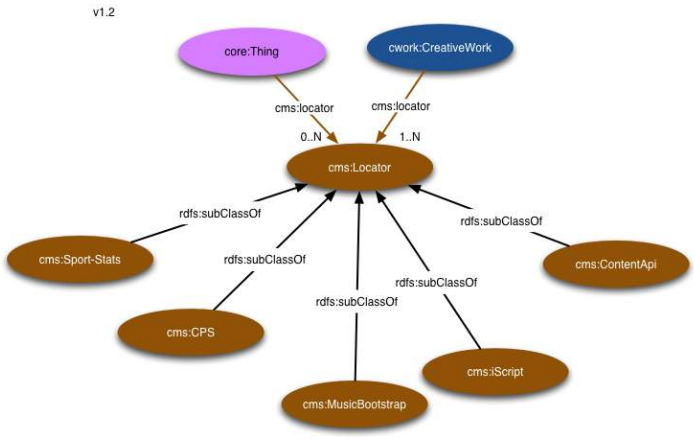

Figure 2.4: CMS Ontology 1.2

Person 0.2 describes information related to *persons*, instances of class `person:Person`, considered to be a *subclass of* class `bbc:Thing`. A person can have a *role*, a *first* and a *last name* (properties `person:role`, `person:firstName`, `person:lastName`). The person ontology is shown in Figure 2.5.

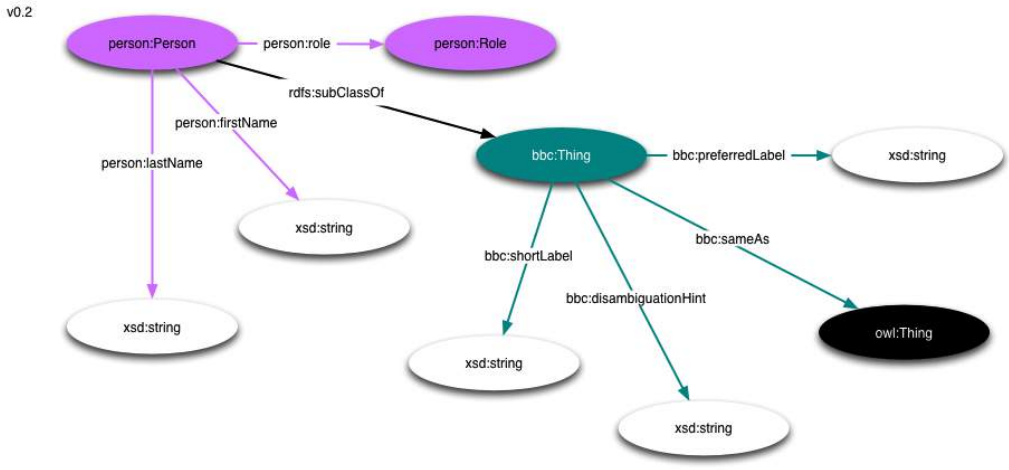

Figure 2.5: Person Ontology 0.2

Provenance 1.1 specifies the main concepts and properties used to describe versioning and change log information for the BBC datasets. The main class of the ontology is `provenance:Graph` that carries information about different versions of a dataset. The information that carries a provenance graph are the *owner* and *provider* of the dataset (properties `provenance:owner`, `provenance:provider`) that can be any web

resource. The provision date, the reason a dataset changed and its version, a canonical location and a previous hash version (properties `provenance:provided`, `provenance:changeReason`, `provenance:version`, `provenance:canonicalLocation`, `provenance:previousVersionHash`). The ontology is shown in Figure 2.6.

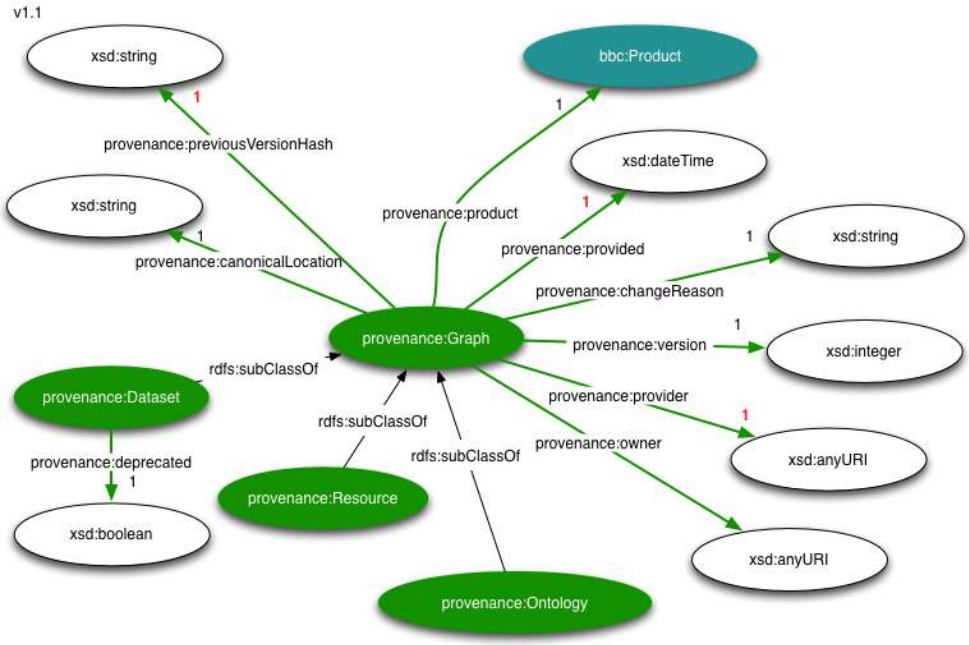

Figure 2.6: Provenance Ontology 1.1

Tagging 1.0: this ontology is used for connecting creative works with concepts from domain ontologies. The main concept is the `tagging:TagConcept` which is a subclass of `bbc:Thing`. A tag concept is associated with a *set of tags* (instances of class `tagging:TagSet`) and a locator of a content management system (instance of class `cms:Locator`). The ontology is shown in Figure 2.7.

Figure 2.8 presents an overview of the ontologies that comprise the BBC schema. The main classes of each of the ontologies are shown, in a color-coded fashion to indicate the ontology where they come from.

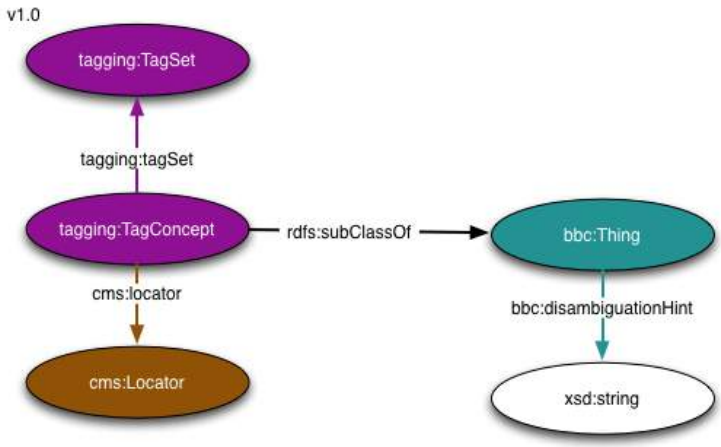

Figure 2.7: Tagging Ontology 1.0

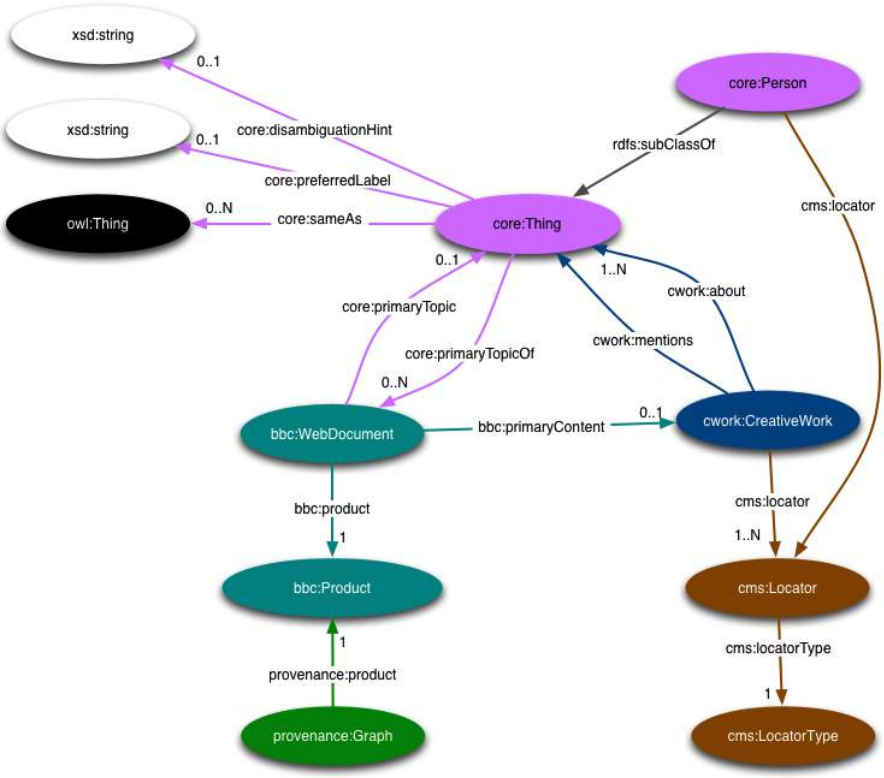

Figure 2.8: Overview Ontology 0.2

In addition to the aforementioned general ontologies, concepts from *domain ontologies* are used as tagging concepts: the *sports* ontology contains concepts for describing sports, competitions and sporting events, the *curriculum* ontology describes entities in academia and finally the *news* ontology describes the basic concepts that a *creative work* can be tagged with.

2.1.2 Statistics

Table 2.1 shows the RDFS and OWL constructs used in the BBC ontologies (core and domain-specific ontologies). The ontologies are relatively simple: they contain few classes, properties and small class and property hierarchies. More specifically, the class hierarchy has a maximum depth of *3* whereas the property hierarchy has a depth of *1*.

Classes and Properties

classes	*data type properties*	*object properties*
74	*88*	*28*

RDFS Constructs

`rdfs:subClassOf`	`rdfs:subPropertyOf`	`rdfs:domain`	`rdfs:range`
60	*17*	*105*	*115*

OWL Property and Class Constraints

`owl:oneOf`	`owl:TransitiveProperty`
8	*1*

Table 2.1: BBC Ontology Characteristics

2.2 Open PHACTS

Open PHACTS [35] (Open PHArmacological Concept Triple Store), is a freely available dataset in the pharmacological domain with data coming from various information sources. It provides tools and services to query this integrated data to support pharmacological research. It uses the state-of-the-art semantic web standards and technologies. It is a major public-private partnership involving organizations from major pharmaceutical companies, academia and small-medium enterprises [12]. The project is funded by the European Federation of Pharmaceutical Industries and Associations (EFPIA) and the European Union through the Innovative Medicines Initiative; it is scheduled to complete in early 2014.

The **Open PHACTS** platform is, at its core, a data integration platform. But instead of imposing a top-down view of data, as the usual approach of data warehousing, a more bottom-up view is considered; information coming from multiple providers is exposed through an adaptive integration. This bottom-up approach is facilitated by the adoption of open web-based data standards and is characterized as *semantic data interoperability*. In essence, data from all relevant sources is published in a semantically interoperable web enabled format, which is extended by community-adopted ontologies. This way, an increasing number of tools and services (including those provided by **Open PHACTS**) can take advantage of the published data layer, with full provenance allowing access to the underlying datasets [46].

The Open Pharmacological Space (OPS) of **Open PHACTS**, offers an open platform to deliver and sustain pharmacological/chemical data added from various sources (e.g., academia, publishers, small and medium-sized enterprises). Very well-known and established datasets have been added and linked to each other such as **ChemSpider** [29], **DrugBank** [31] **ChEMBL** [28], **UniProt** [39, 25], Gene Ontology (GO) [32] Database, Medical Subject Headings (MeSH) [33], Worldwide Protein Data Bank (PDB) [37]. We will briefly present here the datasets that we used in this deliverable, for the *instance matching* and *extract-transform-load* (ETL) tasks. In our experiments we consider RDF datasets (some of them have been extracted from relational databases).

ConceptWiki [30] is a universal open access repository of editable concepts. It focuses on the life sciences and on people working in the life sciences. It is currently being developed as part of the **Open PHACTS** project. It includes concepts from *Literature*, *Proteins*, *Chemicals* and other domains. The dataset can be freely downloaded and used as a thesaurus to identify references to the concepts from the aforementioned topics and in different information sources.

Literature concepts are related to scientific *authors* and their expertise, as well as scientific *publications* and their key subject concepts; these are assembled in a section of **ConceptWiki** called WikiPeople/WikiPublications. Much of this information was originally obtained through the PubMed/Medline[3] databases.

All concepts related to *Proteins*, *Genes* and their relationships, as well as to the species they are found in, are assembled in a section of **ConceptWiki** called WikiProteins/WikiGenes. Much of the information was obtained through the SwissProt/UniProt databases[4]. *Chemical* concepts represent small molecules and are all assembled in a section of **ConceptWiki** called WikiChemicals/WikiCompounds. This information, as well as many concepts related to properties of chemicals and their relationships were originally obtained from **ChemSpider** dataset that we describe below. The dataset contains data for more than 2,5 millions of proteins, genes and chemicals.

ChemSpider [29] is a free chemical structure database providing access to over 28 million structures, properties and associated information. The dataset builds on the collected sources by adding additional properties, related information and links back to the original data sources. It offers text and structure search to find compounds of interest and provides services to improve this data by curation and annotation and to integrate it with users' applications. By integrating and linking compounds from more than 400 data sources, **ChemSpider** enables researchers to discover the most comprehensive

[3] http://www.ncbi.nlm.nih.gov/pubmed
[4] http://web.expasy.org/docs/swiss-prot_guideline.html

view of freely available chemical data. **ChemSpider** integrates compound data with publications, provides publishing platform for the addition and preservation of data in order to make the data accessible and reusable. The dataset is owned by the Royal Society of Chemistry[5].

DrugBank database [31] is a bioinformatics and cheminformatics resource that combines detailed drug data (i.e., chemical, pharmacological and pharmaceutical) with comprehensive drug target information (i.e., sequence, structure, and pathway). The database contains 6811 drug entries, including 1528 FDA-approved small molecule drugs, 150 FDA-approved biotech drugs (protein/peptide), 87 nutraceuticals and 5080 experimental drugs. Additionally, 4294 non-redundant protein sequences (i.e., drug target/enzyme/transporter/carrier) are linked to these drug entries. Each entry contains more than 150 data fields with half of the information being devoted to drug/chemical data and the other half devoted to drug target or protein data.

ChEMBL [28] is a database of bioactive drug-like small molecules. It contains 2D structures, calculated properties and abstracted bioactivities. The data is abstracted and curated from the primary scientific literature, and covers a significant fraction of the structure activity relationship (SAR) and discovery of modern drugs. **ChEMBL** attempts to normalize the bioactivities into a uniform set of end-points and units, and also to tag the links between a molecular target and a published assay with a set of varying confidence levels. Additional data on clinical progress of compounds is being integrated into **ChEMBL**.

Open PHACTS Linksets As mentioned above, **Open PHACTS** aims to semantically bridge the pharmacological/chemical datasets. Specifically, Figure 2.9 presents the relationships between the datasets of the **Open PHACTS** suite, in other words the *linksets*; such a relationship is described by means of the `skos:exactMatch` property. The figure also shows the number of links each linkset has. The black arrows in Figure 2.9 show the direct relationships between the datasets while the dashed arrows show the *transitive relationship* between the datasets. That is, a concept x in a dataset d_1 is linked to a concept z in a dataset d_3 through a concept y in dataset d_2. These linksets will be used as the "golden standard" for the instance matching process discussed in Chapter 5.

[5] http://www.rsc.org/

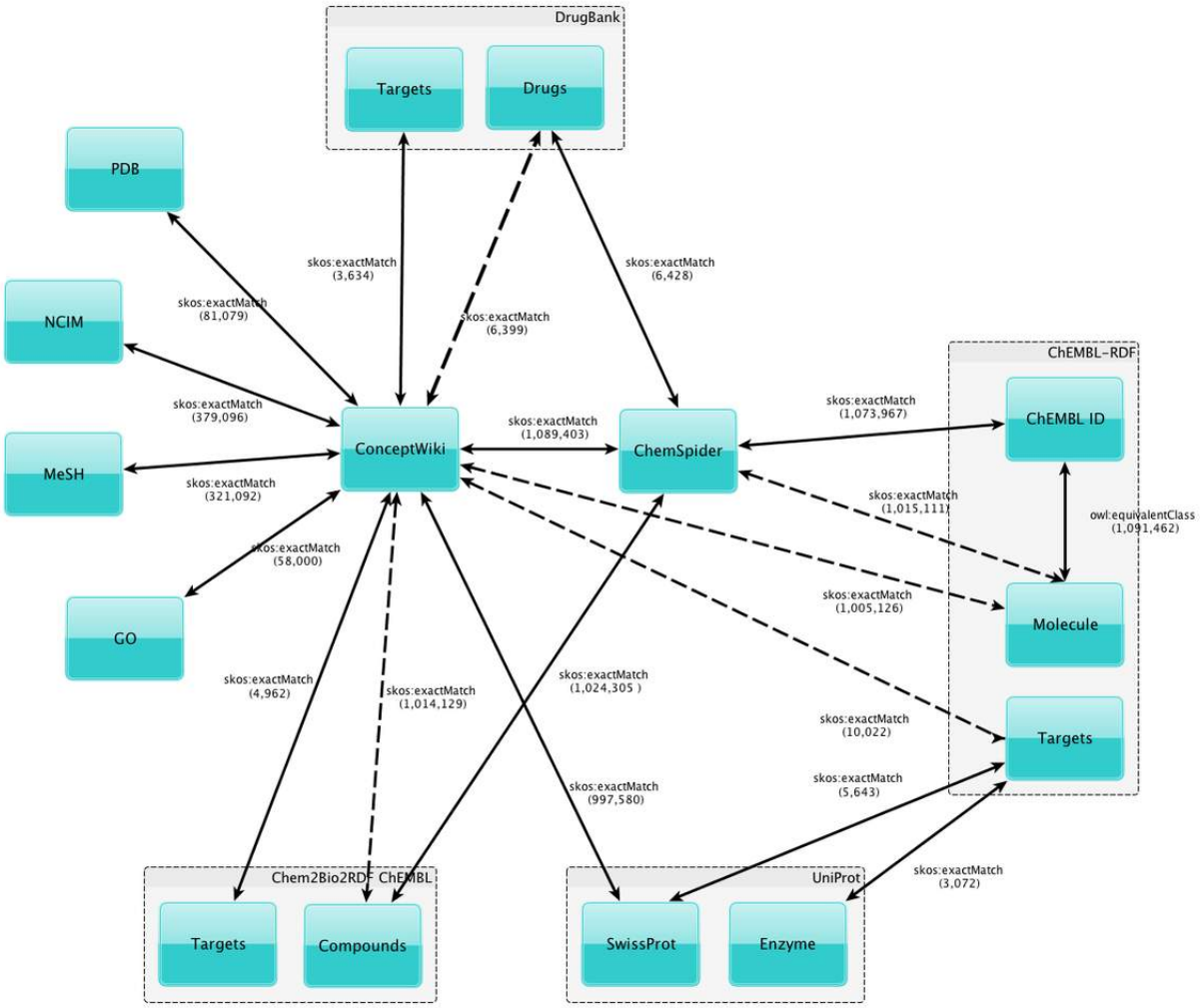

Figure 2.9: **Open PHACTS** Dataset Suite

3 Description of the Systems

In this chapter, we describe the systems that were used for identifying the choke points in the three processes (reasoning, instance matching and ETL). The considered systems are all state-of-the-art in their respective area, and were chosen for being representative of the currently available tools and approaches. The selected tools (described in subsequent sections of this chapter) are **Virtuoso**, **OWLIM**, Limes, Silk, **Triplify** and **D2R**. **Virtuoso** and **OWLIM** were used for the reasoning choke points, Limes and Silk were used for the instance matching choke points, whereas **Triplify** and **D2R** were used for the ETL choke points. As far as the ETL choke points are concerned, we also used a special feature of **Virtuoso**, namely **Virtuoso Views**.

All the above run in a server with 2 Intel® Xeon® CPU E5520 at 2.27GHz (a total of 8 cores/16 threads), 72GB ECC FB Ram and running CentOS release 6.4 x86_64. The system has 8 1TB SATA-2 in 7200RPM disks, which are configured as 4 RAID-1 mirrored disks under Areca ARC-1222 disk controller.

3.1 RDF Query Engines

3.1.1 Virtuoso

Virtuoso [40] is an innovative enterprise grade multi-model data server, developed by Openlink [34]. The main innovation of **Virtuoso** is that it delivers a platform-agnostic solution for data management, access and integration. It supports the management of various types of data, including relational, RDF, XML, text documents and others. This way, the users can employ the hybrid server architecture of **Virtuoso** to get access to all these different types of data.

One of the features of **Virtuoso** is the ability to create *Virtuoso RDF Views*. These views are non-materialized, and they are automatically updated when the source data changes. They essentially allow the up-to-date conversion of relational data into RDF and its exposure in a manner accessible through a SPARQL endpoint. This way, relational data is accessible as RDF data, in a transparent manner. *Virtuoso RDF views* allow the customization of the relational data, via a declarative Meta Schema Language that defines how relational data should be translated into RDF data, possibly after being transformed in various ways. These transformations will be exploited for testing our ETL choke points.

For the purposes of this deliverable, **Virtuoso** was used for the reasoning choke point analysis and for the ETL choke point analysis. We used the OpenLink Virtuoso Universal Server (Enterprise Edition), Version 07.00.3203, Compiled for Linux (i686-generic-linux-glibc212-64).

3.1.2 OWLIM

OWLIM [36] is a state-of-the-art RDF database management system developed by Ontotext[1]. It is a native RDF engine, implemented in Java, that supports the semantics of RDFS [2], OWL 2 RL and OWL 2 QL [20]; this makes it adequate for the reasoning choke points analysis that we undertake in this deliverable. According to their website, **OWLIM** offers great scalability features, and efficient loading and query evaluation times, even for large datasets. **OWLIM** is used in a large number of research projects and software tools. It comes in three different versions, namely OWLIM-Lite, OWLIM-SE and OWLIM-Enterprise.

For the experiments that we conducted in this deliverable we used OWLIM-SE Version 5.4.6287 with Sesame Version 2.6[2] and Tomcat Version 6[3]. We installed and run our experiments with **OWLIM** on the same server we used for **Virtuoso**.

[1] http://www.ontotext.com/
[2] http://www.openrdf.org/
[3] http://tomcat.apache.org/

3.2 Instance Matching Systems

3.2.1 LIMES

LIMES(LInked discovery framework for MEtric Spaces) [21] is a system used for the discovery of links between linked data sources. LIMES utilizes the triangle inequality in metric spaces to compute estimates of instance similarities. The system queries the SPARQL endpoints of the Knowledge Bases to return the selected concept instances and then matches those instances.

LIMES was developed in the AKSW group of the University of Leipzig[4] and is an open-source system. LIMES is configurable via a web interface, but can also be downloaded as a standalone tool for carrying out link discovery locally[5]. For our choke point analysis, we used the standalone tool, in particular version 0.6.

LIMES's strategy is to reduce the complexity of the matching algorithm, by reducing the number of comparisons of concepts' labels using "exemplars", which is a partitioning of the metric space.

3.2.2 SILK

The SILK-Linking Framework [44], like LIMES, is a tool for discovering links between entities within different data sources. It queries the knowledge bases by using SPARQL queries to return the specific required instances and their properties. It runs different distance metrics (*jaro* distance metrics, *jaroWinkler* distance metric, q-grams, etc.) to return the similarity of the instances. In order to be more efficient, SILK tool performs a pre-matching of the instances by indexing the values of the label properties. SILK is a publicly-available open source system that was developed in the context of the LOD2 project. For our analysis, we used SILK single machine version 2.5.4.

3.3 ETL Tools

3.3.1 Triplify

Triplify [38] is a tool for translating relational data into formats that are useful for the web of data, namely RDF, JSON, or Linked Data. Its main motivation is to help users expose relational data in semantical form for use in Semantic Web applications.

In the process of translating relational data to an, for instance, RDF representation, **Triplify** allows the user to define various types of transformations (e.g., string concatenation, or value grouping); this allows us to classify it as an ETL tool. These transformations are based on the use of database queries to create views, which can then be converted into the desired format. Triplify puts special emphasis on usability, so that users have an easy and simple interface to configure the way in which the translation takes place.

Triplify is implemented as a lightweight plugin for Web applications and is licensed under the terms of the GNU Lesser General Public License. For our analysis of ETL choke points, we used version 0.8[6] of **Triplify**.

3.3.2 D2R

D2R [3] is a tool that allows the publication of relational data on the Semantic Web using adequate semantical languages. **D2R** provides various useful features, such as the ability to navigate the contents of the database via a web interface, support for metadata and other annotations, as well as the use of resolvable URIs, so that Semantic Web browsers can follow links in the Linked Data cloud.

Note that, unlike **Triplify**, **D2R** does not perform a physical translation of the data. Instead, it allows the user to define custom *mappings*, which are then used to translate SPARQL queries into SQL queries on-the-fly, essentially allowing access to the relational data in a transparent manner. These mappings correspond

[4] http://aksw.org/About.html

[5] http://sourceforge.net/projects/limes/files/

[6] http://sourceforge.net/projects/triplify/files/latest/download?source=files

to the ETL transformations that we will use in our choke point analysis in Chapter 6. **D2R** (and the defined mappings) can be configured via various parameters, some of which are aimed at tuning the system's behavior towards improving performance. For our analysis of ETL choke points, we used version 0.8.1[7] of **D2R**.

[7] `https://github.com/downloads/d2rq/d2rq/d2rq-0.8.1.tar.gz`

4 Reasoning Choke Point Analysis

4.1 Outline of this Chapter

The purpose of the analysis presented in this chapter is to verify which OWL and RDFS constructs are handled by RDF engines **Virtuoso** and **OWLIM** in accordance to the owl 2 rl semantics [20]. In later versions of the reasoning benchmark (LDBC Deliverable D4.4.2 in M24) we will report on the *performance analysis* of queries that include reasoning intensive tasks and how these are (or not) handled efficiently by the tested RDF engines. These queries will reveal if the underlying optimizers take advantage of, for instance, cardinality constraints to optimize their plans, or property path axioms to eliminate unnecessary joins.

To check the constructs supported by the aforementioned systems, we define a set of *conformance* queries. The queries are formulated against the BBC dataset that contains a rich set of instances, and a set of ontologies that describe them (see Section 2.1 for more details). Because of the fact that the BBC ontologies are relatively simple in terms of the reasoning constructs that they use, they were manually enhanced with certain reasoning-intensive OWL constructs, as described in Section 4.2. Instances of the BBC dataset were produced using an automated data generator, described in Deliverable D2.2.2 [9].

A detailed description of the employed RDF engines **Virtuoso** and **OWLIM** can be found in Sections 3.1.1 and 3.1.2 respectively. Our objective in this exercise is to identify *choke points*, i.e., the most important challenges that current state-of-the-art systems face, in order to be included as (hidden) challenges in the reasoning benchmark that we will design; the ultimate goal is to encourage systems to address these challenges, thus stimulating and encouraging technological progress.

Section 4.2 presents the enhacements for the BBC ontologies. Before describing the choke points, we give a detailed description of the semantics of the RDFS and OWL constructs that are relevant for our analysis (Section 4.3). In Section 4.4 we present the conformance queries, and in Section 4.5 we discuss the results of our experiments for the **Virtuoso** and **OWLIM** engines.

4.2 Enhanced BBC Ontologies

In order to stress the reasoning capabilities of the reasoning engines used (**Virtuoso** [40] and **OWLIM** [36]) and identify choke points, we extended the BBC ontologies by *i)* defining additional properties and classes and *ii)* adding various owl 2 rl constructs.

4.2.1 Property Constraints

(E1) Asymmetric Property (`owl:AsymmetricProperty`**)** : if an *asymmetric property* property p is asserted between objects x and y (triple (x, p, y)) then p cannot be asserted between objects y and x (triple (y, p, x)).

(E2): Irreflexive Property (`owl:IrreflexiveProperty`**)** : if a property p is defined as *irreflexive*, triple (x, p, x) cannot exist in the dataset.

We defined property `ldbc:partOf` with `rdfs:domain` and `rdfs:range` class `bbc:WebDocument` to be both an *asymmetric* and *irreflexive* property.

```
ldbc:partOf rdf:type rdf:Property ;
            rdfs:domain bbc:WebDocument ;
            rdfs:range  bbc:WebDocument ;
            rdf:type owl:AsymmetricProperty ;
            rdf:type owl:IrreflexiveProperty .
```

(E3) Property Chain Axiom (owl:propertyChainAxiom) is a construct that allows one to define properties as a *composition* of others . For instance, one can define the property p as the composition of properties p_1 and p_2. In this case, if there exist triples (x, p_1, y), (y, p_2, z) then, according to the *Semantics of Axioms about Properties* [20] there exist triple (x, p, z).

We extended BBC ontologies with properties ldbc:cworkThumbnailAltText and ldbc:thumbnailType. The former is defined as a chain of properties cwork:thumbnail and cwork:altText, with rdfs:domain cwork:CreativeWork and rdfs:range xsd:string and the latter, as a chain of cwork:thumbnail and cwork:thumbnailType with range cwork:CreativeWork and range cwork:ThumbnailType.

```
ldbc:cworkThumbnailAltText rdf:type rdf:Property ;
                           rdfs:domain cwork:CreativeWork ;
                           rdfs:range xsd:string ;
                           owl:propertyChainAxiom (cwork:thumbnail cwork:altText) .
```

```
ldbc:thumbnailType
        rdf:type rdf:Property ;
        rdfs:domain cwork:CreativeWork ;
        rdfs:range cwork:ThumbnailType ;
        owl:propertyChainAxiom ( cwork:thumbnail cwork:thumbnailType) .
```

4.2.2 Class Constraints

E4 Union Of (owl:unionOf) : this class constraint allows one to define a class c as a *union* of existing classes or individuals. If for instance c is defined as a union of classes c_1 c_2, then an instance of class c_1 (triple $(x, \texttt{rdf:type}, c_1)$) is also an instance of class c (triple $(x, \texttt{rdf:type}, c)$). The same principle holds for properties.

To add this constraint in the BBC ontologies we defined class news:Theme as the union of classes ldbc:Sport, ldbc:Politics, ldbc:Music and ldbc:Art.

```
news:Theme
        owl:unionOf ( ldbc:Sport ldbc:Politics ldbc:Music ldbc:Art ) .
```

E5 Intersection Of (owl:intersectionOf) allows one to define a class c as an *intersection* of existing classes c_1, c_2. Essentially this means that c is a subclass of classes c_1 *and* c_2. To add this constraint in the BBC ontologies, we defined class news:Event to be an intersection of classes news:Person and news:Organization.

```
news:Event
        owl:intersectionOf ( news:Person news:Organisation ) .
```

E6 Disjoint Classes & Properties (owl:disjointWith) : if classes c_1 and c_2 are defined as *disjoint* (triple $(c_1, \texttt{owl:disjointWith}, c_2)$) then they cannot share common instances, that is, for instance, triples of the form $(x, \texttt{rdf:type}, c_1)$, $(x, \texttt{rdf:type}, c_2)$. The same holds for *disjoint properties* (constraint owl:propertyDisjointWith). That is, if triple $(p_1, \texttt{owl:propertyDisjointWith}, p_2)$ exists, then there cannot exist triples (x, p_1, y), (x, p_2, y) in the dataset. Disjointness between two classes/properties is generalized to multiple classes/properties using the owl:AllDisjointProperties and owl:AllDisjointClasses OWL constructs respectively.

We defined class cwork:Audience to be disjoint from class cwork:CreativeWork using the owl:disjointWith construct.

```
cwork:Audience owl:disjointWith cwork:CreativeWork .
```

We also defined pairwise disjointness between properties core:facebook, core:twitter, using the owl:propertyDisjointWith construct.

```
core:facebook
  owl:propertyDisjointWith core:twitter .

core:twitter
  owl:propertyDisjointWith core:facebook .
```

Construct owl:AllDisjointProperties was used to state properties core:tag, cwork:audience, cwork:primaryFormat and cwork:thumbnail as disjoint.

```
_:node1 a owl:AllDisjointProperties ;
        owl:members (cwork:tag cwork:audience cwork:primaryFormat cwork:thumbnail) .
```

Finally, owl:AllDisjointClasses construct is used to define that classes cwork:NewsItem, cwork:BlogPost and cwork:Programme are disjoint.

```
_:node2 a owl:AllDisjointClasses ;
        owl:members (cwork:NewsItem cwork:BlogPost cwork:Programme) .
```

4.2.3 Cardinality Constraints

E7 MinCardinality & MaxCardinality Constraints (owl:maxCardinality **and** owl:minCardinality**)** are used to make a property required, to allow only a specific number of values for that property, or to insist that a property must not occur. Constraints owl:maxCardinality and owl:minCardinality link a restriction class to a data value belonging to the value space of the XML Schema datatype xsd:nonNegativeInteger. owl:minCardinality and owl:maxCardinality constraints for values 0 and 1 are added in the ontology (these are supported by OWL Lite). More specifically, we specified the above constraints for properties ldbc:dateDestroyed and cwork:thumbnail.

```
ldbc:dateDestroyed rdf:type owl:DatatypeProperty ;
                   rdfs:domain cwork:CreativeWork ;
                   rdfs:range xsd:string ;
                   owl:maxCardinality "0"^^xsd:NonNegativeInteger .

cwork:thumbnail
      owl:cardinality  "1"^^xsd:nonNegativeInteger .
```

4.2.4 Key Constraints

(E8) Key (`owl:hasKey`**)** is used to uniquely identify instances based on their property values. To test whether the engines implement correctly this feature we added the following statement:

```
core:Thing
  owl:hasKey (core:shortLabel core:preferredLabel
              core:disambiguationHint core:primaryTopicOf) .
```

4.3 Semantics of RDFS and OWL Constructs

In this section, we describe the semantics of the RDFS and OWL constructs that appear in the enhanced BBC ontologies. These constructs will be used for the definition of the choke points in subsequent sections. Some of these constructs appear in the original BBC ontologies, whereas others were added during the enhancement phase to allow the exploration and definition of more sophisticated reasoning-intensive queries and choke points.

We assume that the reader is familiar with the principles of the RDF [17], RDFS [2] and OWL [18] languages that were presented in detail in Deliverable [8]. The brief presentation we give in Subsection 4.3.1 focuses only on an outline of the semantics of the relevant RDFS and OWL constructs in order to be used as a point of reference for the presentation of conformance queries and choke point analysis in the following sections. The reader is referred to Deliverable [8] for more details.

4.3.1 Summary of the RDFS and OWL Semantics

Both RDFS [2] and OWL [18] are based on RDF syntax and semantics, i.e., they adopt a triple-based representation. RDFS is used to add some limited semantical constructs (such as *subsumption*) to RDF, whereas OWL contains more sophisticated constructs applying on both classes and properties (such as transitivity or functionality requirements on properties, the ability to define new classes by *union*, *intersection* and *enumeration*.

Most of the OWL constructs were initially defined in 2004, in what is now known as OWL1 [18]. The semantics of some of the constructs were slightly refined in a subsequent version, OWL2 [45], introduced in 2012. Both OWL1 and OWL2 define several sublanguages that allow a different set of constructs, and thus adopt a different stance in the tradeoff between expressive power and reasoning complexity. Our presentation and analysis below focus on the constructs used in a specific sublanguage of OWL 2, namely OWL 2 RL [20], which is aimed at applications that require scalable reasoning without sacrificing too much expressive power.

Each construct is associated with specific semantics, which are formally encoded in the form of *if-then* rules. A rule means that if a dataset contains triples that match the triple pattern in the *"if"* part, then it should imply either (a) triples that match the triple pattern in the *"then"* part or (b) when the *"then"* part contains the keyword "FALSE", it means that an ontology containing the triples in the *"if"* part is inconsistent, i.e., it implies everything. The informal description of the involved OWL constructs, as well as the intuition behind their semantics is given in the subsections below.

4.3.2 Class and Property Subsumption

Class and *property subsumption* is the most basic, useful and frequent reasoning-intensive relationship that appears in semantic modeling. Subsumption is denoted using the RDFS constructs `rdfs:subClassOf` and `rdfs:subPropertyOf` for classes and properties respectively.

According to [2, 13] if a class c_1 is a subclass of c_2 (triple $(c_1, \texttt{rdfs:subClassOf}, c_2)$), then the instances of the former $(x, \texttt{rdf:type}, c_1)$ are also instances of the latter $(x, \texttt{rdf:type}, c_2)$. The same holds for subsumption between properties. Rules CAX-SCO and PRP-SPO1 in Table 4.1 describe these semantics.

According to the *Semantics of Schema Vocabulary* [20], class and property subsumption are *transitive* (see Rules SCM-SCO, SCM-SPO respectively in Table 4.1). More specifically, the existence of $(c_1, \texttt{rdfs:subClassOf}, c_2)$ and $(c_2, \texttt{rdfs:subClassOf}, c_3)$ in a dataset should cause the inference of $(c_1, \texttt{rdfs:subClassOf}, c_3)$.

	If	Then
CAX-SCO	$(?c_1, \texttt{rdfs:subClassOf}, ?c_2)$ $(?x, \texttt{rdf:type}, ?c1)$	$(?x, \texttt{rdf:type}, ?c2)$
PRP-SPO1	$(?p_1, \texttt{rdfs:subPropertyOf}, ?p_2)$ $(?x, ?p_1, ?y)$	$(?x, ?p_2, y)$
SCM-SCO	$(?c_1, \texttt{rdfs:subClassOf}, ?c_2)$ $(?c_2, \texttt{rdfs:subClassOf}, ?c_3)$	$(?c_1, \texttt{rdfs:subClassOf}, ?c_3)$
SCM-SPO	$(?p_1, \texttt{rdfs:subPropertyOf}, ?p_2)$ $(?p_2, \texttt{rdfs:subPropertyOf}, ?p_3)$	$(?p_1, \texttt{rdfs:subPropertyOf}, ?p_3)$

Table 4.1: Class and Property Subsumption

4.3.3 Property Domain and Range

The constructs `rdfs:domain` and `rdfs:range` are used to denote the *domain* and *range* of properties respectively. For example, $(p, \texttt{rdfs:domain}, c_1)$/$(p, \texttt{rdfs:range}, c_1)$ indicate that c_1 is the *domain/range* of property p. Rules SCM-RNG1/SCM-DOM1 shown in Table 4.2 state that if a property p has as *range/domain* a class c_1, then it has as *range/domain all superclasses* c_2 of c_1. Range and domain of properties is also inherited along the property subsumption hierarchy: rules SCM-RNG2/SCM-DOM2 (Table 4.2) state that if a property p_2 has as range/domain a class c, then its subproperty p_1 have also as range/domain class c. In addition whenever a subject s is connected via property p to some object o, s should be an instance of the domain of p, and o should be an instance of the range of p (rules PRP-DOM and PRP-RNG resp.).

	If	Then
SCM-RNG1	$(?p, \texttt{rdfs:range}, ?c_1)$ $(?c_1, \texttt{rdfs:subClassOf}, ?c_2)$	$(?p, \texttt{rdfs:range}, ?c_2)$
SCM-RNG2	$(?p_2, \texttt{rdfs:range}, ?c)$ $(?p_1, \texttt{rdfs:subPropertyOf}, ?p_2)$	$(?p_1, \texttt{rdfs:range}, ?c)$
SCM-DOM1	$(?p, \texttt{rdfs:domain}, ?c_1)$ $(?c_1, \texttt{rdfs:subClassOf}, ?c_2)$	$(?p, \texttt{rdfs:domain}, ?c_2)$
SCM-DOM2	$(?p_2, \texttt{rdfs:domain}, ?c)$ $(?p_1, \texttt{rdfs:subPropertyOf}, ?p_2)$	$(?p_1, \texttt{rdfs:domain}, ?c)$
PRP-DOM	$(?p, \texttt{rdfs:domain}, ?c)$ $(?x, ?p, ?y)$	$(?x, \texttt{rdf:type}, ?c)$
PRP-RNG	$(?p, \texttt{rdfs:range}, ?c)$ $(?x, ?p, ?y)$	$(?y, \texttt{rdf:type}, c)$

Table 4.2: Property Domain and Range

4.3.4 Union and Intersection of Classes

The owl:unionOf construct is used to construct a new class, that is the union of two (or more) other classes. Dually, the owl:intersectionOf construct is used to construct a new class that is the intersection of two (or more) other classes. As with all OWL constructs, the semantics of owl:unionOf are intentional, i.e., all instances that are *known* to be instances of either of c_1, c_2 will be also instances of their union, and vice-versa, i.e., known instances of the union will be instances of either c_1 or c_2 (or both). According to rules SCM-UNI and SCM-INT shown in Table 4.3, a class c defined as *union* (respectively *intersection*) of a set of existing classes c_1, c_2 ... c_n, then c is inferred as their *superclass* (respectively *subclass*).

SCM-INT	$(?c, \texttt{owl:intersectionOf}, ?x)$ LIST[$?x$, $?c_1$, ..., c_n]	$(?c, \texttt{rdfs:subClassOf}, ?c_1)$ $(?c, \texttt{rdfs:subClassOf}, ?c_2)$ ... $(?c, \texttt{rdfs:subClassOf}, ?c_n)$
SCM-UNI	$(?c, \texttt{owl:unionOf}, ?x)$ LIST[$?x$, $?c_1$, ..., c_n]	$(?c_1, \texttt{rdfs:subClassOf}, ?c)$ $(?c_1, \texttt{rdfs:subClassOf}, ?c)$ ... $(?c_n, \texttt{rdfs:subClassOf}, ?c)$

Table 4.3: Union and Intersection of Classes

4.3.5 Enumeration

The owl:oneOf construct is used to define a class via enumeration, i.e., by explicitly stating its instances. This implies that all such individuals are instances of the defined class, as shown in CLS-OO in Table 4.4.

	If	Then
CLS-OO	$(?c, \texttt{owl:oneOf}, ?x)$ LIST[$?x$, $?y_1$, ..., $?y_n$]	$(?y_1, \texttt{rdf:type}, ?c)$ ... $(?y_n, \texttt{rdf:type}, ?c)$

Table 4.4: Semantics of Enumerated Classes

4.3.6 Equality of Individuals

The uncontrolled nature of the Web of Data implies that there will be several cases where the *same resource* in the real world (e.g., a human being, an object or an idea) may be described using different URIs in different or even the same dataset. To address this problem, OWL2 proposes the use of the OWL construct owl:sameAs to connect *instances* that represent the same real-world entity[1]. Hence OWL construct owl:sameAs denotes *equality*. The opposite of owl:sameAs is owl:differentFrom, which explicitly states that two individuals are different, i.e., they correspond to a different real-world entity. Table 4.5 presents all rules that hold for owl:sameAs and owl:differentFrom constructs.

One observation is that whatever holds for one resource, holds for the other as well (rules EQ-REP-S, EQ-REP-P, EQ-REP-O). Obviously, a pair of individuals cannot be the same and different at the same time, thus the rule EQ-DIFF1. By its definition, owl:sameAs has the properties of equivalence relations, i.e., it is reflexive, symmetric and transitive. *Reflexivity* implies that $(x, \texttt{owl:sameAs}, x)$ for all resources x (rule EQ-REF). The relation being *symmetric* means that $(x, \texttt{owl:sameAs}, y)$ implies $(y, \texttt{owl:sameAs}, x)$ (rule EQ-SYM). Finally,

[1] Although owl:sameAs construct must be used only to denote that two URIs denote the same real world entity, it is sometimes used to express equality at the schema level.

	If	Then
EQ-REF	$(?s, ?p, ?o)$	$(?s, \texttt{owl:sameAs}, ?s)$ $(?p, \texttt{owl:sameAs}, ?p)$ $(?o, \texttt{owl:sameAs}, ?o)$
EQ-SYM	$(?x, \texttt{owl:sameAs}, ?y)$	$(?y, \texttt{owl:sameAs}, ?x)$
EQ-TRANS	$(?x, \texttt{owl:sameAs}, ?y)$ $(?y, \texttt{owl:sameAs}, ?z)$	$(?x, \texttt{owl:sameAs}, ?z)$
EQ-REP-S	$(?s, \texttt{owl:sameAs}, ?s')$ $(?s, ?p, ?o)$	$(?s', ?p, ?o)$
EQ-REP-P	$(?p, \texttt{owl:sameAs}, ?p')$ $(?s, ?p, ?o)$	$(?s, ?p', ?o)$
EQ-REP-O	$(?o, \texttt{owl:sameAs}, ?o')$ $(?s, ?p, ?o)$	$(?s, ?p, ?o')$
EQ-DIFF1	$(?x, \texttt{owl:sameAs}, ?y)$ $(?x, \texttt{owl:differentFrom}, ?y)$	FALSE

Table 4.5: Semantics of Equality

transitivity implies that from $(x, \texttt{owl:sameAs}, y)$ and $(y, \texttt{owl:sameAs}, z)$ we should infer $(x, \texttt{owl:sameAs}, z)$ (rule EQ-TRANS).

4.3.7 Inverse of Properties

The inverse property construct (`owl:inverseOf`) allows one to define a property as the inverse of another. For example, the property has_parent is the inverse of has_child. More formally, if p_1 is the inverse of p_2 then a triple of the form (x, p_1, y) implies (y, p_2, x). Note that when p_1 is the inverse of p_2, then p_2 is the inverse of p_1, so the above implication holds both ways. Rows PRP-INV1, PRP-INV2 of Table 4.6 expresses these implications.

	If	then
PRP-INV1	$(?p_1, \texttt{owl:inverseOf}, ?p_2)$ $(?x, ?p_1, ?y)$	$(?y, ?p_2, ?x)$
PRP-INV2	$(?p_1, \texttt{owl:inverseOf}, ?p_2)$ $(?x, ?p_2, ?y)$	$(?y, ?p_1, ?x)$

Table 4.6: Inverse Constraints

4.3.8 Constraints on Properties

Several OWL constructs are introduced to allow restricting the values that a property can have. In particular, there are constructs that restrict a property to be *inverse functional* (`owl:InverseFunctionalProperty`), *transitive* (`owl:TransitiveProperty`), *asymmetric* (`owl:AsymmetricProperty`) or *irreflexive* (`owl:IrreflexiveProperty`). The intuitive semantics of such constraints are given below. The formal semantics can be found at Table 4.7.

Inverse functional properties are useful to denote values that uniquely identify an entity. Note that, due to the fact that the semantics of OWL2 do not include the Unique Name Assumption (UNA), inverse functional

	If	then
PRP-IFP	$(?p, \texttt{rdf:type}, \texttt{owl:InverseFunctionalProperty})$ $(?x_1, ?p, ?y)$ $(?x_2, ?p, ?y)$	$(?x_1, \texttt{owl:sameAs}, ?x_2)$
PRP-TRP	$(?p, \texttt{rdf:type}, \texttt{owl:TransitiveProperty})$ $(?x, ?p, ?y)$ $(?y, ?p, ?z)$	$(?x, ?p, ?z)$
PRP-ASYP	$(?P, \texttt{rdf:type}, \texttt{owl:AsymmetricProperty})$ $(?x, ?p, ?y)$ $(?y, ?p, ?x)$	FALSE
PRP-IRP	$(?P, \texttt{rdf:type}, \texttt{owl:IrreflexiveProperty})$ $(?x, ?P, ?x)$	FALSE

Table 4.7: Constraints of Properties

properties should not be viewed as integrity constraints, because they cannot directly (by themselves) lead to contradictions. Instead, they force us to assume (infer) that certain individuals are the same, as indicated by rule PRP-IFP. If a property p is defined as transitive, then the existence of triples (x, p, y) and (y, p, z) should imply (x, p, z) (see also rule PRP-TRP). Transitive properties appear quite often in user properties (e.g., *partOf*), but also in built-in properties (e.g., *subsumption*).

If a property p is defined as *asymmetric*, then whenever x is connected to y via p, then y cannot be connected to x via p. More formally, if p is asymmetric, then the existence of (x, p, y) and (y, p, x) violates the correctness of the database (rule PRP-ASYP). Finally, if a property p is defined as *irreflexive* then, no individual can be connected to itself via p, i.e., a triple (x, p, x) cannot exist in the dataset (cf. rule PRP-IRP).

4.3.9 Keys of Classes

The `owl:hasKey` construct is used to specify a property (or a set of properties) as being the key for a given class (in the sense of primary keys, as defined in relational tables). Thus, the values of said properties uniquely identify a resource that is an instance of the class. For example, if property p is the key for class c, then the triples $(x, \texttt{rdf:type}, c)$, $(y, \texttt{rdf:type}, c)$, (x, p, z) and (y, p, z) imply $(x, \texttt{owl:sameAs}, y)$. A more general form of this statement is shown by rule PRP-KEY of Table 4.8.

	If	then
PRP-KEY	$(?c, \texttt{owl:hasKey}, ?u)$ LIST[$?u$, $?P_1$, …, $?P_2$] $(?x, \texttt{rdf:type}, ?c)$ $(?x, ?p_1, ?z_1)$ … $(?x, ?p_n, ?z_n)$ $(?y, \texttt{rdf:type}, ?c)$ $(?y, ?p_1, ?z_1)$ … $(?y, ?p_n, ?z_n)$	$(?x, \texttt{owl:sameAs}, ?y)$

Table 4.8: Keys

4.3.10 Property Chains

The construct owl:propertyChainAxiom allows one to define properties as a composition of others. As an example, the property $grandparent$ can be defined as the composition of $parent$ with itself. More formally, when a property p is defined as the composition of properties p_1 and p_2, then the triples (x, p_1, y), (y, p_2, z) imply (x, p, z) (rule PRP-SPO2, Table 4.9).

	If	then
PRP-SPO2	$(?p, \texttt{owl:propertyChainAxiom}, ?x)$ LIST[$?x$, $?p_1$, ..., p_n] $(?u_1, ?p_1, ?u_2)$ $(?u_2, ?p_2, ?u_3)$ ... $(?u_n, ?p_n, ?u_{n+1})$	$(?u_1, ?p, ?u_{n+1})$

Table 4.9: Property Chains

4.3.11 Disjoint Classes and Properties

Defining two classes c_1, c_2 as *disjoint* implies that they cannot share common instances. Disjointness is denoted using the owl:disjointWith construct. Disjointness between classes is generalized to multiple ones using the owl:AllDisjointClasses construct. The semantics of said constructs implemented by rules CAX-DW, CAX-ADC are shown in Table 4.10. Similar constructs owl:propertyDisjointWith, owl:AllDisjointProperties exist for specifying *disjoint properties*, i.e., properties that cannot share common instances. Rules PRP-ADP, PRP-PDW of Table 4.10 show some consequences of the semantics of the above constructs.

	If	Then
CAX-DW	$(?c_1, \texttt{owl:disjointWith}, ?c_2)$ $(?x, \texttt{rdf:type}, ?c_1)$ $(?x, \texttt{rdf:type}, ?c_2)$	FALSE
CAX-ADC	$(?x, \texttt{rdf:type}, \texttt{owl:AllDisjointClasses})$ $(?x, owl{:}members, ?y)$ LIST[$?y$, $?c_1$, ..., $?c_n$] $(?z, \texttt{rdf:type}, ?c_i)$ $(?z, \texttt{rdf:type}, ?c_j)$	FALSE
PRP-ADP	$(?x, \texttt{rdf:type}, \texttt{owl:AllDisjointProperties})$ $(?x, owl{:}members, ?y)$ LIST[$?y$, $?P_1$, $?P_2$, ... $?P_n$] $(?u, ?P_1, ?z)$ $(?u, ?P_2, ?z)$	FALSE
PRP-PDW	$(?P_1, \texttt{owl:propertyDisjointWith}, ?P_2)$ $(?x, ?P_1, ?y)$ $(?x, ?P_2, ?y)$	FALSE

Table 4.10: Disjoint Classes and Properties

4.3.12 Cardinalities

Cardinality constraints appear quite often in practice, and are used to allow a specific maximum or minimum number of values for any given property. Constraints `owl:maxCardinality` and `owl:minCardinality` link a restriction class to a data value belonging to the value space of the XML Schema datatype `xsd:nonNegativeInteger`.

The most common type of cardinality constraints are for values 0 and 1, which are simpler to handle; such cardinality constraints correspond to functional or required properties. Note that OWL 2 RL only supports cardinality constraints of this type (i.e., with values 0 or 1), so our analysis in this deliverable will focus on these types of cardinality constraints as well. Table 4.11 shows restrictions implied by `owl:maxCardinality`.

	If	Then
CLS-MAXC1	($?x$,`owl:maxCardinality` "0"`xsd:nonNegativeInteger`) $(?x, owl{:}onProperty, ?p)$ $(?u,$ `rdf:type`$, ?x)$ $(?u, ?p, ?y)$	FALSE
CLS-MAXC2	($?x$,`owl:maxCardinality` "1"`xsd:nonNegativeInteger`) $(?x, owl{:}onProperty, ?p)$ $(?u,$ `rdf:type`$, ?x)$ $(?u, ?p, ?y_1)$ $(?u, ?p, ?y_2)$	FALSE

Table 4.11: Cardinalities

4.4 Conformance Choke Points

In this Section we discuss the conformance choke points which determine whether a certain OWL construct is properly supported by the evaluated system, in the sense of performing sound and complete reasoning as specified by the semantics associated with the construct [20]. This is of crucial importance, because reasoning has only recently started being supported by query engines, and is not yet supported "by default" in all existing systems.

Support of OWL and RDFS constructs is tested using a set of *conformance queries* that were used to test the **Virtuoso** and **OWLIM** engines. Virtuoso is an engine that supports *backward* reasoning, whereas OWLIM is a *forward reasoner*, that is OWLIM *materializes the closure* of the dataset using the rules discussed in Section 4.3. A query is then evaluated on the closure of the dataset, that is, explicit and inferred triples (that appear in the consequence of the **"then"** part of the rules of Section 4.3) are treated in exactly the same manner. On the other hand, a backward reasoner computes the inferred triples during *query time*. Reasoning in Virtuoso is performed by specifying *rule_sets* that implement a small subset of the rules discussed in Section 4.3. Virtuoso recognizes `rdfs:subClassOf` and `rdfs:subPropertyOf`. `owl:sameAs` is considered for arbitrary subjects and objects if specially enabled by a pragma in the query (`owl:sameAs`='yes').

The conformance queries that we refer to here, are mainly SPARQL Ask queries and are expessed for the BBC ontologies discussed in Section 2.1. In particular, the SPARQL queries test whether the semantics described in Section 4.3 are implemented correctly by the underlying engine. The results of these tests will reveal the *limitations* or *choke points* of the tested systems regarding *reasoning constructs*; these are presented in Section 4.5. For each query we give also some explanation on the expected result and the intuition behind it.

4.4.1 Class and Property Subsumption

In order to check the conformance of the RDF engines regarding class and property subsumption we added a set of triples in our dataset shown in Table 4.12. Note that the BBC ontologies have a property hierarchy of depth 1 (see Section 2.1.2). In order to check rule SCM-SPO, we introduced a new property `things:pr-scm-spo-1#id`, as a subproperty of `cwork:about`. The conformance queries for class and property subsumption (`rdfs:subClassOf`, `rdfs:subPropertyOf`) are shown in Table 4.13; these queries implement the semantics of subsumption discussed in Section 4.3.2. Their expected result is *true*.

DESCRIPTION	UPDATE

CLASS SUBSUMPTION	`INSERT DATA {`
	`  things:cw-cax-sco-1#id`
	`  rdf:type cwork:BlogPost ;`
	`  cwork:title "Test for rdfs:subClassOf (cax-sco)" . }`

PROPERTY SUBSUMPTION	`INSERT DATA {`
	`  things:pr-scm-spo-1#id`
	`  rdf:type rdf:Property ;`
	`  rdfs:subPropertyOf cwork:about ;`
	`  cwork:title "Test for rdfs:subPropertyOf (scm-spo)" .`
	
	`  things:cw-cax-sco-1#id`
	`  things:pr-scm-spo-1#id tags:tag-cax-sco-spo-1#id ;`
	`  cwork:title "Test for rdfs:subPropertyOf (prp-spo1)" . }`

Table 4.12: Test Data for Class and Property Subsumption

RULE	DESCRIPTION	CONSTRUCTS	QUERY
CAX-SCO	Check that `things:cw-cax-sco-1#id` is an instance of `cwork:CreativeWork`	`rdf:type`, `rdfs:subClassOf`	`ASK { ?cw a cwork:CreativeWork .` `  FILTER(?cw = things:cw-cax-sco-1#id) }`

PRP-SPO1	Check that `things:pr-scm-spo-1#id` is an instance of `cwork:tag`	`rdfs:subPropertyOf`, `rdfs:subClassOf`, `rdf:type`	`ASK {`
			`  ?cw a cwork:CreativeWork .`
			`  ?cw cwork:tag tags:tag-cax-sco-spo-1#id .`
			`  FILTER(?cw = things:cw-cax-sco-1#id>) }`

SCM-SCO	Check if `cwork:BlogPost` is a subclass of `owl:Thing` (transitivity)	`rdfs:subClassOf`	`ASK { cwork:BlogPost rdfs:subClassOf ?c .` `  FILTER(?c = owl:Thing ) }`

SCM-SPO	Check that a property is a subproperty of `cwork:tag` (transitivity of `rdfs:subPropertyOf`)	`rdfs:subPropertyOf`	`ASK {`
			`  ?s rdfs:subPropertyOf ?o .`
			`  FILTER ( (?o = cwork:tag)  &&`
			`  (?s = things:pr-scm-spo-1#id ) ) }`

Table 4.13: Rules CAX-SCO, SCM-SCO, PRP-SPO1, SCM-SPO

4.4.2 Property Domain and Range

The conformance queries for testing the semantics of rdfs:domain and rdfs:range RDFS properties are shown in Table 4.15; these queries implement the semantics of the aforementioned properties as discussed in Section 4.3.3. These tests take into account the BBC ontologies presented in Section 2.1. Table 4.14 presents the triples we added to check PRP-DOM and PRP-RNG. The expected result of all these conformance queries is *true*.

DESCRIPTION	UPDATE

PROPERTY DOMAIN AND RANGE	`INSERT DATA { events:event-prp-dom-rng-1#id news:person org:org-prp-dom-rng-1#id }`
	
	`INSERT DATA { events:cw-prp-scm-dom2-1#id rdf:type rdf:Property ; rdfs:subPropertyOf cwork:about ; rdfs:comment "Testing rdfs:domain and rdfs:range (scm-dom2)" . }`

Table 4.14: Test Data for Property Domain and Range

RULE	DESCRIPTION	CONSTRUCTS	QUERY
SCM-RNG1	Check that news:person has as range core:Thing	rdfs:range, rdfs:subClassOf	`ASK {news:person rdfs:range core:Thing}`
SCM-RNG2	Check that cwork:about has as range owl:Thing	rdfs:range, rdfs:subClassOf	`ASK {cwork:about rdfs:range owl:Thing}`
SCM-DOM1	Check that news:person has as domain core:Event	rdfs:subClassOf, rdfs:domain	`ASK {news:person rdfs:domain core:Event}`
SCM-DOM2	Check that events:cw-prp-scm-dom2-1#id has as domain cwork:CreativeWork	rdfs:subClassOf, rdfs:domain	`ASK {events:cw-prp-scm-dom2-1#id rdfs:domain cwork:CreativeWork }`
SCM-DOM	Check that events:event-prp-dom-rng-1#id is an instance of core:Event	rdfs:domain, rdfs:subClassOf	`ASK { ?event rdf:type core:Event . ?event news:person ?org FILTER((?event=events:event-prp-dom-rng-1#id) && BOUND(?org)) }`
SCM-RNG	Check that org:org-prp-dom-rng-1#id is an instance of owl:Thing	rdfs:range, rdfs:subClassOf	`ASK { ?org rdf:type owl:Thing . ?event news:person ?org . FILTER ((?org = org:org-prp-dom-rng-1#id) && BOUND(?event)) }`

Table 4.15: Rules SCM-RNG1, SCM-RNG2, SCM-DOM1, SCM-DOM2, PRP-DOM, PRP-RNG

4.4.3 Union and Intersection of Classes

For the conformance choke points of owl:unionOf, owl:intersectionOf, we wrote Ask queries to check the implications of scm-uni, scm-int discussed in Section 4.3.4. For these queries, given in Table 4.17, we use the **Class Constraints** extensions to the BBC ontologies presented in Section 4.2.2. We tested the conformance of the engines using not only schema information but also instances (scm-uni, scm-int (1)). Table 4.16 presents the data used for our conformance queries. The expected result of all those queries is *true*.

Description	Update

Class Union and Intersection	INSERT DATA {
	things:sport-scm-uni-1#id
	rdf:type ldbc:Sport ;
	rdfs:comment "Testing owl:unionOf (scm-uni)" .
	}
	
	INSERT DATA {
	things:news-person-scm-int-1#id
	rdf:type news:Person ;
	rdf:type news:Organisation;
	rdfs:comment "Testing owl:intersectionOf (scm-int)" .
	}

Table 4.16: Test Data for Union and Intersection of Classes

Rule	Description	Constructs	Query

scm-uni (1)	Check whether an instance of ldbc:Sport is also an instance of class news:Theme	owl:unionOf	ASK {
			?t a news:Theme .
			FILTER(?t=things:sport-scm-uni-1#id>)
			}

scm-uni (2)	Check whether ldbc:Sport is a subclass of news:Theme	owl:unionOf	ASK {
			ldbc:Sport rdfs:subClassOf news:Theme
			}

scm-int (1)	Check whether an instance of classes news:Person and news:Organization is an instance of news:Event	owl:intersectionOf	ASK {
			?t a news:Event .
			FILTER (
			(?t=things:person-scm-int-1#id) &&
			(EXISTS{?t a news:Person}) &&
			(EXISTS{?t a news:Organisation}))
			}

scm-int (2)	Check whether news:Event is a subclassOf news:Person	owl:intersectionOf	ASK {
			news:Event rdfs:subClassOf news:Person
			}

Table 4.17: Rules scm-int, scm-uni

4.4.4 Conformance Choke Points for Enumeration

Testing the support of owl:oneOf is done by checking whether the implication CLS-OO discussed in Section 4.3.5 is correctly implemented. The expected result of the query shown in Table 4.18 is *true*. This query is formulated on the basis of the extensions to the BBC ontologies discussed in Section 4.2.

RULE	DESCRIPTION	CONSTRUCTS	QUERY
CLS-OO	Test that bbc:HighWeb and bbc:Mobile are instances of class bbc:Platform	owl:oneOf	ASK { bbc:HighWeb a ?plat . bbc:Mobile a ?plat0 . FILTER ((?plat = bbc:Platform) && (?plat0 = ?plat)) }

Table 4.18: Rule CLS-OO

4.4.5 Conformance Choke Points for Equality Tests

Equality tests involve checking the existence of conflicting statements related to the owl:sameAs and owl:differentFrom constructs. The conformance queries for owl:sameAs and owl:differentFrom are shown in Table 4.20. These SPARQL queries are formulated on the basis of rules EQ-REF, EQ-SYM, EQ-TRANS, EQ-REP-S, EQ-REP-P, EQ-REP-O and EQ-DIFF1 discussed in Section 4.3.6. These queries test whether two resources are specified as being the same and as being different, at the same time. In such a case, a conflict appears, according to the OWL Semantics, so a reasoner that implements such semantics should return *false*. The data we used for these conformance tests are shown in Table 4.19.

DESCRIPTION	UPDATE
EQUALITY CHECKS	INSERT DATA { things:cw-eq-ref-1#id bbc:primaryContentOf things:webdoc-eq-ref-1#id ; rdfs:comment "Testing owl:sameAs (eq-ref)". } INSERT DATA { things:cw-eq-sym-1#id bbc:primaryContentOf things:webdoc-eq-sym-1#id ; owl:sameAs things:cw-eq-sym-2#id ; rdfs:comment "Testing owl:sameAs (eq-sym)". } INSERT DATA { things:cw-eq-trans-1#id bbc:primaryContentOf things:webdoc-eq-trans-1#id ; owl:sameAs things:cw-eq-trans-2#id ; rdfs:comment "Testing owl:sameAs (eq-trans), (eq-rep-o), (eq-rep-p)". things:cw-eq-trans-2#id ldbc:referTo things:bbc-product-eq-trans-1#id ; rdfs:comment "Testing owl:sameAs (eq-trans)" ; owl:sameAs things:cw-eq-trans-3#id . things:cw-eq-trans-3#id rdf:type cwork:CreativeWork ; rdfs:comment "Testing owl:sameAs (eq-trans), (eq-rep-s)". } ldbc:refersTo owl:sameAs ldbc:referTo ; rdfs:comment "Testing owl:sameAs (eq-rep-p)" .

Table 4.19: Test Data for Equality Checks

RULE	DESCRIPTION	CONSTRUCTS	QUERY

EQ-REF	Test if EQ-REF holds for a specific resource and property	owl:sameAs	ASK {
			?s owl:sameAs ?s .
			?o owl:sameAs ?o .
			?p owl:sameAs ?p .
			?s ?p ?o .
			FILTER(
			(?s=things:cw-eq-ref-1#id) &&
			(?p = bbc:primaryContentOf)) .
			}

EQ-SYM	Test if owl:sameAs is symmetric	owl:sameAs	ASK {
			?cWork2 owl:sameAs ?cWork .
			FILTER ((?cWork2 = things:cw-eq-sym-2#id) &&
			(?cWork = things:cw-eq-sym-1#id))
			}

EQ-TRANS	Test if owl:sameAs is transitive	owl:sameAs	ASK {
			things:cw-eq-trans-1#id owl:sameAs ?o .
			FILTER(?o = things:cw-eq-trans-3#id)
			}

EQ-REP-S	Test inheritance of properties along owl:sameAs path (for subjects)	owl:sameAs	ASK {
			things:cw-eq-trans-3#id
			bbc:primaryContentOf ?o .
			FILTER (?o = things:webdoc-eq-trans-1#id)
			}

EQ-REP-O	Test inheritance of properties along owl:sameAs path (for objects)	owl:sameAs,
		owl:inverseOf
			ASK {
			things:webdoc-eq-trans-1#id
			bbc:primaryContent ?o
			FILTER (?o = things:cw-eq-trans-2#id) .
			}

EQ-REP-P	Test inheritance of subjects and objects along the owl:sameAs path (for properties)	owl:sameAs	ASK {
			things:cw-eq-trans-1#id
			ldbc:refersTo ?o .
			FILTER (?o = things:bbc-product-eq-trans-1#id)
			}

EQ-DIFF1	Test satisfiability for owl:sameAs and owl:differentFrom	owl:sameAs,
		owl:differentFrom
			ASK {
			things:cw-eq-trans-1#id owl:sameAs
			things:cw-eq-trans-4#id ;
			owl:differentFrom
			things:cw-eq-trans-4#id
			}

Table 4.20: Rules EQ-REF, EQ-SYM, EQ-TRANS, EQ-REP-S, EQ-REP-O, EQ-REP-P, EQ-DIFF1

4.4.6 Conformance Choke Points for Inverse of Properties

The construction of inverse properties (using the `owl:inverseOf` construct) will be based as usual on checking whether the corresponding implications (PRP-INV1, PRP-INV2 of Section 4.3.7) are properly supported by the query engine. The INSERT statement in Table 4.21 inserts triples for an inverse property (instance of `owl:inverseOf`) and the ASK query checks that the instances reached by the inverse property are the same.

RULE	QUERY

PRP-INV1	`INSERT DATA {`
	`  things:cw-prp-inv1#id a cwork:BlogPost ;`
	`  cwork:title "Test for owl:inverseOf (prp-inv1)" ;`
	`  bbc:primaryContentOf things:cw-prp-inv1-webdocument-1 .`
	`}`
	
	`ASK {`
	`  ?cWork a cwork:BlogPost .`
	`  ?cWork bbc:primaryContentOf ?pco .`
	`  ?pcoInv bbc:primaryContent ?cWork .`
	`  FILTER(?pco = ?pcoInv) .`
	`  FILTER(?cWork = things:cw-prp-inv1#id>) .`
	`}`

Table 4.21: Rule PRP-INV1

4.4.7 Conformance Choke Points for Constraints on Properties

The various constraints on properties (inverse functional, transitive, asymmetric, irreflexive) using the corresponding semantics are presented in Section 4.3.8. If the engine supports checking the constraints on properties, then the INSERT statements are presented in Table 4.23 for PRP-ASYP and PRP-SYM should fail. On the other hand, the ASK queries for PRP-IFP and PRP-TRP should return *true*.

RULE	QUERY

PRP-KEY	`INSERT DATA {`
	`  things:cw-prp-key-1-constraint#id`
	`  rdf:type cwork:BlogPost ;`
	`  core:shortLabel "label1" ;`
	`  core:preferredLabel "label2" ;`
	`  core:disambiguationHint "label3" ;`
	`  core:primaryTopicOf things:cw-prp-key-webdocument-1 .`
	
	`  things:cw-prp-key-2-constraint#id`
	`  rdf:type cwork:CreativeWork;`
	`  core:shortLabel "label1" ;`
	`  core:preferredLabel "label2" ;`
	`  core:disambiguationHint "label3" ;`
	`  core:primaryTopicOf things:cw-prp-key-webdocument-1 .`
	`}`
	
	`ASK {`
	` things:cw-prp-key-1-constraint#id`
	`  owl:sameAs things:cw-prp-key-2-constraint#id`
	`}`

Table 4.22: Rule PRP-KEY

RULE	QUERY
PRP-IFP	INSERT DATA { things:thing-prp-ifp-1#id core:disambiguationHint "hint for things:thing-prp-ifp-1-2#id"; rdfs:comment "Testing owl:InverseFunctionalProperty" ; things:thing-prp-ifp-2#id core:disambiguationHint "hint for things:thing-prp-ifp-1-2#id"; rdfs:comment "Testing owl:InverseFunctionalProperty" . } ASK { things:thing-prp-ifp-1#id owl:sameAs ?o . FILTER (?o = things:thing-prp-ifp-2#id) }
PRP-ASYP	INSERT DATA { things:cw-prp-asyp-constraint-1#id cwork:title "Constraint Violation test for owl:AsymmetricProperty" ; bbc:primaryContentOf things:prp-asyp-webdocument-1#id . things:cw-prp-asyp-constraint-2#id bbc:primaryContentOf things:prp-asyp-webdocument-2#id . things:prp-asyp-webdocument-1#id ldbc:partOf things:prp-asyp-webdocument-2#id . things:prp-asyp-webdocument#2 ldbc:partOf things:prp-asyp-webdocument-1#id . }
PRP-IRP	INSERT DATA { things:cw-prp-irp-constraint#id cwork:title "Constraint Violation test for owl:IrreflexiveProperty" ; bbc:primaryContentOf things:cw-prp-irp-webdocument-1 . things:cw-prp-irp-webdocument-1 ldbc:partOf things:cw-prp-irp-webdocument-1 . }
PRP-TRP	INSERT DATA { sports:sportsdiscipline-prp-trp-1#id rdfs:comment "Testing owl:TransitiveProperty (prp-trp)" ; sport:subDiscipline sports:sportsdiscipline-eq-trans-2#id . sports:sportsdiscipline-prp-trp-2#id sport:subDiscipline sports:sportsdiscipline-prp-trp-3#id . sports:sportsdiscipline-prp-trp-3#id sport:subDiscipline sports:sportsdiscipline-prp-trp-4#id . sports:sportsdiscipline-prp-trp-4#id rdf:type sport:SportsDiscipline . } ASK { ?s sport:subDiscipline ?o . FILTER ((?s = sports:sportsdiscipline-prp-trp-1#id) && (?o = sports:sportsdiscipline-prp-trp-4#id)) }

Table 4.23: Rules PRP-IFP, PRP-ASYP, PRP-IRP, PRP-TRP

4.4.8 Conformance Choke Points for Class Keys

Support for the owl:hasKey construct (as defined by PRP-KEY in Section 4.3.9) is checked using the query shown in Table 4.22. According to the key constraints that we have defined in the ontology, the INSERT statements should fail to execute. Otherwise, the engine should deduce that the two inserted instances are

connected using a owl:sameAs link.

4.4.9 Conformance Choke Points for Property Chains

The conformance queries related to the proper support of property chains (which allow the composition of properties) discussed in Section 4.3.10 is implemented using query shown in Table 4.24. The INSERT statement inserts triples for properties that are part of the property chain as defined in the extensions for the BBC ontologies. The ASK query requests that the same resource is reached either by an explicit join before the respective triples, or by the property chain. The first ASK query checks whether PRP-SPO2 holds for owl:DatatypeProperty properties and the second for owl:ObjectProperty.

RULE	QUERY

PRP-SPO2	INSERT DATA {
	  things:cw-prp-spo2#id
	  rdf:type cwork:CreativeWork ;
	  cwork:title "Test for owl:propertyChainAxiom - owl:DataTypeProperty" ;
	  cwork:thumbnail thumbnail:cw-prp-spo2-thumbnail .
	
	  thumbnail:cw-prp-spo2-thumbnail
	  cwork:altText "AltText for CW : things:cw-prp-spo2#id" .
	}
	
	ASK {
	  ?cWork a cwork:CreativeWork .
	  ?cWork cwork:thumbnail ?thumbnail1 .
	  ?thumbnail1 cwork:altText ?thumbnailAltText1 .
	  ?cWork ldbc:cworkThumbnailAltText ?thumbnailAltText2 .
	  FILTER (?thumbnailAltText1 = ?thumbnailAltText2) .
	  FILTER (?cWork = things:cw-prp-spo2#id) .
	}

PRP-SPO2	INSERT DATA {
	  things:cw-prp-spo2-2#id
	  rdf:type cwork:NewsItem ;
	  cwork:title "Test for owl:propertyChainAxiom - owl:ObjectProperty" ;
	  cwork:thumbnail thumbnail:cw-prp-spo2-2-thumbnail . }
	
	ASK {
	  ?cWork a cwork:NewsItem .
	  ?cWork cwork:thumbnail ?thumbnail1 .
	  ?thumbnail1 cwork:thumbnailType ?thumbnailType1 .
	  ?cWork ldbc:thumbnailType ?thumbnailType2 .
	  FILTER (?thumbnailType1 = ?thumbnailType2) .
	  FILTER (?cWork = things:cw-prp-spo2-2#id> .
	}

Table 4.24: Rule PRP-SPO2

4.4.10 Conformance Choke Points for Disjoint Classes and Properties

Queries in Table 4.25 implement the semantics of CAX-DW, CAX-ADC, PRP-ADP and PRP-PDW are presented in Section 4.3.11. These INSERT queries add triples that violate the disjointness of classes and properties, and the engine should fail to perform the updates.

RULE	QUERY
PRP-PDW	INSERT DATA { things:cw-prp-pdw-constraint#id cwork:title "Constraint Violation test for owl:propertyDisjointWith" ; rdf:type cwork:CreativeWork ; core:facebook things:cw-prp-pdw-webdocument-1 ; core:twitter things:cw-prp-pdw-webdocument-1 .}
PRP-ADP	INSERT DATA { things:cw-prp-adp-constraint#id a cwork:NewsItem ; cwork:title "Constraint Violation test for owl:AllDisjointProperties" ; cwork:about things:value-1#id; cwork:primaryFormat things:value-1#id; cwork:audience things:value-1#id; cwork:thumbnail things:value-1#id . }
CAX-DW	INSERT DATA { things:cw-cax-dw-constraint#id a cwork:Audience ; cwork:title "Constraint Violation test for owl:disjointWith" . things:cw-cax-dw-constraint#id a cwork:CreativeWork ; }
CAX-ADC	INSERT DATA { things:cw-cax-adc-constraint#id a cwork:NewsItem ; cwork:title "Constraint Violation test for owl:AllDisjointClasses" . things:cw-cax-adc-constraint#id a cwork:BlogPost . things:cw-cax-adc-constraint#id a cwork:Programme . }

Table 4.25: Rules PRP-PDW, PRP-ADP, CAX-DW, CAX-ADC

4.4.11 Conformance Choke Points for Cardinalities

The queries shown in Table 4.26 implement rule CLS-MAXC1 and CLS-MAXC2 discussed in Section 4.3.12. Note that to test the latter, we insert two `cwork:thumbnail` triples for the same instance. An engine that performs consistency checking would either reject this update operation, or infer that the object values for these triples are the same (i.e., checks the existence of a `owl:sameAs` triple). This is the role of the ASK query showin in Table 4.26.

RULE	QUERY

CLS-MAXC1	INSERT DATA {
	things:cw-cls-maxc1-constraint#id
	rdf:type cwork:NewsItem ;
	ldbc:dateDestroyed "1.1.1990" .
	}

CLS-MAXC2	INSERT DATA {
	things:cw-cls-maxc2-constraint#cwork-id1
	rdf:type cwork:BlogPost ;
	cwork:thumbnail things:cw-cls-maxc2-constraint#thumbnail-id1 .
	cwork:thumbnail things:cw-cls-maxc2-constraint#thumbnail-id2 . }
	
	ASK {
	things:cw-cls-maxc2-constraint#thumbnail-id1
	owl:sameAs things:cw-cls-maxc2-constraint#thumbnail-id2 . }

Table 4.26: Rules CLS-MAXC1, CLS-MAXC2

4.5 Conformance Choke Points Results

In this section we discuss the results for the *conformance queries* discussed in Section 4.4. For our experiments we used the **OWLIM** and **Virtuoso** engines. The **OWLIM** repository we used to run our performance tests was configured with (i) *inconsistency checks* set to *true*, and (ii) the OWL-2RL OPTIMIZED rule set. Recall that **OWLIM** uses *forward reasoning* to compute the *closure of the dataset*; the queries (reasoning or not) are then evaluated on the pre-computed set of triples. On the other hand, **Virtuoso** uses *backward reasoning*: the inferred triples are computed at *query time* when reasoning is involved. Towards this purpose, Virtuoso uses special purpose *rule-sets* and *options*. To answer queries involving subsumption, preamble `define:input inference 'graph_name'` should be incorporated in the query. Option `define input:same-as 'yes'` is used to handle queries that involve `owl:sameAs` links and option `option (transitive, t_in(?x), t_out(?syn), t_distinct, t_min(0))` to trigger *transitivity*. Without the incorporation of the above options, **Virtuoso** is unable to handle any form of reasoning. In addition, **Virtuoso** does not perform consistency checks (e.g., cardinality constraints, property constraints etc.) and hence updates that result in an inconsistent state of the database are successful. On the other hand, **OWLIM** succeeds in most of the conformance tests. In the following we will present the results for every class of choke points that we discussed in Section 4.4.

4.5.1 Class and Property Subsumption

Table 4.27 shows the results for **Virtuoso** and **OWLIM** and for the *class* and *property subsumption* (`rdfs:subClassOf`, `rdfs:subPropertyOf`) conformance queries presented in Section 4.4.1. We have to note here that **Virtuoso** was successful in running the tests when option `transitive` was added in the query.

RULE	OWLIM	Virtuoso
CAX-SCO	SUCCESS	SUCCESS
PRP-SPO1	SUCCESS	FAIL
SCM-SCO	SUCCESS	SUCCESS
SCM-SPO	SUCCESS	SUCCESS

Table 4.27: Class and Property Subsumption Results

4.5.2 Property Domain and Range

RULE	OWLIM	Virtuoso
SCM-RNG1	SUCCESS	FAIL
SCM-RNG2	SUCCESS	FAIL
SCM-DOM1	SUCCESS	FAIL
SCM-DOM2	SUCCESS	FAIL
SCM-DOM	SUCCESS	FAIL
SCM-RNG	SUCCESS	FAIL

Table 4.28: Property Domain and Range Results

The results for the *property domain* and *range* (`rdfs:domain`, `rdfs:range`) conformance tests for both RDF engines discussed in Section 4.4.2 are given in Table 4.28.

4.5.3 Union and Intersection of Classes

RULE	OWLIM	Virtuoso
SCM-UNI (1)	SUCCESS	FAIL
SCM-UNI (2)	SUCCESS	FAIL
SCM-INT (1)	SUCCESS	FAIL
SCM-INT (2)	SUCCESS	FAIL

Table 4.29: Union and Intersection of Classes

The results for the *union* and *intersection* of classes (`owl:unionOf` and `owl:intersectionOf` resp.) conformance tests for both RDF engines discussed in Section 4.4.3 are given in Table 4.29.

4.5.4 Enumeration

RULE	OWLIM	Virtuoso
CLS-OO	SUCCESS	SUCCESS

Table 4.30: Enumeration Results

Table 4.30 presents the results for the conformance query considering `owl:oneOf` construct that was presented in Section 4.4.4.

4.5.5 Equality

RULE	OWLIM	Virtuoso
EQ-REF	FAIL	FAIL
EQ-SYM	SUCCESS	SUCCESS
EQ-TRANS	SUCCESS	SUCCESS
EQ-REP-S	SUCCESS	SUCCESS
EQ-REP-O	SUCCESS	SUCCESS
EQ-REP-P	SUCCESS	SUCCESS
EQ-DIFF1	FAIL	FAIL

Table 4.31: Equality

The results for *equality tests* regarding `owl:sameAs` links discussed in Section 4.4.5 are shown in Table 4.31. Note here that both **OWLIM** and **Virtuoso** are successful in all tests except EQ-REF and EQ-DIFF1.

4.5.6 Class Keys, Property Chains, Inverse Properties

Table 4.32 presents the results for conformance queries regarding key constraints, property chains, inverse properties discussed in Sections 4.4.8, 4.4.9 and 4.4.6 respectively. As expected **OWLIM** succeeded in all tests, whereas **Virtuoso** only for the `owl:inverseOf` property test.

Rule	OWLIM	Virtuoso
PRP-KEY	SUCCESS	FAIL
PRP-SPO2 (1)	SUCCESS	FAIL
PRP-SPO2 (2)	SUCCESS	FAIL
PRP-INV1	SUCCESS	SUCCESS

Table 4.32: Class Keys, Property Chains, Inverse Properties Results

4.5.7 Constraints on Properties

Rule	OWLIM	Virtuoso
PRP-IFP	SUCCESS	FAIL
PRP-ASYP	SUCCESS	FAIL
PRP-IRP	SUCCESS	FAIL
PRP-TRP	SUCCESS	SUCCESS

Table 4.33: Property Constraints Results

The results for the conformance tests on *property constraints* (Section 4.4.7) are presented in Table 4.33. **Virtuoso** fails in all texts except for `owl:TransitiveProperty` construct; the query in this case run with option `transitive`.

4.5.8 Disjoint Classes and Properties

Rule	OWLIM	Virtuoso
PRP-PDW	SUCCESS	FAIL
PRP-ADP	SUCCESS	FAIL
CAX-DW	SUCCESS	FAIL
CAX-ADC	SUCCESS	FAIL

Table 4.34: Disjointness Results

The results for constraints regarding disjointness of classes and properties (see Section 4.4.10) are shown in Table 4.34.

4.5.9 Cardinalities

RULE	OWLIM	Virtuoso
CLS-MAXC1	FAIL	FAIL
CLS-MAXC2	SUCCESS	FAIL

Table 4.35: Cardinalities Results

Table 4.35 presents the results for the tests concerning cardinality constraints discussed in Section 4.4.11. **Virtuoso** fails for both tests, and **OWLIM** succeeds only for CLS-MAXC2.

5 Instance Matching Choke Point Analysis

In order to identify choke points for the *instance matching* task we conducted a set of experiments using *i)* the **Open PHACTS** dataset presented in Section 2.2 and *ii)* the state of the art instance matching systems Limes [21] and Silk [44] presented in Sections 3.2.1 and 3.2.2 and respectively.

5.1 Evaluation Criteria and Choke Points Identification

Our analysis of the instance matching choke points is targeted on identifying cases where the tested systems perform poorly in the matching task (with respect to the golden standards) regarding a set of *evaluation criteria*:

Precision/ Recall / F-measure These metrics are used to determine the *effectiveness* of the instance matching systems. In information retrieval, *precision* is the fraction of the *intersection* of the *relevant* and *retrieved* instances over the *retrieved instances*, whereas *recall* is the fraction of the *intersection* of *relevant* and *retrieved instances* over the *relevant instances*.
In the case of instance matching, retrieved instances are the instances matched by the used systems, and the relevant instances are the matched instances that are also reported in the *golden standard* provided by the test case. Precision can be seen as a measure of *exactness* or *quality*, whereas recall is a measure of *completeness*. *F-measure* is a metric that combines precision and recall. It is calculated as their *harmonic mean*.
When comparing the results of the instance matching process with the golden standard, one can calculate the *true positive* (tp) (correct), the *false positive* (fp) (unexpected) and the *false negative* (fn) (missing) results. Precision, recall and f-measure can then be computed as follows:

$$
\begin{aligned}
precision &= \frac{tp}{tp + fp} \\
recall &= \frac{tp}{tp + fn} \\
fmeasure &= 2 \times \frac{precision \times recall}{precision + recall}
\end{aligned}
$$

Run Times We record the times that the systems needed to compute the matching instances for the datasets. This is a standard metric for measuring the *performance* of the tested systems. The time that it takes for an instance matching system to compute the matching instances is an important criteria, but not as important as precision, recall and F-measure since such systems are judged primarily on the basis on their results: systems with higher quality results are more preferrable than ones with lower quality, even if the latter compute matches faster.

Scalability In the context of Linked Data, it is essential that RDF instance matching systems can deal with billions of triples. For instance, according to the latest diagram of the Linked Open Data cloud, there are over 31 billion triples published online that originate from different sources. In this context, *scalability* is one of the most important characteristics of an instance matching system.

Distance Measures Distance Measures include *string matching algorithms* and *transformation methods*. String matching algorithms are distinguished into *character-based*, *token-based* and *special-purpose* ones.
Character-based distance measures compare strings on a character basis, whereas token-based ones compare strings on a token basis, a token being a string of two or more characters that is significant as

a group. These measures work well for *typographical* mistakes. There exist a number of tasks where token-based distance measures are better suited (e.g., strings where substrings are reordered e.g. "John Doe" and "Doe, John") than character-based ones. Special purpose distance measures are developed for matching specific types of strings such as dates. SILK has a large variety of matching metrics for different use cases if compared to LIMES that has a slightly smaller variety. The character-based measures are shown in Table 5.1, the token-based ones in Table 5.2 and finally the special-purpose ones in Table 5.3. Some of the transformation methods that SILK offers are shown in Table 5.4.

SILK

METRIC	DESCRIPTION
levenshteinDistance	The *minimum number of edits* needed to transform one string into the other, using the *allowed* operations (insertion, deletion and substitution of a single character).
levenshtein	The levensthein distance normalized to the interval $[0, 1]$.
jaro	Jaro is a simple distance metric originally developed to compare person names.
jaroWinkler	The JaroWinkler distance metric is designed and best suited for short strings (e.g., person names).
equality	This metric returns 0 if strings are equal, 1 otherwise.
inequality	Complementary of *equality* metric.

LIMES

levenshtein	The levensthein distance normalized to the interval $[0, 1]$.
exactMatch	This metric returns 0 if strings are equal, 1 otherwise.

Table 5.1: Character-based Distance Metrics

SILK

METRIC	DESCRIPTION
jaccard	Jaccard distance coefficient.
dice	Dice distance coefficient.
softjaccard	This metric is the same as Jaccard distance but values within an levenhstein distance of `maxDistance` are considered equivalent.

LIMES

jaccard	Jaccard distance coefficient.
overlap	Overlap distance coefficient.
trigrams	N-grams distance coefficient.
cosine	Cosine distance coefficient.

Table 5.2: Token-based Distance Metrics

VOCABULARIES/THESAURI Instance matching systems can use these information resources for discovering matches between the different datasets. Wordnet [41] is such an example that encodes the synonyms and antonyms of terms. Such information can be used for finding or even rejecting potential matches.

SCHEMA INFORMATION In the context of Linked Data, datasets in different domains are accompanied by *ontologies*: Uniprot [39], PubMed[1] and GO [32] in the biomedical domain, MusicBrainz [2] in the entertainment domain. Schema information can be used to guide and optimize the matching process: instances of equivalent classes (`owl:equivalentClass`) can be considered as possible matches, whereas those of disjoint classes (`owl:disjointWith`) as improbable matches.

[1] PubMed:`http://www.ncbi.nlm.nih.gov/pubmed`
[2] MusicBrainz:`http://musicbrainz.org/`

SILK

METRIC	DESCRIPTION
num(*float* `minValue`, *float* `maxValue`)	Computes the numeric difference between two numbers: `minValue`, `maxValue` are the minimum and maximum values that occur in the dataset.
date	Computes the distance between two dates ("YYYY-MM-DD" format). Returns their difference in days.
dateTime	Computes the distance between two date time values (`xsd:dateTime` format). Returns the difference in seconds.
wgs84(*string* `unit`, *string* `curveStyle`)	Computes the geographical distance between two points. `unit` is the unit in which the distance is measured (e.g., "meter", "kilometer")

LIMES

METRIC	DESCRIPTION
wgs84(*string* `unit`, *string* `curveStyle`)	Computes the geographical distance between two points. `unit` is the unit in which the distance is measured (e.g., "meter", "kilometer")

Table 5.3: Special-Purpose Distance Metrics

FUNCTIONS AND PARAMETERS	DESCRIPTION
removeBlanks	Remove whitespace from a string
removeSpecialChars	Remove special characters (including punctuation) from a string
lowerCase	Convert to lower case
upperCase	Convert to upper case
capitalize(allWords)	Capitalizes the string i.e. converts the first character to upper case
stem	Apply word stemming to the string

Table 5.4: List of a subset of the transformation processes for SILK

5.2 Test Cases

For the instance matching choke points analysis, we used the **Open PHACTS** datasets provided by the **Open PHACTS** [35] FP7 European Project and discussed in Section 2.2. **Open PHACTS** provided a set of *golden standards* or *linksets* presented in Section 2.2, used to evaluate the quality of the matching process performed by instance matching systems. These golden standards were created by *domain experts (curators)* in the pharmacology and chemistry domains. The results of the instance matching process were compared against the golden standards to compute the precision/recall and f-measure values as discussed previously. For our experiments, we focused on the following two **Open PHACTS** test cases:

TC1 : instances of the **ConceptWiki** [30] dataset were matched with instances from the **DrugBank–Targets** dataset.

TC2 : **ConceptWiki** [30] instances were matched with instances from the **ChemSpider** [29] dataset.

We focused on the **ConceptWiki** dataset since it is the central dataset of the **Open PHACTS** project that is linked to all the others datasets as shown in Figure 5.1. The **DrugBank – Targets** dataset was used in our experiments because it is a dataset that is widely known and well established. More importantly, the dataset was already used for the SILK and LIMES systems that we employed in our experiments. Last but not least, the **ChemSpider** dataset was used to test the *scalability* aspect of the systems since it is the dataset with the highest number of reported matchings.

The **ConceptWiki** dataset is an RDF dataset that is comprised of 2.65 million triples that all use property `skos:prefLabel`[3] where `skos` is the namespace for the SKOS vocabulary [19] .

The DRUGBANK dataset, contained information about *targets* and *drugs* (in total 522000 triples); we removed information about drugs, hence creating the **DrugBank – Targets** dataset. These triples were

[3] `http://www.w3.org/2004/02/skos/core\#prefLabel`

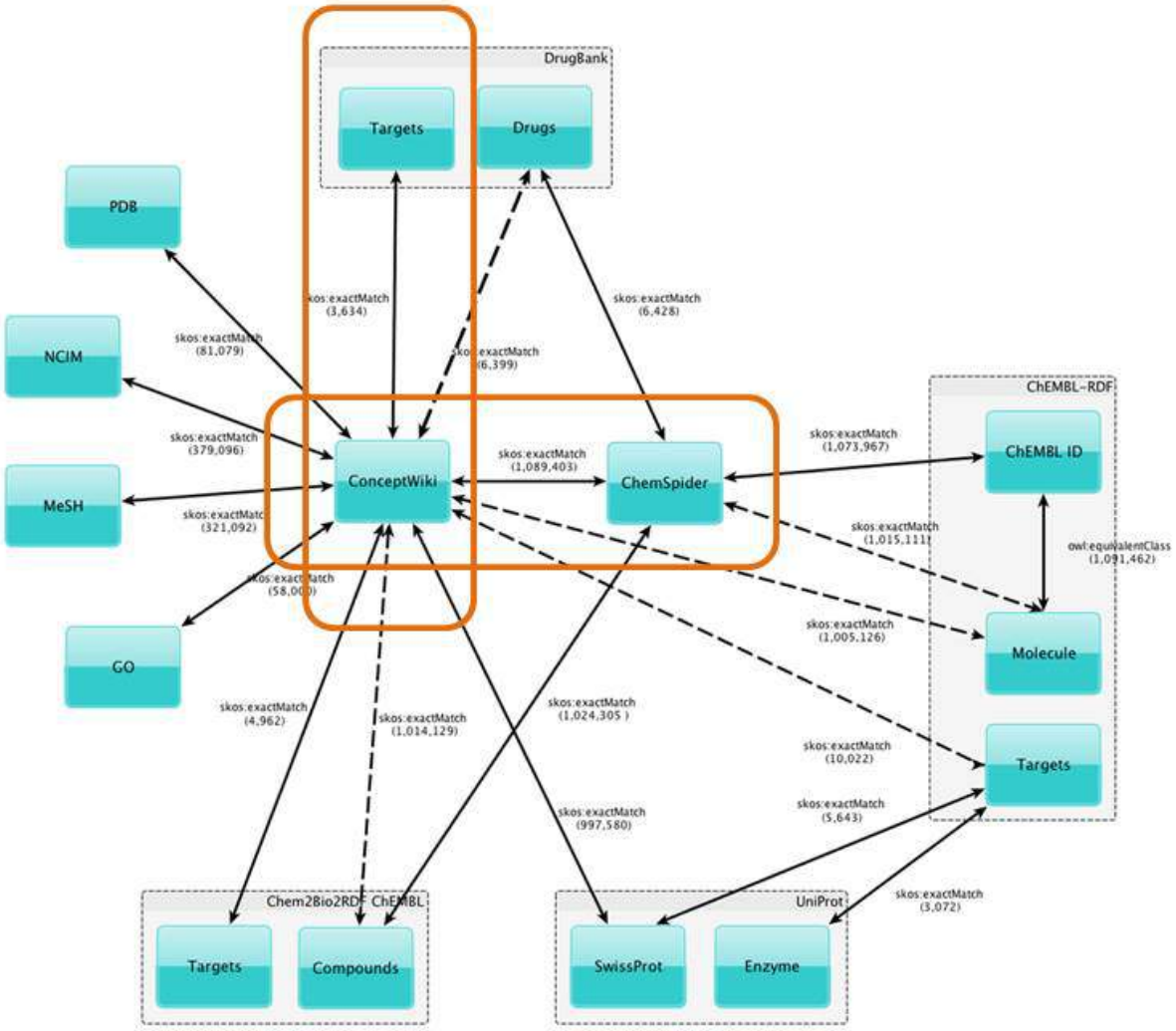

Figure 5.1: Instance Matching Test Cases

removed because they caused some matches with **ConceptWiki** instances, which although reasonable and most propably correct in most cases they were not recorded in the provided golden standard; thus the extra triples were introducing noise in our measurements (especially in the case of precision) affecting the performance of the systems.

The **DrugBank−Targets** dataset contained 6 classes and 117 properties. The classes, properties and their instances are shown in Table 5.5. In total the **DrugBank−Targets** dataset contains 273259 triples. **ChemSpider** contains chemical compounds, but also chemical structures (SMILES[4], InCHI[5]). The dataset contains 1,14 million different compounds that are distinguished by their unique *title*, i.e., RDF property `chemspider:title` where `chemspider` is the namespace for `http://rdf.chemspider.com`. In total it contains $9,35$ million triples with 6 concept properties that are shown with the number of their instances in Table 5.6.

[4]SMILES: `http://en.wikipedia.org/wiki/Simplified_molecular-input_line-entry_system`
[5]InCHI: `http://www.iupac.org/home/publications/e-resources/inchi.html`

Classes	Instances
drugbank:drug_interactions	10096
drugbank:drugs	0
drugbank:enzymes	53
drugbank:references	96
drugbank:targets	4553
vocab:Offer	0

Table 5.5: **DrugBank – Targets** Dataset Classes and their instances

Properties	Instances
chemspider:smiles	1148672
chemspider:csid	1148672
chemspider:inchi	1148672
chemspider:inchikey	1148672
chemspider:title	1148672
chemspider:non − validated − synonym	3614705

Table 5.6: **ChemSpider** Properties

The golden standards for the **ConceptWiki/DrugBank – Targets** datasets and **ConceptWiki/ChemSpider** datasets contain 3634 and 1089403 RDF triples of the form $(x, \texttt{skos:exactMatch}, y)$ respectively, where each triple represents a match for the specific test case.

5.3 Experimental Set Up

For our tests we used two open source, publicly available instance matching systems, namely Limes [21] and Silk Single Machine [14, 15, 44].

Both Limes and Silk obtain their data from RDF repositories using SPARQL endpoints. In our experiments we stored the RDF datasets in OpenLink Virtuoso Universal Server Version 7. Java memory was configured to 8GB.

An important task for our experiments was to find the information on the basis of which the datasets could be matched. After the analysis of the golden standards, we indicated this information:

- Tc1: instances from the **ConceptWiki** and **DrugBank – Targets** datasets were matched using:
  1. the **ConceptWiki** skos:prefLabel property, instances of which were obtained by executing the SPARQL query

```
select distinct ?prefLabel
from <http://conceptwiki.data>
where {?cw skos:prefLabel ?prefLabel}
```

  2. the **DrugBank – Targets** drugbank:name property, instances of which are obtained with the following SPARQL query where drugbank is the namespace for http://drugbank-targets.data.

```
select distinct ?name
from <http://drugbank-targets.data>
where {?db skos:prefLabel ?name }
```

- TC2, instances from **ConceptWiki** and **ChemSpider** datasets were matched using the

  1. the **ConceptWiki** `skos:prefLabel` property, instances of which were obtained by executing the SPARQL query stated previously.
  2. the **ChemSpider** `chemspider:title` property, instances of which were obtained by executing the SPARQL query:

```
select distinct ?title
from <http://chemspider_compounds.data>
where {?db chemspider:title ?title}
```

The SILK and LIMES systems were then configured to match instances (i.e., triple subjects) by comparing their:

- ?`prefLabel` with ?`name` values for TC1
- ?`prefLabel` with ?`title` values for TC2.

We conducted experiments with the above test cases using the different *string matching algorithms* that are offered by the tested systems: *exactMatch* (for LIMES), *equality* (for SILK), *trigrams* [4], *cosine* [26], *jaccard* [42] and *levenshtein* [11] algorithms (discussed in Tables 5.1 and 5.2). We run each of the above algorithms using different values for the *threshold* that determines whether two instances are considered as a *true match*. Specifically we experimented with the LIMES system using the following algorithms *exactMatch*, *cosine*, *trigrams* and *jaccard*. We tested SILK with the *equality*, *jaccard* and *levenshtein* algorithms. We failed to conduct experiments with LIMES using the *levenshtein* algorithm, although it is mentioned in the available matching algorithms. For our experiments we used a threshold between values 0 and 1.

A threshold is perceived in two ways:

- to represent the *similarity* between two inputs. Values above a certain threshold imply a higher similarity between the instances. Exact matches are generated for similarities *greater* than a certain threshold. This threshold perception is used in LIMES. In our experiments we are interested in higher similarities, so we used two threshold values $t = 0.9$ and $t = 0.8$.
- to represent the *distance* between two inputs. Exact matches are generated for distances that they are *below* a given threshold (intuitively, the smaller the distance, the better the match): a distance $t = 0$ indicates that an exact match is found, whereas distance $t = 1$ indicates that there is no match. Consequently values below a low threshold indicate a higher similarity between instances. This threshold perception is used in SILK. As we are interested in higher similarities we experiment with threshold $t = 0.1$ and $t = 0.2$.

5.4 Experimental Results

5.4.1 Precision, Recall and F-measure

The results for LIMES, for thresholds 0.9, 0.8 and for test case TC1 are shown in Tables 5.7 and 5.8 respectively. The results for test case TC2 and for thresholds 0.9, 0.8 are given in Tables 5.9 and 5.10 respectively.

From the results one can see that LIMES did not perform very well for TC1. A possible reason is that the queries we used to match the instances were not sufficient for the kind of data in hand, and hence a more detailed analysis is needed, probably with the help of the authors of the golden standards.

Another interesting point is that *i)* for the string matching algorithms *exactMatch*, *cosine* and *trigrams* and *ii)* for both test cases, LIMES returned the same results for a given threshold. Without having access to the source code, we are unable to provide here any explanation. *jaccard* performed worse than the other

LIMES

ALGORITHM	PRECISION	RECALL	F-MEASURE
exactMatch	0.098	0.0047	0.009
cosine	0.098	0.0047	0.009
trigrams	0.098	0.0047	0.009
jaccard	0	0	0

Table 5.7: **ConceptWiki/DrugBank−Targets** - TC1 (threshold $t = 0.9$)

LIMES

ALGORITHM	PRECISION	RECALL	F-MEASURE
exactMatch	0.04	0.17	0.065
cosine	0.04	0.17	0.065
trigrams	0.04	0.17	0.065
jaccard	0	0	0

Table 5.8: **ConceptWiki/DrugBank−Targets** - TC1 (threshold $t = 0.8$)

LIMES

ALGORITHM	PRECISION	RECALL	F-MEASURE
exactMatch	1	0.15	0.26
cosine	1	0.15	0.26
trigrams	1	0.15	0.26
jaccard	1	0.04	0.08

Table 5.9: **ConceptWiki/ChemSpider** - TC2 (threshold $t = 0.9$)

LIMES

ALGORITHM	PRECISION	RECALL	F-MEASURE
exactMatch	1	0.62	0.77
cosine	1	0.62	0.77
trigrams	1	0.62	0.77
jaccard	1	0.17	0.29

Table 5.10: **ConceptWiki/ChemSpider** - TC2 (threshold $t = 0.8$)

three string matching methods. We assume that this is because the strings are not "normalized" as in other domains (e.g., person names, addresses etc.). Furthermore, LIMES for both test cases returns better results with 0.8 threshold. This shows that the compared instances had some but not important differences.

SILK

ALGORITHM	PRECISION	RECALL	F-MEASURE
equality	0.0	0.0	0.0
levenshtein	0.0	0.0	0.0
jaccard	0.0	0.0	0.0

Table 5.11: **ConceptWiki/DrugBank−Targets** - TC1 (threshold $t = 0.1$)

SILK

ALGORITHM	PRECISION	RECALL	F-MEASURE
equality	0.0	0.0	0.0
levenshtein	0.0	0.0	0.0
jaccard	0.0	0.0	0.0

Table 5.12: **ConceptWiki/DrugBank−Targets** - TC1 (threshold $t = 0.2$)

The results for SILK, for thresholds 0.1, 0.2 and for test case TC1 are shown in Tables 5.11 and 5.12 respectively. The results for test case TC2 and for thresholds 0.1, 0.2 are given in Tables 5.13 and 5.14 respectively. From the results one can observe that SILK did not perform well for precision and recall returning zero true positive results. On the other hand, LIMES returned a non empty set of true positive results, showing that LIMES outperformed SILK for this experiment. One can also see that SILK performed better than LIMES for the test case TC2 and for threshold 0.1 (corresponding to a threshold of 0.9 for LIMES). Specifically it returned high scores for precision *and* recall. An interesting observation one can make is that *equality* and *jaccard* distance metrics give the same results in SILK system. This was also observed in LIMES for the testcases by using the metrics *exactMatch* and *cosine*.

SILK

ALGORITHM	PRECISION	RECALL	F-MEASURE
equality	1	0.51	0.68
levenshtein	–	–	–
jaccard	1	0.51	0.68

Table 5.13: **ConceptWiki/ChemSpider** - TC2 (threshold $t = 0.1$)

SILK

ALGORITHM	PRECISION	RECALL	F-MEASURE
equality	1	0.51	0.68
levenshtein	1	0.037	0.071
jaccard	1	0.51	0.68

Table 5.14: **ConceptWiki/ChemSpider** - TC2 (threshold $t = 0.2$)

As far as threshold 0.2 is concerned (corresponding threshold to 0.8 for LIMES), SILK performs slightly better than LIMES for all the distance metrics, apart from *jaccard*. Again, here SILK gives the same results for both *equality* and *jaccard*, and the results are the same as for threshold 0.1.

Table 5.15 below a synopsis of the performance of SILK and LIMES in terms of *f-measure* and for the algorithms that are implemented in both systems.

TEST CASE	DISTANCE METHOD	LIMES		SILK	
		0.9	0.8	0.1	0.2
TC1	*jaccard*	0.0	0.0	0.0	0.0
	exactMatch/equality	0.009	0.065	0.0	0.0
TC2	*jaccard*	0.08	0.29	0.68	0.68
	exactMatch/equality	0.26	0.77	0.68	0.68

Table 5.15: A summary of the comparison for SILK and LIMES

Algorithms	**ConceptWiki/DrugBank−Targets**(Tc1)		**ConceptWiki/ChemSpider**(Tc2)	
	Retrieval	*Matching*	*Retrieval*	*Matching*
exactMatch	983.142	1,050.841	331.643	437.396
trigrams	981.457	1,048.487	289.792	384.036
cosine	960.148	1,027.482	284.683	381.621
jaccard	251.889	318.058	100.420	200.597

Table 5.16: Run times for Limes- Threshold 0.9 (time in seconds)

Algorithms	**ConceptWiki/DrugBank−Targets**	**ConceptWiki/ChemSpider**
	Retrieval/Matching	*Retrieval/Matching*
equality	4213	4417
levenshtein	5590	− ($t = 0.1$) / 9829 ($t = 0.2$)
jaccard	912	4589

Table 5.17: Run times for Silk - Threshold 0.1 (time in seconds)

5.4.2 Run Times

For this experiment, we measured the time it took the systems to create the matching results for each of the test cases discussed in Section 5.2.

The run times of Limes for all the different string matching algorithms and the two test cases with threshold 0.9 are shown in Table 5.16. The runtimes for threshold 0.8 are comparable, so we do not report them for Limes or Silk. We distinguish between the time to obtain the data (*Retrieval*) from the SPARQL endpoints, and the time need to compute the matches (*Matching*).

Limes needs around 16 minutes to retrieve the data and 17 minutes to compute the matchings for the *exactMatch*, *cosine* and *trigrams* algorithms and for testcase Tc1. For the same test case it needs less time to compute the matching with the *jaccard* distance metric. In fact it takes around 4 minutes to retrieve data and approximately 5 minutes to compute the matchings.

Regarding test case Tc2, Limes needs about 5 minutes to retrieve the data and about 6 minutes to compute the matchings except for *jaccard* for which it needs one and a half minute to retrieve data and 3 minutes to compute the matches. This difference in the data retrieval time can be explained by the use of "exemplars"; recall that Limes works with exemplars to reduce the size of the input data for the matching process.

The run times of Silk for all the different algorithms and test cases with threshold 0.1 (corresponds to threshold 0.9 for Limes) are shown in Table 5.17. The runtimes for threshold 0.2 are pretty much the same, so we do not report them. Note that for *levenshtein* and for threshold 0.1, Silk failed to produce any results (the used threshold is very low indicating high similarity) With the high number of exact matches (up to millions for this test case), it seems that Silk fails to scale for large datasets and linksets.

Silk needs around 70 minutes to retrieve data and compute the matchings for the *equality* algorithm, around 90 minutes for the *levenshtein* and around 15 minutes for *jaccard* and for test case Tc1. As one can notice Limes is notably faster than Silk in this testset. Regarding test case Tc2, Silk needs around 73 minutes to retrieve data and compute the matchings for the *equality*, around 165 minutes for the *levenshtein* and around 76 minutes for *jaccard*. Again in this test case Limes calculates faster the mappings.

5.4.3 Scalability

Scalability is a major issue for instance matching systems in the era of Linked Open Data. In our case, the biggest test case of **Open PHACTS** is Tc2. The matching process compares 1,65 million `skos:prefLabel` triples from **ConceptWiki** with 1,14 millions `chempider:title` triples from **ChemSpider**. In total it conducts more than 3 trillion matches. Both systems, Limes and Silk, did cope well with this exigent matching process. The only problem was that in one case the matching process did not end. Specifically

when running *levenshtein* distance in SILK with threshold=0.1 and test case **ConceptWiki/ChemSpider**, it ran many hours and at the end we always got the message "Connection abort" without any other information or error.

5.4.4 Support Matching with Thesaurus

Both LIMES and SILK instance matching systems are domain independent systems, which means they conduct the matchings without any knowledge of the data, depending only on the string matching methods mentioned above. Other instance matching systems have been reported, like the OTO matching system [5] that makes use of general purpose thesaurus, e.g. Wordnet [41] to enhance the matching process. This way the matching outputs do not only depend on the string matching algorithms used, but also on the semantic process of comparing synonyms and antonyms that exist in the thesaurus. LIMES and SILK do not take into account such information in order to enhance the matching process and to ameliorate the quality of the results.

5.4.5 Reasoning functionalities

LIMES and SILK systems do not use any kind of semantic information expressed in a *schema* or *ontology*. In this way the matching process is not "optimized" since it might consider instances that belong to disjoint classes, that should never be considered as possible matches. For example, in one of our test cases we want to match data from **ConceptWiki** and **DrugBank−Targets**. As mentioned in Section 2.2 **ConceptWiki** includes concepts from Literature, Proteins and Chemicals and others and **DrugBank−Targets** includes drug targets. When using LIMES and SILK to match the instances of these datasets, concepts from Literature will be considered as possible matches with concepts from drugs, which is semantically incorrect since these classes are disjoint.

5.4.6 Concluding Remarks

CRITERIA	SYSTEMS	
	LIMES	SILK
PERFORMANCE	√	√
SCALABILITY	√	√
SCHEMA	−	−
THESAURI	−	−
DISTANCE MEASURES	√	√
TRANSFORMATIONS	−	√

Table 5.18: Overall comparison for LIMES and SILK

To sum up, we conducted instance matching experiments on systems LIMES and SILK, by using two different test cases, namely **ConceptWiki** with **DrugBank−Targets** and **ConceptWiki** with **ChemSpider**. The experiments showed that LIMES performed better in the former case by giving better results in terms of precision,recall and F-measure, and worse for the second test case for the same criteria. Furthermore, LIMES conducted the matching task much faster than SILK in both testsets.

LIMES does not support any transformation processes unlike SILK that supports a larger set as discussed in Table 5.4. Neither of the systems uses any kind of semantic information expressed in a *schema* (e.g., *ontology*), or a domain specific or general thesaurus to enhance the matching process with semantic information. Table 5.18 shows the overall evaluation results of the systems.

6 ETL Choke Point Analysis

The datasets that we are dealing with in ETL Choke Point Analysis, are **ChEMBL** [28] and **ChEBI** [6]. **ChEMBL**, whose description can be found in Section 2.2 and not repeated here, consists of 6.2 GB of data, stored in 28 relational tables. **ChEBI** consists of 1.5 GB of data, stored in 12 relational tables. **ChEBI** is a dictionary of molecular entities which are defined as natural or synthetic products which are used to intervene in the processes of living organisms. **ChEBI** also includes an ontological classification, which is used to specify the relationships between molecular entities or classes of entities and their parents and/or children [6].

6.1 D2R

6.1.1 Extract

As described in Section 3.3.2, **D2R** is a tool that provides a mechanism through which relational data are mapped to RDF. To achieve this, **D2R** provides a declarative mapping language upon which the translations are based. The main components of this mapping language are the class maps (`d2rq:ClassMap`) and the property bridges (`d2rq:PropertyBridge`). A class map represents a class of an ontology whereas the property bridges represent a set of properties of a certain class. For instance, assuming the relational table of **ChEBI** database, `AUTOGEN_STRUCTURES(id, structure_id)` its appropriate mappings are shown in Table 6.1.

```
1   map:autogen_structures a d2rq:ClassMap;
2           d2rq:dataStorage map:database;
3           d2rq:uriPattern "autogen_structures/@@autogen_structures.id@@";
4           d2rq:class vocab:autogen_structures;
5           d2rq:classDefinitionLabel "autogen_structures" .
6
7   map:autogen_structures_id a d2rq:PropertyBridge;
8           d2rq:belongsToClassMap map:autogen_structures;
9           d2rq:property vocab:autogen_structures_id;
10          d2rq:propertyDefinitionLabel "autogen_structures id";
11          d2rq:column "autogen_structures.id";
12          d2rq:datatype xsd:integer .
13
14  map:autogen_structures_structure_id__ref a d2rq:PropertyBridge;
15          d2rq:belongsToClassMap map:autogen_structures;
16          d2rq:property vocab:autogen_structures_structure_id;
17          d2rq:refersToClassMap map:structures;
18          d2rq:join "autogen_structures.structure_id => structures.id" .
19
```

Table 6.1: **D2R** Mappings for the relational table `CHEBI.AUTOGEN_STRUCTURES`

In more details, for the relational table `AUTOGEN_STRUCTURES` we create the class map `map:autogen_structures` (line 1 of Table 6.1) which represents an RDFS schema class that correspond to this table. This class map defines that its instances are identified by the column `autogen_structures.id` of the relational table (line 3). In addition, it is connected to a `d2rq:Database`, namely `map:database` (line 2) and has a set of `d2rq:PropertyBridges` (lines 7-18) which attach properties to instances. The first of these attached properties is the `map:autogen_structures_id` (line 7), whose RDF datatype is integer (line 12), and corresponds to the relational column `autogen_structures.id` (line 11). The second one is the `map:autogen_structures_structure_id__ref` (line 14), which corresponds to a foreign key in the re-

lational schema (structures.id), so it has to reference another class map map:structures (line 17) that creates the instances which are used as the values of this bridge.

Table 6.3 shows the result of applying the mappings shown in Table 6.1 on the relational data of Table 6.2. As we can see in Table 6.3 the mapping creates 6 triples. In the same manner all mappings for both **ChEBI** and **ChEMBL** datasets are produced. By evaluating these mappings on the entire datasets, the extract process translates the relational representation into 281.348.982 triples.

AUTOGEN_STRUCTURES

id	structure_id
2680136	2707183

Table 6.2: A sample row for the relational table CHEBI.AUTOGEN_STRUCTURES

```
1    chebi_baseuri:autogen_structures a rdfs:Class .
2    chebi_baseuri:autogen_structures_id a rdf:Property .
3    chebi_baseuri:autogen_structures_structure_id a rdf:Property .
4    chebi_baseuri:autogen_structures/2680136 a chebi_baseuri:autogen_structures;
5        vocab:autogen_structures_id "2680136"^^xsd:integer;
6        vocab:autogen_structures_structure_id chebi_baseuri:structures/2707183 .
```

Table 6.3: Resulting triples after the application of the mappings shown in Table 6.1 on the sample row of Table 6.5 for D2R

6.1.2 Transform

To aggregate information of the same nature found in the datasets, we can apply a set of transformations in the original data; these transformations must be applied during the extraction stage. To achieve this in D2R, we use the d2rq:sqlExpression property of d2rq:PropertyBridge. So, for properties with literal values (in which such transformations make sense), these values are generated by evaluating a specific SQL expression.

For instance, considering the relational table DOCS of the **ChEMBL** dataset, the mappings along with the chosen transformations shown in Table 6.4 could be applied. In particular, the transformations shown in Table 6.4 aggregate all the important information (journal, year etc.) of a document in relational table CHEMBL.DOCS in a concatenated string. For example, for row of CHEMBL.DOCS shown in Table 6.5, the transformation of Table 6.4 would produce the triple shown in Table 6.6.

```
1        map:docs_important_info a d2rq:PropertyBridge;
2                d2rq:belongsToClassMap map:docs;
3                d2rq:property vocab:docs_important_info;
4                d2rq:sqlExpression "CONCAT(docs.journal, ', ',
5                        docs.year, ', ',
6                        docs.volume, '(',
7                        docs.issue, ') ',
8                        docs.first_page, '-',
9                        docs.last_page)" .
```

Table 6.4: **D2R** A part of mappings for relational table CHEMBL.DOCS

DOCS

doc_id	journal	year	volume	issue	first_page	last_page	...
1	J. Med. Chem.	2004	47	1	1	9	...

Table 6.5: A sample row for CHEMBL.DOCS relational table

```
chebi_baseuri:docs/1 vocab:docs_important_info "J. Med. Chem. 2004, 47(1) 1-9" .
```

Table 6.6: Resulting triple after the application of the mappings shown in Table 6.4 on the sample row of Table 6.5 for D2R

Despite the fact that **D2R** provides the option to use SQL expressions to manipulate literal values, its mapping language is not rich enough to evaluate more complex queries. For example, we can easily aggregate information in the level of the row (e.g. CONCAT aggregate function), as described above, but not in the column one (e.g. GROUP BY aggregate functions).

Another type of transformation that can be applied is related to the inter-linkage between the two datasets **ChEMBL** and **ChEBI**. As the linkage information is stored in a table of the **ChEMBL** dataset itself, as shown in Table 6.7 it was trivial to produce the appropriate triples by querying this relational table. So, assuming the row of the relational table MOLECULE_DICTIONARY of the **ChEMBL** dataset shown in Table 6.7, the produced link should be the one shown in Table 6.8.

MOLECULE_DICTIONARY

molregno	...	chebi_par_id
97	...	8364

Table 6.7: A sample row for CHEMBL.MOLECULE_DICTIONARY relation table

```
chembl_baseuri:compounds/97 skos:exactMatch chebi_baseuri:compounds/8364 .
```

Table 6.8: Resulting triple after the application of the inter-linkage transformation in the sample row of Table 6.7

6.1.3 Load

After the process of Export and Transform, all relational data, transformed or not, are stored in the disk in separate RDF files. These files are then loaded to Virtuoso Server.

6.2 Triplify

6.2.1 Extract

Triplify (described in Section 3.3.1) is another tool for publishing RDF data from relational databases that can be used as an ETL tool. To achieve this data publication, relational database queries have to be defined, which can be thought of as database views. These queries are expressed in SQL with some additions (SQL aliases) in order to retrieve valuable information and to convert the results into RDF triples. For example, assuming that we want to publish data from the relational table AUTOGEN_STRUCTURES(id, structure_id) of the **ChEBI** database, the query that has to be evaluated is shown in Table 6.9.

As shown from this query, in order for **Triplify** to be able to convert its results into RDF, the query is required to have a certain structure. More specifically, the first column (id) must be used in order to generate

instance URIs, while the other ones could be used to represent properties. Some of the property columns, such as autogen_structures.structure_id, may contain references to other objects rather than literal values. So, a configuration array that specifies which columns are references to objects of which type has to be defined, as shown in Table 6.10.

For example, if we apply the query of Table 6.9 on the relational row shown in Table 6.2, we would get the RDF triples shown in Table 6.11. In the same manner are produced all SQL queries for both **ChEBI** and **ChEMBL** datasets. By evaluating these queries the extract process translates the relational representation of these datasets into 267.960.975 triples.

```
triplify[‘queries’]=array(
        ‘autogen_structures’=>"
                SELECT autogen_structures.id AS id,
                        autogen_structures.id
                                AS ‘vocab:autogen_structures_id^^xsd:integer’,
                        autogen_structures.structure_id
                                AS ‘vocab:autogen_structures_structure_id’
                FROM autogen_structures"
);
```

Table 6.9: **Triplify** Mapping query for the relational table CHEBI.AUTOGEN_STRUCTURES

```
triplify[‘objectProperties’]=array(
        ‘vocab:autogen_structures_structure_id’=>‘structures’
);
```

Table 6.10: **Triplify** Object Properties array for the relational table CHEBI.AUTOGEN_STRUCTURES

```
chebi_baseuri:autogen_structures a rdf:Class .
chebi_baseuri:autogen_structures_id a rdf:Property .
chebi_baseuri:autogen_structures_structure_id a rdf:Property .
chebi_baseuri:autogen_structures/2680136 a chebi_baseuri:autogen_structures;
        vocab:autogen_structures_id "2680136"^^xsd:integer;
        vocab:autogen_structures_structure_id chebi_baseuri:structures/2707183 .
```

Table 6.11: Resulting triples after evaluating the SQL query shown in Table 6.9 upon the sample row of Table 6.2

6.2.2 Transform

As in **D2R**, a set of transformations on the relational data must be applied during the extraction stage. The first transformation that we applied was the same as in the case of **D2R** in which we aggregated values of the same nature. For this type of transformation and the relational table DOCS of the **ChEMBL** dataset, the appropriate SQL query is the one shown in Table 6.12.

Contrary to **D2R**, we can take advantage of the fact that pure SQL is used in order to translate relational data to their appropriate RDF triples, in order to create more expressive transformations, for example in the column level by using GROUP BY aggregate functions.

```
'docs'=>"
        SELECT docs.doc_id as id,
                docs.doc_id AS 'vocab:docs_doc_id^^xsd:integer',
                CONCAT(docs.journal, ', ',
                        docs.year, ', ',
                        docs.volume, '(',
                        docs.issue, ') ',
                        docs.first_page, '-',
                        docs.last_page) AS 'vocab:docs_important_info',
                docs.pubmed_id AS 'vocab:docs_pubmed_id^^xsd:integer',
                docs.doi AS 'vocab:docs_doi',
                docs.title AS 'vocab:docs_title',
                docs.doc_type AS 'vocab:docs_doc_type',
                docs.chembl_id AS 'vocab:docs_chembl_id'
        FROM docs"
```

Table 6.12: **Triplify** Mapping query for relational table CHEBI.DOCS

Such a transformation has been applied in the relational table MOLECULE_SYNONYMS of the **ChEMBL** database and the corresponding SQL query is shown in Table 6.13. This query aggregates all synonyms grouped by their molregno identifier. For example, taking into consideration the SQL query shown in Table 6.13 and the relational rows shown in Table 6.14 the produced RDF triples are the ones shown in Table 6.15.

```
'molecule_synonyms'=>"
        SELECT molecule_synonyms.molregno as id,
                GROUP_CONCAT(molecule_synonyms.synonyms, ' (',
                        molecule_synonyms.syn_type, ')')
                                AS 'vocab:molecule_synonyms_synonyms_and_type',
                molecule_synonyms.research_stem
                        AS 'vocab:molecule_synonyms_research_stem',
                molecule_synonyms.molregno
                        AS 'vocab:molecule_synonyms_molregno'^^xsd:integer
        FROM molecule_synonyms
        GROUP BY molregno"
```

Table 6.13: **Triplify** Mapping query for the relational table MOLECULE_SYNONYMS

MOLECULE_SYNONYMS

molregno	synonyms	syn_type	research_stem
1620	Brumetadina	TRADE_NAME	NULL
1620	Cimetidine	BAN	NULL
1620	Cimetidine	FDA_ALTERNATIVE_NAME	NULL

Table 6.14: Sample rows for CHEMBL.MOLECULE_SYNONYMS relation table

The final transformation that is applied is the inter-linkage of **ChEBI** and **ChEMBL**, which is similar to **D2R**.

```
chembl_baseuri:molecule_synonyms a rdf:Class .
chembl_baseuri:molecule_synonyms_synonyms_and_type a rdf:Property .
chembl_baseuri:molecule_synonyms_research_stem a rdf:Property .
chembl_baseuri:molecule_synonyms_molregno a rdf:Property .
chembl_baseuri:molecule_synonyms_synonyms_and_type/1620
  vocab:molecule_synonyms_molregno "1620"^^xsd:integer;
  vocab:molecule_synonyms_synonyms_and_type
    "Brumetadina (TRADE_NAME), Cimetidine (BAN), Cimetidine (FDA_ALTERNATIVE_NAME)" .
```

Table 6.15: Resulting triples after evaluating the SQL query shown in Table 6.13 in the sample rows of Table 6.14

6.2.3 Load

As in the case of **D2R**, all the data, after Export and Transform has finished, are stored in the disk and then loaded to Virtuoso Server.

6.3 Virtuoso Linked Data Views

In the case of Virtuoso Views, the process of ETL can be omitted because the extract and transform processes are performed within the Virtuoso database engine using a SPARQL-based Meta Schema Language to provide RDBMS-to-RDF mapping functionality (called Linked Data Views). Through this language we can map relational database schema components, such as tables, views, columns, rows or foreign keys, to Classes, Attributes, Relationships, and Instances defined by RDF Schemas or OWL Ontologies.

Therefore, all that needs to be done in order to define the transformations described in previous sections is to define appropriate SQL queries that generate materialized SQL views. On top of these SQL views and the existing SQL tables of both the **ChEMBL** and **ChEBI** databases, the Linked Data Views of Virtuoso are built. The SQL Views that we defined for this purpose are shown in Table 6.16.

6.4 Experiments for ETL process

For both **D2R** and **Triplify** we measured the time it took ETL process to finish. Also we got some statistics about CPU and I/O utilization of the process.

6.4.1 D2R

In **D2R**, the entire process of Extract, Transform and Load took about one hour as shown in Figure 6.1. Regarding the stages of Extract and Transform, **D2R** has the benefit that all the mapping components of its mapping language are discrete defined in the level of class maps and property bridges as described in Section 6.1.1. So, the mappings can be divided in separate files in order to parallelize this process. By following this approach, we divided the mappings of **ChEBI** and **ChEMBL** datasets in 16 files, and we run in parallel 16 processes, one for each file. By doing this, the extraction along with the transformation process took about 14 minutes in which 281.348.982 triples were produced. In addition, as shown in Figure 6.1 the parallelization of the process led to almost full CPU and I/O usage. Loading the above triples in **Virtuoso** took 41 minutes.

6.4.2 Triplify

In **Triplify**, the entire process of Extract, Transform and Load took about one and a half hour as shown in Figure 6.2. Contrary to **D2R** where mappings are *fine grained* (i.e, can be specified per table and column),

```
-- docs table transformed
CREATE VIEW DB.DBA.CHEMBL_docs_transformed AS
SELECT doc_id,
        CONCAT(journal, ', ',
                year, ', ',
                volume, '(',
                issue, ') ',
                first_page, '-',
                last_page) AS docs_important_info,
        pubmed_id,
        doi,
        title,
        doc_type,
        chembl_id
        FROM DB.DBA.CHEMBL_docs;

-- molecule_synonyms table transformed
CREATE VIEW DB.DBA.CHEMBL_molecule_synonyms_transformed AS
SELECT molregno,
        GROUP_CONCAT(CONCAT(synonyms,' (',syn_type, ')'),',')
                AS molecule_synonyms_synonyms_and_type,
        research_stem
FROM DB.DBA.CHEMBL_molecule_synonyms
GROUP BY molregno;
```

Table 6.16: SQL views for Virtuoso

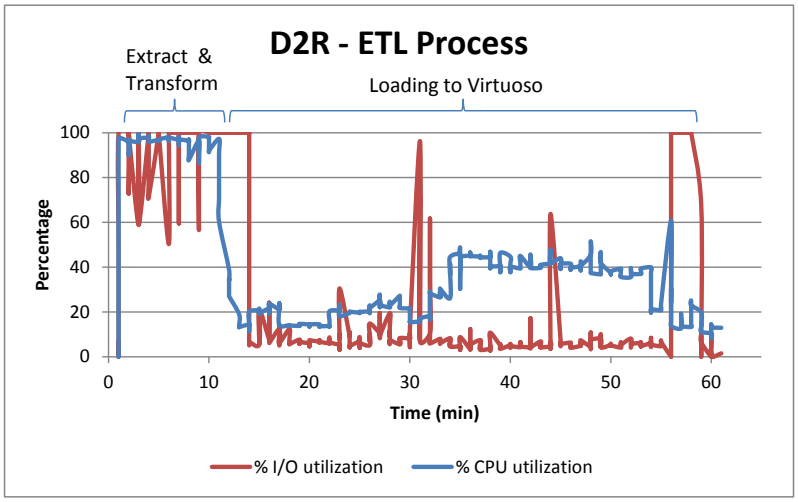

Figure 6.1: ETL process for **D2R**

Triplify enforces restrictions on the SQL query structure. Hence, the mapping queries cannot be specified in different files in order to parallelize the extraction process. This can only be done at table level (i.e., specify one mapping query per table). So, we divided the mapping SQL queries in 8 files, taking into consideration the size of each table. By doing this, the extraction of RDF triples, along with the transformation process, took about 50 minutes in which 267.960.975 triples triples were produced.

One can observe that the time **Triplify** needed to perform the *extract-transform* sequence is larger than

the respective time for **D2R** (13 min.). Note that we were able to parallelize this task for **D2R** in 16 different jobs, whereas this was not the case for **Triplify** (8 different jobs running in parallel - one job per set of tables). In addition, in our datasets we have two large tables that produced 65% of the resulting triples; to extract those triples we were able to run 2 jobs in parallel (one for each such table). Note also that there is a difference in the resulting triples for the two systems. The reason is that **Triplify** does not produce triples for properties with zero values for the attributes (not the case for **D2R**). For instance, assuming the sample row of relational table `ASSAY2TARGET` of **ChEMBL** dataset shown in Table 6.17, the triples for columns `ASSAY2TARGET.complex` and `ASSAY2TARGET.multi` will not be generated.

Loading the generated triples in **Virtuoso** took 33 minutes (expected since the size of the resulting triples is smaller).

`ASSAY2TARGET`

assay_id	tid	relationship_type	complex	multi	confidence_score	curated_by
1	12052	H	0	0	8	Autocuration

Table 6.17: Sample rows for `CHEMBL.ASSAY2TARGET` relation table

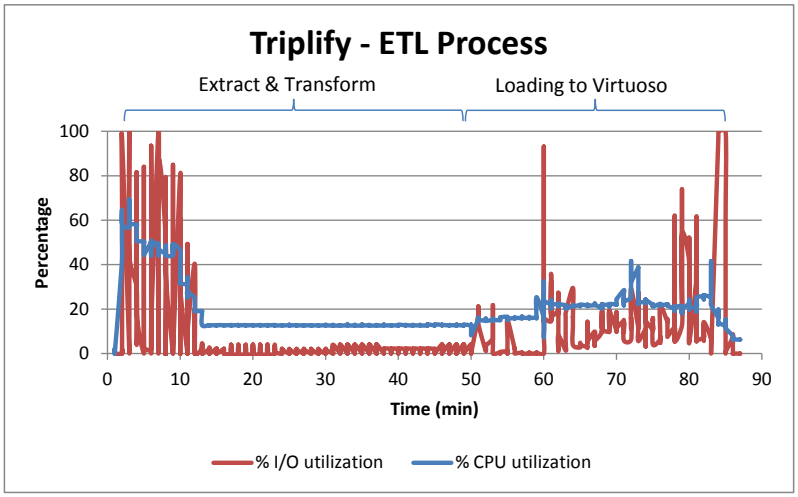

Figure 6.2: ETL process for **Triplify**

7 Conclusion

This deliverable focused on identifying the main challenges (choke points) related to three semantical tasks, namely *reasoning*, *instance matching* and *ETL processing*. To identify these choke points, we tested existing state-of-the-art systems in their respective tasks, using datasets from the Semantic Publishing Domain (as provided by the respective Task Force) and the Life Sciences domain, appropriately modified in order to enhance the challenges associated with each respective task.

The identification of a choke point involves understanding the most important challenges that current systems face in their respective tasks, in order to be included as (hidden) challenges in the benchmarks that we will design; the ultimate goal is to encourage systems to address these challenges, thus stimulating and encouraging technological progress.

As far as reasoning is concerned, we focused on *conformance* choke points to test whether the RDF engines were able to implement the semantics of OWL 2 RL fragment of the Ontology Web Language (OWL). As expected, our experiments show that **OWLIM** supports more reasoning constructs than **Virtuoso**. Some preliminary performance results (not reported in this deliverable) showed that **OWLIM** outperforms **Virtuoso** for reasoning intensive queries. Recall that **OWLIM** implements forward reasoning, that is it materializes the closure of the dataset; reasoning intensive queries are then evaluated against this dataset. On the other hand, **Virtuoso** computes at query time the inferred triples needed by a reasoning intensive query, hence adding an overhead during query processing.

Our analysis of the instance matching choke points targeted on identifying cases where the tested systems perform poorly in the matching task (with respect to the golden standards) under a set of well defined evaluation criteria (performance, precision, recall, F-measure etc.), for different parameterizations of the tested systems. Our experiments show that existing systems do not handle well very large datasets, and do not take into account any schema or vocabulary information during the matching process.

The ETL choke point analysis identified transformations that stressed the tested systems. We measured the performance of the various tools, as well as the richness of possible transformations that can be defined in each tool. The main conclusions from our work on ETL choke points were that some transformations are not supported by all systems, and that the additional extraction and loading overhead imposed by the ETL process for **D2R** and **Triplify** is large, causing a disadvantage compared to **Virtuoso Views** that avoids it.

This deliverable is intended as a prelude to the upcoming full benchmark specifications for the above tasks (reasoning, instance matching, ETL processing) which are planned to be reported in M24 of this project, as part of Deliverables D4.4.2, D4.4.3 and D4.4.4 respectively; the choke points identified here will be used as the building blocks of said benchmarks.

REFERENCES

[1] I. Bhattacharya and L. Getoor. *Entity resolution in graphs. Mining Graph Data*. Wiley and Sons, 2006.

[2] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. `www.w3.org/TR/2004/REC-rdf-schema-20040210`, 2004.

[3] D2R Server: Accessing databases with SPARQL and as Linked Data. http://d2rq.org/d2r-server.

[4] Frederick J. Damerau. *Markov Models and Linguistic Theory: : An Experimental Study of a Model for English (Janua Linguarum Series Minor No 95)*. Mouton, 1971.

[5] E. Daskalaki and D. Plexousakis. OtO Matching System: A Multi-strategy Approach to Instance Matching. *AISE*, 2012.

[6] P. De Matos, R. Alcántara, A. Dekker, M Ennis, J. Hastings, K. Haug, I. Spiteri, S. Turner, and C. Steinbeck. Chemical entities of biological interest: an update. *Nucleic acids research*, 38(suppl 1), 2010.

[7] A. K. Elmagarmid, P. G. Ipeirotis, and V.S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1), 2007.

[8] I. Fundulaki. D1.1.1: Overview and Analysis of Existing Benchmark Frameworks. LDBC Deliverable D1.1.1, 2013.

[9] I. Fundulaki. D2.2.2: Data Generator. LDBC Deliverable D2.2.2, 2013.

[10] A. Gubichev and T. Neumann. D2.2.1: Analysis and Classification of Choke Points. LDBC Deliverable D2.2.1, 2013.

[11] D. Gusfield. Algorithms on strings, trees, and sequences: computer science and computational biology. *Cambridge University Press*, 1997.

[12] L. Harland. Open PHACTS: A Semantic Knowledge Infrastructure for Public and Commercial Drug Discovery Research. *Knowledge Engineering and Knowledge Management Lecture Notes in Computer Science*, 7603, 2012.

[13] P. Hayes. RDF semantics. `http://www.w3.org/TR/rdf-mt/`, 2004. W3C Recommendation, 10 February 2004.

[14] R. Isele, A. Jentzsch, and C. Bizer. Silk Server - Adding missing Links while consuming Linked Data. In *COLD*, 2010.

[15] A. Jentzsch, R. Isele, and C. Bizer. Silk: Generating RDF Links while publishing or consuming Linked Data. In *ISWC*, 2010. Poster.

[16] C. Li, L. Jin, and S. Mehrotra. Supporting efficient record linkage for large data sets using mapping techniques. In *WWW*, 2006.

[17] F. Manola, E. Miller, and B. McBride. RDF Primer. `www.w3.org/TR/rdf-primer`, February 2004.

[18] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language. `http://www.w3.org/TR/owl-features/`, 2004.

[19] A. Miles and S. Bechhofer. SKOS Simple Knowledge Organization System Reference. `http://www.w3.org/TR/skos-reference/`. W3C Recommendation.

[20] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language Profiles (Second Edition). http://www.w3.org/TR/owl2-profiles/. W3C Recommendation 11 December 2012.

[21] A.-C. Ngonga Ngomo and Soren Auer. LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. *IJCAI*, 2011.

[22] J. Noessner, M. Niepert, C. Meilicke, and H. Stuckenschmidt. Leveraging Terminological Structure for Object Reconciliation. In *ESWC*, 2010.

[23] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. `www.w3.org/TR/rdf-sparql-query`, January 2008.

[24] Y. Raimond, T. Scott, S. Oliver, P. Sinclair, and M. Smethurst. Use of Semantic Web technologies on the BBC Web Sites. `http://3roundstones.com/led_book/led-raimond-et-al.html`.

[25] N. Redaschi and UniProt Consortium. UniProt in RDF: Tackling Data Integration and Distributed Annotation with the Semantic Web. In *Biocuration Conference*, 2009.

[26] A. Singhal. Modern Information Retrieval: A Brief Overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2001.

[27] BBC Ontologies. http://www.bbc.co.uk/ontologies/.

[28] CheMBL DB. https://www.ebi.ac.uk/chembldb/.

[29] ChemSpider. http://www.chemspider.com/.

[30] ConceptWiki. http://ops.conceptwiki.org/.

[31] DrugBank. http://www.drugbank.ca.

[32] Gene ontology (GO). http://www.geneontology.org/.

[33] MeSH. http://www.nlm.nih.gov/mesh/.

[34] OpenLink. http://www.openlinksw.com/.

[35] Open Pharmacological Space (Open PHACTS). http://www.openphacts.org.

[36] OWLIM. http://www.ontotext.com/owlim.

[37] Protein Data Bank - PDB. http://www.rcsb.org/pdb/home/home.do.

[38] Triplify. http://triplify.org/Overview.

[39] UniProt. http://www.uniprot.org/.

[40] Virtuoso. http://virtuoso.openlinksw.com/.

[41] Wordnet. http://wordnet.princeton.edu/.

[42] P.-N. Tan, M. Steinbach, and V. Kumar. Introduction to Data Mining. *ISBN 0-321-32136-7*, 2005.

[43] Mapping Relational Data to RDF with Virtuoso's RDF Views. http://virtuoso.openlinksw.com/whitepapers/relational%20rdf%20views%20mapping.html.

[44] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and Maintaining Links on the Web of Data. In *ISWC*, 2009.

[45] W3C OWL Working Group. OWL 2 Web Ontology Language. http://www.w3.org/TR/owl2-overview/, 2012.

[46] Antony J. Williams and et al. Open PHACTS: semantic interoperability for drug discovery. *Drug Discovery Today*, 17, 2012.