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

DIACHRON

Managing the Evolution and Preservation of the Data Web DIACHRON

Dissemination level:	
Type of Document:	
Contractual date of delivery:	
Actual Date of Delivery:	
Deliverable Number:	
Deliverable Name:	
Deliverable Leader:	
Work package(s):	
Status & version:	
Number of pages:	
WP contributing to the deliverable:	
WP / Task responsible:	
Coordinator (name / contact):	
Other Contributors:	
EC Project Officer:	Federico Milani
Keywords:	
Abstract: Some meaningful abstract here.	

Document History			
Ver.	Date	Contributor(s)	Description
0.1	24.06.2014	Jeremy Debattista	Created TOC and LaTeX Setup

Table of Contents

Table of Figures

List of Tables

1 Introduction

1.1 Scope and Objectives

1.2 Context of this Document

1.3 Document Structure

2 Data Quality Framework

2.1 High-Level Architecture

2.2 Stream Processor

2.3 The Dataset Quality Ontology

The idea behind the Dataset Quality Ontology [?][1] (daQ) is to provide a comprehensive generic vocabulary framework, allowing a uniform definition of specific data quality metrics and thus suggest 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 1 depicts the current state of the daQ vocabulary.

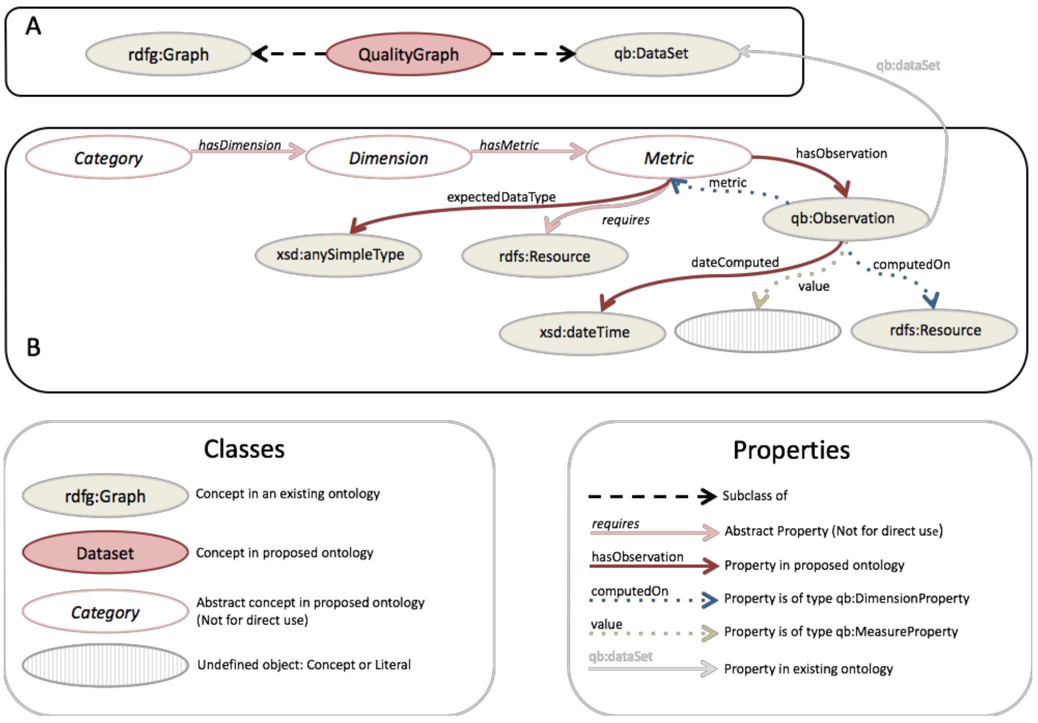

Figure 1: The extended Dataset Quality Ontology (daQ)

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` [?]. 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;

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

- their use in the Semantic Web Publishing vocabulary [?] 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. [?]. As shown in Figure 1 Box B, a quality graph comprises of a number of different *Categories*, which in turn possess a number of quality *Dimensions*[2]. 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 $x, z, y \in \mathbb{N}$, then:

Definition 1

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

Figure 2 shows this formalisation in a pictorial manner using Venn diagrams.

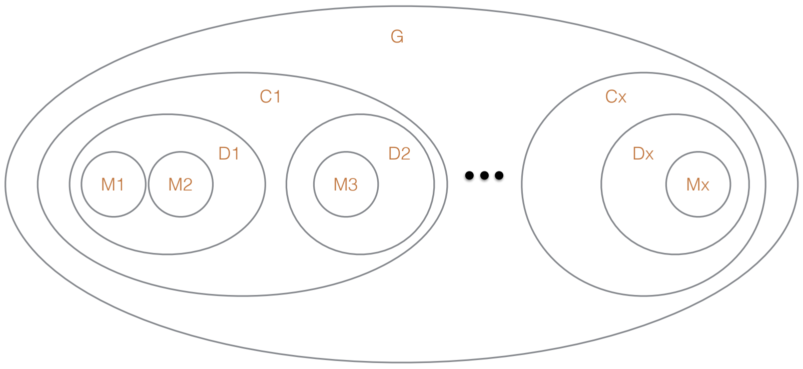

Figure 2: Venn Diagram depicting Definition 1

2.3.1 Extending daQ for Multi-Dimension Representation and Statistical Evaluation

The Data Cube Vocabulary [?] 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[3].

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 1.

[2]In this deliverable we will refer to these as quality dimensions, in order to distinguish between the data cube dimensions

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

```
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 1: The Data Structure Definition (Turtle Syntax)

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 2 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 2: The Quality Graph Definition (Turtle Syntax)

2.3.2 Abstract Classes and Properties

This ontology framework (Figure **??**) 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 protocols. 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 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 concrete 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 A typical Quality Metadata Graph

The excerpt listing in 3 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 3: 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 2), the value for the `qb:structure` property is defined as `daq:dsd` (see Listing 1). 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 Quality RESTful API Design

3 Libraries Used

4 Ranking Service

4.1 Data Quality Assessment Process

4.2 Data Quality Metrics

- Metric input is a quad ¡?s, ?p, ?o, ?g¿ -

4.2.1 Accessibility Category

4.2.2 Availability

Dereferenceability Metric
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. In Linked Data, there are two possible ways which allow publishers make URIs dereferencable. These are the `303` URIs and the `hash` URIs[4]. Yang et. al [**?**] 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.

This metric (listing **??**) will count 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 [**?**]. The metric will return a ratio of the number of dereferenced URIs (deref) against the total number of triples in a

Algorithm 1 Dereferenceablity Algorithm

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

dataset (totalTriples). The expected range is [0..1], where 0 is the worst rating and 1 is the best rating.

4.2.3 Intrinsic Category

4.2.4 Accuracy

Malformed Datatype Literals

Literals are nodes in an RDF graph, used to identify values such as numbers and dates. The RDF specifies two types of literals: plain and typed. A plain literal is a string combined with an optional language tag. A typed literal comprises a string (the lexical form of the literal) and a datatype (identified by a URI) which is supposed to denote a mapping from lexical forms to some space of values. In the Turtle syntax typed literals are notated with syntax

[4]http://www.w3.org/TR/cooluris/

such as: *"13"8sd:int* This Malformed Datatype Literals metric intends to check if the value of a typed literal is valid with regards to the given xsd datatype.

Calculates the ratio of typed literals not valid regarding its given xsd datatype to all literals

Algorithm 2 Malformed Datatype Literals Algorithm

```
1: procedure INIT
2:     totalLiterals = 0 ;
3:     malformedLiterals = 0 ;
4: procedure COMPUTEMALFORMEDDATATYPELITERALS($\langle ?s, ?p, ?o, ?g \rangle$)
5:     if (isLiteral(?o)) then totalLiterals++ ;
6:         if (isTypedLiteral(?o)) && (!hasValidDatatype(?o)) then malformedLiterals++ ;
7:     metric=malformedLiterals/totalLiterals;
```

Literals Incompartible with Datatype Range

Defined Ontology Author

POBO Definition Usage

Synonym Usage

4.2.5 Consistency

Entities As Members of Disjoint Classes

Homogeneous Datatypes

The literals assigned to a certain property should be the same datatype. E.g. lite This metric reports outliers or conflicts of literal datatypes. Since it is * meant to measure the homogeneity and not the validity possible rdfs:range * restrictions are not evaluated. * * This metric makes two distinctions as far as homogeneity is concerned: * 1) if there are just a few triples of one predicate having a datatype * different from all other triples, they are considered as outliers and * are all reported * 2) if there is no obvious ratio of possibly wrong and possibly right triples * the conflicting view candidates are reported *

Misplaced Classes or Properties

Misused Owl Datatype or Object Properties

Obsolete Concepts in Ontology

Ontology Hijacking

Undefined Classes Metric

Oftentimes a terms which is used in the object position of a triple and is not a literal is not formaly defined as being a class. 'Being defined' means that the term is defined either in some external ontology or at an earlier position in the given dataset. Regarding to Hogar [?] to the most used undefined classes belong `foaf:UserGroup`, `rss:item`, `linkedct:link`, `politico:Term` . The probability for undefined class in the subject position is very low, because the subject of a quad never references classes or properties in external vocabularies. Therefore they is no need to analyse the subject for this metric. For the most LOD data sets is sufficient to check object by the predicate `rdf:type`. In the case when LOD data set 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 define in corresponding ontology or in a separate namenspace.

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

Algorithm 3 Undefined Classes Metric Algorithm

```
1: procedure INIT
2:     totalClasses = 0 ;
3:     undefinedClasses = 0 ;
4: procedure CLASSNOTDEFINED($\langle ?s, ?p, ?o, ?g \rangle$)
5:     if (isClassProperty(?p) && (isURI(?o)) then ;
6:         if (!isDefined(?o)) undefinedClasses++ then;
7:     totalClasses++ ++;
```

Undefined Properties

Similar to the Undefined Classes metric 4.2.5 the Undefined Properties metric identifies terms in the predicate position that are used without any formal definition. Hogan [?] identified the following properties that are often used whithout being defined: *foaf:image*, *cycann:label* , *foaf:tagLine*. The following list of predicates indicate that the object of the quad must be a defined property:*rdfs:subPropertyOf*, *owl:onProperty*, *owl:assertionProperty*, *owl:equivalentProperty*, *owl:propertyDisjointWith*.

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

4.2.6 Conciseness

Duplicate Instance

Extensional Conciseness

Ontology Versioning Conciseness

Algorithm 4 Undefined Properties Algorithm

```
1: procedure INIT
2:     totalProperties = 0 ;
3:     undefinedProperties = 0 ;
4: procedure PROPERTYNOTDEFINED($\langle ?s, ?p, ?o, ?g\rangle$)
5:     if (isURI(?p) totalProperties++ then ;
6:         if (!isDefined(?p)) udefinedProperties++ then;
7:         if (isFromList(?p)) && !isDefined(?o) udefinedProperties++ then;
8:         totalProperties ++;
```

4.2.7 Representational Category

4.2.8 Representational Conciseness

Short URIs

4.2.9 Understandability

Empty Annotation Value The metric Empty Annotation Value consider the following widely used annotation properties (labels, comments, notes, etc.) and identifies triples whose property is an annotation property and whose object is an empty string.

Calculates the ratio of annotations with empty objects to all annotations in the data set.

Algorithm 5 Empty Annotation Value Algorithm

```
1: procedure INIT
2:     totalAnnotations = 0 ;
3:     emptyAnnotations = 0 ;
4: procedure ISEMPTYANNOTATION($\langle ?s, ?p, ?o, ?g\rangle$)
5:     if (isAnnotation(?p)) then
6:         if (isEmpty(?o)) then emptyAnnotations++ ;
7:         totalAnnotations++;
```

Human Readable Labelling

Labels Using Capitals

Low Blank Node Usage

Whitespace in Annotation

4.3 Visualisation of Quality Assessment

4.4 Ranking of Quality-Computed Datasets

5 Crawling Service

6 Conclusions