Rheinische Friedrich-Wilhelms-Universität
Bonn

Institut für Informatik III

Master Thesis

RDF Quality Extension for OpenRefine

Author:
Muhammad Ali Qasmi

Supervisor:
Professor Dr. Sóren Auer

A thesis submitted in fulfilment of the requirements
for the degree of Master of Science

in

Computer Science

December 2014

Matrikel-Nr.:2616574

Declaration of Authorship

I, Muhammad Ali Qasmi, declare that this thesis titled, 'RDF Quality Extension for OpenRefine' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"A computer lets you make more mistakes faster than any invention in human history..."

Mitch Ratcliffe

Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III

Abstract

Computer Science

Master of Science

RDF Quality Extension for OpenRefine

by Muhammad Ali Qasmi

The process of improving the quality of data by removing erroneous and ambiguous data is known as data cleaning. This includes both identifying and removing problematic, duplicate or out-dated data. Since the requirements to the data quality vary for different application domains, it is difficult to define a general cleaning approach. Several approaches were developed in the last years for cleaning relational data, while there is still a need for methods that help to improve quality of RDF data. In this thesis we have restricted our investigations to the domain of RDF data. The objective of the master thesis is to develop an approach for cleaning RDF data in simply but effective manner. In order to show a technical realization of our proposed cleaning approach for RDF data, we have also implemented a prototype as an extension to Open Refine cleaning framework.

Keywords: Cleaning, Data Quality, OpenRefine, RDF.

Acknowledgements

First of all, I would like to praise and thank Almighty Allah for helping me in every phase of this life. There is no doubt that He has bestowed countless blessing upon me.

I would like to further extend my gratitude to everyone who supported me throughout the course of writing this master thesis. I am thankful for their precious time, guidance, advice and positive criticism during the thesis work. I am sincerely grateful to them for sharing their insight and generous views on a number of problems related to the implementation of the project and writing the master thesis.

In the end, I would like to specially express my warm thanks to my supervisor Prof. Dr. Sóren Auer and my mentor Ms. Natalja Friesen for their support and guidance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

The advent of Semantic Web technologies and acceptance of RDF (Resource Description Framework) as a W3C recommendation has resulted in publishing huge amount of open and machine readable Linked Data on Web. This can be perceived from the fact that in 2007 Linked Open Data (LOD)[1] consists of 2 billion RDF triples interlinked with over 2 million RDF links [3]. However, in September 2011 it has grown up to 31 billion RDF triples, interlinked by around 504 million RDF links [2]. Though this rapid accumulation of Linked Data has created a huge and dense Web of Data. But data in this Web comprised of huge difference with respect to its quality [4]. Due to this variation in quality, data may contain misrepresented and incomplete information. Applications widely depending on processing this kind of data may not produce the desired results. Sometimes many important business, social or economical decisions are based on the information extracted from the Linked Data. Therefore inconsistent and incomplete information may lead to undesirable conclusions.

In order to maintain the high quality of data it is important to avoid gathering erroneous data. However, despite of reliable quality rules, well defined processes for verification and validation of data it is still possible to collect data that may consist invalid, duplicate, or expired pieces of information. Therefore a continuous refinement or cleaning of data is not only essential but also unavoidable.

Data cleaning is a process that involves removal of random or systematic errors from data through filtering, merging and translation [5]. In the process of knowledge discovery often the largest fraction of time is invested in data cleaning [6–8]. Furthermore, the

[1] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[2] http://lod-cloud.net/

process of cleaning involves careful inspection of data so that important or required information is not lost.

On the other hand, the recent developments in the field computer science has given us immense computational power and huge storage capacity. Furthermore the rapid growth of Semantic Web technologies indicates that soon or later there will be a paradigm shift in the Internet. The idea of Semantic Web technologies integrating into large scale Web applications is already on the move and is termed as Web 3.0 [9]

The backbone of these semantic technologies is the Resource Description Framework (RDF) that provide meaning to Linked Data for various websites or other data sources accessible through Web.

Nevertheless, having sufficient computational power, wide storage capacity and bulk amount of RDF triples is not sufficient to process information and extract required knowledge from it if the quality of data and information carried by the RDF data is not reliable.

A reasonable amount of work is already done to address the quality issues and data cleaning problems in relational databases. There are numerous data warehouse solutions that perform data transformation and cleaning for relational databases [10–12]. Recently, few publications propose methods and approaches to access the quality of RDF or Linked Data [4, 13, 14]. However, to this date no signification contribution is made to clean the RDF data in terms of its approach and implementation.

1.2 Problem Specification

One way to ensure sound quality of RDF data is to take preemptive measures by defining and placing concrete checks and balances before data is accepted and stored for later use. Another way is to clean RDF data and remove invalid, incorrect, duplicate and not reliable information from it.

The amount of RDF data available on Web. It is not possible to maintain its quality without the support of any data cleaning software. However, development tools supporting cleaning of RDF data will help to overcome this kind of problems. The majority of existing cleaning techniques were developed to deal with relational databases. Some of them are domain specific and hard to extend to another application fields and different data structure, e.g. RDF data. While several approaches have been developed recently to assess the quality of RDF data [4, 13] the question of how to clean RDF data in order to improve its quality remains unresolved. The master thesis address this issue by

proposing an approach for cleaning RDF data. The developed approach is then demonstrated in a software prototype that enables both identification of quality problems and cleaning dataset in a user friendly and effective manner.

1.3 Thesis Contributions

In this thesis we investigate the question of how to perform cleaning of RDF and demonstrate the proposed approach by implementing a prototype.

To develop a cleaning approach for RDF data and to implement a prototype the following steps are needed:

1. Investigate the question of how to assess the quality of RDF data. For this theories about data quality were studied, followed by research concerning with quality of RDF data. We identified a set of quality problems that could be addressed by cleaning. These problems address several aspects of data quality: it verifies whether small pieces of information are logically consistent in themselves, whether they are well-formed, and whether their representation follows good practices of data engineering and data publishing.
2. Finding a way to support the user in cleaning process by providing him with cleaning suggestions for the selected quality problems.
3. Design of cleaning workflow supporting the proposed cleaning approach.
4. Implementation of prototype demonstrating the proposed cleaning approach. The main purpose of the developed cleaning tool is to support the user in the following data cleaning issues:
   (a) Automatic identification of inconsistencies/anomalies in the data and
   (b) Generation of suggestions addressing the identified problems.

1.4 Thesis Structure

The rest of the thesis is organized in the following manner. In Chapter 2 we present related work in the area of data quality and briefly discuss some existing cleaning frameworks. In Chapter 3 we present an overview of RDF data model. Chapter 4 introduces RDF data quality and discusses in detail existing problem in RDF dataset. Followed by Chapter 5 we propose and explain our cleaning approach for RDF dataset. And in

Chapter 6 we discuss our implementation of prototype for cleaning RDF dataset. Finally, in Chapter 7 we summaries our work and propose some suggestions for further work.

Chapter 2

Related Work

The development of tool for cleaning RDf data includes several issues, like assessing quality of RDF data, techniques for data cleaning as well as design and architecture of cleaning tool, we consider the related work from these different perspectives. First we introduce existing approaches on RDF data quality, then we present the current techniques for data cleaning and finally we provide a brief overview of existing cleaning frameworks.

2.1 RDF Quality Assessment

Since the concept of data quality is multi-dimensional.The process of quality assessment requires analyzing and evaluating various quality parameters known as data quality dimensions. [15–19]. Therefore to solve any specific issue companies develop quality measures on an ad hoc basis [17, 20]. In order to make this process easy many researcher has conducted various surveys on several datasets to formulate a comprehensive list of quality dimensions. For instance Wang et al. [19] has in general defined and categorized various quality dimensions for all kind dataset.

Similarly, Zaveri et al. [4] and Hogan et al. [13] has listed and categorized various quality metrics under different quality dimensions (commonly know as quality criteria) to measure quality of RDF data. These quality metrics provides a mean to access the quality of RDF dataset with respect to various data quality dimensions and quantify the results.

Using these existing research on RDF data quality we can directly use them in the master thesis for the part concerning with quality assessment.

2.2 Cleaning Approaches

Several attempts are made to define a concrete approach to clean data that may satisfy all kind of data. However, due to huge diversity in data, variation in its quality requirements (for different applications) and high influence of user specific needs the problem to define a cleaning approach is quite inconceivable.

Erhard and Hong Hai [10] has given a approach to clean relational data in which they have divided their cleaning problem in to two broad categories i.e. schema level and instance level problem. The idea is separate schema related issues from the actual data or content, because problems related schema should be addressed and resolved with making changes in schema of data base. Still simply changing the schema is not sufficient because data that was stored with problematic schema has to be corrected. Schema related problems has direct impact on the content of data. Therefore simply resolving the schema related problems is not sufficient. For example lack of integrity constraints may results storing in invalid values to the data base.

Lup Low at el. [21] has proposed a cleaning approach which user is allowed to define cleaning rules with respect to domain knowledge and these rules are used to select different cleaning criteria that are applied to eliminate duplicate, erroneous values from the relational databases.

2.3 Cleaning Frameworks

There already exists several tools or frameworks that identifies problems in data and takes appropriate measures to clean them.

For instance, AJAX is a declarative framework that extends SQL99 and provide five atomic transformations for data cleaning i.e. mapping, view, matching, clustering and merging. These transformations can be combined together to perform further complex transformations. The framework detects and remove duplicate or redundant data by applying above mentioned transformations [7].

Potter's Wheel is an interactive data cleaning system that uses a spreadsheet like interface for performing transformations and identifying inconsistencies in data for cleaning. It has three main component: a data source, a transformation engine, an online reorderer that assists in interactive scrolling and sorting at the user interface and an Automatic discrepancy detector. Data is fetched through the data source and the transformation engine transforms data to be displayed on spreadsheet like interface. This is done at

run time as only number of rows visible to user are transformed. The automatic discrepancy detector detects inconsistency in the data already transformed and visible to user. This approach reduces initial response time for the user. User can then select various transformations through graphical user interface and see results immediately. By repeating the last two steps user can create a complex transformation to clean data in an interactive manner [22].

Intelliclean presented in [23] has three stages for data cleaning i.e. Pre-processing stage, Processing Stage and Validation and Verification Stage. In pre-processing stage data records are processed to transform in standardise format which includes converting date into a single format and removing abbreviations. The outcome of this stage is consumed by processing stage. In processing stage duplicate or synonymous records are identified based on predefined rules and their certainty factor. Each rule is broadly classified in one of the four categories i.e. Duplicate Identification Rule, Merge/Purge Rule, Update Rule, Alert Rule. Duplicate identification rules consist of rules that define conditions for two records to be considered as duplicate of each other. Merge/Purge Rule category consists of rules the defined conditions as how to merge or remove records. Update rule category consists of rules that define conditions for data to updated based on other available information. And finally Alert rule category consists of rules that defined conditions where system notify user about specific anomaly in records that may or may not be correct. In the last stage a log is generated for user specifying all actions taken for cleaning the data and along with reasons for taking those actions in previous two stages.

OpenRefine[1] (or ex-Google Refine) is a desktop based web application with spreadsheet like interface that allows users to perform various cleaning operations for example identify and remove duplicate records, filter records based on their value type i.e. text, numeric, or date, clustering, sorting, etc. It uses Google Refine Expression Language (GREL) for data transformation expression language. It maintains versions of changes in data so that if required user may revert or undo any change or cleaning operation. It has the capability to import data in various formats which include CVS, doc, xml, json, etc. An RDF extension [24, 25] is written for OpenRefine but it simply converts data from other formats in to RDF by providing a clean graphical user interface to the user where they can define relations of each column from the dataset to the resource or property of RDF graph. But it does not use any feature of RDF to assess it quality and identify quality problems or clean them from RDF dataset.

To sum up, many other cleaning approaches are proposed for relational or text based data but to the best of our knowledge there is no cleaning approach or framework for

[1] http://openrefine.org/

RDF data. The reason for selecting and discussing these four cleaning framework is that the general ideas or concepts given by these frameworks are utilized in our approach for RDF data cleaning.

Chapter 3

Resource Description Framework

In this chapter we will explain the concepts of Resource Description Framework (RDF) because it will help in understanding the data structure of RDF as well as next chapters. Knowledge of RDF data model is necessary to understand the concepts related RDF data quality and its cleaning.

3.1 Background

In February 1999, the Resource Description Framework and Syntax Specification [1] became World Wide Web Consortium (W3C) recommendation. The specification uses abstract mathematical structures (such as triples and sets) to lay down the basis for the model [26]. Therefore the RDF Data Model defined in the specification is abstract in nature. It has separated RDF data model from its syntax [27]. As a result data in RDF model can be presented in various formats (e.g. XML, JSON, etc). Non-RDF resources are easily accommodated in the model by simply assigning unique resource identifiers (URIs) to resources in non-RDF data source. The objective of RDF data model is to acts as a semantic bridge among various data and it sources. All these features make RDF data model as one of the popular frameworks to exchange data on Web.

3.2 RDF Data Model

The RDF data model is roughly similar to Object Oriented data model [27]. The model consists of entities that are interconnected through a binary relationship. When two entities are connected with each other they form a statement (also know as triple). In

[1] http://www.w3.org/TR/REC-rdf-syntax/

a statement the source entity is called subject, the relationship is called predicate (or property) and the other entity is know as object. These statements and predicates are themselves treated as first-class object and can be use as subject or object by other statements. Subjects and predicates in a statement are always a resource while objects can be both a resource or a literal. A resource always has a unique identifier (URI), a literal is an ordinary string. In a graph resources are commonly drawn as ovals and literals are represented by boxes.

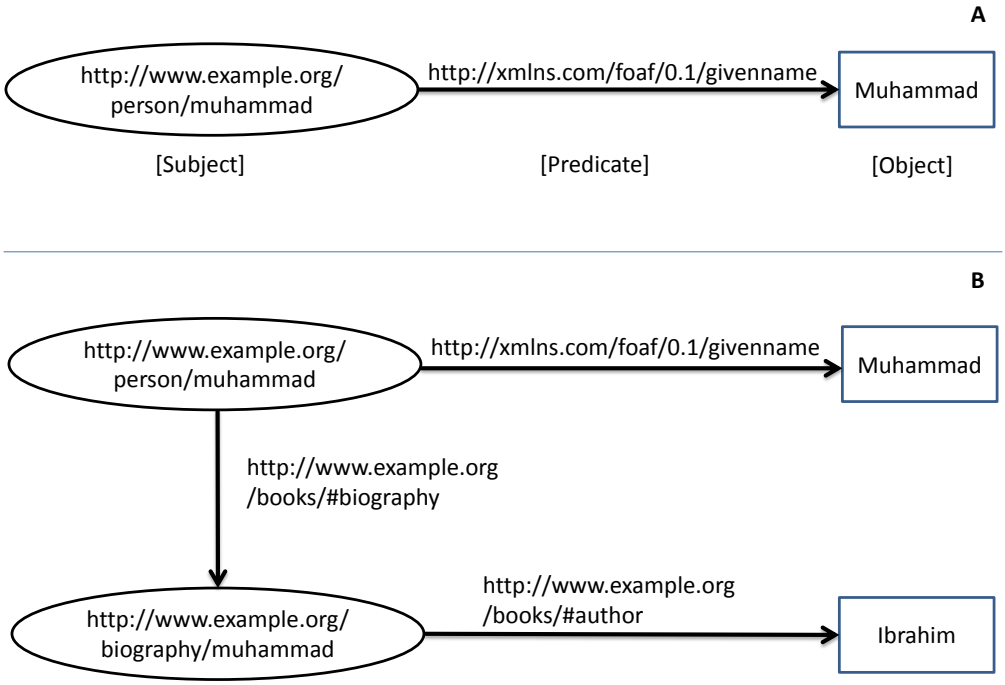


FIGURE 3.1: RDF statements example.

In figure 3.1-A shows an example for a RDF statement. It states that the resource 'http://www.example.org/person/muhammad' has a property *givenname* (defined by 'http://xmlns.com/foaf/0.1/givenname') and literla value 'Muhammad' is of type string. The statement consists of three parts. The resource 'http://www.example.org/person/muhammad' is the subject. The property is also a resource 'http://xmlns.com/foaf/0.1/givenname' which is the predicate. And the literal 'Muhammad' is the object of the statement. Similarly more statements can be added as shown in figure 3.1-B. Here another statement is added and both of them are connected by the property 'biography'. Thus figure 3.1-B consists of three RDF statements.

The URIs can be abbreviated using XMLnamespace syntax [2]. For example, in figure 3.1 we can write ex:muhammad instead of 'http://www.example.org/person/muhammad'. Here 'ex' is the abbreviated prefix of 'http://www.example.org/person/'.

[2] http://www.w3.org/TR/REC-xml-names/

3.3 RDF and RDF-Schema properties

RDF has defined it own set of properties in Resource Description Framework And Syntax Specification. The properties defined in Resource Description Framework And Syntax Specification are commonly use with namespace prefix 'rdf' which can be expanded into 'http://www.w3.org/1999/02/22-rdf-syntax-ns'. One of the most common and extensively used property is 'rdf:type' property. It indicates that the properties are related and defines the relation between them. For example sometimes we need to create statements about collection of resources. For this purpose we use 'rdf:type' property to define a resource as container.

In RDF we can define three different type of containers.

1. Bags are unordered list of resources or literals. For example a list of team members in a football team. Bags allow repetition of values therefore values in bag are not necessarily unique.

2. Sequences are ordered list of resources or literals. For example a list of months, where order of month is important in a calendar. However, sequences also allow duplicate values.

3. Alternatives is a kind of list of resources or literals in which a property can pick only one value. For example when selecting a gender, an application can only choose one value i.e. either male or female.

There are many other properties defined in Resource Description Framework And Syntax Specification. They are important because these properties are extensively used in RDF datasets. We use these properties in our cleaning approach to analyze RDF data for inconsistencies and anomalies. Properties are also used to filter the RDF data for specific data quality problems. Many common problems identified in RDF dataset are based on the incorrect usage of properties. A list of RDF properties along with their brief description is presented in table 3.1.

In order to enhance the capability of RDF data model the RDF properties are extended by Resource Description Framework Schema Specification[3]. The properties defined in Resource Description Framework Schema Specification are commonly used with namespace prefix 'rdfs' which can be resolved into 'http://www.w3.org/2000/01/rdf-schema'. RDF Schema provides a mechanism to define classes and subclasses in hierarchical fashion. It further allows to restrict a property by defining their domain

[3] http://www.w3.org/TR/2000/CR-RDF-schema-20000327

rdf:html	The data type of RDF literals storing fragments of HTML content
rdf:langString	The data type of language-tagged string values
rdf:type	The subject is an instance of a class.
rdf:Property	The class of RDF properties.
rdf:Statement	The class of RDF statements.
rdf:subject	The subject of the subject RDF statement.
rdf:predicate	The predicate of the subject RDF statement.
rdf:object	The object of the subject RDF statement.
rdf:Bag	The class of unordered containers.
rdf:Seq	The class of ordered containers.
rdf:Alt	The class of containers of alternatives.
rdf:value	Idiomatic property used for structured values.
rdf:List	The class of RDF Lists.
rdf:first	The first item in the subject RDF list.
rdf:rest	The rest of the subject RDF list after the first item.
rdf:XMLLiteral	The data type of XML literal values.

TABLE 3.1: List of RDF properties [1].

and range. This becomes very important from RDF quality perspective because defining class with hierarchies and properties with domain and range restrict RDF data to grow in a defined manner and analyze RDF data for errors. The properties defined in RDF Schema are not local to the class. They are global and are described with respect to classes they connect [27]. RDF Schema gives additional properties like 'rdfs:comment', 'rdfs:label', 'rdfs:seeAlso' to add further description in a resource.

There are many other properties defined in Resource Description Framework Schema Specification. A list of these properties along with their brief description is given in table 3.2.

rdfs:Resource	The class resource, everything.
rdfs:Class	The class of classes.
rdfs:subClassOf	The subject is a subclass of a class.
rdfs:subPropertyOf	The subject is a subproperty of a property.
rdfs:comment	A description of the subject resource.
rdfs:label	A human-readable name for the subject.
rdfs:domain	A domain of the subject property.
rdfs:range	A range of the subject property.
rdfs:seeAlso	Further information about the subject resource.
rdfs:isDefinedBy	The definition of the subject resource.
rdfs:Literal	The class of literal values, eg. textual strings and integers.
rdfs:Container	The class of RDF containers.
rdfs:member	A member of the subject resource.
rdfs:Datatype	The class of RDF data types.

TABLE 3.2: List of RDFS properties [2].

We have briefly discussed RDF data model and its properties because they play a significant role in identifying problems and cleaning RDF data. They are extensively used in the prototype of RDF quality extension and understanding them is a prerequisite to understand the workflow and other implementation related details.

Chapter 4

RDF Data Quality and Assessment

In the previous chapter we have introduced the main principles of RDF data model that are required to address the problem related to RDF data quality. In this chapter we introduce and discuss in detail the concepts related to RDF data quality, its problems and its assessment based on previous researches.

4.1 Data Quality

The quality of data is defined as the suitability or fitness of data to be used by a certain application [28, 29]. Due to huge diversity in data and its properties, previous researches have defined data quality as a multi-dimensional concept [15–18]. Factors such as accuracy, timeliness, completeness, relevancy, objectivity, believability, understandability, consistency, conciseness, availability, and verifiability can affect the quality of data [19]. Furthermore, when dealing with quality of data one must take into account the subjective perception of persons or stakeholders involved with data because the individual needs and experiences vary with data so it may effect the quality of data [16, 30]. However, only subjective perception is not sufficient. For evaluating the quality of data we also need some objective measurements. Objective assessment involves company, government or any other organization's business rules and regulations. It further includes database constraints applied in an application by its developer or database administrator [31].

RDF data is presented in triples (see chapter 3 section 3.2) so the quality of RDF data further includes the usefulness of links presents in them [32]. Due to this the quality

of RDF data is not only dependent on the information provided by individual links but also on the way they are interconnected to each other.

4.2 Quality Assessment

A data quality assessment is a process to examine the quality of data with respect to user needs under certain circumstances [33]. As the user needs vary from one application to another therefore the factors the affect quality of data are not the same for all applications. Due to this many data quality measures were created on ad-hoc basis to solve a particular problem [17, 20]. As a result many fundamental principles to create data quality metrics are lacking [31].

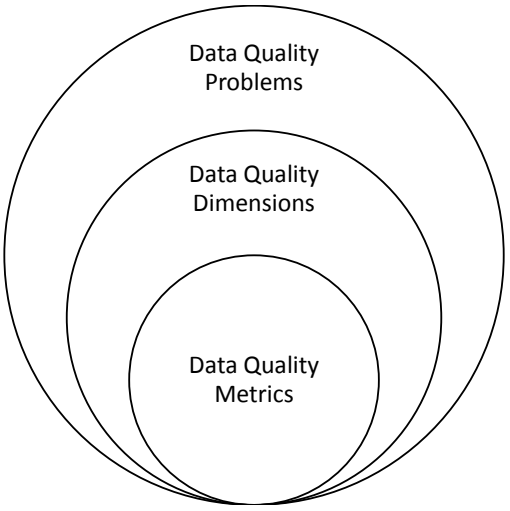


FIGURE 4.1: Connection between data quality problems, dimensions and metrics.

Assessing the quality of RDF data means to analyze the data for different kind of quality problems. Due to diversity of data the nature of quality problems differ therefore these quality problems are further classified and grouped into quality dimensions. Each quality dimension consists of a number of quality metrics to measure the quality of data in their respective dimension. Figure 4.1 shows the connection between data quality problems, dimensions, metrics in a Venn diagram. Since the introduced terms play a very important role for cleaning, we will explain each of them in more details below.

4.3 Quality Problems

Data quality problems are defined to be the problems that may occur due to diversity of data and values being in conflict from multiple data sources [14]. It also includes errors,

noise and modelling problems in data that may effect the potential of an application. [4, 13, 34].

RDF data quality problems derived from problems in information presented by different links and they way they are interconnected. As mention earlier (see chapter 3 section 3.3) RDF data model allows to define a set of reusable classes and properties for a vocabulary or ontology, many problems occur due to misrepresentation of data due incorrect usage of these classes or properties. Hogan et al. [13] has analyzed RDF dataset for quality problems and listed four key issues that are the root cause for all other related issues in any RDF dataset.

- **URI/HTTP: accessibility and derefencability**: RDF data is not available, accessible or identification of resources is not unique.
- **Syntax errors**: Software agent is unable to parse RDF data because it is not encoded in correct format.
- **Reasoning: noise and inconsistency**: A piece of RDF data is not correctly interpret because of invalid or improper usage of classes or properties.
- **Non-authoritative contributions**: Contradiction is detected in RDF dataset due to redefining the classes or properties already used in the dataset.

All other quality problems related RDF data are derived from one of the above mentioned issues. Thus these four issues represent a summaries view of RDF data quality problems.

4.4 Quality Dimensions

A data quality criteria with respect to the need of consumers is called Data Quality Dimensions. These dimensions usually represent the characteristics or properties of a dataset [4]. Some quality dimensions are specific to a special kind of dataset. For instance statical data and textual data does not necessarily need to have exactly same quality dimensions to access its quality. A statical data may consists of quality dimensions based on properties of mean, median or mode of the dataset. But same thing does not apply for textual data set.

Despite of these variations and differences there are some quality dimensions that are independent of any data model and are common to all kinds of dataset. A comprehensive list of these quality dimensions is collected in [19, 31].

Similarly, Zaveri et al. [4] has performed an survey for existing quality dimensions on RDF dataset and listed quality dimensions specific to RDF dataset. They have grouped these quality dimensions into the six categories as shown in fig 4.2.

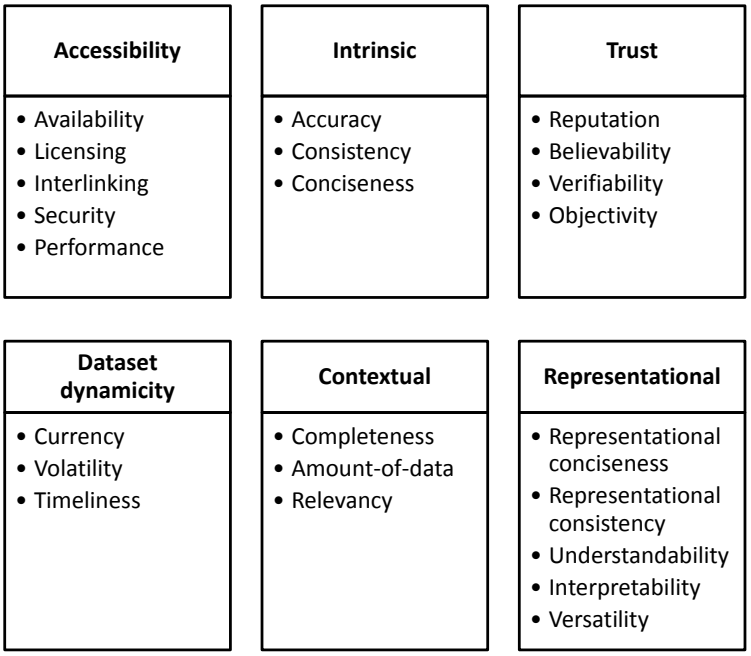


FIGURE 4.2: Data quality dimension categories.

These quality dimensions are important because they lay down the basis for any data cleaning application or framework.

4.5 Quality Metrics

The process to measure the quality of data with respect to data quality dimensions is known as data quality assessment metric (or data quality assessment measure) [35]. Each metric is a process or method to assess a specific state or condition in the dataset [36]. These states or conditions are detected based on various signs or symptoms that reflects occurrence of a certain data quality problem as defined in one of the data quality dimensions. In order to quantify the assessment results, usually a score is computed by these quality metrics using a scoring function based on a number of symptoms and its severity [4].

For RDF data the quality metrics given below are derived from the same four main RDF data quality problems given by Hogan et al. [13]. And Zaveri et al. [4] has identified

and placed them in to their respective quality dimensions. The quality of RDF data in a dimension for a metric can be computed by :

$$metric\,value = \frac{number\,of\,triples\,with\,quality\,problem}{total\,number\,of\,triples} \tag{4.1}$$

4.5.1 URI/HTTP: accessibility and derefencability

- **De-referencability issues** : RDF uses HTTP URIs to identify and look up for the resources. There URIs are used to retrieve content about the specific resource. If any of these URIs are not assessable or simply does not exist (e.g. 4xx/client and 5xx/server errors) then it causes de-referencability issue.
- **No structured data available** : Provided URI is valid and does refer to valid resource. But it does not contain any structure data to extract information from it e.g. data is simply in text instead of XML format.
- **Misreported content types** : The HTML response header does have a content type field. This issues is caused if content type is misreported or not reported at all e.g. content type text/html is reported instead of application/rdf+xml.

4.5.2 Syntax errors

- **RDF Syntax Errors** : RDF data is not presented in correct format e.g. incorrect usage of RDF/XML causes problem for software agent to parse XML and extract data from it.

4.5.3 Reasoning: noise and inconsistency

- **A typical use of collections, containers and reification** : In correct usage of collections, containers and reification e.g. a collection is defined without First, or rest properties, containers are not defined to be bag, sequence or alternative and in reification the subject of a statement is null.
- **Use of undefined classes and properties** : Usage of classes or properties in RDF data that are not defined e.g. FOAF vocabulary [1] defines a property FOFA:knows. But in RDF dataset FOFA:know is used instead of FOAF:knows.

[1] http://xmlns.com/foaf/spec/

- **Misplaced classes/properties** : This issues occurs when a resource is defined as a class is used as a property or a resource is defined as a property is used as a class e.g. rdf:type is used as subject of a statement.

- **Members of deprecated classes/properties Category** : Some classes and properties are deprecated and are no longer recommended for use. owl:DeprecateClass and owl:DeprecateProperty is used to mark a class or property obsolete in a vocabulary, respectively. However, often in dataset these obsolete classes or properties are used that effects the quality of RDF dataset.

- **Malformed data type literals** : RDF allows to define data type for literal values. However, sometimes these literal values do not obey the lexical syntax as define for its respective data type e.g. usage of xsd:dateTime with a string the represents date only (or the literal value does not give data in required format).

- **Literals incompatible with data type range** : RDF allows to define data type for literal values. However, often incorrect data types are used e.g. usage of xsd:nonNegativeInteger data type with a negative integer literal value or plain non-numeric string.

4.5.4 Non-authoritative contributions

- **Ontology hijacking** : This kind of problem occurs when a predefine class or property is redefine by third party and used along with previously defined vocabularies. It causes an ambiguity between old and new definition of the class or property.

As mentioned earlier each of these quality metrics represent a specific state of problem in dataset. And thus are used to identify quality problems in RDF dataset. Details related computation of these metrics are discussed in chapter 6 section 6.4.

Chapter 5

RDF Data Cleaning

In the previous chapter we have introduced the concepts related to RDF data quality and RDF quality assessment. We also introduced and discussed in detail various terms related to these concepts e.g. RDF data quality dimension and RDF data quality metrics. In this chapter we will focus on problems related to RDF data cleaning and propose an approach to solve them.

5.1 Cleaning

Cleaning is defined as a process of improving the quality of data by identifying and eliminating errors and inconsistencies from data. Data cleaning is also referred as data cleansing or data scrubbing [10].

By the definition of data quality the requirements for quality of data may vary from one application to another. As a consequence data cleaning approaches and strategies will also vary from on context of an application to another. It is important to thoroughly understand and clearly establish application's requirements with respect to the quality of data. Otherwise in the light of garbage in and garbage out principle misleading or incorrect information will further produce defective and inconsistent results. As a consequence the time, effort and cost invested in processing the data will go in vain.

Data cleaning problems are by large solved through the combination of two major processes i.e. data transformation and data cleaning. According to [37] data transformation is a process of amending the structure, representation and content of data. It becomes increasing critical in places where different data source are integrated together. Furthermore, it is also required to deal with problems raised due to data migration from on system to another.

Data transformation plays a vital role in the process of cleaning of data. Sometimes this term is interchangeably used with data cleaning because the outcome of both processes is almost similar i.e. data with improved quality. However, technically they both are separate processes and despite of many overlapping features. They have some differences. Therefore, it is important differentiate between them.

Erhard Rahm at el. [10] has worked on data cleaning problems for relational databases and they have classified them into two broad categories i.e. single-source problems and multiple-source problems. Each of these categories are further divided into schema level and instance level problems. Problems that can be addressed by only updating the schema of data comes under the flag of schema level problems. e.g. lack of integrity constrain. These kind of problems are often resolve using data transformation technique.

However, problems that need changes or improvements in the actual content of data are placed in the category of instance level problem. e.g. data entry problems, misspellings. These kind of problems are not simply identified by analyzing the schema of data. Since problems found in this category are related to actual content of data instead of schema. Therefore they cannot be resolved by using data transformation techniques. And thus these problems are primary addressed or related to data cleaning.

We adopt these data cleaning concepts of data transformation and cleaning from relation databases for RDF dataset.

The transformation process is used to remove syntax errors from RDF data and ensure all data is in valid RDF format. Otherwise software will not be able to parse the data and as a consequence cleaning of content present in RDF dataset will not take place. In the master thesis we don't consider quality problems caused by data transformation and related to invalid RDF format e.g. RDF syntax errors (see chapter 4 section 4.5).

As far as data cleaning of RDF dataset is concerned, Hogan et al. [13] has structured all RDF quality issues into four main categories:

1. **incomplete data** : Part of content presented in RDF dataset is not complete because it is either not assessable or not available due to improper, broken or invalid HTTP URIs links.
2. **incoherent data** : Local content present in RDF dataset cannot be interpret properly by a software agent because invalid, improper usage of classes or properties.
3. **hijack data** : Remote content present in RDF dataset cannot be interpret properly by a software agent because local content has re-defined some classes or properties.

4. **inconsistent data** : A contradiction is detected in RDF dataset by the software agent.

These four categories of symptoms encapsulates all major issues related to RDF data cleaning. Therefore we primarily assess RDF data for these four symptoms. With respect to each symptom we identify different quality problems and apply appropriate method to clean them.

In order to develop a general cleaning approach we choose some cleaning problems representative for each of the four categories shown in table 5.1.

Category	Quality Problem(s)
Incomplete data	Empty annotation value.
Incoherent data	Undefined classes and properties, misplaced classes or properties, malformed data type literals.
Hijack data	Ontology hijacking.
Inconsistent data	Incompatible with data type range, labels using capitals, white space in annotation.

TABLE 5.1: Quality problems in their respective categories.

Quality problems in these four categories of issues also require user involvement to clean them. For example in issue like empty annotation value we need user to fill in the missing piece of information, similarly in undefined classes or properties user is required to define classes or properties. Carefully evaluating each of these quality problems lead us to define a cleaning process for RDF dataset.

5.2 Cleaning Process

Data cleaning process refers to the steps, phases or stages of various processes through which data is passed to remove errors, inconsistency and improve its quality. The outcome of each stage is the input for other. However the sequence of these stages is important and usually data has to at least once pass through all the stages to complete its cycle of cleaning.

We studied existing cleaning approaches for relational and text data (see chapter 2 section 2.2)) and we also examine some existing cleaning frameworks (for details see chapter 2 section 2.3) to define our cleaning process for RDF data by the following steps:

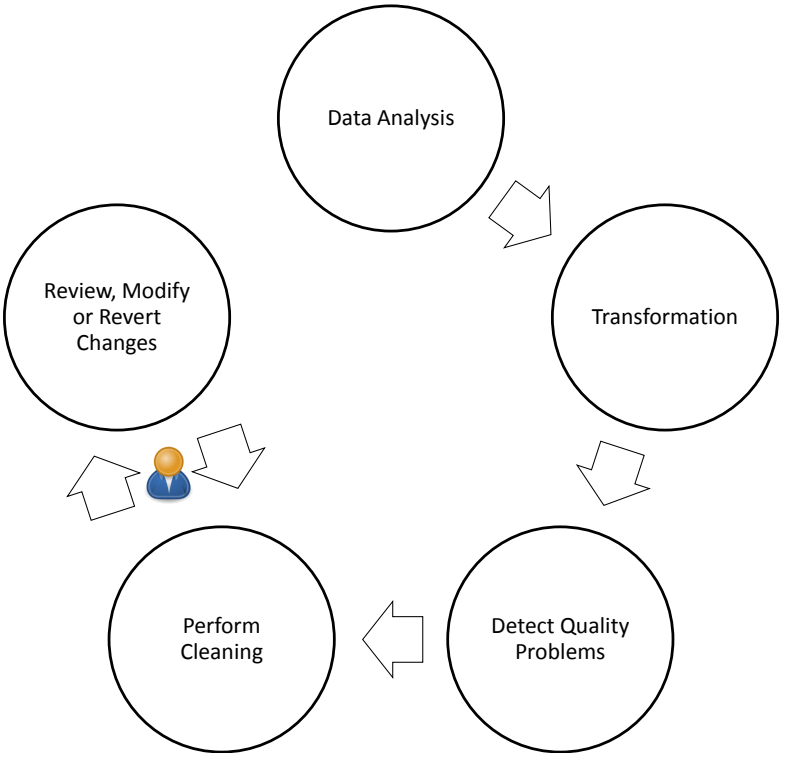


FIGURE 5.1: Main steps of the cleaning process.

1. **Data Analysis** : Detail data analysis is performed to extract information related to metadata, external vocabularies or any other property of RDF data. This includes structural information like data presentation format e.g. XML,Turtle, N3, etc. It also includes storing information that we may loose due to data transformation for cleaning like URI prefix, grouping of triples, etc. Syntax errors are also checked here so that data is easily parsed for transformation and cleaning.

2. **Transformation** : Transformation is performed to bring RDF data in a simpler state that can be easily utilized to detect quality problems and perform cleaning in an optimized way. The may include replacing all URI prefixes with their actual URI and de-coupling all groups of triples into statements where each triple contains only single subject, single predicate and single object.

3. **Detect Quality Problems** : Once transformation is complete, data is further processed to identify quality problems found in the dataset. RDF quality metrics are used to identify the quality problems in the dataset. The information related to the type of quality problem is added to all identified quality problems. This information is then later used for cleaning the identified quality problems. Some triples may have more than one quality problems. Therefore grouping these quality problems is also performed in this step.

4. **Perform Cleaning** : Based on quality problems found in the previous step the information about the quality problems, cleaning suggestion for the specific problems and for some cases regular expression that should be applied to the triple in order to clean it is provided to support the user in cleaning process.

5. **Review, Modify or Revert Changes** : During the cleaning process user is provided with interface to review the dataset. This interface clearly reflect all the changes done in dataset and allows the user to freely review, modify and revert any changes in the dataset.

The last two stages are repeated until a satisfactory level of data quality is achieved. As user satisfaction is quite subjective. Therefore number of repetition for these two stages is to be decided by user.

Chapter 6

RDF Quality Extension

In the previous chapter we have discussed about RDF data cleaning problems and proposed an approach for RDF data cleaning. In this chapter we will describe in detail the implementation of our cleaning approach for RDF data as an extension of OpenRefine.

6.1 OpenRefine

OpenRefine (formally known as Google Refine) is an open source data manipulation tool. It has features that allow users to clean, transform, reshape, smartly modify unstructured and messy data. It has an easy to use spreadsheet like interface. It stores all the data locally therefore it gives leverage to its user to work offline. It also provide various end points for developers to extent its features. Among them DBpedia[1], Geoxtension[2], Named-Entity Recognition[3], Crowdsourcing[4] are few notable extensions for OpenRefine.

The RDF quality extension is an extension of OpenRefine for identifying and cleaning RDF datasets. We have chosen OpenRefine as our basic platform because it has many salient features that we can use in our advantage for cleaning RDF datasets. They are mentioned below :

- **Elegant Interface**: OpenRefine has elegant and easy to use spread sheet like interface that allows user to have direct interaction with data. Users can manipulate data and see the changes immediately. Furthermore, all controls are visible, position and arranged in accordance to their likelihood for usage. For instances,

[1] https://github.com/sparkica/dbpedia-extension
[2] https://github.com/giTorto/geoXtension
[3] http://freeyourmetadata.org/named-entity-extraction/
[4] https://github.com/sparkica/crowdsourcing

controls for the operations related to columns are grouped together and placed above each column of the dataset in a drop down list (see figure 6.6). This naturally establishes the relationship between these controls and data in column for users.

- **Efficient Memory Usage**: OpenRefine does not load all the data from the disk in to the memory of system at once. Because dataset can be huge and usually a system has limited memory as compare to dataset. Therefore, Open Refine only loads number of rows that are visible to user from the disk. Users are able to manipulate with rows visible to them and thus system is able to efficiently utilize it resources.

- **Search, Filter and Cluster**: OpenRefine supports columns based search and filter operation that allows user to quickly and efficiently view the portion of dataset of their interest. Search operation includes keywords text search where as filter includes text facets, numeric facet, time line facets and custom facets (see figure 6.7). These facets allows user to extract a portion of data from whole dataset based on its feature. For example user can extract all rows for a column that contains numeric values only; using numeric facet. OpenRefine also allows its users to apply multiple filter at the same time. Furthermore based on applied filters it automatically presents cluster of data with in the dataset to the user. These clusters are formed based on textual similarity of in the dataset. Search, filter and cluster operations become very crucial for user to view data from different aspects and re-view the effect of cleaning operations.

- **History Management** : History management is feature of Open Refine that keeps the track of all changes applied on dataset. User can revert or re-do those changes as per their requirements. This gives user a safe way to manipulate and clean data without having the fear for important or required data being lost while performing any clean operation.

- **Project Management** : OpenRefine treats every individual data set as a project. These projects are portable and the state of these projects are maintained through out the cleaning phase of dataset. As a result user can backup, import and export these project to other machines and perform the cleaning activity in multiple sessions.

As mentioned in chapter 2 there exist many other cleaning frameworks and OpenRefine is not the only option. But all the above mentioned features and OpenRefine build-in mechanism to implement and extend new feature in the form of plug-in or extension

makes it quite suitable for being selected over others. This is the reason we choose OpenRefine tool to implement our prototype for RDF data cleaning.

6.2 Architecture

RDF quality extension has a client-server architecture (as shown in figure 6.1) because it is based on the architecture[5] of OpenRefine. However, since OpenRefine is intended to be used locally on user's machine therefore both client and server side are installed on the same system.

The client side of the extension is integrated with graphical user interface of OpenRefine. Therefore it functions and deals in the same manner as defined by Open Refine. The only difference comes when user selects any RDF cleaning operation and as a consequence an Ajax post call with respect selected command is send to the server which is handled by server side implementation of the extension. This Ajax post call consist of all necessary information required by server-side of the extension to execute the requested command. It includes project id, command id, column id, etc. The entire client side is implemented in java script.

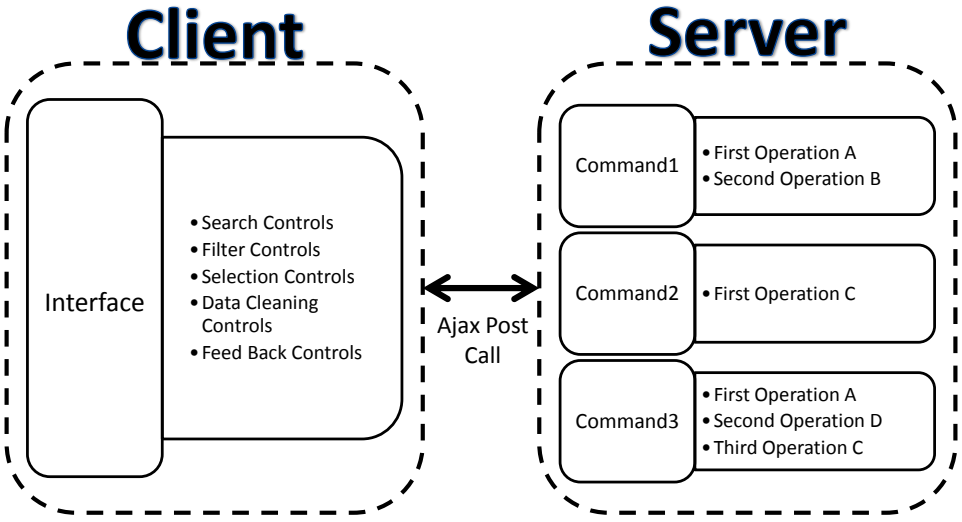


FIGURE 6.1: OpenRefine's Architecture for Quality Extension.

[5] https://github.com/OpenRefine/OpenRefine/wiki/Architecture

The server side of the extension is responsible for retrieving, maintaining, updating versions of dataset with respect to operations applied on them. Each command is identified based on its 'command id' a received through Ajax post call from the client. The particular portion of dataset is identified using 'project id' from Ajax post call and project's state. This portion of data is retrieved and processed with accordance to command executed over it. Each command consists of one or more than one operations. An operation is a unit task executed over the dataset. The sequence of the execution of operation as defined in the command is important because the changes made by one operation over the dataset is required by the other. Once the command is executed successfully, an Ajax notification is sent to the client side to notify the user. Furthermore, state of the project is updated and a record about changes in the dataset is stored as history.

6.3 Cleaning Workflow

The cleaning workflow of RDF quality extension is based on the cleaning process discussed in chapter 5. Over here we describe each step that user will perform to clean any RDF dataset using RDF quality extension. In a nutshell the user launches the OpenRefine with RDF quality extension, loads RDF dataset in new project, has its quality assessed, then cleans the dataset, review changes and finally exports clean dataset.

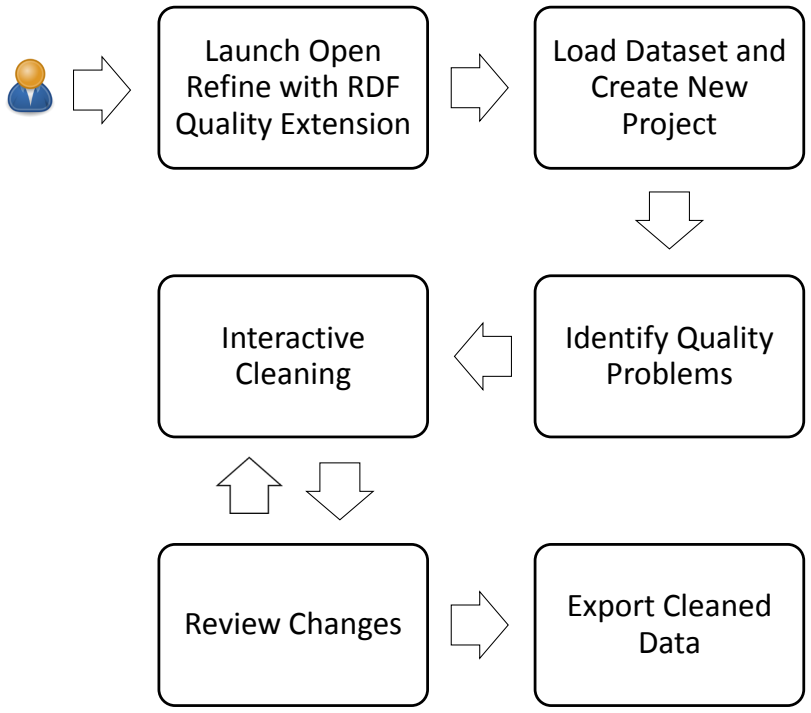


FIGURE 6.2: Cleaning Workflow of RDF Quality Extension.

1. **Launch OpenRefine with RDF Quality Extension**: For the first time user has to download, install and configure OpenRefine with RDF quality extension. The installation and configuration steps are same as for any extension for OpenRefine and they are provided in OpenRefine documentation [6]. Once this is done successfully user may run OpenRefine with RDF quality extension.

2. **Load Dataset and Create New Project**: User may load data in Open Refine from local disk or web. Both options are provided by OpenRefine. User may manually choose from various parse options. In our case default parse option works fine. However, in any case OpenRefine will copy the data locally into its own repository and create a new project. This locally copied dataset is only accessible to the new project created by OpenRefine. This is the default behaviour of OpenRefine and it cannot be changed by any external extension because OpenRefine is designed to be used offline. Furthermore, it projects original data from being corrupted and allows user to work with OpenRefine without any security concerns.

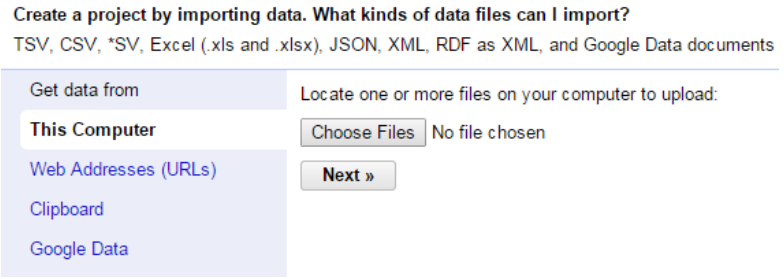


FIGURE 6.3: Load Data and Create new project in OpenRefine.

3. **Identify Quality Problems**: Once data is loaded in new project successfully. OpenRefine can now access the quality of RDF dataset by using features implemented by RDF quality extension. RDF quality extension is well integrated with OpenRefine interface. Therefore, user simply needs to click on quality button and then identify quality problems button. OpenRefine will start the process of assessing the quality of RDF dataset loaded in the current project (see figure 6.4).

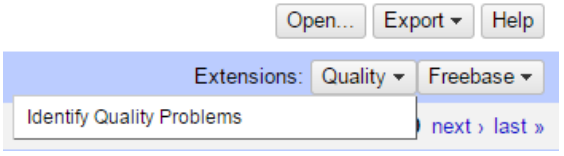
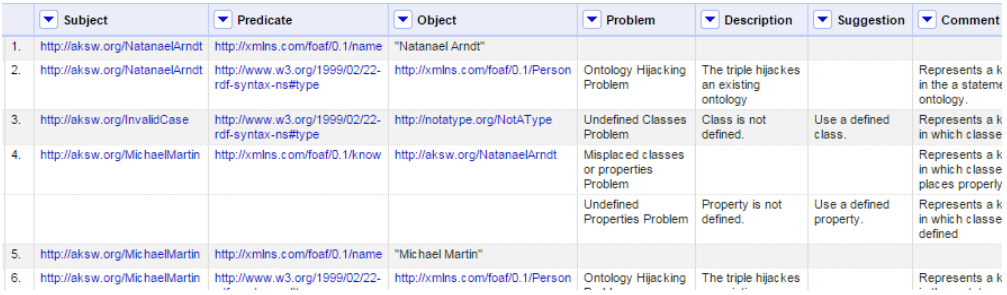


FIGURE 6.4: Identify quality problems of dataset.

Once quality assessment is complete a cleaning report is generated. This cleaning report consists of cleaning suggestion based on predefined cleaning ontology, discussed in more detail in section 6.5.

[6] https://github.com/OpenRefine/OpenRefine/wiki/Write-An-Extension

4. **Interactive Cleaning**: For cleaning the dataset, the original data is presented along with the cleaning report in the OpenRefine spreadsheet like interface. Each row consists of the following Columns. Subject, Predicate, Object, Problem, Description, Suggestion and Comments (see figure 6.5).



	Subject	Predicate	Object	Problem	Description	Suggestion	Comment
1.	http://aksw.org/NatanaelArndt	http://xmlns.com/foaf/0.1/name	"Natanael Arndt"				
2.	http://aksw.org/NatanaelArndt	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://xmlns.com/foaf/0.1/Person	Ontology Hijacking Problem	The triple hijackes an existing ontology		Represents a k in the a statem ontology.
3.	http://aksw.org/InvalidCase	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://notatype.org/NotAType	Undefined Classes Problem	Class is not defined.	Use a defined class.	Represents a k in which classe
4.	http://aksw.org/MichaelMartin	http://xmlns.com/foaf/0.1/know	http://aksw.org/NatanaelArndt	Misplaced classes or properties Problem			Represents a k in which classe places properly
				Undefined Properties Problem	Property is not defined.	Use a defined property.	Represents a k in which classe defined
5.	http://aksw.org/MichaelMartin	http://xmlns.com/foaf/0.1/name	"Michael Martin"				
6.	http://aksw.org/MichaelMartin	http://www.w3.org/1999/02/22-	http://xmlns.com/foaf/0.1/Person	Ontology Hijacking	The triple hijackes		Represents a k

FIGURE 6.5: Columns in a row of OpenRefine cleaning interface.

In this view user has the option to inspect each component of a RDF triple and view the problems associated with it side by side. Multiple quality problems with a RDF triple are also group together (see figure 6.5 forth row). Furthermore, user may apply various transformation for individual columns or cells (see figure 6.6).

FIGURE 6.6: Guided manual cleaning using column transformation feature.

5. **Review Changes**: As mentioned in section 6.1; OpenRefine also support facet browsing to extract a subset of rows to be transformed for cleaning. This feature allows user to view and deal with individual quality problems separately (see figure 6.7). And easily apply the cleaning suggestion to the relevant subset of RDF triples.

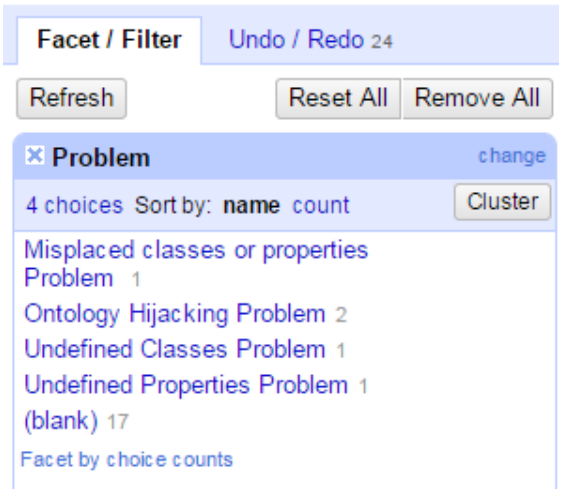


FIGURE 6.7: Facet Browsing

OpenRefine also support text, numeric search and filter operations. Furthermore it gives the feature to forms the clusters of similar data found in a column. User may revert any change done by previous transformations for cleaning (see figure 6.8).

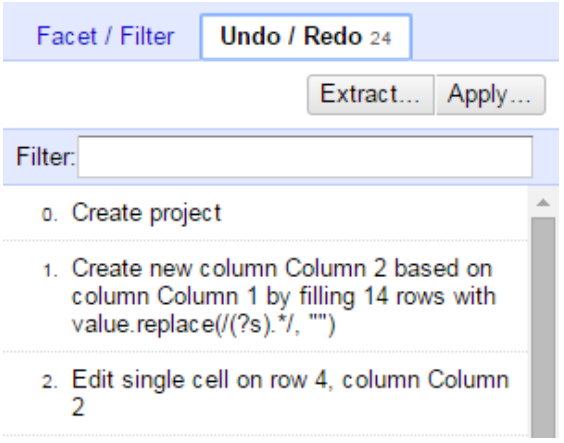


FIGURE 6.8: Undo/Redo Operations

User many also define custom cleaning rules using Google Refine Expression Language (also know as GREL) supported by OpenRefine (see figure 6.9) e.g. for White Space In Annotations problem user may use GREL expression like 'value.trim()' to remove leading and ending white spaces.

In a nutshell, the triples identified with quality problems are indicated with their type of quality problem followed by a cleaning suggestion. With facets browsing user can filter out quality problems of same kind. User may apply cleaning suggesting (e.g. GREL expression) to remove the problems. This supports user to quickly and efficiently clean the dataset. Since changes are reflected on the interface therefore user may review all the changes done during the cleaning phase

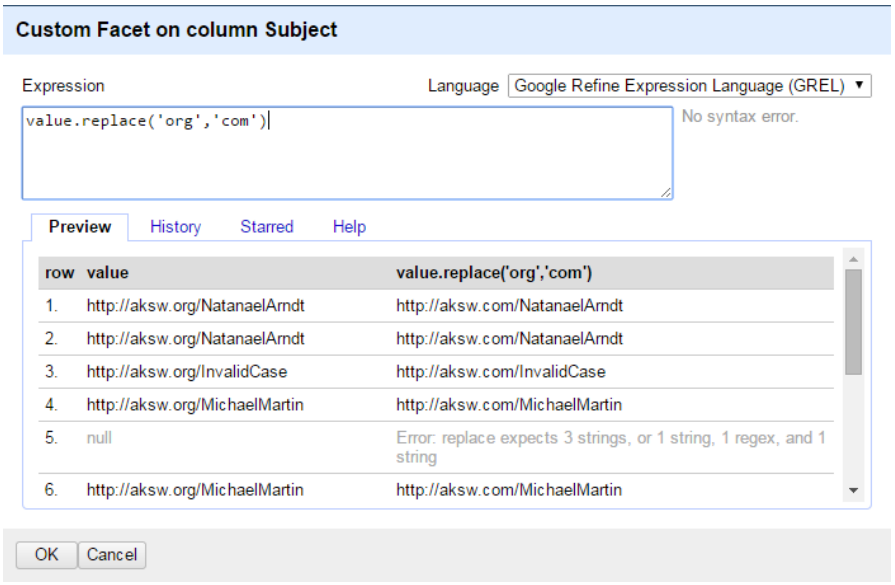


FIGURE 6.9: GREL Interface

and base on the review user may decide to further clean the data and resolve any ambiguity left in the dataset.

6. **Export Cleaned Data**: Once cleaning is done user may export all the clean dataset in the required format. Due to time limitation, currently we were not able to implement this feature completely. We have done some worked to export data in format like RDF/XML and Turtle format by storing the structure related information in the meta data of project and applying those structural information on the dataset. But, this feature is not functional yet. For now our export feature simply dump the clean data for all the triples in file in which each row consists of a triple (subject, predicate and object).

6.4 Metric Implementation

The RDF quality extension implements quality metrics to identify different kind of problems in RDF dataset. These metrics are selected, developed and implemented based on previous discussed quality metrics in chapter 4. Since quality metrics are the heart of RDF quality extension and they are the primary reason that differentiate RDF quality extension from rest of cleaning features provided by OpenRefine or other extensions of OpenRefine therefore it is vital to briefly discuss the implementation of all current implemented metrics.

6.4.1 Empty Annotation Value

Empty Annotation Value considers the widely used RDF annotation properties e.g. labels, comments, notes, etc and identifies triples whose property's object value is empty string. The algorithms to compute RDF triples with empty annotation value is given in algorithm 1.

Algorithm 1 Algorithms to compute triples with Empty Annotation Value problem

Data: list of RDF triple li, list of RDF predicate (for annotation properties) la
Result: list of RDF triples with Empty Annotation Value problem lp
initialization triple t = null, predicate p = null, object o = null, string v = null;
while *NOT at end of li* **do**
    read triple t from li;
    p = predicate in t;
    if *p is in la* **then**
        o = object in t;
        if *o is a literal* **then**
            v = value in o;
            if *v is empty* **then**
                add t in lp;
            end
        end
    end
    increment index for li counter;
end

6.4.2 Incompatible Data type Range

Incompatible data type Range detects incompatible data type of literals by comparing its data type URI with the data type URI specified in the range of the Object's predicate.The algorithms to compute RDF triples with incompatible data type range is given in algorithm 2.

Algorithm 2 Algorithms to compute triples with Incompatible Data Type Range problem

Data: list of RDF triple li

Result: list of RDF triples with Incompatible Data Type Range problem lp

initialization triple t = null, predicate p = null, object o = null, string v = null, subject r = null, vocabulary v = null;

```
while NOT at end of li do
    read triple t from li;
    p = predicate in t;
    o = object in t;
    if o is a literal then
        if p is a URI then
            v = get vocabulary from URI in p;
            r = read rdfs property range of p from v;
            \\check URI of r is not same as URI of p
            if r is NOT equal to p then
                add t in lp;
            end
        end
        increment index for li counter;
    end
end
```

6.4.3 Labels Using Capitals

Labels Using Capitals considers the widely used RDF annotation properties e.g. labels, comments, notes, etc and identifies triples whose property's object uses a bad style of capitalization. The algorithms to compute RDF triples with labels using capitals is given in algorithm 3

Algorithm 3 Algorithms to compute triples with Labels Using Capitals problem

Data: list of RDF triple li, list of RDF predicate (for annotation properties) la, list of bad style capitalization regex string lr

Result: list of RDF triples with Labels Using Capitals problem lp

initialization triple t = null, predicate p = null, object o = null, string v = null, string r = null;

```
while NOT at end of li do
    read triple t from li;
    p = predicate in t;
    if p is in la then
        o = object in t;
        if o is a literal then
            v = value in o;
            while NOT at end of lr do
                read r from lr;
                \\using regex to detect bad style capitalization
                if v NOT satisfy r then
                    add t in lp;
                    break;
                end
                increment index for lr counter;
            end
        end
    end
    increment index for li counter;
end
```

6.4.4 Malformed Data type Literals

Malformed Data type Literals detects whether the value of a typed literal is valid with respect to its given data type. The algorithms to compute RDF triples with malformed data type literals is given in algorithm 4

Algorithm 4 Algorithms to compute triples with Malformed Data type Literals problem

Data: list of RDF triple li, list of regex string for RDF data type lr

Result: list of RDF triples with Malformed Data type Literals problem lp

initialization triple t = null, object o = null, string v = null, string d = null, string r = null;

while *NOT at end of li* **do**
    read triple t from li;
    o = object in t;
    if *o is a literal* **then**
        d = get data type of o;
        r = get regex of data type d from lr;
        v = value in o;
        \\apply regex r on v to check value is malformed
        if *v NOT satisfy r* **then**
            add t in lp;
        end
    end
    increment index for li counter;
end

6.4.5 Misplaced Class or Property

Misplaced Property find resources that are defined as a property but also appear on subject or object positions in other triples. Similarly, Misplaced Class find resources that are defined as a class but also appear on predicate position in other triples.The algorithms to compute RDF triples with misplaced class is given in algorithm 5 and for misplace property is given in algorithm 6.

Algorithm 5 Algorithms to compute triples with Misplaced Class problem

Data: list of RDF triple li, list of RDF triple in li with predicate of value RDF 'type' la

Result: list of RDF triples with Misplaced Class problem lp

initialization triple t = null, subject s = null, object o = null, vocabulary v = null

boolean b = FALSE;

while *NOT at end of li* **do**

read triple t from li; s = subject in t; o = object in t;

if *s is URI* **then**

\\look for definition in given data

if *s exists in la* **then**

if *s is defined as property in la OR la has property of domain for s OR la has property of range for s* **then**

b = TRUE;

end

else

\\look for definition in external vocabulary

v = get vocabulary from URI in s;

\\if domain and range exists then it is NOT a class

if *s is defined as property in v OR v has property of domain for s OR v has property of range for s* **then**

b = TRUE;

end

end

end

if *b is FALSE AND o is a URI* **then**

\\look for definition in given data

if *o exists in la* **then**

if *o is defined as property in la OR la has property of domain for o OR la has property of range for o* **then**

b = TRUE;

end

else

\\look for definition in external vocabulary

v = get vocabulary from URI in s;

\\if domain and range exists then it is NOT a class

if *o is defined as property in v OR v has property of domain for o OR v has property of range for o* **then**

b = TRUE;

end

end

end

if *b is TRUE;* **then**

add t in lp; b = FALSE;

end

increment index for li counter;

end

Algorithm 6 Algorithms to compute triples with Misplaced Property problem

Data: list of RDF triple li, list of RDF triple in li with predicate of value RDF 'type' la

Result: list of RDF triples with Misplaced Property problem lp

initialization triple t = null, predicate p = null, vocabulary v = null;

while *NOT at end of li* **do**
    read triple t from li;
    p = predicate in t;
    if *p is URI* **then**
        \\look for definition in given data
        if *p exists in la* **then**
            if *p is defined as class in la OR (la do NOT have property of domain for p AND la do NOT have property of range for p)* **then**
                add t in lp;
            end
        else
            \\look for definition in external vocabulary
            v = get vocabulary from URI in p;
            \\if p is defined as a class then it is NOT a property
            if *p is defined as class in v OR (v do NOT have property of domain for p AND v do NOT have property of range for p)* **then**
                add t in lp;
            end
        end
    end
    increment index for li counter;
end

6.4.6 Ontology Hijacking

The Ontology Hijacking detects the redefinition by analyzing defined classes or properties in dataset and looks of same definition in its respective vocabulary. The algorithms to compute RDF triples with ontology Hijacking is given in algorithm 7

Algorithm 7 Algorithms to compute triples with Incompatible Data Type Range problem

Data: list of RDF triple li, RDF 'type' property a

Result: list of RDF triples with Ontology Hijacking problem lp

initialization triple t = null, subject = null, predicate p = null, object = null, vocabulary v = null;

while *NOT at end of li* **do**
    read triple t from li;
    p = predicate in t;
    if *p is equal to a* **then**
        s = subject in t;
        v = get vocabulary from URI in s;
        if *s is defined in v* **then**
            add t in lp;
        end
    end
    increment index for li counter;
end

6.4.7 Undefined Classes

Undefined Classes detects undefined classes from dataset by checking for its definition in their respective referred vocabulary.The algorithms to compute RDF triples with undefined classes is given in algorithm 8

Algorithm 8 Algorithms to compute triples Undefined Classes problem

Data: list of RDF triple li, list of RDF triple in li with predicate of value RDF 'type' la

Result: list of RDF triples with Undefined Classes problem lp

initialization triple t = null, subject s = null, subject s' = null, vocabulary v = null, boolean b = FALSE;

```
while NOT at end of li do
    read triple t from li;
    s = subject in t;
    \\look for definition in given data
    while NOT at end of la do
        s' = subject in la;
        if s equals s' then
            b = TRUE; break;
        end
        increment index for la counter;
    end
    if f equals FALSE then
        if s is a URI then
            v = get vocabulary from URI in s;
            \\look for definition in external vocabulary
            if s is NOT defined in v then
                add t in lp;
            end
        end
    end
    b = FALSE; increment index for li counter;
end
```

6.4.8 Undefined Properties

Undefined Properties detects undefined properties from dataset by checking for its definition in their respective referred vocabulary.The algorithms to compute RDF triples with undefined properties is given in algorithm 9.

Algorithm 9 Algorithms to compute triples with Undefined Properties problem

Data: list of RDF triple li, list of RDF triple in li with predicate of value RDF 'type' la

Result: list of RDF triples with Undefined Properties problem lp

initialization triple t = null, predicate p = null, predicate p' = null, vocabulary v = null, boolean b = FALSE;

```
while NOT at end of li do
    read triple t from li;
    p = predicate in t;
    \\look for definition in given data
    while NOT at end of la do
        p' = predicate in la;
        if p equals p' then
            b = TRUE; break;
        end
        increment index for la counter;
    end
    if f equals FALSE then
        if p is a URI then
            v = get vocabulary from URI in p;
            \\look for definition in external vocabulary
            if p is NOT defined in v then
                add t in lp;
            end
        end
    end
    b = FALSE; increment index for li counter;
end
```

6.4.9 White space in Annotation

White space In Annotation consider the widely used annotation properties e.g. labels, comments, notes, etc and identifies triples whose property's object value has leading or ending white space.The algorithms to compute RDF triples with empty annotation value is given in algorithm 10.

Algorithm 10 Algorithms to compute triples with White space in Annotation problem

Data: list of RDF triple li, list of RDF predicate (for annotation properties) la, list of regex for leading and ending white space lr

Result: list of RDF triples with White space in Annotation problem lp

initialization triple t = null, predicate p = null, object o = null, string v = null, string r = null;

```
while NOT at end of li do
    read triple t from li;
    p = predicate in t;
    if p is in la then
        o = object in t;
        if o is a literal then
            v = value in o;
            while NOT at end of lr do
                read r from lr;
                \\using regex to detect leading or ending white spaces
                if v NOT satisfy r then
                    add t in lp;
                    break;
                end
                increment index for lr counter;
            end
        end
    end
    increment index for li counter;
end
```

6.4.10 Performance Optimization

Some of the metrics mentioned the previous section (e.g. Misplaced Class or Property, Ontology Hijacking) are depended on the vocabularies that are not available locally and thus they are to be downloaded before the computation of these metrics. Since many times these vocabularies are huge and download them from Internet takes considerable amount of time. Therefore two level cache is also implemented to optimize the over all performance of RDF quality extension.

Whenever a metric is required to download any vocabulary it first checks whether the required vocabulary is available in the local memory. If found then that vocabulary from that local memory is used otherwise required vocabulary is search with in the file cache and if found then this vocabulary is loaded into local memory and used. Similarly, if

required vocabulary is not found in the file then it is downloaded from the Internet and stored in file cache as well as local memory cache and then used by the metric.

In this way required vocabularies are only downloaded once. File cache ensures that these vocabularies are stored permanently on local machine for later use by other metrics or on other datasets.

6.5 Ontologies for Cleaning

The formal representation of shared concepts and their relations for an specific domain is known as ontology [38]. An ontology is a collection of terms with exact definition of the meaning of terms defined in the vocabulary and a set of axioms.

We have designed a Cleaning Ontology to describe the concepts related to different kind of quality problems in RDF dataset and to also provide possible solution for them. Figure 6.10 gives a general structure of cleaning ontology.

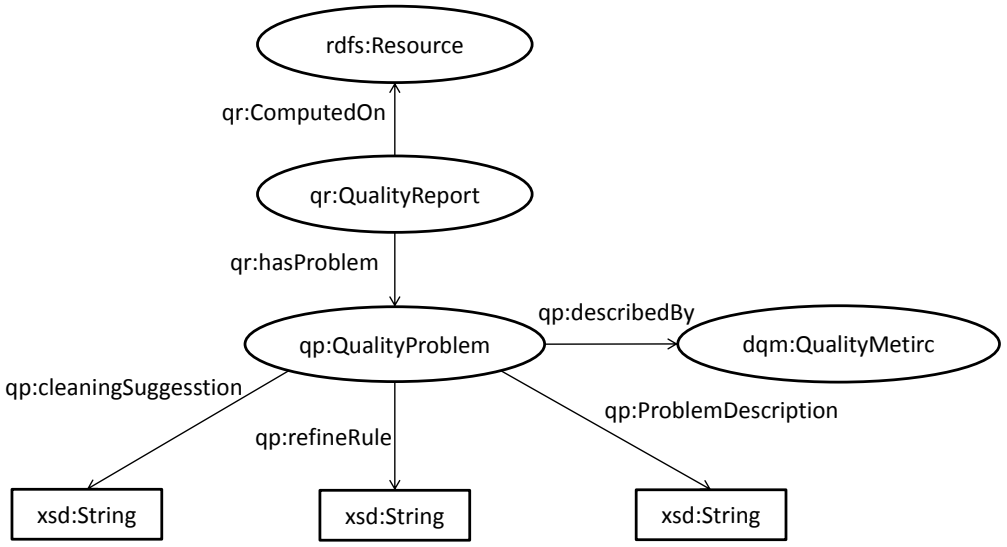


FIGURE 6.10: Cleaning Ontology General Structure

In this cleaning ontology we have defined two ontologies Quality Report (QR) and Quality Problem (QP). QR is for general terms and concepts and QP for detailed description of quality problems. QR has class qr:QualityReport and QP has a class qp:QualityProblem. A quality report consists of multiple quality problems. Each quality problem has a three basic set of properties attached to it. They are :

- **qp:describedBy**: This property is used to link the quality problem with its associated quality metric define in dqm[7] (data quality metric) ontology.
- **qp:ProblemDescription** : This property is used to describe in detail the type and nature of quality problem.
- **qp:CleaningSuggestion** : This property is used to provide any cleaning suggestion related to the quality problem.
- **qp:RefineRule** : Refine rule property is used to give any cleaning rule in GREL (Google refine expression language) for that quality problem.

We now further describe this cleaning ontology with the help of an concrete example.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dqm: <http://www.diachron-fp7.eu/dqm#>.
@prefix qr: <http://purl.org/eis/vocab/qr#>.
@prefix qp: <http://purl.org/eis/vocab/qp#>.

qp:WhitespaceInAnnotationProblem a rdfs:Class ;
        rdfs:subClassOf qr:QualityProblem ;
        qp:problemDescription "Literal value contains white spaces";
        qp:cleaningSuggestion "Trim leading and trailing whitespace" ;
        qp:describedBy dqm:WhitespaceInAnnotationMetric;
        rdfs:comment "Represents a kind of quality problems in
        which object in annotation property contains white spaces." ;
        qp:qrefineRule "value.replace(" ","").trim();";
        rdfs:label "Whitespace in Annotation Problem".
```

These RDF triples has define white space annotation problem in our cleaning ontology. We can see that qp:WhitespaceInAnnotationProblem is subclass qp:QualityProblem class. Therefore it consists of all the four basic properties define by qp:QualityProblem class i.e. qp:describedBy, qp:ProblemDecription, qp:CleaningSuggestion and qp:RefineRule. We can see in the example that these properties provide information with respect its property and relevant to that quality problem. Additionally, we have used two common rdfs properties to add more details for this quality problem i.e. rdfs:comments, rdfs:label.

Defining these quality problems in an ontology is an established approach to extend our ontology for other quality problems. It also enables us to easily add more information to the existing quality problems.

[7] http://www.diachron-fp7.eu/dqm

6.6 Cleaning Report

Our cleaning report uses quality report (see section 6.5) to describe the identified quality problems and provide cleaning suggestion for it. As discussed earlier in section 6.3 once user click on the identify quality problem button the RDF quality extension starts assessing the quality of the dataset. During this quality assessment process RDF triples with quality problems are collected from the dataset. The cleaning report takes these problematic triples and add information from the cleaning ontology with respect to the problem assist user in further cleaning the RDF dataset (see figure 6.5).

6.7 Evaluation and Results

In order to test our implementation of RDF Quality Extension, we have developed some test cases for each quality metric. For each quality metric we have created a dataset of RDF triples that has the quality problems for the quality metric implementation under test. If the metric implementation is able to identify the triple with the quality problem then the test is passed. However, since the implementation is a prototype therefore currently our test cases do not consider the scalability of the solution. Furthermore, for clarity correct or tipples without any problem are removed from the results in the figures.

6.7.1 Empty Annotation Value

Empty annotation value checks the annotation properties for empty string. As in our test case the two properties rdfs:label and rdfs:comments are empty string. Therefore they are identified as Empty Annotation Value problem.

```
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rel: <http://www.perceive.net/schemas/relationship/> .

<#spiderman>
        rel:enemyOf <#green-goblin> ;
        a foaf:Person ;
# Test Case : empty literal
        rdfs:label "" ;
# Test Case: empty literal
        rdfs:comment "  " .
```

Result: We can see in the image below that the two empty annotation value problems are identified correctly.

Subject	Predicate	Object	Problem	D
http://example.org/#spiderman	http://www.w3.org/2000/01/rdf-schema#comment	" "	Empty Annotation Value Problem	Literal empty
http://example.org/#spiderman	http://www.w3.org/2000/01/rdf-schema#label	""	Empty Annotation Value Problem	Literal empty

6.7.2 Incompatible Data type Range

Incompatible data type range checks the compatibility for the data type of literal with predicate's range data type. In the following test case pav:createdOn, pav:lastUpdateOn are predicates and their data type is specified for their range value i.e. xsd:dateTime. However, we can see that in our test case we have specified incorrect data type for the literal. Therefore, they are identified as Incompatible Data type Range problem.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix void: <http://rdfs.org/ns/void#> .
@prefix pav: <http://purl.org/pav/> .
@prefix ex: <http://example.org/> .

ex:dataset a void:Dataset ;
# Test Case: required data type is DateTime not Integer
        pav:createdOn "2"^^xsd:integer ;
# Test Case : required data type is DateTime not String
        pav:lastUpdateOn "wrongValue"^^xsd:string .
```

Result: We can see in the image below that the two incompatible data type range problems are identified correctly.

Subject	Predicate	Object	Problem
http://example.org/dataset	http://purl.org/pav/lastUpdateOn	"wrongValue"^^http://www.w3.org/2001/XMLSchema#string	Incompatible Datatype Range Problem
http://example.org/dataset	http://purl.org/pav/createdOn	"2"^^http://www.w3.org/2001/XMLSchema#integer	Incompatible Datatype Range Problem

6.7.3 Labels Using Capitals

Labels Using Capitals checks labels literal value improper usage of capital letters. Since in our test case rdfs : label has a literal value in Camel Cascade format therefore it is identified as Label Using Capitals problem.

```
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rel: <http://www.perceive.net/schemas/relationship/> .

<#green-goblin>
        rel:enemyOf <#spiderman> ;
        a foaf:Person ;
        foaf:name "Green Goblin" ;
#  Test Case : Literal is in Camel Cascade format.
        rdfs:label "GreenGoblin" ;
        rdfs:comment "Name of Green Goblin" .
```

Result: We can see in the image below that the one labels using capitals problem is identified correctly.

Subject	Predicate	Object	Problem
http://example.org/#green-goblin	http://www.w3.org/2000/01/rdf-schema#label	"GreenGoblin"	Labels Using Capitals Problem

6.7.4 Malformed Data type Literals

Malformed Data type Literals checks whether the value of a literal is compatible to the data type specified for that literal. In our test case property pav:createdOn has literal value '2' but the specified data type is xsd:dateTime. Therefore it is identified as Malformed Data type Literals problem.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix void: <http://rdfs.org/ns/void#> .
@prefix pav: <http://purl.org/pav/> .
@prefix ex: <http://example.org/> .

ex:dataset a void:Dataset ;
# Test Case: Data type defined for literal is DateTime
        pav:createdOn "2"^^xsd:dateTime ;
        pav:lastUpdateOn "2013-05-07T00:00:00.000+01:00"^^xsd:dateTime .
```

Result: We can see in the image below that the one malformed data type literals problem is identified correctly.

Subject	Predicate	Object	Problem
http://example.org/dataset	http://purl.org/pav/createdOn	"2"^^http://www.w3.org/2001/XMLSchema#dateTime	Malformed Datatype literals Problem

6.7.5 Misplaced Class or Property

Misplaced Class or Property checks whether is class is used as a property or a property is used as a class. Since in our test case foaf:knows is a property but it is used as a class. Therefore, it is identified as Misplaced Class or Property problem.

```
@base <http://aksw.org/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix people: <http://aksw.org/> .

people:MichaelMartin a foaf:Person ;
        foaf:name "Michael Martin" .

people:NatanaelArndt a foaf:Person ;
        foaf:name "Natanael Arndt" .

# Test Case : foaf:knows is a property.
foaf:knows people:MichaelMartin people:NatanaelArndt .
```

Result: We can see in the image below that the one misplaced class or property problem is identified correctly.

Subject	Predicate	Object	Problem
http://xmlns.com/foaf/0.1/knows	http://aksw.org/MichaelMartin	http://aksw.org/NatanaelArndt	Misplaced classes or properties Problem

6.7.6 Ontology Hijacking

Ontology Hijacking checks whether a user defined class or property is already defined in previously used vocabularies in the dataset. In our test case rdf:type and rdfs:Class are already defined in RDF syntax and RDF schema. Therefore they are identified as Ontology Hijacking problem.

```
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<#Person>
        a rdfs:Class .

# Test Case : rdf:type is hijacked
rdf:type
        a rdfs:Class;
        rdfs:subClassOf <#Person> .

# Test Case : rdfs:Class is hijacked
rdfs:Class
        rdfs:domain <#sister>;
    rdfs:range <#Person> .
```

Result: We can see in the image below that the two ontology hijacking problems are identified correctly.

Subject	Predicate	Object	Problem
http://example.org/#sister	http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/2000/01/rdf-schema#Class	Ontology Hijacking Problem
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class	Ontology Hijacking Problem

6.7.7 Undefined Classes

Undefined Classes checks whether a class that is not defined is used in the data set. In our test case notatype:NotAType is not defined therefore it is identified as Undefined Class problem.

```
@base <http://aksw.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix people: <http://aksw.org/> .
@prefix notatype: <http://notatype.org/>.

people:MichaelMartin a foaf:Person ;
                     foaf:name "Michael Martin" .

people:NatanaelArndt a foaf:Person ;
                     foaf:name "Natanael Arndt" .

# Test Case : foaf:know is not defined (it's foaf:knows)
people:MichaelMartin foaf:know people:NatanaelArndt .
```

```
# Test Case : notatype:NotaType is not defined
people:InvalidCase a notatype:NotAType .
```

Result: We can see in the image below that the one undefined classes problem is identified correctly.

Subject	Predicate	Object	Problem
http://aksw.org/InvalidCase	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://notatype.org/NotAType	Undefined Classes Problem

6.7.8 Undefined Properties

Undefined Properties checks whether a class that is not defined is used in the data set. In our test case foaf:know is not defined therefore it is identified as Undefined Property problem.

```
@base <http://aksw.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix people: <http://aksw.org/> .
@prefix notatype: <http://notatype.org/>.

people:MichaelMartin a foaf:Person ;
                     foaf:name "Michael Martin" .

people:NatanaelArndt a foaf:Person ;
                     foaf:name "Natanael Arndt" .

# Test Case : foaf:know is not defined (it's foaf:knows)
people:MichaelMartin foaf:know people:NatanaelArndt .

# Test Case : notatype:NotaType is not defined
people:InvalidCase a notatype:NotAType .
```

Result: We can see in the image below that the one undefined properties problem is identified correctly.

Subject	Predicate	Object	Problem
http://aksw.org/MichaelMartin	http://xmlns.com/foaf/0.1/know	http://aksw.org/NatanaelArndt	Undefined Properties Problem

6.7.9 White space in Annotation

White space in annotation checks for leading and ending white space in annotation's literal value. In our test case properties rdfs:comment literal value has a leading white space, rdfs:label literal value has an ending white space and rdfs:comment has white space at both ends. Therefore, all of them are identified as White space in Annotation problem.

```
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rel: <http://www.perceive.net/schemas/relationship/> .

<#green-goblin>
    rel:enemyOf <#spiderman> ;
    a foaf:Person ;
    foaf:name "Green Goblin" ;
        rdfs:label " " ;
# Test Case : white space at the start of literal.
        rdfs:comment " Name of Green Goblin" .

<#spiderman>
    rel:enemyOf <#green-goblin> ;
    a foaf:Person ;
# Test Case : white space at the end of the literal.
    rdfs:label "Spiderman " ;
# Test Case : white space at both ends of the literal.
        rdfs:comment " Name of Spiderman " .
```

Result: We can see in the image below that the three white space in annotation problems are identified correctly.

Subject	Predicate	Object	Problem
http://example.org/#spiderman	http://www.w3.org/2000/01/rdf-schema#comment	" Name of Spiderman "	Whitespace in Annotation Problem
http://example.org/#spiderman	http://www.w3.org/2000/01/rdf-schema#label	"Spiderman "	Whitespace in Annotation Problem
http://example.org/#green-goblin	http://www.w3.org/2000/01/rdf-schema#comment	" Name of Green Goblin"	Whitespace in Annotation Problem

As mentioned earlier, the basic purpose of this prototype was to demonstrate a technical realization of our propose approached for analyzing and cleaning RDF dataset in a simple, efficient and interactive manner. By implementing this prototype as an extension for OpenRefine, we have also utilized the general capabilities or features of OpenRefine for

cleaning RDF dataset. The source code and other related matter for our implementation is available at https://github.com/diachron/quality-extension.

6.8 Limitation Of OpenRefine Extension

Despite of many features provide by OpenRefine for data cleaning there are some limitations; specially when it comes to extend OpenRefine by implementing an extension because OpenRefine has a set of well defined but few extension points to implement any extension for it.

Furthermore OpenRefine uses it own data model and data storage to store, retrieve and modify the dataset. This means that for any RDF data that we want to clean we have to first load it into the local repository of OpenRefine. Though this approach has some advantages i.e. user can access the data offline, the data security issues are minimized and OpenRefine can independently maintain multiple sub-versions of data so that user may later (if required) revert any changes. However, this kind of solution is not scalable for very large amount of data. Because every time new project is created (see 6.3) a copy of entire dataset is maintained. Furthermore, if data is placed on the remote server then it has to be first downloaded locally which might take a lot of network, time and storage resources. On the top of it converting huge amount data from it original format into OpenRefine data model takes considerable amount of time and memory. Not only this, once data is cleaned then it has to be converted back into it original or required format and exported to the remote server.

Chapter 7

Summary and Further work

In the previous chapter we have discussed about the implementation details of RDF quality extension for OpenRefine. We gave reasons to choose Open Refine for implementing it extension, described the architecture of extension and workflow. In this chapter we will summaries all our work for this thesis and finally conclude on with some suggestion for further work.

7.1 Summary

In this thesis we have described Resource description framework, its importance and role in Semantic Web technologies. We further discussed about RDF data quality, its assessment and the problems that effects the quality of RDF data set. Based on previous research we discussed about different categories of RDF data quality problems from the perspective of data quality dimensions and then looked into different quality metrics. We also examine symptoms for RDF data quality problems. Using this available knowledge we developed an approach to clean and improve the quality of RDF data by identifying and removing the RDF data quality problems. We further extended our approach by implementing a workable prototype as an extension of OpenRefine for cleaning and evaluating RDF dataset.

7.2 Further work

As mentioned earlier to the best of our knowledge we do not find any cleaning approach or framework for RDF dataset therefore in this thesis our primary focus was to propose

an approach for cleaning RDF dataset and also demonstrate it by implementing a workable prototype. However, this is only the first step towards cleaning any RDF dataset therefore we also gave some concrete suggestions for further work in the area.

- **Add more metrics**: Our current implementation uses RDF data quality metrics to identify quality problems for cleaning in RDF dataset. However, we have restricted our implementation to the metrics that are directly related to RDF or RDFS properties (see Chapter 3 section 3.3) only. However there are numerous vocabularies and ontologies available for RDF framework e.g. FOAF[1], OWL[2] and to improve the capability of RDF quality extension it is important to add quality metrics for them in the current implementation of RDF quality extension.

- **Export in required format**: OpenRefine has defined it own data model to store and retrieve any data in project (see Chapter 6 section 6.8). Therefore, an export feature has to be implemented that converted stored data in required RDF format e.g. N3, RDF/XML, etc. For this purpose there do exists an OpenRefine extension that converts some data format in RDF [25] and this can be used in parallel to our current implementation of RDF quality extension. But in-order to automate the entire process and improve the performance of extension some work is required in this region.

- **Scalability for huge amount of data**: OpenRefine has the limitation of being operated on a single local machine therefore considerable amount of work is required to make this solution scalable because usually datasets are huge and cannot be accommodated by a single machine. For this integrations of technologies related to distributed system is important that allows the cleaning framework to distribute and clean RDF dataset on multiple systems and store them in managed repository.

[1] http://xmlns.com/foaf/spec/
[2] http://www.w3.org/TR/owl-ref/

Bibliography

[1] List of rdf properties. http://www.w3.org/1999/02/22-rdf-syntax-ns, . Accessed: 2014-12-06.

[2] List of rdfs properties. http://www.w3.org/2000/01/rdf-schema, . Accessed: 2014-12-06.

[3] Dieter Fensel, Federico Michele Facca, Elena Simperl, and Ioan Toma. *Semantic web services*. Springer, 2011.

[4] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sóren Auer. Quality assessment for linked open data: A survey. *Semantic Web journal - IOS PRESS*, 2012.

[5] Joseph F Hair, Ronald L Tatham, Rolph E Anderson, and William Black. *Multivariate data analysis*, volume 6. Pearson Prentice Hall Upper Saddle River, NJ, 2006.

[6] Y Bither. Cleaning and matching of contact lists including nicknames and spouses, 2005.

[7] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, Cristian Saitee, et al. Declarative data cleaning: Language, model, and algorithms. *CCSD - HAL Inira*, 2001.

[8] Mauricio A Hernández and Salvatore J Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data mining and knowledge discovery*, 2 (1):9–37, 1998.

[9] Jim Hendler. Web 3.0 emerging. *Computer*, 42(1):111–113, 2009.

[10] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[11] Timothy E Ohanekwu and CI Ezeife. A token-based data cleaning technique for data warehouse systems. In *IEEE Workshop on Data Quality in Cooperative Information Systems, Siena, Italy*, 2003.

[12] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. Ajax: an extensible data cleaning tool. In *ACM Sigmod Record*, volume 29, page 590. ACM, 2000.

[13] Aidan Hogan, Andreas Harth, Alexandre Passant, Stefan Decker, and Axel Polleres. Weaving the pedantic web. In *3rd International Workshop on Linked Data on the Web (LDOW2010), in conjunction with 19th International World Wide Web Conference*. CEUR, 2010.

[14] Pablo N. Mendes, Hannes Múhleisen, and Christian Bizer. Sieve: Linked data quality assessment and fusion. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, EDBT-ICDT '12, pages 116–123, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1143-4. doi: 10.1145/2320765.2320803. URL http://doi.acm.org/10.1145/2320765.2320803.

[15] Donald P Ballou and Harold L Pazer. Modeling data and process quality in multi-input, multi-output information systems. *Management science*, 31(2):150–162, 1985.

[16] Donald Ballou, Richard Wang, Harold Pazer, and Giri Kumar Tayi. Modeling information manufacturing systems to determine information product quality. *Management Science*, 44(4):462–484, 1998.

[17] Kuan-Tse Huang, Yang W Lee, and Richard Y Wang. *Quality information and knowledge*. Prentice Hall PTR, 1998.

[18] Yair Wand and Richard Y Wang. Anchoring data quality dimensions in ontological foundations. *Communications of the ACM*, 39(11):86–95, 1996.

[19] Richard Y Wang and Diane M Strong. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, pages 5–33, 1996.

[20] Kenneth C Laudon. Data quality and due process in large interorganizational record systems. *Communications of the ACM*, 29(1):4–11, 1986.

[21] Wai Lup Low, Mong Li Lee, and Tok Wang Ling. A knowledge-based approach for duplicate elimination in data cleaning. *Information Systems*, 26(8):585–606, 2001.

[22] Vijayshankar Raman and Joseph M Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.

[23] Mong Li Lee, Tok Wang Ling, and Wai Lup Low. Intelliclean: a knowledge-based intelligent data cleaner. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 290–294. ACM, 2000.

[24] Fadi Maali, Richard Cyganiak, and Vassilios Peristeras. Re-using cool uris: Entity reconciliation against lod hubs. *LDOW*, 813, 2011.

[25] Mateja Verlic. Lodgrefine-lod-enabled google refine in action. In *I-SEMANTICS (Posters & Demos)*, pages 31–37. Citeseer, 2012.

[26] Brian McBride. Jena: Implementing the rdf model and syntax specification. In *SemWeb*, 2001.

[27] Stefan Decker, Prasenjit Mitra, and Sergey Melnik. Framework for the semantic web: an rdf tutorial. *Internet Computing, IEEE*, 4(6):68–73, 2000.

[28] Joseph M Juran. Quality control handbook. In *Quality control handbook*. McGraw-Hill, 1962.

[29] Shirlee-ann Knight and Janice M Burn. Developing a framework for assessing information quality on the world wide web. *Informing Science: International Journal of an Emerging Transdiscipline*, 8(5):159–172, 2005.

[30] Richard Y Wang. A product perspective on total data quality management. *Communications of the ACM*, 41(2):58–65, 1998.

[31] Leo L Pipino, Yang W Lee, and Richard Y Wang. Data quality assessment. *Communications of the ACM*, 45(4):211–218, 2002.

[32] Christophe Guéret, Paul Groth, Claus Stadler, and Jens Lehmann. Assessing linked data mappings using network measures. In *The Semantic Web: Research and Applications*, pages 87–102. Springer, 2012.

[33] Yang W Lee, Diane M Strong, Beverly K Kahn, and Richard Y Wang. Aimq: a methodology for information quality assessment. *Information & management*, 40 (2):133–146, 2002.

[34] Aidan Hogan, Júrgen Umbrich, Andreas Harth, Richard Cyganiak, Axel Polleres, and Stefan Decker. An empirical survey of linked data conformance. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:14–44, 2012.

[35] Christian Bizer and Richard Cyganiak. Quality-driven information filtering using the wiqa policy framework. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(1):1–10, 2009.

[36] Leo Pipino, Richard Wang, David Kopcso, and William Rybold. Developing measurement scales for data-quality dimensions. *ME Sharpe, New York*, 2005.

[37] Elke Rundensteiner. Special issue on data transformation. *IEEE Techn. Bull. Data Engineering*, 22(1), 1999.

[38] Xin Wang, Howard John Hamilton, and Yashu Bither. *An ontology-based approach to data cleaning.* Regina: Department of Computer Science, University of Regina, 2005.