EU Project No:601043 (Integrated Project (IP))

DIACHRON

Managing the Evolution and Preservation of the Data Web DIACHRON

Dissemination level:	Public
Type of Document:	Report
Contractual date of delivery:	M16
Actual Date of Delivery:	
Deliverable Number:	D5.2
Deliverable Name:	Software prototype of the crawling, ranking and appraisal services
Deliverable Leader:	UBONN
Work package(s):	WP5
Status & version:	1.0
Number of pages:	50
WP contributing to the deliverable:	WP5
WP / Task responsible:	T5.2, T5.3
Coordinator (name / contact):	Jeremy Debattista / jeremy.debattista@iais-extern.fraunhofer.de (UBONN)
Other Contributors:	Christoph Lange, Natalja Friesen, and Santiago Londoño (UBONN), Ioannis Chrysakis, Kostas Stefanidis, and Giorgos Flouris (FORTH)
EC Project Officer:	Federico Milani

Keywords: data quality framework, quality metadata, quality metrics, dataset ranking, dataset crawling

Abstract:
The report describes the Data Quality Framework required to analyse and assess datasets for quality which would enable crawling and ranking.

Document History			
Ver.	Date	Contributor(s)	Description
0.1	24.06.2014	Jeremy Debattista	Created TOC and LaTeX Setup
0.5	18.07.2014	All Contributors	Added contribution from partners
0.8	24.07.2014	Jeremy Debattista / Christoph Lange	Finalisation of first draft
1.0	31.07.2014	Jeremy Debattista	Reviewing comments and finalising final draft

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

1 Introduction

1.1 Scope and Objectives

This document describes the Diachron quality assessment service and its general applications. Taking a broad view of quality as "fitness for use" [14], we have designed a generic, extensible quality assessment framework, which is intended to serve all purposes envisaged in the Diachron project but also beyond. We initially support quality assessment on RDF data, no matter whether they are streamed, e.g. from a big dataset dump, or obtained from a SPARQL endpoint. We represent the output of quality assessment as an RDF graph of *quality metadata*, which can be stored as annotations to the original dataset. This approach enables flexible reuse of the quality metadata for visual quality analysis, quality-based filtering and ranking of datasets, and many other applications, without unnecessary recomputation. This document also describes the implementation of a large number of concrete quality metrics within the framework. We have so far implemented a substantial number of the quality metrics required for the Diachron pilot applications, focusing on those metrics that can be computed by analysing the dataset itself, the vocabularies it uses, and its immediate outgoing links to other datasets. The dataset itself is analysed in a triple-by-triple streaming fashion, which ensures scalability over big datasets at the expense of ruling out certain complex quality metrics.

1.2 Context of this Document

The overall architecture of Diachron, as outlined in Deliverable 6.1 [17, section 5], comprises i) a platform layer of modules that access and operate on diachronic data, ii) an integration layer that makes these modules accessible as web services, as well as iii) pilot applications that solve domain-specific problems and make use of the services to handle data specific to these problems. An initial specification of the interfaces of these services – web services with a RESTful API and JSON input/output, to be exact – has also been provided in Deliverable 6.1. Here, we describe the *implementation* of the *quality assessment module* and the *quality assessment service* and explain, in a generic, domain-independent way, how to use it to assess the quality of data.

The concrete quality metrics that we have so far implemented within our quality assessment framework, and which the service can compute for any given dataset, follow the definitions from Deliverable 5.1 [25]. We prioritised the implementation of those metrics that Deliverable 5.1 identified relevant to Diachron [25, section 4].

1.3 Document Structure

Section 2 defines a generic, extensible data quality framework, which is not specific to any Diachron application domain. The framework allows for implementing arbitrary concrete quality metrics; any such metric is implemented partly in Java and partly as an extension of the data quality (daQ) core vocabulary. The input dataset, currently assumed to be represented in RDF, is fed into all quality metric implementations of the user's choice; the output is represented as a set of RDF metadata in terms of daQ and its concrete, metric-specific extensions, so that it can be stored and reused flexibly. This core quality assessment step is accessible through a web service interface. Supported uses of a dataset's quality metadata include visual quality analysis (over multiple revisions of a dataset if desired) as well as filtering and ranking datasets by their quality.

Section 3 lists the libraries that served as technical prerequisites for implementing the quality assessment service.

Section 4 describes in detail the general processes of data quality assessment and quality-based ranking, the design of a self-contained user interface giving access to the quality assessment service (optionally also the cleaning service described in Deliverables 3.1 [8] and 3.2 [9], as well as all concrete metrics that we have implemented so far, with a special focus on those required for the Diachron pilot applications.

2 Data Quality Framework

2.1 High-Level Architecture

The purpose of the Quality Framework is to provide an integrated platform that:

1. assesses Linked Open Data (via RDF Dumps or SPARQL endpoints) in a scalable manner;
2. provides queryable quality metadata on the assessed datasets;
3. provides visualisations on the quality metadata.

Furthermore, we aim to create an infrastructure and a platform that (i) can be easily extensible by different third parties by creating their custom and more specific pluggable metrics required to assess their particular dataset domain, and (ii) having the the necessary ontology framework to represent the metadata about the quality of the assessed linked datasets.

Currently, there is no uniform infrastructure to assess quality of a linked dataset, allowing the extension or redefinition of custom-specific metrics such as those required by the DIACHRON use cases. Tools such as Trellis [11], WIQA [1] and Sieve [18] implement a number of metrics but lack flexibility w.r.t. the level of automation, and user friendliness [26]

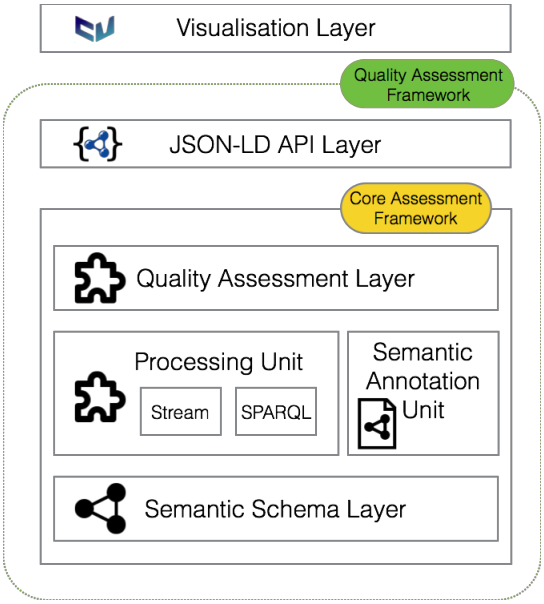

Figure 1: Quality Framework High Level Architecture Design

Figure 1 illustrates the high level architecture of the Quality Framework. The core assessment framework is made up of four modules: *Semantic Schema Layer*, *Processing Unit*, *Semantic Annotation Unit*, *Quality Assessment Layer*, and an optional *Visualisation Layer* which we will discuss in Section 4.3.

2.1.1 Semantic Schema Layer

The Quality Framework is based on semantic technologies and thus has an underlying semantic vocabulary layer which currently consists of two ontologies: (i) the Dataset Quality Ontology (daQ)[1]; and (ii) the Quality Problem Report Ontology (qr)[2]. The former describes the quality metadata representation whilst the latter describes quality

[1] http://purl.org/eis/vocab/daq
[2] http://purl.org/eis/vocab/qr

problems found in the dataset itself. The semantic schema layer is meant to be domain independent, where it could be reused in other similar frameworks. The daQ ontology (cf. Section 2.3) is the core vocabulary of this schema layer, and any other ontology part of this layer builds upon it.

The daQ ontology is a comprehensive generic vocabulary framework, based on three abstract concepts (Category, Dimension and Metric). Any newly implemented specific metric should have its representation in RDF, extending the daQ ontology. In DIACHRON, all metrics are defined in the Diachron Quality Metric vocabulary (DQM)[3]. Such vocabularies are straightforward to integrate into the Quality Framework, since they adopt and extend the generic daQ vocabulary (by inheriting classes and properties) as the way quality metadata is represented (cf. Section 2.3.3). The Quality Problem Report Ontology (qr) is made up of two classes a `qr:QualityReport` and `qr:QualityProblem`. The former represents a report on the problems detected during the assessment of quality on a dataset, whilst the latter represents the individual quality problems making up that report. Four properties are also defined in the ontology. The `qr:computedOn` represents the dataset URI on which quality assessment has been made. This property is attached to a `qr:QualityReport`. `qr:hasProblem` links a `qr:QualityProblem` to a `qr:QualityReport`. The mentioned property identifies problem instances in a report. Each `qr:QualityProblem` `isDescribedBy` an instance of a `daq:Metric`[4]. The property `qr:problematicThing` represents the actual problematic instance from the dataset. This could be a list of resources (`rdf:Seq`) or a list of reified statements. Listing 1 represents an excerpt from a typical dataset showing the instance of `ex:JoeDoe` who is a `foaf:Researcher` working for `ex:UniBonn`. In these two instances there are three problematic triples:

(A) ⟨ **`ex:JoeDoe a foaf:Researcher`** ⟩ - The problem in this triple is caused by the usage of an undefined class, in this case `foaf:Researcher`;

(B) ⟨ **`ex:JoeDoe rdfs:label "JoeDoe"`** ⟩ - The literal ("JoeDoe") in the triple causes the malformed capitalisation metric to point out a problem in this triple (as labels should be written in a human-friendly style);

(C) ⟨ **`ex:UniBonn rdfs:label "UniBonn"`** ⟩ - The literal ("UniBonn") in the triple causes the malformed capitalisation metric to point out a problem in this triple.

Listing 2 represent these three problems using the Quality Problem Report ontology.

```
ex:JoeDoe a foaf:Researcher ;
 rdfs:label "JoeDoe" ;
 ex:worksFor ex:UniBonn .

ex:UniBonn rdfs:label "UniBonn" ;
 foaf:name "University of Bonn" .
```

Listing 1: An excerpt of a typical dataset

```
ex:QualityReport a qr:QualityReport ;
 qr:computedOn <uri:datasetResearchers> ;
 qr:hasProblem <#prob1>,<#prob2>,<#prob3> .

<#prob1> a qr:QualityProblem ;
 qr:isDescribedBy <urn:metric/UndefinedClasses123> ;
 qr:problematicThing [
    diachron:hasSubject ex:JoeDoe ;
  diachron:hasPredicate rdf:type ;
  diachron:hasObject foaf:Researcher ;
 ] .

 <#prob2> a qr:QualityProblem ;
  qr:isDescribedBy <urn:metric/Capitalisation789> ;
  qr:problematicThing [
    diachron:hasSubject ex:JoeDoe ;
    diachron:hasPredicate rdfs:label ;
    diachron:hasObject "JoeDoe" ;
  ] .

  <#prob3> a qr:QualityProblem ;
```

[3] `http://purl.org/eis/vocab/dqm`
[4] refer to Section 2.3

```
  qr:isDescribedBy <urn:metric/Capitalisation789> ;
  qr:problematicThing [
   diachron:hasSubject ex:UniBonn ;
   diachron:hasPredicate rdfs:label ;
   diachron:hasObject "UniBonn" ;
  ] .
```

Listing 2: A corresponding Quality Report for Listing 1

2.1.2 Processing Unit

The Processing Unit is an integral part of the Quality Framework. In this framework, we provide two main scalable processing units: a sequential stream processor (cf. Section 2.2) and a SPARQL processor[5]. The former streams triples from RDF data dumps one by one in a sequential fashion. The latter allows the framework to assess quality on data that is available only in SPARQL endpoints. This unit is one of the two extensible modules (the other being Quality Assessment Layer) in the Quality Framework. For DIACHRON, the plan is to extend the sequential stream processor, enabling the de-reification of RDF statements into RDF triples.

Typically, an initialised processor has 2 inputs: the dataset URI (for the sequential stream processor) or the dataset SPARQL endpoint (in the case of the SPARQL processor), and a metric configuration file selecting the metrics to be computed on the dataset. Listing 3 shows an example of a typical metric configuration file.

```
@prefix diachron:     <http://www.diachron-fp7.eu/diachron#> .
@prefix qf:     <http://www.diachron-fp7.eu/qualityFramework#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfg:  <http://www.w3.org/2004/03/trix/rdfg-1> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .

_:metricAssessment a qf:metricConfiguration ;
  diachron:metric "accessibility.availability.SPARQLAccessibility" ;
  diachron:metric "accessibility.availability.RDFAccessibility" ;
  diachron:metric "accessibility.availability.Dereferencibility" ;
  diachron:metric "accessibility.performance.DataSourceScalability" ;
  diachron:metric "accessibility.performance.HighThroughput" ;
  diachron:metric "accessibility.performance.LowLatency" ;
  diachron:metric "intrinsic.accuracy.DefinedOntologyAuthor" ;
  diachron:metric "intrinsic.accuracy.SynonymUsage" ;
  diachron:metric "intrinsic.accuracy.POBODefinitionUsage" ;
  diachron:metric "intrinsic.consistency.EntitiesAsMembersOfDisjointClasses" ;
  diachron:metric "intrinsic.consistency.HomogeneousDatatypes" ;
  diachron:metric "intrinsic.consistency.MisplacedClassesOrProperties" ;
  diachron:metric "intrinsic.consistency.ObsoleteConceptsInOntology" ;
  diachron:metric "intrinsic.conciseness.OntologyVersioningConciseness" ;
  diachron:metric "dynamicity.timeliness.TimelinessOfResource" ;
  diachron:metric "dynamicity.currency.CurrencyDocumentStatements" ;
  diachron:metric "dynamicity.currency.TimeSinceModification" ;
  diachron:metric "representational.understandability.HumanReadableLabelling" ;
  diachron:metric "representational.understandability.LowBlankNodeUsage" .
```

Listing 3: An typical metric configuration file

Each data processor in the Quality Framework has a defined 3-stage procedure (Listing 4): (i) processor initialisation; (ii) processing; and (iii) memory clean up. In the first process (processor initialisation), the processor create the necessary objects in memory to process data and load the required metrics that are instructed in the configuration file. Once the initialisation is ready, the processing is done by passing the streamed triples into the metrics. A final memory cleanup ensures that no unused objects are using unnecessary computational resources.

```
public interface IOProcessor {
  // Initialise the io processor with the necessary in-memory objects and metrics
  void setUpProcess();

  // Process the dataset for quality assessment
  void startProcessing() throws ProcessorNotInitialised;

  // Cleans up memory from unused objects after processing is finished
```

[5]The SPARQL processor is still being investigated and will not be ready by the deliverable deadline.

```
  void cleanUp() throws ProcessorNotInitialised;
}
```

Listing 4: IO Processor Interface

2.1.3 Quality Assessment Layer

The Quality Assessment Layer is undeniably the most important layer of the Quality Framework. The framework can be extended by any third party providing their own custom specific metric. This is already done in the DIACHRON project, where a number of metrics (cf. Section 4.2) required to assess the various use cases specified in [25] are implemented. The Quality Assessment Layer provides two interfaces and an abstract class (cf. Figure 2) which facilitate the quality framework to be a pluggable and extensible platform. The interface `QualityMetric` is the core interface class which describes the metric classes. Each metric implementing this interface, must implement the following methods:

compute assesses the quad[6]/triple which is passed by the stream processor by the defined metric;

metricValue returns the value computed by the quality metric;

toDAQTriples returns a list of daQ triples, containing quality metadata about the assessed metric, which will be used to create the quality metadata for the assessed dataset;

getMetricURI returns the URI of the Quality Metric from the ontology description (e.g. `http://purl.org/eis/vocab/dqm#DereferenceablityMetric`);

getQualityProblems returns a typed list of either Resources or Quads, which will be used to create a quality report of the metric;

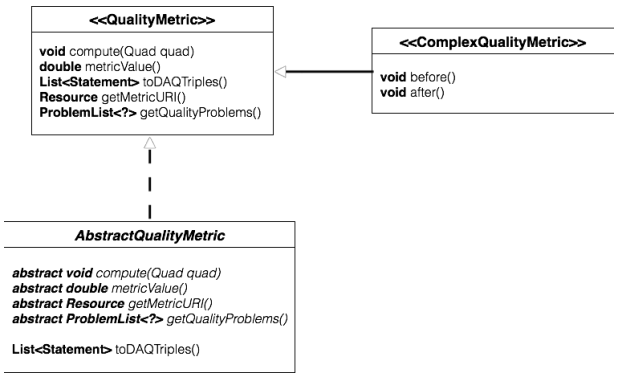

Figure 2: Quality Assessment Layer Class Diagram – A Quality Framework as a Pluggable Platform

Furthermore, computing a metric might require some pre-processing or post-processing. Therefore, an interface (`ComplexQualityMetric`) extending `QualityMetric` was developed. This interface allow metric developers to perform such processing using the **`void`** `before(Object ... args)` and **`void`** `after(Object ... args)` methods.

In order to facilitate further such development of pluggable metrics, the `AbstractQualityMetric` class was developed, implementing the `QualityMetric` interface. In this abstract class, the method `List⟨Statement⟩ toDAQTriples()` is implemented, generating quality metadata observation instances (cf. Section 2.3) for the metric being assessed. Having this generic implementation of the method would ensure that the quality metadata is described correctly with respect to its definition in the Dataset Quality Ontology.

[6] A quad is a triple with an added context value (eg. a named graph URI where the triple is found).

2.1.4 Semantic Annotation Unit

The Semantic Annotation Unit takes the triples generated by the `toDAQTriples()` method in order to create the quality metadata for an assessed dataset. This unit will handle the necessary communications with the datastore to insert the generated quality metadata in the dataset itself. The unit provides a number of helper classes that provide inferencing queries on vocabularies that describe metrics (in the case of DIACHRON, the DQM vocabulary) based on the core ontology - Dataset Quality Ontology (daQ). Therefore, RDF descriptions of metrics extending the daQ (cf. Section 2.3.3) ontology are absolutely required. These inferencing queries enable the framework to create a complete metadata description (cf. Section 2.3) of an assessed quality metric.

2.2 Sequential Stream Processor

In order to accurately assess linked datasets for quality measures, the assessment should be performed on all triples in the datasets. One must keep in mind that the computation of metrics on large datasets might be expensive; thus, such stream processors computing dataset's quality must be scalable. Figure 3 shows closer look towards the quality assessment process is illustrated. A user chooses a dataset and the metrics which are required for the assessment of quality. The submitted information is passed to the Quality Framework via its API and initialise the processing unit (stream processor) in the core framework. The stream processor is then initialised by: (1) creating the necessary objects in memory, and (2) initialising the chosen metrics. In Figure 3, "Metric 1" is shadowed out – to illustrate that it was not chosen by the user for this particular use case. Once the objects are created, the stream processor fetches the dataset and streams quads one by one to all initialised metrics in parallel. After all statements are assessed, the Semantic Annotation unit requests the metric value for each metric and creates (or updates – in case the dataset already has one) the Quality Graph. This named graph represents the quality metadata of a dataset using the representation defined in Section 2.3, and it is stored in the dataset itself. Having this metadata, it will allow us to rank and crawl datasets based on different quality attributes.

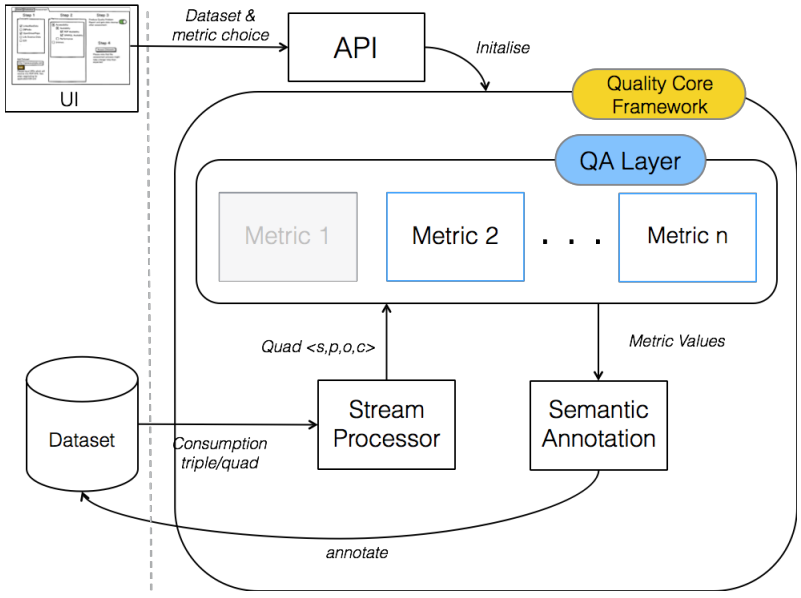

Figure 3: Closer look at the Quality Assessment process

2.2.1 The Initialisation Process – `setUpProcess()`

The *setUpProcess()* method initialises the stream processor. Algorithm 1 is the pseudocode of the the process. The sequential stream processor starts its initialisation by first trying to identify the serialisation used by the available

RDF data dump. The method `guessRDFSerialisation` analyses the file serialisation by mapping the file name to one of the RDF serialisations for which Jena has built-in support (e.g. RDF/XML, NTriples, Turtle, NQuads, etc...) According to the file's serialisation, the process then assigns different types of `PipedRDFIterator`[7] and either a`PipedQuadsStream`[8] or `PipedTriplesStream`[9] These two objects are required for the scalable execution of the sequential stream processor as together they act as a the "producer"[10] of sequential RDF triples from the RDF data dump. Once these are initialised, a flag is set to true to signal that the processor unit is in progress. Finally, the chosen metrics are loaded into memory. Metrics are loaded dynamically at runtime using `newInstance()`[11].

Algorithm 1 The Initialisation of the Sequential Stream Process

```
1:  procedure SETUPPROCESS
2:      rdfSerialisation = guessRDFSerialisation(datasetURI) ;
3:      if rdfSerialisation is Quads then
4:          iterator = new PipedRDFIterator⟨Quad⟩() ;
5:          rdfStream = new PipedQuadsStream((PipedRDFIterator⟨Quad⟩) iterator) ;
6:      if rdfSerialisation is Triple then
7:          iterator = new PipedRDFIterator⟨Triple⟩() ;
8:          rdfStream = new PipedTriplesStream((PipedRDFIterator⟨Triple⟩) iterator) ;
9:      set initialised boolean to true ;
10:     loadMetrics() ;
```

2.2.2 The Processing of Triples – `startProcessing()`

After the initialisation process, the method `startProcessing()` is invoked. The RDFStream `rdfStream` object starts parsing the RDF dump and producing triple or quad statements in the `iterator`. On a different thread, the "consumer" – the sequential stream processor – consumes these statements from the `iterator`, converts them into quads of $\langle s, p, o, c \rangle$, and passes them to all initialised metrics. The consumption process is repeated until all statements are exhausted from the `iterator`. The semantic annotation unit is then signalled to start its annotation. Algorithm 2 describes this process in pseudocode.

Algorithm 2 Processing Triple/Quad Statements

```
1:  procedure STARTPROCESSING
2:      if initialised == false then
3:          throw exception ;
4:      create new producer thread for rdfStream ;
5:      while (iterator has another statement) do
6:          quad = Object2Quad(iterator.next()) ;
7:          pass quad to all metrics and compute metric ;
8:      invoke semantic annotation unit ;
```

2.2.3 Clean Up – `cleanUp()`

The final process is to clean up the objects from memory. The processor follows a simple approach by assigning `null` to all objects and shutting down all running threads.

[7] `https://jena.apache.org/documentation/javadoc/arq/org/apache/jena/riot/lang/PipedRDFIterator.html`

[8] `https://jena.apache.org/documentation/javadoc/arq/org/apache/jena/riot/lang/PipedQuadsStream.html`

[9] `https://jena.apache.org/documentation/javadoc/arq/org/apache/jena/riot/lang/PipedTriplesStream.html`

[10] As in the producer in the "Producer-Consumer problem" `http://en.wikipedia.org/wiki/Producer?consumer_problem`. The consumer is on a separate thread, feeding the metrics.

[11] `http://docs.oracle.com/javase/8/docs/api/java/lang/Class.html#newInstance--`

2.3 The Dataset Quality Ontology

The idea behind the Dataset Quality Ontology [5][12] (daQ) is to provide a comprehensive generic vocabulary framework, allowing a uniform definition of specific data quality metrics and thus suggesting how quality metadata should be represented in datasets. This metric definition would then allow publishers to attach data quality metadata with quality benchmarking results to their linked dataset. Figure 4 depicts the current state of the daQ vocabulary.

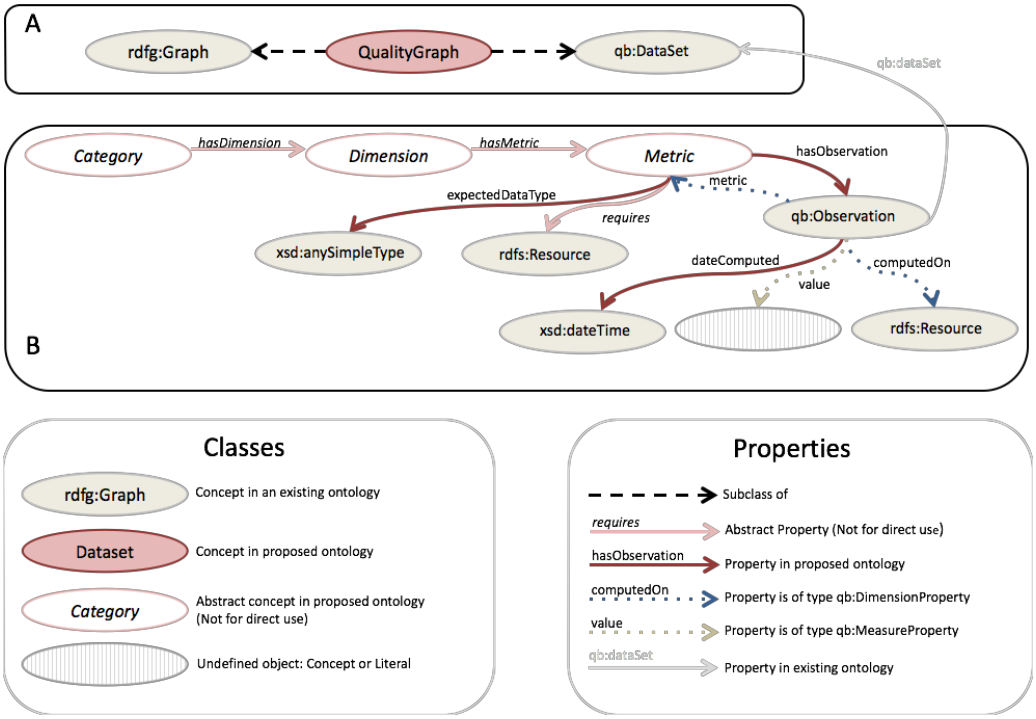

Figure 4: The extended Dataset Quality Ontology (daQ)

Using daQ, the quality metadata is intended to be stored in what we defined to be the *Quality Graph*. The latter concept is a subclass of `rdfg:Graph` [2]. This means that the quality metadata is stored and managed in a separate named graph from the assessed dataset. Named graphs are favoured due to

- the capability of separating the aggregated metadata with regard to computed quality metrics of a dataset from the dataset itself;
- their use in the Semantic Web Publishing vocabulary [3] to allow named graphs to be digitally signed, thus ensuring trust in the computed metrics and defined named graph instance. Therefore, in principle each `daq:QualityGraph` can have the following triple `:myQualityGraph swp:assertedBy :myWarrant .`

The daQ ontology distinguishes between three layers of abstraction, based on the survey work by Zaveri et al. [26]. As shown in Figure 4 Box B, a quality graph comprises of a number of different *Categories*, which in turn possess a number of quality *Dimensions*[13]. A quality dimension groups one or more computed quality *Metrics*. To formalise this, let G represent the named Quality Graph (`daq:QualityGraph`), $C = \{c_1, c_2, \ldots, c_x\}$ is the set of all possible quality categories (`daq:Category`), $D = \{d_1, d_2, \ldots, d_y\}$ is the set of all possible quality dimensions (`daq:Dimension`) and $M = \{m_1, m_2, \ldots, m_z\}$ is the set of all possible quality metrics (`daq:Metric`); where $x, y, z \in \mathbb{N}$, then:

[12] `http://purl.org/eis/vocab/daq`
[13] In this deliverable we will refer to these as quality dimensions, in order to distinguish between the data cube dimensions

Definition 1.

$$G \subseteq C,$$
$$C \subset D,$$
$$D \subset M;$$

Figure 5 shows this formalisation in a pictorial manner using Venn diagrams.

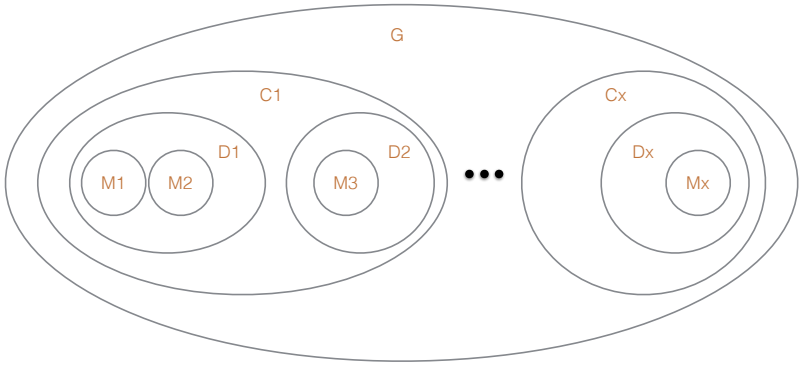

Figure 5: Venn Diagram depicting Definition 1

2.3.1 Extending daQ for Multi-Dimension Representation and Statistical Evaluation

The Data Cube Vocabulary [4] allows the representation of statistics about observations in a multidimensional attribute spaces. Multidimensional analysis of these observations, e.g. across the revision history of a dataset, would thus have required complex querying. Extending daQ with the standardised Data Cube Vocabulary allows us to represent quality metadata of a dataset as a collection of *Observations*, dimensions being the different quality metrics computed, the resources whose quality is assessed, revisions of these resources, and arbitrary further dimensions, such as the intended application scenario. It also permits applying the wide range of tools that support data cubes to quality graphs, including the CubeViz visualisation tool[14].

A *Quality Graph* is a special case of `qb:DataSet`, which allows us to represent a collection of quality observations complying to a defined dimensional structure. Each observation represents a quality metric measured out against a particular resource (e.g. a specific revision of a dataset). daQ defines the structure of such observations by the `qb:DataStructureDefinition` shown in Listing 5.

```
daq:dsd a qb:DataStructureDefinition ;
  # Dimensions: metrics and what they were computed on
  qb:component [
    qb:dimension daq:metric ;
    qb:order 1 ; ] ;
  qb:component [
    qb:dimension daq:computedOn ;
    qb:order 2 ; ] ;
  # Measures (here: metric values)
  qb:component [ qb:measure daq:value ; ] ;
  # Attribute (here: the unit of measurement)
  qb:component [
    qb:attribute sdmx-attribute:unitMeasure ;
    qb:componentRequired false ;
    qb:componentAttachment qb:DataSet ; ] .
```

Listing 5: The Data Structure Definition (Turtle Syntax)

[14] `http://cubeviz.aksw.org`

The `daq:QualityGraph` also defines one restriction that controls the property `qb:structure` and its value to the mentioned definition, thus ensuring that all *Quality Graph* instances make use of the standard definition. Having a standard definition ensures that all *Quality Graph*s conform to a common data structure definition, thus datasets with attached quality metadata can be compared. Listing 6 describes the definition of `daq:QualityGraph`.

```
daq:QualityGraph
  a rdfs:Class, owl:Class  ;
  rdfs:subClassOf rdfg:Graph , qb:DataSet ,
    [ rdf:type owl:Restriction ;
      owl:onProperty qb:structure ;
      owl:hasValue daq:dsd ];
  rdfs:comment "Defines a quality graph which will contain all metadata about quality metrics on the dataset." ;
  rdfs:label "Quality Graph Statistics" .
```

Listing 6: The Quality Graph Definition (Turtle Syntax)

2.3.2 Abstract Classes and Properties

This ontology framework (Figure 4) has three abstract classes/concepts (`daq:Category`, `daq:Dimension`, `daq:Metric`) and three abstract properties (`daq:hasDimension`, `daq:hasMetric`, `daq:requires`) which should not be used directly in a quality instance. Instead these should be inherited as parent classes and properties for more specific quality metrics. The abstract concepts (and their related properties) are described as follows:

daq:Category represents the highest level of quality assessment. A category groups a number of dimensions.

daq:Dimension – In each dimension there is a number of metrics.

daq:Metric is the smallest unit of measuring a quality dimension. Each metric instance is linked to one or more observations. Each observation has a value (`daq:value`), representing a score for the assessment of a quality attribute. This attribute is defined as a `qb:MeasureProperty`. Since this value is multi-typed (for example one metric might return true/false whilst another might require a floating point number), the value's `daq:hasValue` range is inherited by the actual metric's attribute defined by the property `daq:expectedDataType`. An observation must have the Dimension Properties (`qb:DimensionProperty`) `daq:computedOn` and `daq:metric`, which defines the assessed resource and the metric the mentioned resource was assessed by respectively. A metric might also require additional information (e.g. a gold standard dataset to compare with). Therefore, a instance of a metric representation can also define such properties using subproperties of the `daq:requires` abstract property. Another important attribute for any observation is the `daq:dateComputed`, where it records the date of the observation's creation.

2.3.3 Extending daQ for Custom/Specific Quality Metrics

The classes of the core daQ vocabulary can be extended by more specific and custom quality metrics. In order to use the daQ, one should define the quality metrics that characterise the "fitness for use" [14] in a particular domain. We are currently in the process of defining the quality dimensions and metrics described in Deliverable 5.1. **Extending** the daQ vocabulary means adding new quality protocols that inherit the abstract concepts (Category-Dimension-Metric). Custom quality metrics do not need to be included in the daQ namespace itself; in fact, in accordance with LOD best practices, we recommend extenders to make them in their own namespaces. In Figure 6 we show an illustrative example of extending the daQ ontology (TBox) with a more specific quality attribute, i.e. the RDF Availability Metric as defined in [26], and an illustrative instance (ABox) of how it would be represented in a dataset.

The `Accessibility` concept is defined as an `rdfs:subClassOf` the abstract `daq:Category`. This category has five quality dimensions, one of which is the *Availability* dimension. This is defined as an `rdfs:subClassOf` `daq:Dimension`. Similarly, *RDFAvailabilityMetric* is defined as an `rdfs:subClassOf` `daq:Metric`. The specific properties *hasAvailabilityDimension* and *hasRDFAccessibilityMetric* (sub-properties of `daq:hasDimension` and `daq:hasMetric` respectively) are also defined (Figure 6).

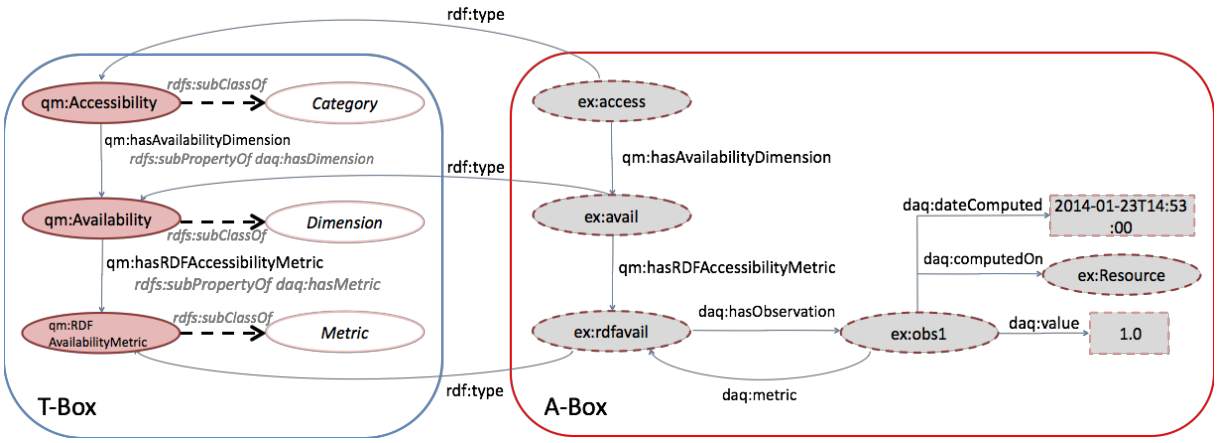

Figure 6: Extending the daQ Ontology – TBox and ABox

2.3.4 A typical Quality Metadata Graph

The excerpt listing in 7 show a typical quality graph metadata in a dataset.

```
# ... prefixes

# ... dataset triples

ex:qualityGraph1 a daq:QualityGraph ;
  qb:structure daq:dsd .

ex:qualityGraph1 {

  # ... quality triples
  ex:accessibilityCategory a dqm:Accessibility ;
    dqm:hasAvailabilityDimension ex:availabilityDimension .

  ex:availabilityDimension a dqm:Availability ;
    dqm:hasEndPointAvailabilityMetric ex:endPointMetric ;
    dqm:hasRDFAvailabilityMetric ex:rdfAvailMetric .

  ex:endPointMetric a dqm:EndPointAvailabilityMetric ;
    daq:hasObservation ex:obs1, ex:obs2 .

  ex:obs1 a qb:Observation ;
    daq:computedOn <efo-2.43> ;
    daq:dateComputed "2014-01-23T14:53:00"^^xsd:dateTime ;
    daq:value "1.0"^^xsd:double ;
    daq:metric ex:endPointMetric ;
    qb:dataSet ex:qualityGraph1 .

  ex:obs2 a qb:Observation ;
    daq:computedOn <efo-2.44> ;
    daq:dateComputed "2014-01-25T14:53:00"^^xsd:dateTime ;
    daq:value "1.0"^^xsd:double ;
    daq:metric ex:endPointMetric ;
    qb:dataSet ex:qualityGraph1 .

  ex:rdfAvailMetric a dqm:RDFAvailabilityMetric ;
      daq:hasObservation ex:obs3, ex:obs4 .

  ex:obs3 a qb:Observation ;
    daq:computedOn <efo-2.43> ;
    daq:dateComputed "2014-01-23T14:53:01"^^xsd:dateTime ;
    daq:value "1.0"^^xsd:double ;
    daq:metric ex:rdfAvailMetric ;
    qb:dataSet ex:qualityGraph1 .

  ex:obs4 a qb:Observation ;
```

```
    daq:computedOn <efo-2.44> ;
    daq:dateComputed "2014-01-25T14:53:01"^^xsd:dateTime ;
    daq:value "0.0"^^xsd:double ;
    daq:metric ex:rdfAvailMetric ;
    qb:dataSet ex:qualityGraph1 .

  # ... more quality triples
}
```

Listing 7: A Quality Graph Excerpt (Turtle Syntax)

The instance *ex:qualityGraph1* is a named `daq:QualityGraph`. The defined graph is automatically a `qb:DataSet`, and due to the restriction placed on the `daq:QualityGraph` (see Listing 6), the value for the `qb:structure` property is defined as `daq:dsd` (see Listing 5). In the named graph, instances for the `daq:Accessibility`, `daq:Availability`, `daq:EndPointAvailabilityMetric` and `daq:RDFAvailabilityMetric` are shown. A metric instance has a number of observations. Each of these observations specifies the metric value (`daq:value`), the resource the metric was computed on (`daq:computedOn` – here: different datasets, which are actually different revisions of one dataset), when it was computed (`daq:dateComputed`), the metric instance (`daq:metric`) and finally to what dataset the observation is defined in (`qb:dataSet`).

2.4 Design of the RESTful Quality API

The RESTful API design and activity diagrams are explained in Deliverable 6.1 [17, Section 6.1.7]. The only minor change is in the input parameters for the `/diachron/compute_quality` API call. In Deliverable 6.1 we define the following two parameters:

Dataset – An instance of a DIACHRON dataset URI;

QualityReportRequired – A boolean indicating whether a quality report is required.

The input parameter we introduce in this Deliverable is **MetricsConfiguration**. This parameter is an object with a list of metrics (cf. Listing 3) in JSON-LD format, identifying the metrics required to be used for the dataset quality assessment. Listing 8 shows a sample input message format with the newly added parameter **MetricsConfiguration**.

```
"Dataset": "http://exampleuri.com/rdfdump",
"QualityReportRequired": true,
"MetricsConfiguration" : [
    {
        "@id": "_:f4212571792b1",
        "@type": [
            "http://www.diachron-fp7.eu/qualityFramework#metricConfiguration"
        ],
        "http://www.diachron-fp7.eu/diachron#metric": [
            {
                "@value": "intrinsic.accuracy.DefinedOntologyAuthor"
            },
            {
                "@value": "accessibility.availability.RDFAccessibility"
            },
            {
                "@value": "representational.understandability.HumanReadableLabelling"
            },
            {
                "@value": "intrinsic.consistency.ObsoleteConceptsInOntology"
            },
            {
                "@value": "accessibility.availability.SPARQLAccessibility"
            },
            {
                "@value": "accessibility.performance.HighThroughput"
            },
            {
                "@value": "intrinsic.consistency.EntitiesAsMembersOfDisjointClasses"
            },
            {
                "@value": "intrinsic.accuracy.SynonymUsage"
```

```
        },
        {
            "@value": "dynamicity.currency.CurrencyDocumentStatements"
        },
        {
            "@value": "intrinsic.accuracy.POBODefinitionUsage"
        },
        {
            "@value": "accessibility.availability.Dereferencibility"
        },
        {
            "@value": "intrinsic.conciseness.OntologyVersioningConciseness"
        },
        {
            "@value": "dynamicity.currency.TimeSinceModification"
        },
        {
            "@value": "representational.understandability.LowBlankNodeUsage"
        },
        {
            "@value": "accessibility.performance.DataSourceScalability"
        },
        {
            "@value": "intrinsic.consistency.HomogeneousDatatypes"
        },
        {
            "@value": "intrinsic.consistency.MisplacedClassesOrProperties"
        },
        {
            "@value": "accessibility.performance.LowLatency"
        },
        {
            "@value": "dynamicity.timeliness.TimelinessOfResource"
        }
    ]
  }
]
```


Listing 8: API Call Input Message Format

3 Tools and Libraries Used

In this section we discuss the main tools and libraries used in our solution to help us achieve our goal.

3.1 OntoWiki

OntoWiki[15] is a tool providing support for agile, distributed knowledge engineering scenarios. Based on semantic technologies, OntoWiki provides an easy to use control management system (CMS) that allows users to manage the knowledge base (RDF data) underlying the application. OntoWiki is part of the LOD2 Stack, available under the GPL open source license. The Quality Framework is currently based on OntoWiki.

3.2 CubeViz

CubeViz[16] is an RDF DataCube browser and also an extension to OntoWiki. This extension allows users to visually represent statistical data represented in RDF, specifically data which is modelled by the RDF DataCube vocabulary. The Dataset Quality Vocabulary (daQ) is defined to use the mentioned RDF statistical vocabulary, thus CubeViz was a suitable extension to use to visualise statistical results about quality metadata.

[15] http://ontowiki.net/
[16] http://cubeviz.aksw.org

3.3 Apache Jena

Apache Jena[17] is an open source Java framework for building Semantic Web and Linked Data applications. The framework is composed of different APIs interacting together to process RDF data. If you are new here, you might want to get started by following one of the tutorials. Apache Jena is licensed under the Apache License, Version 2.0. This framework is the underlying technology used for the "Quality Core Framework".

3.4 OpenRDF Sesame

OpenRDF Sesame[18] is a de-facto standard framework for processing RDF data. This includes parsers, storage solutions (RDF databases a.k.a. triplestores), reasoning and querying, using the SPARQL query language. It offers a flexible and easy to use Java API that can be connected to all leading RDF storage solutions. The framework is licensed under BSD.

3.5 Virtuoso

Virtuoso[19] is an innovative enterprise grade multi-model data server for agile enterprises and individuals. In simple terms, Virtuoso is a triple store. It delivers an unrivaled platform agnostic solution for data management, access, and integration. Virtuoso is licensed under GNU General Public License (GPL) Version 2.

4 Assessment and Ranking Service

4.1 Data Quality Assessment Process

In Deliverable 6.1 (Section 6.1.7.3), we described the assessment process in an activity diagram. Since we need to cater for datasets modelled on the DIACHRON Data Model (cf. D1.3), the Stream Processor described in Section 2.2 requires some adaptation. The DIACHRON data model is used as an underlying schema to store and query data from various heterogeneous sources using one standardised representation. The schema features a number of key aspects that enable DIACHRON services, in particularly making data traceable (indicating the provenance) and reproducible (cf. D1.3).

Due to the heterogenous nature of data from different models (e.g. triple-based, tuple-based, multidimensional etc.), the DIACHRON model is highly based on reified RDF. This enables the description of resources as RDF statements (expanding a triple to a reification quad[20]), recording further information such as provenance, attributes, relations and also changes. For example, if consider the triple[21]:

```
ex:resourceSubject ex:property "value" .
```

it will be represented as a set of `rdf:Statement` triples:

```
ex:reifiedResource rdf:type rdf:Statement .
ex:reifiedResource rdf:subject ex:resourceSubject .
ex:reifiedResource rdf:predicate ex:property .
ex:reifiedResource rdf:object "value" .
```

In Section 2.2 we described how the *Stream Processor* sequentially passes quads to the metrics in the *Quality Assessment Layer*. Although data based on the DIACHRON data model would be sufficient for the *Stream Processor*, metric assessment will not give the desired results. The main problem is that most metrics would not just require the *object* value, but would also require the *subject* and the *predicate*. Therefore, DIACHRON resources have to be de-reified into RDF triple prior to them being passed to the metric.

[17] `https://jena.apache.org`
[18] `http://www.openrdf.org/`
[19] `http://virtuoso.openlinksw.com/`
[20] `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#reification`
[21] This example does not follow the DIACHRON data model, but rather to illustrate a simple example of reification

In order to support the DIACHRON data model, datasets would require preprocessing to be de-reified to a flat RDF structure. This can be done in two ways: (1) Constructing a new dataset using SPARQL CONSTRUCT; (2) Parsing a dump of the dataset. These are explained further below.

Constructing de-reified RDF triples

This is the preferred method of flattening down RDF triples from the DIACHRON data model. For this, a SPARQL endpoint is desirable so that we could query the model directly and do not require loading the dataset into memory for the construction of the model. Listing 9 illustrates the SPARQL query used to de-reify a dataset. The place holder `::datasetUri::` will be replaced by the dataset URI provided.

```
CONSTRUCT {
  ?s ?p ?o
}
WHERE{
  ::datasetUri:: diachron:hasRecordSet ?rs .
  GRAPH ?rs {?rs diachron:hasRecord [diachron:subject ?s ;
  diachron:hasAttribute [diachron:Property ?p ; diachron:object ?o] ]
  }
}
```

Listing 9: "SPARQL Query to de-reify the DIACHRON data model into flat RDF"

Once the graph is constructed, it is passed to the stream processor which in turn starts the assessment.

Parsing a dump of the dataset

In the unlikely event that a SPARQL endpoint is not available, a data wrapper is provided with the stream processor module. This data wrapper parses the RDF statements and creates triples on the fly, which are then passed to the stream processor to sequentially deliver into the *Quality Assessment Layer*.

4.2 Data Quality Metrics

In this subsection we describe the metrics implemented as of the deliverable's deadline which are included in the first prototype of this service. For this purpose, those metrics which are generally required by DIACHRON (**GEN**) and those specific by the pilot partners (Data Publica – **DP**; Scientific Pilot – **EBI**; cf. Deliverable 5.1 [25, section 4]) were analysed and prioritised accordingly, also keeping in mind metrics required by the cleaning application discussed in Deliverable 3.2 [9]. Some pilot partners required custom metrics (such as checking for a specific property) to be implemented. These will be identified with a $\star$. Each implemented metric is expected to conform to the interface described in Section 2.1.2.

4.2.1 Accessibility Category

Availability Dimension

Availability of a dataset is the extent to which data (or some portion of it) is present, obtainable and ready for use [26].

Dereferenceability Metric (required for: **GEN**, **EBI**)

HTTP URIs should be dereferencable, i.e. HTTP clients can retrieve the resources identified by the URI. A typical web URI resource would return a `200 OK` code indicating that a request is successful and `4xx` or `5xx` if the request is unsuccessful. In Linked Data, a successful request should return a document (RDF) containing the description (triples) of the requested resource. There are two possible ways which allow publishers make URIs dereferencable. These are the `303` URIs and the `hash` URIs [23]. Yang et. al [24] describes a mechanism to identify the dereferenceability process of linked data resource.

Calculates the number of valid redirects (303) or hashed links according to LOD Principles.

The dereferenceability metric (listing 5) counts the number of valid dereferenceable URI resources found in the subject (?s) and object (?o) position of a triple. The `isDereferenceable(resource)` method uses the rules defined in [24]. The metric returns a ratio of the number of dereferenced URIs (deref) against the total number

Algorithm 3 Dereferenceability Algorithm

```
1: procedure INIT
2:     totalTriples = 0 ;
3:     deref = 0 ;
4: procedure DEREFERENCE(⟨?s, ?p, ?o, ?g⟩)
5:     if (isURI(?s)) && (isDereferenceable(?s)) then deref++ ;
6:     if (isURI(?o)) && (isDereferenceable(?o)) then deref++ ;
7:     totalTriples++;
```

of triples in a dataset (totalTriples). The expected range is $[0,\ldots,1]$, where 0 is the worst rating and 1 is the best rating.

RDF Accessibility Metric (required for: **EBI**)

A data source should be available as a LOD format (e.g. RDF) dump which can be downloaded by a consumer. This dump should be stated in the dataset and accessible under the given URI.

Check if data dumps (void:dataDump) exists and are reachable and parsable.

Algorithm 4 RDF Accessibility Algorithm

```
1: procedure INIT
2:     lodDumps = 0 ;
3:     accessibleLodDumps = 0 ;
4: procedure RDFACCESSIBILITY(⟨?s, ?p, ?o, ?g⟩)
5:     if ?p is void:dataDump then lodDumps++ ;
6:     if isAccessible(?o) then accessibleLodDumps++ ;
```

The metric returns a ratio between the number of defined LOD data dumps (lodDumps) and the total number of accessible data dumps (accessibleLodDumps). The expected range is $[0,\ldots,1]$, where 0 is the worst rating and 1 is the best rating. If no data dump is found, then the metric would return 0 and thus the dataset fails this metric.

SPARQL Accessibility Metric (required for: **EBI**)

This metric follows the previous (RDF Accessibility Metric), where a dataset is assessed to check if a SPARQL endpoint is defined.

Check if a SPARQL endpoint (matching void:sparqlEndpoint) is available and returns a result.

The metric returns a ratio between the number of defined SPARQL endpoints (endPoints) and the total number of accessible SPARQL endpoints (accessibleEndPoints). The expected range is $[0,\ldots,1]$, where 0 is the worst rating and 1 is the best rating. If no endpoint is defined, then the metric would return 0 and the dataset fails this metric

Performance Dimension

In a broad sense, performance refers to the ability to get access to the Linked Data source efficiently, without

Algorithm 5 SPARQL Accessibility Algorithm

```
1: procedure INIT
2:     endPoints = 0 ;
3:     accessibleEndPoints = 0 ;
4: procedure SPARQLACCESSIBILITY(⟨?s, ?p, ?o, ?g⟩)
5:     if ?p is void:sparqlEndpoint then
6:         endPoints++ ;
7:         query ?o ;
8:         if returnResult(?o) then accessibleEndPoints++ ;
```

causing any major delays in the client application when querying the data. It constitutes a highly important quality feature, as low performance can seriously affect the ability of clients to access the data (availability) and its usability. There are a variety of factors that can have an effect on performance, such as networking issues, server configurations and usage of complex RDF features.

Low Latency Metric (required for: **EBI**)
Latency is a measure of the response-time of a data source, defined by Bizer as "the delay between submission of a request by the user and reception of the response from the system" [1]. Achieving low latency should be one of the main goals, from a performance perspective, of a Linked Data service, as it negatively affects the responsiveness of client applications and hence, their ability to provide the user with timely information.

> *Estimates the efficiency with which a system can bind to the dataset, by measuring the delay between the submission of a request and the reception of the corresponding response, sent back from the system.*

As shown in algorithm 6, the implementation of this metric consists in figuring out the URI of the dataset from which the triples were obtained, to afterwards send several HTTP requests to their source. The respective response times are then averaged to obtain a measure of the latency. Note that this metric refers to the resource itself, not to its contents. The result of the metric is a real number in the range $[1, +\infty)$, as it represents the average time (in

Algorithm 6 Low Latency Algorithm

```
1: procedure INIT
2:     totalDelay = −1;
3:     requestsToSend = 2;
4: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
5:     if isDataSetURI(?s) then
6:         startTimer();
7:         for i = 0 to requestsToSend do
8:             sendSynchronousRequestTo(?s);
9:         timeElapsedSinceStart = stopTimer();
10:        totalDelay += timeElapsedSinceStart;
```

milliseconds) elapsed between the issuing of the request and the reception of its response. The lower the value, the better, as it represents how long does it take to get access to the dataset.

High Throughput Metric (required for: **EBI**)
As latency, throughput is a determining factor when assessing the performance of a data source, since it measures the rate at which a service can provide data as response to client requests. If a service is unable to handle a reasonable amount of data requests, in a timely fashion, its usability and the performance of clients themselves

could be severely affected. Therefore, it is desirable that the Linked Data source is able to properly respond to as many requests as possible during a limited period of time.

> *Measures the efficiency with which a system can access the dataset, as the average number of requests responded by the service hosting it, per second.*

The throughput of the resource is measured by sequentially sending it a fixed number of requests and by totalling the response time of all of them (time elapsed between the sending of the request and the reception of the response). Afterwards, the total number of requests is divided by their total response time. Algorithm 7 provides additional details. The value of the metric is in the range $[1+\infty)$ and represents the average number of requests

Algorithm 7 High Throughput Algorithm

```
1:  procedure INIT
2:      totalDelay = −1;
3:      requestsToSend = 3;
4:  procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
5:      if isDataSetURI(?s) then
6:          startTimer();
7:          for i = 0 to requestsToSend do
8:              sendSynchronousRequestTo(?s);
9:          timeElapsedSinceStart = stopTimer();
10:         totalDelay += timeElapsedSinceStart;
```

successfully served by the resource's host per millisecond. The higher the value, the better, as it represents the number of requests per second, that the data-source is able to serve.

Scalability of a Data Source Metric (required for: **EBI**)

A Linked Data resource that scales well will be able to handle a high, growing amount of requests in an acceptable period of time. This capability will prevent the service from becoming overloaded and hence unresponsive under heavy demand. Thus, scalability helps assure that the data source will be accessible at any time and is an important quality factor regarding performance.

> *Measures the scalability of the data source, by determining whether the average response time of several requests (ten by default), sent simultaneously, is approximately equal to the response time of a single request [7].*

All the requests whose response times (delays) are to be averaged, are sent simultaneously (in parallel), in order to increase the workload on the server. After all these parallel requests have been responded, their respective delays are averaged. Subsequently, a single request is sent, and its delay is compared with the average delay of the parallel requests. This process is outlined in algorithm 8. The value of the metric is in the range $[0,+\infty)$ it consist of the difference in milliseconds, between the average response time of the data source, when exposed to several, simultaneous requests and the response time of a single, isolated request. The lower the value, the better. Higher values suggest poor scalability of the data source, as they mean that it gets overwhelmed by multiple, simultaneous requests.

Security Dimension

Security refers to the capability of restricting access to the data and of guaranteeing that communication between the Linked Data source and its consumers is confidential and protected against tampering, as defined by Flemming in [7]. Conspicuously, the importance of security strongly depends on the nature of the data contained in the

Algorithm 8 Scalability of a Data Source Algorithm

```
1:  procedure INIT
2:      numSimultaneousRequests = 10;
3:      totalDelaySimultaneousReqs = −1;
4:      totalDelaySingleReq = −1;
5:  procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
6:      if isDataSetURI(?s) then
7:          startTimer();
8:          sendAsynchronousRequestsTo(?s, numSimultaneousRequests);
9:          totalDelaySimultaneousReqs = stopTimer();
10:         startTimer();
11:         sendSynchronousRequestTo(?s);
12:         totalDelaySingleReq = stopTimer();
```

resource and also on the application domain, up to the extent that this dimension becomes a critical quality feature when dealing with sensitive information.

HTTPS Data Access Metric ⋆ (required for: **EBI**)

A dataset hosted under a properly configured HTTPS service ensures that communications with its clients are secured by the SSL/TLS protocol. This prevents unauthorised parties from intercepting and getting access to sensitive information and from impersonating the legitimate data provider. Offering a secure connection through HTTPS, has a positive effect on the accessibility of Linked Data resources containing sensitive data, as consumers could otherwise be reluctant to even consult them.

> *Verifies whether the authenticity of the data source is assured and the communication channel is confidential, by verifying that access to it is carried out through a sound, HTTPS connection.*

As shown in algorithm 9, once the URI of the dataset has been determined, it is tested to correspond to the HTTPS protocol, if so, an HTTPS connection is attempted to be established. If no errors occur, and a request is successfully sent through such connection, the data source is considered to be safe. The value of the metric is

Algorithm 9 HTTPS Data Access Algorithm

```
1:  procedure INIT
2:      httpsConnectionSucceeded = 0;
3:  procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
4:      if isDataSetURI(?s) && isHTTPS(?s) then
5:          httpsConnectionSucceeded = sendTestRequestHTTPS(?s);
```

binary: it will return 1 if an HTTPS connection was successfully established with the data source and 0 otherwise. Thus, a value of 1 corresponds to the best possible quality rating.

Licensing Dimension

In an open data world, such as that of Linked Open Data, it is important to be aware of the terms under which data sources can be accessed and used. In order to prevent any inconveniences and to assure that information is distributed according to the publisher's rules, consumers should make sure that the datasets they use are available under clear legal terms.

Machine-readable Indication of a License Metric

As stated by Hogan et al. [13], each dataset should contain a license specifying how the content can be used. Such a license can be machine-readable, if provided by means of any of the several properties defined for that purpose.

Having access to the license this way, enables customers to automatically check that the permissions granted upon the data have been specified and hence leverages accessibility to the dataset. The dataset is assessed by checking if triples (with the subject being the base URI) contain any of the following predicates:

- `http://purl.org/dc/terms/license`
- `http://purl.org/dc/terms/accessRights`
- `http://purl.org/dc/terms/rights`
- `http://purl.org/dc/elements/1.1/rights`
- `http://www.w3.org/1999/xhtml/vocab#license`
- `http://creativecommons.org/ns#license`

> *Checks whether consumers of the dataset are explicitly granted permission to re-use it, under defined conditions, by annotating the resource with a machine-readable indication of the license.*

Algorithm 10 details how the metric is computed. Firstly, the URI of the dataset is determined from the triples as they are processed. Having the dataset's URI, the triples providing licensing information about it can be looked for. The value of the metric is binary: a value of 1 indicates that machine-readable licensing information was found

Algorithm 10 Machine-readable Indication of a License Algorithm

```
1: procedure INIT
2:     dataSetURI = null;
3:     hasMachineReadLicense = 0;
4: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
5:     if isDataSetURI(?s) then
6:         dataSetURI = ?s;
7:     if isLicensingProperty(?p) && ?s == dataSetURI then
8:         hasMachineReadLicense = 1;
9:
```

as part of the dataset, whereas 0 indicates that it was not. Therefore, 1 elicits a higher quality ranking.

4.2.2 Intrinsic Category

The intrinsic category metrics are independent of the user's context. They reflect whether information presented in data correctly represent the real world and whether information is logically consistent itself.

Accuracy Dimension

Accuracy dimension metrics reflect the degree of correctness and precision with which the given dataset represent the real world facts.

Malformed Datatype Literals Metric (required for: **GEN**, **DP**, **EBI**)

Literals in RDF are values, plain or typed, containing a textual string (eg. "13"^^xsd:int). It is common problem in LOD datasets that the literal value does not match its describing data type. The Malformed Datatype Literals metric intends to check this inconsistency. The algorithm 11 describes the metric computation in more details.

> *Calculates the ratio of invalid typed literals against all literals*

Only the typed literals are considered by the metric. The metric values vary in the interval $[0, \ldots, 1]$, where the 0 indicates the best quality.

Algorithm 11 Malformed Datatype Literals Algorithm

```
1: procedure INIT
2:     totalLiterals = 0 ;
3:     malformedLiterals = 0 ;
4: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
5:     if (isLiteral(?o)) then totalLiterals++ ;
6:         if (isTypedLiteral(?o)) && (!hasValidDatatype(?o)) then malformedLiterals++ ;
```

Literals Incompatible with Datatype Range Metric (required for: **GEN**, **DP**, **EBI**)

Similar to the previously described metric, this verifies the correctness of literals regarding their datatype. By correctness, we mean that the data type attached to the literal is different than the one specified by the range of the used property in the ontology. Algorithm 12 describes this metric.

Calculates the ratio of incorrectly defined literals against all literals in the dataset.

Algorithm 12 Literals Incompatible with datatype range

```
1: procedure INIT
2:     totalLiterals = 0
3:     incompatibleLiterals = 0
4: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
5:     if isLiteral(?o) then totalLiterals++ ;
6:         if hasRange(?p) && (literalDatatype(?o)!=rangeDatatype(?p)) then imcompartibleLiterals++;
```

Only the literals referring by the property with the range characteristic are considered by the metric. The metric values vary in the interval $[0, 1]$, where 0 indicates the best quality.

Defined Ontology Author Metric⋆ (required for: **EBI**)

This metric is required of the Scientific Linked Data scenario as defined in Deliverable 5.1. Imported ontologies should have common metadata which includes the definition of an author.

Checks whether the creator `efo:creator` is defined in the ontology

Algorithm 13 Defined Ontology Author Algorithm

```
1: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
2:     if ?o is owl:Ontology then ontologyList.add(?s);
3:     if ?p is efo:creator && ?s in ontologyList then definedCreator = 1 ;
```

The metric will return a 1 (best rating) if the creator of the ontology is defined or 0 if otherwise.

POBO Definition Usage Metric⋆ (required for: **EBI**)

This metric is required of the Scientific Linked Data scenario as defined in Deliverable 5.1. For this use case, it is required to check that each ontology class has a corresponding human readable definition.

Provides a measure for an ontology to check the usage of `pobo:def` in defined classes

The metric will return a ratio of the number of classes without definitions against the total number defined classes. The expected range is $[0, \ldots, 1]$, where 0 is the worst rating and 1 is the best rating

Algorithm 14 POBO Definition Usage Algorithm

```
1: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
2:     if ?o is owl:Class then ontologyClassList.add(?s);
3:     if ?p is pobo:def && ?s in ontologyClassList then
4:         definitionHashMap.put(?s,?o);
```

Synonym Usage Metric⋆ (required for: **EBI**)

This metric is required of the Scientific Linked Data scenario as defined in Deliverable 5.1. Scientific terms usually have a number of synonyms. Each ontology term should have at least one synonym, thus in this metric we check if defined terms have a synonym attached to their description.

> *Measures the number of classes which has a synonym* `efo:alternative_term` *described*

Algorithm 15 Synonym Usage Algorithm

```
1: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
2:     if ?o is owl:Class then ontologyClassList.add(?s);
3:     if ?p is efo:alternative_term && ?s in ontologyClassList then
4:         synonomHashMap.put(?s,?o);
```

The metric will return a ratio of the number of classes without synonyms against the total number defined classes. The expected range is $[0, \ldots, 1]$, where 0 is the worst rating and 1 is the best rating

Consistency Dimension

Consistency metrics intend to identify any kinds of contradictions in data.

Homogeneous Datatypes (required for: **GEN**, **DP**, **EBI**)

In some cases, the range of a property in a vocabulary is either (i) not defined; or (ii) given an open/generic type (such as `rdfs:Literal`). This could cause inconsistencies in a dataset since values defined by the same property can be used with literals of different datatypes. In contrast to the *Malformed Literal* metric and *Incompatible Datatype* metric, this metric deals with the object in a triple where its property has an undefined/open range. The metric computation therefore contains the following steps:

- Count the frequency of different datatypes occurring with a particular predicates.
- Identify the properties corresponding to heterogeneous datatype literals.

> *Calculates the ratio of inconsistent object values in a dataset*

The metric values vary between $[0, \ldots, 1]$, when the 0 indicates no properties containing heterogeneous datatype literals.

Misplaced Classes or Properties

In some cases a URI that occurs in the predicate position of a triple is defined in the corresponding vocabulary as a class, or a contrariwise a URI in the subject or object position is a defined property. These kind of inconsistencies make machine interpretation of the data more complex. More details about the metric computation are shown in algorithm 17.

> *Identifies the percentage of misplaced classes and properties in a dataset.*

The metric values vary between $[0, \ldots, 1]$, where the 0 indicates the best quality.

Algorithm 16 Homogeneous Datatypes

```
1: procedure INIT
2:     totalProperties = 0;
3:     heterogeneousDatatypeProperties = 0
4:     propertyMap = map(Property, List<Datatype>)
5: procedure ANALYZEPROPERTIES(⟨?s, ?p, ?o, ?g⟩)
6:     if isLiteral(?o) then propertyMap.put(?p, Set.add.datatypeOf(?o));
7: procedure COMPUTE
8:     for all Properties in propertyMap do
9:         if size(Set⟨Datatype⟩) ⟩ 1 then heterogeneousDatatypeProperties++;
           totalProperties ++;
```

Algorithm 17 Misplaced Classes or Properties Metric Algorithm

```
1: procedure INIT
2:     totalClassesAndProperties = 0 ;
3:     misplacedClasses = 0 ;
4:     misplacedProperties = 0 ;
5: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
6:     if isURI(?p) then totalClassesAndProperties++ ;
7:     if !isProperty(?p) then misplacedProperties++ ;
8:     if isURI(?o) then totalClassesAndProperties++;
9:     if !isClass(?o) then misplacedClasses++ ;
```

Misused OWL Datatype or Object Properties (required for: **EBI**)

The OWL language defines additional characteristics to some properties. A property can be either an instance of the *owl:ObjectProperty* or *owl:DatatypeProperty* class. A datatype property relates some resource to a literal value, while an object property describes a relation between two resources. Wrong usage of the datatype and object properties indicates inconsistencies in the data. The following algorithm 18 presents more details about how the metric is computed.

Ratio of misused datatype and object properties to all properties

Algorithm 18 Misused OWL Datatype or Object Properties Metric Algorithm

```
1: procedure INIT
2:     totalProperties = 0 ;
3:     misusedObjectProperties = 0 ;
4:     misusedDatatypeProperties = 0 ;
5: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
6:     if isURI(?p) then totalProperties++ ;
7:     if isDatatypeProperty(?p) && isURI(?o) then misusedDatatypeProperties++ ;
8:     if isObjectProperty(?p) && isLiteral(?o) then misusedObjectProperties++ ;
```

The metric gives a value between $[0, \ldots, 1]$, where the 0 indicates the best quality.

Obsolete Concepts in Ontology⋆ (required for: **EBI**)

Over time and versions, classes and properties change and old versions get deprecated. If an ontology is creating a considerable amount of obsolete concepts between different versions, then this is an indicator that the ontology is

going through numerous changes and is potentially in a state of poor quality. Typically, in LOD datasets deprecated classes and properties are identified by `owl:DeprecatedClass` and `owl:DeprecatedProperty` respectively. The pilot use case requires that this metric assess datasets (which contain an ontology) for the identification of deprecated classes and properties. These deprecated concepts should be `rdfs:subClassOf` by the `oboInOwl:ObsoleteClass` and `oboInOwl:ObsoleteProperty`.

Measures the number of obsolete classes and properties in a dataset.

Algorithm 19 Obsolete Concepts Metric

1: **procedure** COMPUTE($\langle ?s, ?p, ?o, ?g \rangle$)
2: **if** (?p is rdfs:subClassOf) $\wedge$ ((?o is oboInOwl:ObsoleteClass) $\vee$ (?o is oboInOwl:ObsoleteProperty)) **then** obsoleteConcept++;
3: **if** (?p is rdf:type) $\wedge$ (?o is owl:Class) **then** concept++;

The value of this metric is computed by measuring the percentage of deprecated classes and properties (obsoleteConcept) against the total number of classes and properties (concept) in the ontology. Value is given between the range of [0..1], with 0 being the best possible value.

Ontology Hijacking (required for: **EBI**)

Ontology Hijacking was introduced by Hogan et. al [12] and is defined as "the contribution of statements about classes and/or properties in a non-authoritative source such that reasoning on those classes and/or properties is affected". In other words ontology hijacking refers to cases where external concepts are redefined in a local ontology. Defining new super classes or properties of third-party classes or properties is an example for this problem, e.g. declaring *rdfs:subPropertyOf* which is defined as a property, to be a *rdfs:Class*. To identify hijacked terms we define a set of properties *P*:

- *rdf:type*,
- *rdf:domain*,
- *rdf:range*,
- *rdfs:subClassOf*,
- *rdfs:subPropertyOf*
- *owl:equivalentClass*,
- *owl:equivalentProperty*,
- *owl:inverseOf*,
- *owl:onProperty*,
- *owl:hasValue*,
- *owl:someValuesFrom*,
- *owl:allValuesFrom*,
- *owl:intersectionOf*,
- *owl:unionOf*,
- *owl:maxCardinality*,

- *owl:cardinality*,
- *owl:oneOf*;

and a set of classes *C*:

- *owl:FunctionalProperty*,
- *owl:InverseFunctionalProperty*,
- *owl:TransitiveProperty*,
- *owl:SymmetricProperty*.

Metric check the two following cases:

- Classes in **C** appear in a position other than the object of a *rdf:type* triple.
- Properties in **P** appear in a position other than the predicate position.

Checks for the re-definition of defined concepts breaking the consistency.

Algorithm 20 Ontology Hijacking Algorithm

```
1: procedure INIT
2:     hijackedTriples = 0 ;
3:     totalTriples = 0 ;
4: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
5:     if ((?s ∈ C) or (?o ∈ C))&& (?p == rdf:type) then hijackedTriples++ ;
6:     if (?p ∈ P) && isAuthority(?s) && !isAuthority(?o) then hijackedTriples++;
```

Undefined Classes Metric (required for: **GEN**)

A class is defined in a vocabulary or dataset by the predicate `rdf:type` and an object value of `rdf:Class` or `owl:Class`. In the case when LOD dataset defines its own vocabulary the following predicates indicate that the object must be a defined class: `rdfs:domain`, `rdfs:range`, `rdfs:subClassOf`, `owl:allValuesFrom`, `owl:someValuesFrom`, `owl:equivalentClass`, `owl:complementOf`, `owl:onClass`, `owl:disjointWith`. The undefined classes problem occurs due to spelling or syntactic mistakes resolvable through minor fixes to the respective ontologies. The missing classes should be defined in the dataset itself or an external ontology.

Calculates the ratio of undefined classes to all classes in the object position in a dataset

Algorithm 21 Undefined Classes Metric Algorithm

```
1: procedure INIT
2:     totalClasses = 0 ;
3:     undefinedClasses = 0 ;
4: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
5:     if (isClassProperty(?p) && (isURI(?o)) then totalClasses++ ;
6:     if (!isDefinedClass(?o)) undefinedClasses++ then;
```

Undefined Properties (required for: **GEN**)
Similar to the Undefined Classes metric, this metric identifies terms in the predicate position that are used without any formal definition.

> *Calculates the ratio of undefined properties to all properties in the given data set*

Algorithm 22 Undefined Properties Algorithm

```
1: procedure INIT
2:     totalProperties = 0 ;
3:     undefinedProperties = 0 ;
4: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
5:     if (isURI(?p) then totalProperties++;
6:     if !isDefined(?p) then undefinedProperties++;
7:     if isFromList(?p) && !isDefined(?o) then undefinedProperties++ ;
```

The metric value ranges between $[0, \ldots, 1]$, where 0 indicates the best quality, namely no undefined classes in the data set.

Conciseness Dimension

High quality data sources should not contain redundant information, since it could lead to contradictions and in general would affect the clarity and manageability of data. Mendes et al. [18] identifies two levels of conciseness: intensional conciseness, which refers to the schema level, that is, to the absence of redundant properties, and extensional conciseness, referring to the extent to which instances (i.e. objects) described by the dataset, are unique.

Duplicate Instance Metric (required for: **GEN**)
The information contained in Linked Data resources should not be redundant, which means that the instances contained in a dataset should, ideally, be unique. As stated by Yuangui Lei et al in [16], the mapping between the real-world objects described by the data sources and the instances contained in the semantic metadata should be one to one. That is, each statement about the existence of a real-world object should correspond to one and only one instance declaration. Resources are divided into groups called classes. The members of a class are known as instances of the class. A triple of the form: R rdf:type C, states that C is an instance of rdfs:Class and R is an instance of C, as defined in the RDF Schema specification [22].

> *One minus the ratio of the number of instances violating the uniqueness rule to the total number of instances in the dataset.*

As shown in algorithm 23, this metric is implemented by computing the subtraction one minus the ratio of the number of non-unique instances to the total number of declared instances. An instance is regarded as non-unique, if there is another instance declaration (i.e. rdf:type annotation) with its same subject URI and object value. The metric will return one minus the ratio of the number of unique instance declarations in the dataset, to the total number of instance declarations existing in the dataset. The expected range is $[0, \ldots, 1]$, where 0 is the best rating (no duplicate instance declarations exist) and 1 is the worst rating (all instance declarations in the dataset are redundant).

Extensional Conciseness Metric (required for: **DP**)
The conciseness of a dataset can be considered at the data level, as the redundancy of objects (i.e. instances) contained into the dataset. As defined by Mendes et al. [18], a dataset is concise (on the extensional or instance

[22] http://www.w3.org/TR/rdf-schema/#ch_type

Algorithm 23 Duplicate Instance Algorithm

```
1: procedure INIT
2:     mapDeclaredInstances = new Map⟨URI, Instance⟩();
3:     countNonUniqueInstances = 0;
4:     countTotalInstances = 0;
5: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
6:     if containsKey(mapDeclaredInstances, ?s) then countNonUniqueInstances++;
7:     countTotalInstances++;
8:
```

level), if it does not contain redundant objects, that is, objects being equivalent in their contents, yet having different identifiers. Ideally, Linked Data resources should not contain redundant information, which implies that all the objects described by them should be unique. Uniqueness of objects is determined from their properties: one object is said to be unique if and only if there are no other objects with the same set of properties and corresponding values.

> *Calculates the ratio of the number of unique objects (i.e. instances) to the Total Number of objects. Two objects are equivalent if they have the same set of properties, all with the same values (but not necessarily the same ids).*

In the implementation of this metric (algorithm 24), objects are identified by their URI (the value of the subject attribute of the triples). The uniqueness of objects is determined from its properties: one object is said to be unique if and only if there is no other subject equivalent to it. Note that two equivalent objects may differ in their URI. The metric will return a ratio of the number of unique objects in the dataset (i.e. objects whose properties and their

Algorithm 24 Extensional Conciseness Algorithm

```
1: procedure INIT
2:     mapDescribedObjs = new Map⟨URI, Object⟩();
3: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
4:     curDescribedObj = getElementWithKey(mapDescribedObjs, ?s);
5:     setObjectProperty(curDescribedObj, ?p, ?o);
6:
```

values are not duplicated in another object), to the total number of objects described in the dataset. The expected range is $[0, \ldots, 1]$, where 0 is the worst rating (all objects are the same) and 1 is the best rating (all objects are unique).

Ontology Versioning Conciseness⋆ (required for: **EBI**)

In order to make sure that redundancy of entities is avoided, checking for a single defined ontology version is desirable.

> *Checks if the ontology has only one instance of ⟨owl:versionInfo⟩.*

Algorithm 25 Ontology Versioning Conciseness Metric

```
1: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
2:     if (?p is owl:versionInfo ) then versionInfo++;
```

This metric returns 1 if there is only one instance of the property, 0 if more.

4.2.3 Representational Category

Representational dimensions reflect the quality aspects like conciseness, consistency and interpretability of information.

Representational Conciseness Dimension

Representational conciseness is determined by how compact, properly formatted and clear is the data. As defined by Pipino et al. in [20], this dimension is defined as "the extent to which information is compactly represented".

Short URIs

URIs play a key role in how information is represented in Linked Data resources, as they are used to name the entities being described. Therefore, having compact, well formatted URIs has a positive effect in the clearness and conciseness of data. As suggested by [13], data providers that locally mint (on average) shorter URIs are deemed as being more compliant with Linked Data best practices.

> *Detects whether, in average, short URIs are being used, which suggests that information is compactly represented and that readability is favored.*

Implementation details regarding these metric are provided in algorithm 26. All URIs identifying instances, that are defined locally, are considered by the metric. The calculation is performed as the average of the lengths of the URIs corresponding to the subjects of all instance declarations (i.e. statements using the *rdf:type* predicate). The metric will return the average length of all the URIs locally defined in the dataset. The expected range is a real

Algorithm 26 Short URIs Algorithm

```
1: procedure INIT
2:     accumulatedURIsLength = 0;
3:     countLocallyDefURIs = 0;
4: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
5:     if equalsURI(?p, rdf:type) && isURI(?s) then
6:         accumulatedURIsLength += lengthOfURI(?s);
7:         countLocallyDefURIs++;
8:
```

number in the range $[0, +\infty)$. Lower values represent better rankings.

Understandability Dimension Understandability is the quality of information that enables users to comprehend its meaning. It is an ultimate prerequisite for information consumer. The better users understand data, the more effectively they can use it.

Empty Annotation Value (required for: **DP**, **EBI**)

In some languages, eg. OWL, annotation properties are distinguished. Annotation properties are predicates that provide informal documentation annotations about ontologies, statements, or IRIs. A simple example for annotation property is *rdfs:comment* which is used to provide a comment. The problem can be solved by the corresponding triples or by replacing empty literals by annotation strings. The following annotation properties were used in this metric:

- *skos:altLabel*
- *skos:hiddenLabel*
- *skos:prefLabel*

- *skos:changeNote*
- *skos:definition*
- *skos:editorialNote*
- *skos:example*
- *skos:historyNote*
- *skos:note*
- *skos:scopeNote*
- *dcterms:description*
- *dc:description*
- *rdfs:label*
- *rdfs:comment*

> *Measures a percentage of triples whose property is an annotation property and whose object is an empty string.*

The algorithm 27 provides detailed description of metric computation.

Algorithm 27 Empty Annotation Value Algorithm

```
1: procedure INIT
2:     totalAnnotations = 0 ;
3:     emptyAnnotations = 0 ;
4: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
5:     if isAnnotation(?p) then totalAnnotations++;
6:     if isEmpty(?o) then emptyAnnotations++ ;
```

The metric gives a percentage (as a decimal) in the range of $[0, \ldots, 1]$, where 0 indicates the best quality.

Whitespace in Annotation

In contrast to the previous metric which identifies triples with empty annotation value, this metric deals with the case when the annotation value is available, but contains leading or trailing whitespaces. The metric is defined as follows:

> *Calculates the ratio of annotations with leading or trailing whitespace to all annotations in the data set.*

A predefined list of annotation properties, described in the previous metric, are used.

The metric value ranges between $[0, \ldots, 1]$, where 0 indicates that data set is free of leading or trailing whitespace in annotations.

Algorithm 28 Whitespace in Annotation Algorithm

```
1: procedure INIT
2:     totalAnnotations = 0 ;
3:     whitespaceInAnnotations = 0 ;
4: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
5:     if isAnnotation(?p) then totalAnnotations++;
6:     if containsLeadingORTrailingWhitespace(?o) then whitespaceInAnnotations++ ;
```

Labels Using Capitals

This metric identifies the triples with label property whose object uses a bad style of capitalisation. We define "bad" capitalisation as "camel case" where compound words or phrases are written such that each next word or abbreviation begins with a capital letter, e.g. *InterestingThing*. Regular expression rules are defined for this metric. The following widely used label properties are considered by the metric:

- *skos:altLabel*
- *skos:hiddenLabel*
- *skos:prefLabel*
- *rdfs:label*

> *Calculates the ratio of labels with "bad capitalisation" to all labels*

Algorithm 29 Labels Using Capitals Algorithm

```
1: procedure INIT
2:     totalLabels = 0 ;
3:     badCapitalisedLabels = 0 ;
4: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
5:     if isLabel(?p) then totalLabels++;
6:     if isBadCapitalised(?o) then badCapitalisedLabels++ ;
```

Human Readable Labelling (required for: **DP**, **EBI**)

Having human readable labels enable human consumers to better understand the data.

> *Measures the percentage of entities having a ⟨rdfs:label⟩ or ⟨rdfs:comment⟩ defined*

Algorithm 30 Human Readable Labelling Metric

```
1: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
2:     if ?p is rdf:type then instanceMap.put(?s,0)
3:     if (?p is rdfs:label ∨ ?p is rdfs:comment) ∧ isNotEmptyLiteral(?o) then in-
   stanceMap.update(?s,humanLabel++);
```

The metric returns a value between [0..1]. This value is calculated by finding a ratio of total number of human readable labels against total number of instances.

Low Blank Node Usage (required for: **DP**, **EBI**)
Blank nodes reduce the understandability of a dataset. The lower the number of blank nodes is, the higher the quality is.

> *Provides a measure calculating the number of blank nodes used in a dataset.*

Algorithm 31 Human Readable Labelling Metric

```
1: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
2:     if blankNode(?s) then blankNodes++;
3:     if blankNode(?o) then blankNodes++;
```

The metric returns a value between [0..1], i.e. the result of dividing the number of blankNodes against the total number of non-literal subject and object entities. The lowest the score, the better.

4.2.4 Dynamicity Category

Currency Dimension
Virtually all application domains are interested in getting access to data that is as up-to-date as possible. Actually, it is generally the case that data sources decrease in value as they become outdated. Therefore, the currency dimension is a key feature of Linked Data resources. The following metrics, intended to assess the currency of datasets, are based in the definition of this dimension provided by Kahn et al. [15] as "the degree to which information is up-to-date".

Currency of Documents/Statements Metric (required for: **GEN**, **DP**, **EBI**)
This metric is based in the definition of currency provided by Rula et al. as "the age of a value, where the age of a value is computed as the difference between the current time (the observation time) and the time when the value was last modified" [21]. The age of a dataset can be computed at both, the resource level (by comparing its last time of modification with the observation time) and the triples level (by comparing the value of last-modified statements instead).

> *Measures the degree to which data is up to date, by comparing the time when the data was observed (approximately the current time), with the time when the data (the document and each triple) was last modified.*

As listed in algorithm 32, the metric is computed as the average of the comparisons of the observation time versus the last modification time of each triple, normalized by the time elapsed since the publication of the document and the observation time. The last modification time is extracted from the properties: *http://purl.org/dc/terms/modified* and *http://semantic-mediawiki.org/swivt/1.0#wikiPageModificationDate*. Likewise, the publication time of the document is extracted from *http://purl.org/dc/terms/created*, *http://purl.org/dc/terms/issued* and *http://semantic-mediawiki.org/swivt/1.0#creationDate*, as suggested by the study conducted by Rula et al. [21]. The metric will return the average of the difference between one and the ratio of the time elapsed since the last modification of each triple to the total time the dataset has been available. Since the former time span cannot be larger that the latter, the expected range is $[0, \ldots, 1]$. The higher the value, the better, as it reflects how recently have the resources been updated. A value of 0 indicates that the resources have never been updated after their publication.

Time Since Modification Metric
As suggested above, the difference between the observation time (the time at which the resource is examined) and the time when the data was last modified, is a natural measure of how current is the information provided by a dataset. In contrast with the Currency of Documents/Statements metric, a simpler approach can be taken in order

Algorithm 32 Currency of Documents/Statements Algorithm

```
1:  procedure INIT
2:      accumTimeDiffs = 0;
3:      countModifiedObjs = 0;
4:      observationTime = getCurrentTime();
5:      publishingTime = null;
6:  procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
7:      if isLastModifiedTimeURI(?p) then
8:          accumTimeDiffs += (observationTime - parseAsTime(?o));
9:          countModifiedObjs++;
10:     if isPublishingTimeURI(?p) then
11:         publishingTime = parseAsTime(?o);
12:
```

to measure the degree to which data is up to date. This metric does so by computing the plain difference between the observation time and the time when the data values were last modified (note that the normalization factor, based on the publishing time of the document, is omitted here).

> *Provides a measure of the degree to which information is up to date, by taking the average difference between the observation time (i.e. the instant when the present calculation of the metric was initiated) and the time when the data (each triple in the dataset) were last modified.*

The computation performed by the metric is detailed in algorithm 33. The observation time corresponds to the instant when the calculation of the metric was initiated and only the triples providing a last time of modification are processed during the computation. The result is in the range $[0, +\infty]$ and represents an amount of time in

Algorithm 33 Time Since Modification Algorithm

```
1:  procedure INIT
2:      accumTimeDiffs = 0;
3:      countModifiedObjs = 0;
4:      observationTime = getCurrentTime();
5:  procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
6:      if isLastModifiedTimeURI(?p) then
7:          accumTimeDiffs += (observationTime - parseAsTime(?o));
8:          countModifiedObjs++;
9:
```

milliseconds. The lower the value, the better, as this metric measures the average time elapsed since the last modification of the information contained in the resource. That is, high values indicate that data provided by the resource was last updated long ago.

Exclusion of Outdated Data Metric

The currency of a Linked Data resource can also be measured relatively to the amount of outdated entities it contains. However, in order to do so, it is necessary to somehow be able recognize when an entity is to be deemed as outdated, which in turn requires that temporal metadata is available and represented according to an appropriate model, such as those proposed by Rula et al. in [22]. In this particular case, we consider an entity to be outdated, if information about its period of validity is provided and according to it, data has already expired. The proportion of outdated data present in a dataset is an important quality factor, since it is usually the case that outdated data is no longer valid.

Determines the extent to which information provided in a dataset is outdated, by comparing the total number of entities described by the dataset, versus how many of those that are recognized to be outdated.

The implementation of the metric, as presented in algorithm 34, makes use of the property *http://purl.org/dc/terms/valid* (when provided) to determine the expiration time of each examined triple and then, whether the described entity is outdated. After processing all triples of the dataset, the metric is computed as the ratio of the number of outdated entities to the total number of entities in the dataset. The metric values vary in the

Algorithm 34 Exclusion of Outdated Data Algorithm

```
1:  procedure INIT
2:      countTotalOutdatedObjs = 0;
3:      countTotalDdescribedObjs = 0;
4:      observationTime = getCurrentTime();
5:  procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
6:      if isValidTimeURI(?p) then
7:          if observationTime < parseAsTime(?o) then
8:              countTotalOutdatedObjs++;
9:      if equalsURI(?p, rdf:type) && isURI(?s) then
10:         countTotalDescribedObjs++;
11:
```

interval $[0, 1]$, where a value of 1 indicates the best quality (no entities were found to be outdated).

Volatility Dimension

In Linked Data evolution appeared almost at each new published version of data. Following the idea of Papavassiliou et al. [19] curators could define a list of changes that occur frequently and correspond to one or more low-level changes (added or deleted triples). These changes termed as Simple Changes also in the context of DIACHRON and comprise an upper abstract level of changes which is pilot-specific to describe group of changes that appear a special interest for each pilot. The detection of Simple Changes achieved accordingly to the methodology presented in [19] and followed in change detection service of DIACHRON (cf. Deliverable 3.1 [8]). The following three volatility metrics take into account these assumptions and background information.

Versions Volatility Metric (required for: **GEN**, **DP**, **EBI**)

The comparison of two sequential (or not) versions of datasets could contain a number of simple changes for each pilot. In other cases, it makes sense to compare an old version of a dataset with the newest one.

Calculates the number of simple changes between two specified versions.

The Versions Volatility Metric can be applied to a pair of defined versions to count the detected number of Simple Changes. This is achieved by querying the corresponding named graph where the the total number of Simple Changes have been stored which are returned as result.

Algorithm 35 Versions Volatility Algorithm

```
1:  procedure INIT
2:      numberOfChanges = 0
3:  procedure COMPUTE
4:      numberOfChanges = countSimpleChanges(v1,v2)
```

The metric will return the total number of Simple Changes between two versions [integer number].

Average Volatility Metric (required for: **GEN**, **DP**, **EBI**)
The number of detected simple changes could be varied across different published versions for each curator/pilot. According to different application scenarios some versions are too similar (appear small number of detected simple changes) or different enough (appear large number of detected simple changes). In other terms the distribution of deltas (that resulted from the detected simple changes) across versions is not necessarily uniform. Thus, it is meaningful to examine all available published versions of datasets in order to find the average detected Simple Changes across each sequential pair of versions.

Calculates the average number of simple changes detected across the published versions.

The Average Volatility Metric firstly calculates the total number of published versions through a SPARQL query. Afterwards, it calculates the detected Simple Changes per versions pair and aggregates the sum of changes. Finally, it calculates and returns the ratio between aggregated sum and the number of examined pairs.

Algorithm 36 Average Volatility Metric Algorithm

1: **procedure** INIT
2: changesTotal = 0
3: versionsNo = 0
4: retValue = 0
5: **procedure** COMPUTE
6: $Versions[] = countVersions(SPARQL)$
7: **for all** $v[i], v[i+1] \in Versions$ **do**
8: $changesTotal = changesTotal + countSimpleChanges(v[i], v[i+1])$
9: retValue = changesTotal / versionsNo -1

The metric will return the ratio $[0,\ldots,1]$ of average detected simple changes across the published dataset versions.

Weighted Volatility Metric (required for: **GEN**, **DP**, **EBI**)
In some applications pilots are interested more in evolution of specified versions. By applying a weighted sum model [6] for each sequential pair of versions, we could adapt this preference for each pilot.

Calculates the average weighted sum of simple changes that has been detected across the published versions.

The Weighted Volatility Metric after finding the total number of published versions it loads the weights from the curator's preference table. Afterwards, it calculates the simple changes per pair and multiply with the corresponding weight. Finally it calculates the ratio of weighted sum of changes to the examined pairs of versions.

The metric will return the ratio of $[0,\ldots,1]$ aggregated weighted sum detected simple changes across the published dataset versions.

Time Validity Interval Metric (required for: **GEN**, **DP**, **EBI**)
An alternative to measure how frequently the data varies with time, is to make use of information about the expiration of entities in order to assess how long does data remain valid. Consequently, such measurement would require two pieces of information about entities described in the dataset: the expiry time of entities (i.e. validity) and the time when data became available (i.e. publishing time). By combining these two components, the interval during which the information will remain valid can be computed.

Estimates the frequency with which data will be updated, by calculating the average length of the time interval during which entities remain valid.

Algorithm 37 Weighted Volatility Metric Algorithm

```
1:  procedure INIT
2:      aggregSChanges = 0
3:      versionsNo = 0
4:      retValue = 0
5:  procedure LOADWEIGHTS
6:      weights[] = fetchWeights()
7:  procedure COMPUTE
8:      versions[] = countVersions(SPARQL)
9:      for all v[i], v[i+1] ∈ versions do
10:         for all w[j] ∈ weights do
11:             changesTotal = changesTotal + w[j] * countSimpleChanges(v[i], v[i+1])
12: retValue = aggregSChanges / versionsNo -1
```

As shown in algorithm 38, information about the expiration time and issued time of entities is extracted from the triples, thereby accumulating the data necessary to compute the average length of the validity interval of all entities for which such information is provided in the dataset. The result is in the range $[0, +\infty]$ and represents

Algorithm 38 Time Validity Interval Algorithm

```
1:  procedure INIT
2:      mapObjsWithValidityInfo = new Map⟨URI, Object⟩();
3:  procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
4:      if isValidityTimeURI(?p) then
5:          setValidityTimeOfObj(getElementWithKey(mapObjsWithValidityInfo, ?s), parseAsTime(?o));
6:      if isPublishingTimeURI(?p) then
7:          setIssueTimeOfObj(getElementWithKey(mapObjsWithValidityInfo, ?s), parseAsTime(?o));
8:
```

an amount of time in seconds. Lower values correspond to better quality rankings, as they indicate that data is expected to be updated more frequently and therefore would be fresher.

Timeliness Dimension

The degree to which Linked Data is considered to be up-to-date or outdated can heavily depend on the task at hand. Depending on the application, information updated one month ago can be considered highly current (e.g. the list of pet shops in a city) or unacceptably outdated (e.g. a pricing list of foreign currencies). Therefore, in order to align the quality measures related to the age of data with the application of interest, it makes sense to provide currency measures relative to a specific task. The metrics for this dimension, are based on the concept of timeliness as defined by Gamble et al.: "Timeliness is a measure of utility, is a comparison of the date the annotation was updated with the consumer's requirement" [10].

Timeliness of the Resource Metric (required for: **GEN**, **DP**, **EBI**)

If the expiration time of the data comprised by a Linked Data resource is provided, it is reasonable to expect such data to be updated before or shortly after that time. Otherwise, it would suggest that the data might be outdated. This, added up to the fact that the expiration time of information is closely associated to the application domain, allows to define Timeliness of the Resource as an application-domain-related metric that measures the currency of data.

Indicates how up-to-date data is, relative to a specific task, by measuring the difference between the invalid time (expiry time of the data) and the observation time (current time).

Algorithm 39 illustrates how this metric is computed. The procedure determines whether the triple contains temporal information stating when the described entity expires (i.e. its validity). This is done by evaluating the property regarded as source of the Expiration/Valid Time (namely, *http://purl.org/dc/terms/valid*, as suggested by Rula et al. [21]), if such a property is found, its value is subtracted from the current observation time, and the result is accumulated. The total accumulated differences will be used afterwards, to calculate the final value of the metric. The result of the metric is a real number in the range $[-\infty, +\infty]$, as it represents the average length of the

Algorithm 39 Timeliness of the Resource Algorithm

```
1: procedure INIT
2:     accumValidTimeDiffs = 0;
3:     countTotalAccountedObjs = 0;
4:     observationTime = getCurrentTime();
5: procedure COMPUTE(⟨?s, ?p, ?o, ?g⟩)
6:     if isExpirationTimeURI(?p) then
7:         accumValidTimeDiffs += (observationTime - parseAsTime(?o));
8:         countTotalAccountedObjs++;
```

gap (in milliseconds) between the expiration time of the data and the actual time. The lower the value, the better, since a higher, positive value indicates that the resources are possibly outdated.

4.2.5 Contextual Category

Amount of Data Dimension

Amount of data, similar to relevancy, is all about the volume of appropriate data for the task at hand. Flemming defined this dimension as the "criterion influencing the usability of a data source" [7]. The only metric available in this dimension is purely statistical.

Amount of Triples Metric (required for: **GEN**)

This metric counts the number of triples present in the dataset. It will check on the size of the dataset which is the lower bound for the number of triples present in the dataset. In order to identify the volume of the data a range[23] was defined as follows:

- high (metric value 1): $>$ 1,000,000,000 triples;
- mediumHigh (metric value 0.8): 10,000,000 triples $< x \leq$ 1,000,000,000 triples;
- medium (metric value 0.6): 500,000 triples $< x \leq$ 10,000,000 triples;
- mediumLow (metric value 0.4): 10,000 triples $< x \leq$ 500,000 triples;
- low (metric value 0.2): $<$ 10,000 triples.

Measures the size of the dataset in terms of the number of triples.

The algorithm counts the number of triples and returns a value according to the range defined above.

[23] Taken from `http://lod-cloud.net`

4.3 The User Interface

In this section, we will describe a set of mockups which will be implemented as a Web User Interface (UI) for the software prototype for assessment and ranking of datasets. The User Interface will be made up of 3 parts: Details, Statistics and Assessment. The Quality Framework Web-UI will be a mix of PHP and JavaScript and is envisioned to run on top of OntoWiki, a wiki based on semantic technologies. The current technologies and extensions available in OntoWiki allow us to reach both the main objectives of this deliverable, and eventually to commence our initial prototypes of the quality framework (e.g. as an enterprise add-on tool for linked data publishers). Further investigation upon the usability of OntoWiki is still necessary in order to discover whether the mentioned application can fulfill further the quality framework's ambitions, beyond the initial prototypes. The CubeViz extension is used to visualise statistical graphs about the quality metadata of the datasets. The UI will communicate with the core assessment framework (cf. Figure 1) via the defined RESTful APIs (cf. Section 2.4).

4.3.1 Details Tab

The Details Tab (Figure 7) will contain the ranked datasets according to a chosen metric by the user from the facet section. This is very similar to the datahub.io[24] interface. The user will also be able to enable filters to the datasets, and also to search through the resultant datasets. This tab will only display those datasets which have a corresponding Quality Graph, in order to enable the ranking and retrieval of datasets based on quality criteria.

The Details Tab will have the following functionality:

1. **Facets –** When the datasets are retrieved, a SPARQL query is performed to get all available quality categories used. The result set is used to populate the Category box in the facet. Once a category is chosen, the dimension box is populated dynamically via a SPARQL query which fetches the dimensions related to the chosen category. A similar procedure is done to populate the metric box. When a metric is chosen, then the datasets are ranked in ascending order on the right hand side of the container.

2. **Filters –** Filters will dynamically change according to the chosen metric. That is, if the `daq:expectedDatatype` (cf. Section 2.3) of a metric is boolean then we have a filter which can be a true/false checkbox, whilst if we have a metric with a double type then we could have some slider regulating the minimum value for the metric.

3. **Buttons –** The buttons have the same functionality as in CKAN. The additional button *quality metadata/⟨file type⟩* returns the quality graph in a specific format. The *Visualise Quality* button takes the user to the Statistics tab in the web page.

4.3.2 Statistics Tab

The Quality Assessment Framework Web-UI will also give the opportunity to its users to visualise statistical information about a dataset quality. The Statistical Tab (Figure 8) will present the user with a number of graphs (cf. Visualisation Layer – Figure 1 for the description of the different visualisation graphs). Users can compare different datasets together or even how a dataset changed in its quality over time.

4.3.3 Visualisation Layer

CubeViz is an extension of the OntoWiki[25] data wiki for visualising data cubes (observation instances). Figures 9, 10, 11, and 12 depicts four different CubeViz chart visualisations from computed quality metadata[26]. While these are generic chart types, we briefly discuss their specific suitability for data quality analysis.

[24] `http://www.datahub.io`

[25] `http://ontowiki.eu/Welcome`

[26] The quality metadata used can be found in `https://raw.githubusercontent.com/diachron/quality/master/src/test/resources/cube_qg.trig`

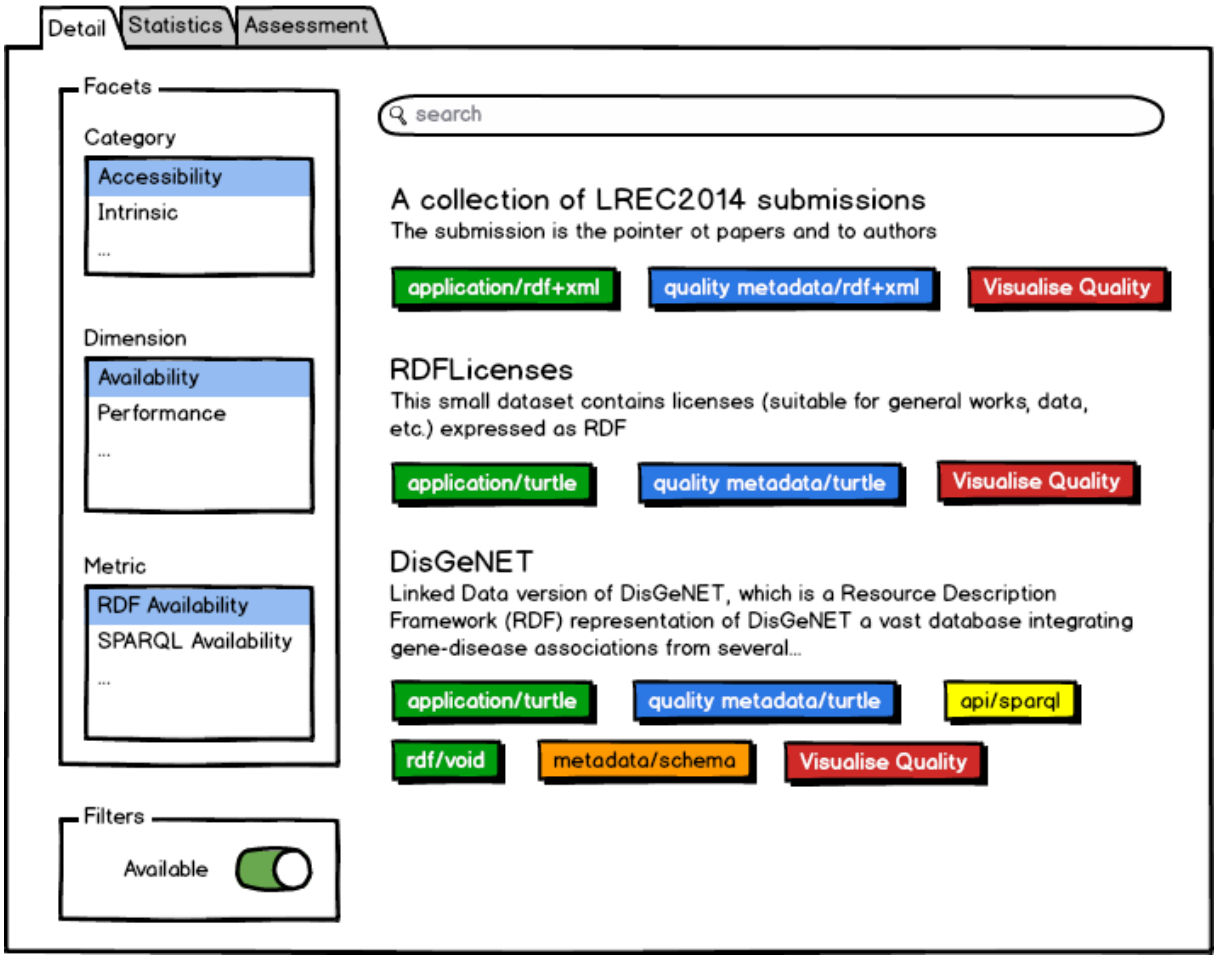

Figure 7: Mockup for the Details Tab

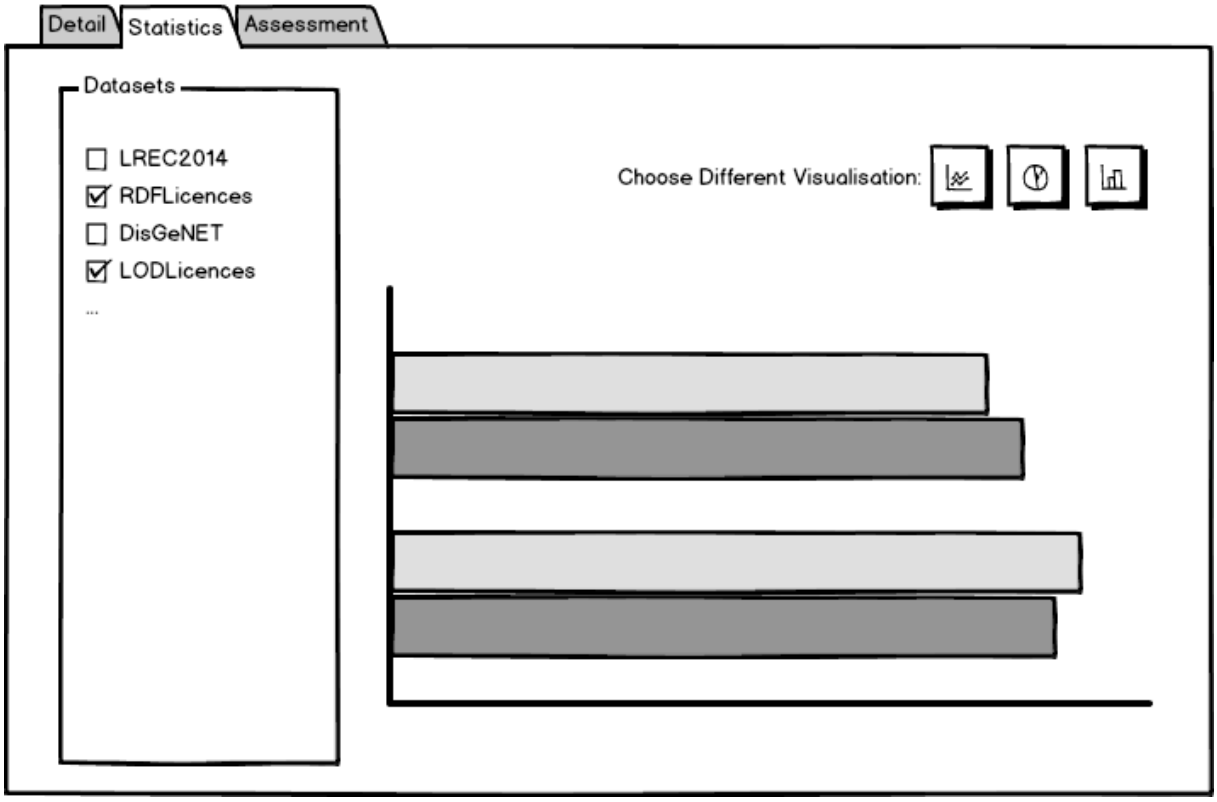

Figure 8: Mockup for the Statistics Tab

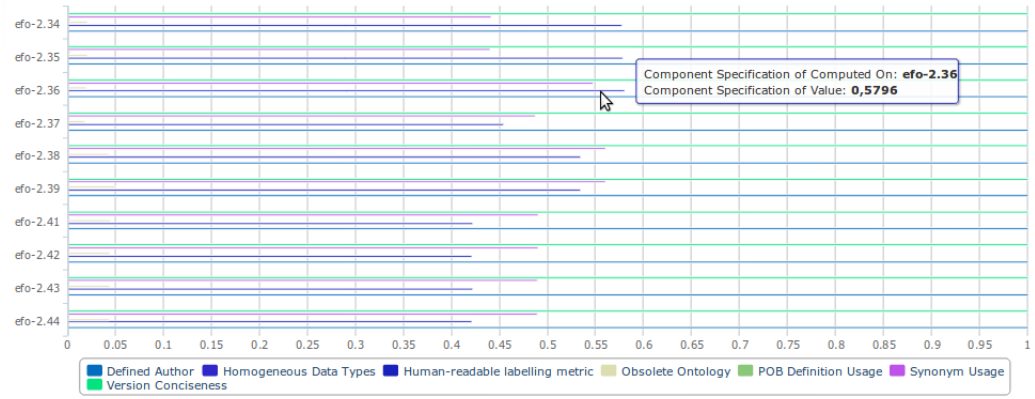

Figure 9: Horizontal Bar Chart

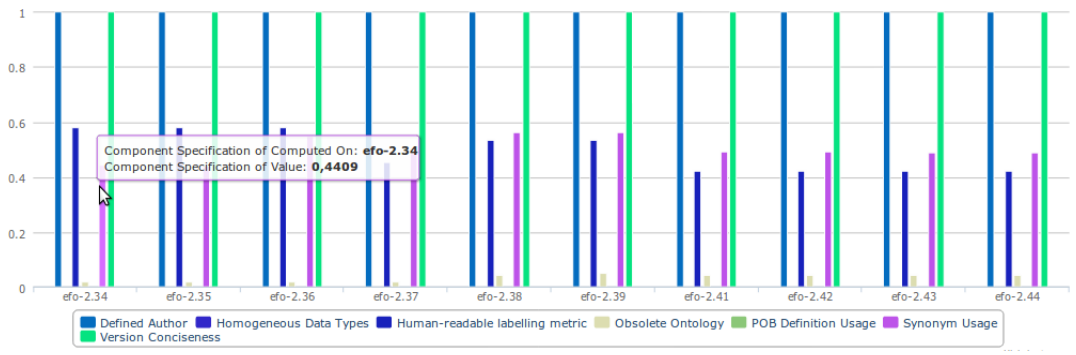

Figure 10: Vertical Bar Chart

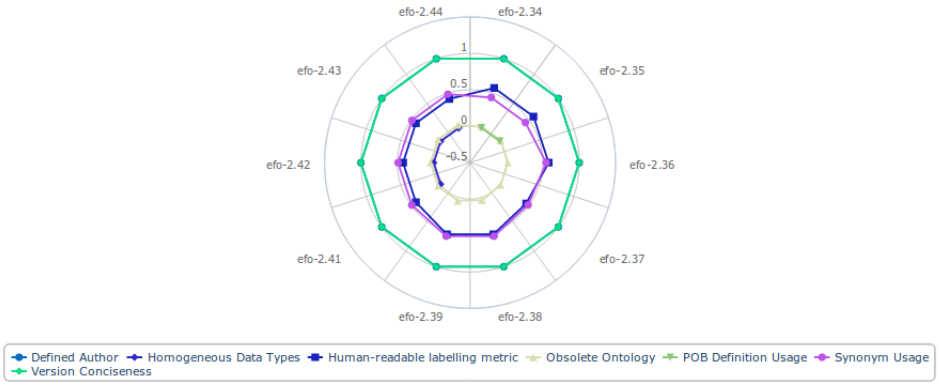

Figure 11: Radar Chart

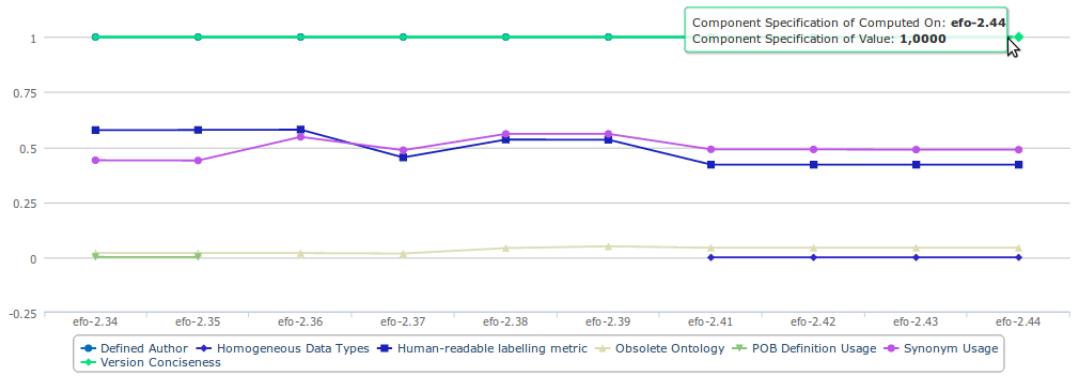

Figure 12: Lines Plot

A *horizontal bar* represents each metric (Figure 9) and shows its value (x-axis) with respect to the dataset (y-axis). Here, the different "datasets" analysed are actually successive revisions of one dataset. This chart provides a clear view of how the value associated to each one of the measured metrics changes as the dataset evolves. The horizontal layout is appropriate when the range of metric values is wide, and the number of different datasets is relatively small.

Similar to the horizontal bars chart, the *vertical bar chart* (Figure 10) allows the user to compare the values computed for each of the metrics (y-axis), with respect to the dataset (x-axis). In contrast with its horizontal counterpart, this chart is more appropriate when there are many datasets analysed but the range of metric values is not so wide.

In the *radar chart* (Figure 11), the datasets are represented as slices of a circle and the values corresponding to the metrics are depicted as points and lines of a particular colour. This chart provides a clear view of how the values of the metric differ from each other for each particular dataset. Furthermore, it allows one to assess the overall quality of a dataset, by showing whether the values of the metrics are concentrated around sections of the circle regarded as "good" or "bad".

The lines plot (Figure 12), lists the different datasets against the values of the metrics. Here, where "different datasets" are actually different revisions in the evolution of one dataset, this plot provides a comparison of the evolution of the quality of the dataset, with respect to each metric. The lines emphasise the points where the values of the metrics changed noticeably from one version to the next.

4.3.4 Assessment Tab

The Quality Framework Web-UI will also allow users to assess datasets for their quality. Figure 13 shows a mockup of how the Assessment Tab will look like. The user is guided step by step to assess a dataset:

1. The user chooses the dataset to be assessed or give a dataset URI;
2. User decides what metrics to be assessed;
3. User decides if a quality problem report is required, which would allow the user to semi-automatically clean up the data (cf. Deliverable 3.1);
4. User clicks the *assess* button to start assessing the chosen datasets.

4.4 Ranking of Datasets by Quality

Tools for data consumers, such as the CKAN data portal software[27], usually provide features such as faceted browsing and sorting, in order to allow prospective dataset users to search within the large dataset archive. Using faceted browsing, datasets could be filtered according to tags or values of metadata properties. The datasets could also be ranked or sorted according to values of properties such as relevance, size or the date of last modification. With many datasets available, filtering or ranking by quality can become a challenge. Talking about "quality" as a whole might not make sense, as different aspects of quality matter for different applications. It does, however, make sense to restrict quality-based filtering or ranking to those quality categories and/or dimensions that are relevant in the given situation, or to assign custom weights to different dimensions, and compute the overall quality as a weighted sum. The daQ vocabulary enables flexible filtering and ranking possibilities in that it facilitates access to dataset quality metrics in these different dimensions and thus facilitates the (re)computation of custom aggregated metrics derived from base metrics. To keep quality metrics information easily accessible, each assessed dataset contains the relevant daQ metadata graph in the dataset itself. We provide two ranking possibilities to the user: (i) Unbiased Automatic Quality Ranking, and (ii) User-Driven Quality Ranking.

[27] `http://ckan.org`

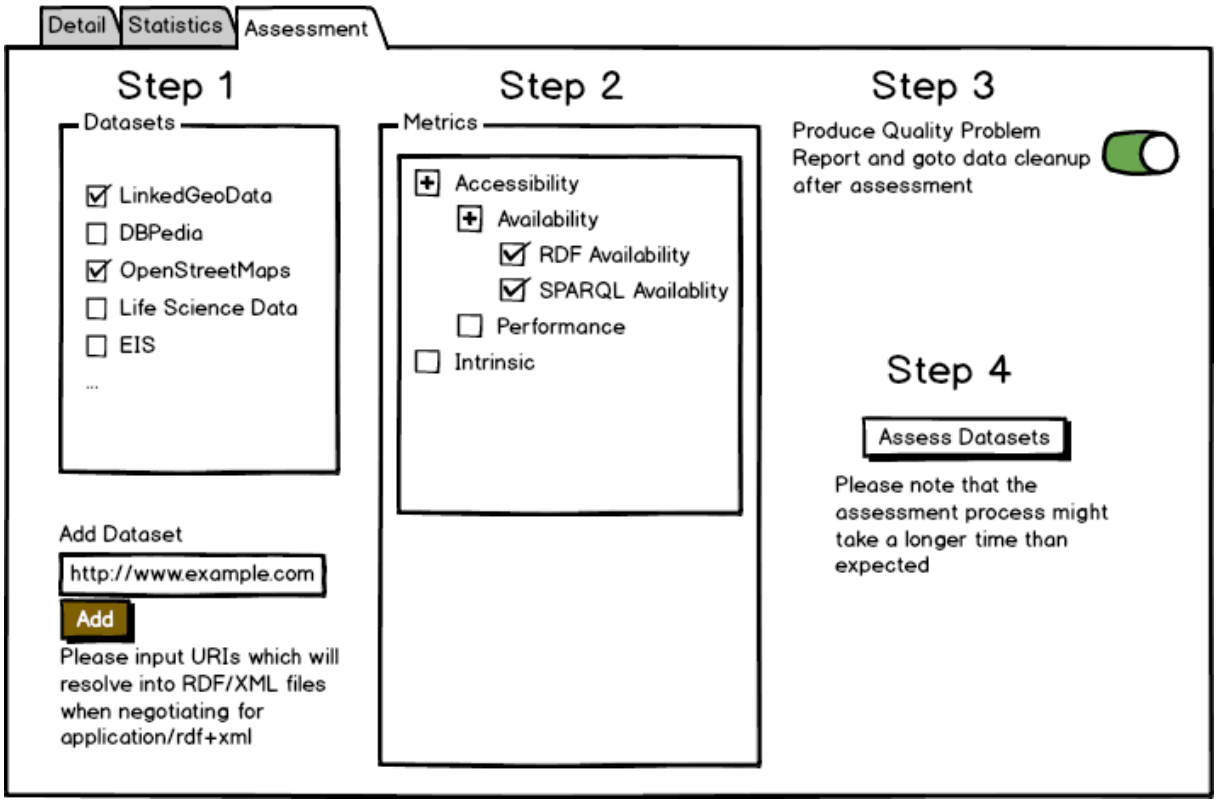

Figure 13: Mockup for the Assessment Tab

4.4.1 Unbiased Automatic Quality Ranking

We aim to achieve a quality-biased ranking of LOD datasets, promoting:

1. High quality datasets;
2. Datasets on which a larger number of quality metrics is calculated.

Therefore, datasets of poor quality but having more quality metadata might end up ranked higher than those with excellent quality on the only metric assessed on. Publishers might be doubtful about the data they publish and they will hide their doubt in the dataset itself. One possibility is that the publishers might hide this psychological nature by claiming that their dataset is of good quality in certain aspects. Quality assessment frameworks should not only be about positive or negative assertions of a dataset, but should also place the publishers' doubts in the assertion spectrum. With our proposed ranking algorithm we do not just promote those datasets that have a high value to one particular metric, but we promote those datasets, which although might lack in particular metric assessment, have more coverage of assessed metrics.

The approach we take for ranking takes into consideration the total number of metrics assessed by the most complete dataset available in the datastore[28]. It also takes into consideration any facet filters chosen by the user, enabling the dynamic change of weights and thus the final ranking. Consider $F_m = \{m_1, m_2, \ldots, m_n\}$ where F_m is the set of all metrics m available. The ranking algorithm is split into a number of steps:

1. Get the total number of metrics assessed (n);
2. Adjust weight for those metrics selected in the facet (Definition 2);
3. Adjust weight for the rest of the metrics (Definition 3);
4. Calculate the metric value by weight to find out ranking (Definition 4) and rank;

4.4.2 Weight Assignment

The weight assignment is the most crucial aspect of the ranking algorithm. It should be evenly distributed amongst the chosen (filtered) metrics, whilst also giving a share of the weight to the other metrics. In this way we ensure the quality-bias ranking and promote not just the high quality datasets, but also giving a smaller share to those that give more information about quality than others. In Definition 2 we calculate a weight value for the number of metrics chosen ($\#X$) i.e. those metrics given a preference by the user. Together with the number of metrics chosen, we add 1 to represent a small share which will be divided equally with the rest of the metrics ($\#\overline{X}$ represents the metrics not chosen). This is defined in Definition 3. To explain this in a simpler manner, if we have five metrics and only one was chosen, then the chosen metric will have a weight of 0.5 whilst the rest will have 0.5 shared equally (i.e. 0.125).

Definition 2.

$$Let\ X \subseteq F_m$$
$$\theta = \frac{1}{\#X + 1}$$

Definition 3.

$$Let\ \overline{X} = F_m \setminus X$$
$$\rho = \frac{1}{(\#X + 1) \cdot \#\overline{X}}$$

[28] We are assuming that the metrics used are common to all datasets in the domain

4.4.3 Ranking Datasets

After the weights have been distributed, the ranking algorithm retrieves all the metric values from the quality graph. Let $v : F_m \rightarrow \{\mathbb{R} \cup \mathbb{N} \cup \mathbb{B} \cup \ldots\}$ be the function that yields the value of a metric (which is, most commonly, a real number, but could also be an integer, a boolean, or any other simple type). The values of chosen metrics, i.e. the set X, are added together and multiplied by the weight θ. Similarly, the complement set ($\overline{X}$) are multiplied by the weight ρ. These two are added together which gives us a value τ for the dataset (Definition 4). This is repeated for all possible datasets; they are then ranked accordingly.

Definition 4.

$$\tau = \theta \cdot \sum_{m \in X} v(m) + \rho \cdot \sum_{m \in \overline{X}} v(m)$$

Listing 10 shows a typical configuration of the retrieval of metric assessment values from the Quality Graph. In this query the metric value of the latest observation is taken into consideration.

```
SELECT ?metric, ?value WHERE {
  ?graph a daq:QualityGraph .
  GRAPH ?graph {
    ?metric a ?metricType .
    ?metric daq:hasObservation ?obs .
    ?obs daq:value ?value .
    ?obs daq:dateComputed ?dateComputed .
  }
  GRAPH <http://www.diachron-fp7.eu/dqm#> {
    ?metricType rdfs:subClassOf daq:Metric .
  }
} ORDER BY DESC(?dateComputed) LIMIT 1
```

Listing 10: Retrieving metric assessment value from the Quality Graph.

4.4.4 Ranking Example

In the following subsection we introduce an example to further help the reader to understand how ranking will work. Consider the following dataset scenario in Table 4.4.4. In our datastore we have four datasets each having a Quality Graph with a number of assessed metrics and their values. *Dataset D* has been assessed by all eight metrics whilst the others have been assessed with six, five and five respectively.

Metric	Dataset A	Dataset B	Dataset C	Dataset D
A	0.2	0.9	0.5	0.3
B	0.5	0.9	0.8	0.4
C	0.9	0.3	0.7	0.7
D	0.8	–	0.9	0.1
E	0.7	–	0.4	0.1
F	–	0.2	–	0.5
G	–	0.8	–	0.9
H	0.3	–	–	0.9

Table 1: Dataset example

In Table 2 we show how the ranking value τ would dynamically change for different scenarios. Initially, no filter is chosen by the user. A possible scenario for this is on page load. When no filter is chosen (in this example) all weights have a ρ value of 0.125. *Dataset D* will be ranked first, followed by *Dataset A*, *Dataset C* and *Dataset B*. This shows that although at first glance it seems that *Dataset B* is a high quality dataset (three out of five assess quality metrics are over 0.8), it got penalised by the fact that other datasets had more quality metadata about their dataset. The single filters **A** and **E** gave expected results, with *Dataset B* and *Dataset A* being top

ranked respectively. The multiple filter **A,B** also gave expected results, with *Dataset D* being ranked third ahead of *Dataset A* due to the it having more quality metadata. More examples of different scenarios are in the Table 2, where the blue coloured cells show the top-ranked dataset.

Chosen Filter	Dataset A	Dataset B	Dataset C	Dataset D		θ	ρ
No Filter	0.425	0.3875	0.4125	0.4875		1	0.125
A	0.32857143	0.60714286	0.45	0.40714286		0.5	0.07142857
E	0.54285714	0.22142857	0.40714286	0.32142857		0.5	0.07142857
A,B	0.38333333	0.67222222	0.54444444	0.41111111		0.33333333	0.05555556
A,B,C	0.49	0.575	0.565	0.475		0.25	0.05
A,E	0.43888889	0.42222222	0.43333333	0.32777778		0.33333333	0.05555556
B,D,E,H	0.515	0.29	0.48	0.42		0.2	0.05

Table 2: Dataset Ranking for different Scenarios

4.4.5 User-Driven Quality Ranking

We will also allow the user to define custom weights to their preferred metrics. At the moment we are also investigating how ranking can be applied to the category and dimension levels.

5 Conclusions

This document describes the software prototype for crawling, ranking and appraisal services which is required as part of the DoW in M16 of the project. It provides a detailed high level description on the Quality Assessment Framework (Section 2.1), which provides the basis of this prototype.

During the initial phase, pilot partners identified the required metrics for their use cases (cf. Deliverable 5.1 [25]). Having these in mind, a framework was identified to facilitate the quality assessment of the pilots' datasets together with providing enough metadata to enable the crawling and ranking of these datasets. After describing the Quality Assessment framework, Assessment and Ranking services were described in Section 4.4. The sources of the prototype of the assessment framework can be found at `https://github.com/diachron/quality`[29].

The approach taken in the QA framework is a generic pluggable approach, therefore it can be used even beyond the scope of DIACHRON. Additionally, a number of research tasks were also addressed in the framework, particularly the representation of quality metadata.

References

1. Bizer, C. Quality-Driven Information Filtering in the Context of Web-Based Information Systems. PhD thesis. FU Berlin, Mar. 2007. `http://www.diss.fu-berlin.de/diss/receive/FUDISS_thesis_000000002736`.
2. Carroll, J. J. et al. Named Graphs, Provenance and Trust. In: *Proceedings of the 14th WWW conference*. (Chiba, Japan, May 10–14, 2005). Ed. by A. Ellis, T. Hagino. ACM Press, 2005, pp. 613–622.
3. Carroll, J. J. et al. Semantic Web Publishing using Named Graphs. In: *ISWC Workshop on Trust, Security, and Reputation on the Semantic Web*. Ed. by J. Golbeck et al. Vol. 127. CEUR Workshop Proceedings. CEUR-WS.org, May 9, 2005. `http://dblp.uni-trier.de/db/conf/semweb/iswc2004trust.html#CarrollBHS04`.
4. Cyganiak, R., Reynolds, D., Tennison, J. The RDF Data Cube Vocabulary. W3C Recommendation. World Wide Web Consortium (W3C), Jan. 16, 2014. `http://www.w3.org/TR/2014/REC-vocab-data-cube-20140116/`.

[29] Please note that this URL can change without any notice.

5. Debattista, J., Lange, C., Auer, S. daQ, an Ontology for Dataset Quality Information. In: *Linked Data on the Web (LDOW)*. (Seoul, Apr. 8, 2014). Ed. by C. Bizer et al. 2014. http://events.linkeddata.org/ldow2014/.
6. Fishburn, P. C. Additive Utilities with Incomplete Product Set: Applications to Priorities and Assignments. Tech. rep. 1967.
7. Flemming, A. Quality Characteristics of Linked Data Publishing Datasources. http://sourceforge.net/apps/mediawiki/trdf/index.php?title=Quality_Criteria_for_Linked_Data_sources. [Online; accessed 13-February-2014]. 2010.
8. Flouris, G. et al. LOD evolution: change typology, update languages and integrity rules. Deliverable D3.1. DIACHRON: Managing the Evolution and Preservation of the Data Web, Mar. 2014.
9. Flouris, G. et al. Specification and Implementation of Monitoring, Synchronization and Repairing Services. Deliverable D3.2. DIACHRON: Managing the Evolution and Preservation of the Data Web, July 2014. Forthcoming.
10. Gamble, M., Goble, C. Quality, Trust, and Utility of Scientific Data on the Web: Towards a Joint Model. In: *Proceedings of the 3rd International Web Science Conference*. WebSci '11. Koblenz, Germany: ACM, 2011, 15:1–15:8. http://doi.acm.org/10.1145/2527031.2527048.
11. Gil, Y., Ratnakar, V. TRELLIS: An Interactive Tool for Capturing Information Analysis and Decision Making. In: *Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*. 13th International Conference, EKAW (Sigüenza, Spain, Oct. 1–4, 2002). Ed. by A. Gómez-Pérez, V. R. Benjamins. LNCS 2473. Springer, 2002, pp. 37–42.
12. Hogan, A., Harth, A., Polleres, A. SAOR: Authoritative Reasoning for the Web. In: 2008, pp. 76–90.
13. Hogan, A. et al. An empirical survey of Linked Data conformance. In: Web Semantics: Science, Services and Agents on the World Wide Web 14(0) (2012). Special Issue on Dealing with the Messiness of the Web of Data, pp. 14–44. http://www.sciencedirect.com/science/article/pii/S1570826812000352.
14. Juran, J. M. Juran's Quality Control Handbook. 4th ed. McGraw-Hill (Tx), 1974. http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0070331766.
15. Kahn, B. K., Strong, D. M., Wang, R. Y. Information Quality Benchmarks: Product and Service Performance. In: Commun. ACM 45(4) (Apr. 2002), pp. 184–192. http://doi.acm.org/10.1145/505248.506007.
16. Lei, Y., Uren, V., Motta, E. A Framework for Evaluating Semantic Metadata. In: *Proceedings of the 4th International Conference on Knowledge Capture*. K-CAP '07. Whistler, BC, Canada: ACM, 2007, pp. 135–142. http://doi.acm.org/10.1145/1298406.1298431.
17. Liapis, A. et al. Development Environment and Detailed Design of the DIACHRON Platform. Deliverable D6.1. DIACHRON: Managing the Evolution and Preservation of the Data Web, Apr. 30, 2014.
18. Mendes, P. N., Mühleisen, H., Bizer, C. Sieve: Linked Data Quality Assessment and Fusion. In: *Proceedings of the 2012 Joint EDBT/ICDT Workshops*. EDBT-ICDT '12. Berlin, Germany: ACM, 2012, pp. 116–123. http://doi.acm.org/10.1145/2320765.2320803.
19. Papavassiliou, V. et al. High-Level Change Detection in RDF(S) KBs. In: Transactions on Database Systems (TODS) 38(1) (2013).
20. Pipino, L. L., Lee, Y. W., Wang, R. Y. Data Quality Assessment. In: Commun. ACM 45(4) (Apr. 2002), pp. 211–218. http://doi.acm.org/10.1145/505248.506010.
21. Rula, A., Palmonari, M., Maurino, A. Capturing the Age of Linked Open Data: Towards a Dataset-Independent Framework. In: *Semantic Computing (ICSC), 2012 IEEE Sixth International Conference on*. Sept. 2012, pp. 218–225.
22. Rula, A. et al. On the Diversity and Availability of Temporal Information in Linked Open Data. English. In: *The Semantic Web – ISWC 2012*. Ed. by P. Cudré-Mauroux et al. Vol. 7649. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 492–507. http://dx.doi.org/10.1007/978-3-642-35176-1_31.

23. Sauermann, L., Cyganiak, R. Cool URIs for the Semantic Web. W3C Interest Group Note. World Wide Web Consortium (W3C), Dec. 3, 2008. http://www.w3.org/TR/2008/NOTE-cooluris-20081203/.

24. Yang, Y. et al. Dereferencing Semantic Web URIs: What is 200 OK on the Semantic Web? https://dl.dropboxusercontent.com/u/4138729/paper/dereference_iswc2011.pdf (visited on 2014-07-28).

25. Zaveri, A. et al. LOD quality assessment and quality-driven acquisition. Deliverable D5.1. DIACHRON: Managing the Evolution and Preservation of the Data Web, Nov. 29, 2013.

26. Zaveri, A. et al. Quality Assessment Methodologies for Linked Open Data. In: Semantic Web Journal (2014). This article is still under review. http://www.semantic-web-journal.net/content/quality-assessment-methodologies-linked-data-survey.