Enabling suggestions in tabular data cleaning and RDF mapping validation

Xianglin Ye
Master’s Thesis Spring 2016
Abstract

Linked open data provides a powerful way to publish data on the Web. However, most publishers still choose to publish their data in tabular formats. Whereas tools for transforming tabular data to linked data are useful, they are still immature and do not provide enough support to address this issue. Firstly, the process of tabular data cleaning is complex and involves using a wide variety of operations on tuples. Nevertheless, there are a number of common operations that users tend to need in certain situations, but that information is not taken advantage of. Furthermore, the process of transforming data to RDF is not trivial in itself – it involves RDF mapping, which requires a deep knowledge and prior research of relevant ontologies. However, few tools currently provide functions to facilitate RDF mapping by providing relevant information for ontologies, making it an intricate process. Even when data are successfully mapped, due to the process being largely manual, and, thus, error-prone, there are very few facilities for validating the produced mappings.

This thesis aims to address the aforementioned issues by identifying relevant algorithms, tools and methodologies, and applying them in the context of linked data transformation. Firstly, we propose a methodology for providing suggestions for data cleaning operations based on measurements of their use in given contexts. Secondly, we apply an existing algorithm for analysing the content of a table for suggesting appropriate RDF annotations. Furthermore, we describe facilities to help effectively manage RDF ontologies and, based on the constraints described within the ontologies, also validate RDF mappings.

As a proof-of-concept, this thesis provides a working prototype of the aforementioned functionalities, organised as a web service. The prototype has been partially integrated in the live version of the DataGraft platform.
Acknowledgment

First of all, I would like to thank my supervisor Dumitru Roman, Arne Berre for the help he provided for the thesis. And I also want to thank Nikolay Nikolov and Antoine Pultier for the suggestions with this thesis.

Thanks to all team members in DataGraft, the thesis will not exist without the DataGraft project.

I also want to thank the support of my family and friend. They give me the motivation to finish the thesis.
## Contents

1 **Introduction** 1  
1.1 Motivation ........................................ 1  
1.2 Goal ........................................ 2  
1.3 Research Question ............................... 2  
1.4 Structure of Thesis ............................... 3  

2 **Background** 5  
2.1 Semantic Web ................................... 5  
2.1.1 Linked Open Data ............................ 5  
2.1.2 RDF Triple ................................ 6  
2.1.3 Ontology ................................... 7  
2.1.4 RDFS and OWL ............................... 8  
2.1.5 SPARQL ................................... 8  
2.2 Related Techniques .............................. 8  
2.2.1 Programming Languages ................. 9  
2.2.2 Apache Jena ............................... 9  
2.2.3 Rest API and Jersey ................... 9  

3 **DaPaaS** 11  
3.1 DataGraft ..................................... 11  
3.1.1 Grafterizer .................................. 12  

4 **Software Components Overview** 15  
4.1 Software Components Overview ............... 15  
4.1.1 Data Cleaning Suggestion ................. 16  
4.1.2 RDF Mapping Suggestion ................. 16  
4.1.3 Ontology Management ................... 16  
4.1.4 RDF Triple Validation ................... 17  
4.2 Software Techniques and Implementation .... 17  
4.3 Software API ................................. 18  
4.3.1 Rest API ................................ 18  
4.3.2 JSON ................................... 19  
4.3.3 API Template ............................. 20  
4.3.4 Unit Testing .............................. 20
5 Tabular Data Cleaning Suggestions ........................................... 23
  5.1 Problems Description .................................................. 23
  5.2 Related Work ............................................................ 24
  5.3 Design Pattern ............................................................ 25
      5.3.1 Singleton ............................................................ 25
      5.3.2 Strategy Pattern .................................................. 26
  5.4 Data Cleaning Suggestions Component Design .................. 28
      5.4.1 Back-end API ....................................................... 29
      5.4.2 Overview of Transformations ................................. 29
      5.4.3 Data Cleaning Suggestions ................................. 30
  5.5 Component Implementation ........................................... 31
      5.5.1 Code Sample ....................................................... 32

6 RDF Mapping Suggestions .................................................. 35
  6.1 Previous Work ............................................................ 35
  6.2 Assign RDF triple to column ...................................... 35
      6.2.1 Problems Description .......................................... 35
  6.3 Apache Lucene Introduction ...................................... 36
      6.3.1 KS Test Introduction ............................................. 38
  6.4 Integration and Implementation ..................................... 39

7 Ontology Management ....................................................... 43
  7.1 Software Component Requirement .................................. 43
  7.2 Jena TDB ................................................................. 44
  7.3 Software Component Design ........................................ 44
  7.4 Keyword Search ........................................................ 45
      7.4.1 Jena Text Search ................................................ 45
      7.4.2 Query using Jena ................................................ 46
  7.5 API Design ............................................................... 46
  7.6 Implementation .......................................................... 48
      7.6.1 Graphical User Interface ..................................... 48
      7.6.2 Sample implementation ....................................... 49

8 RDF Validation ............................................................... 51
  8.1 Related Logic Terms .................................................. 51
      8.1.1 Open World Assumption ...................................... 51
      8.1.2 Close World Assumption ...................................... 52
      8.1.3 Unique Name Assumption ..................................... 53
  8.2 Data Quality Problem in RDF ................................ ..... 54
  8.3 Reasoning for Constraints in RDFS and OWL .................... 55
      8.3.1 Reasoner ............................................................ 55
      8.3.2 Using Jena for reasoning .................................... 55
      8.3.3 Constraints in OWL and RDFS ............................... 56
  8.4 Problem Definition ..................................................... 57
  8.5 Related Work ............................................................ 57
  8.6 RDF Validation Process .............................................. 58
  8.7 Software Component Implementation ............................ 59
      8.7.1 Code sample ........................................................ 60
9 Evaluation

9.1 Test Data Description ........................................ 63
9.2 Evaluation of RDF Suggestion ............................. 64
  9.2.1 Test Case 1 .............................................. 64
  9.2.2 Test Case 2 .............................................. 65
  9.2.3 Test Case 3 .............................................. 66
  9.2.4 Sample Code ............................................ 68
9.3 Evaluation of Data cleaning Prediction .................... 69
  9.3.1 Test Data Source ........................................ 69
  9.3.2 Data Preparation ....................................... 69
  9.3.3 Training .................................................. 69
  9.3.4 Testing ................................................... 70
  9.3.5 Result .................................................... 71
  9.3.6 Discussion .............................................. 71

10 Conclusion ....................................................... 73
  10.1 Summary .................................................... 73
  10.2 Contribution ............................................... 74
  10.3 Improvement and Future work ............................ 74

A Code and Application ........................................... 77
## List of Figures

2.1 RDF Graph Example ........................................... 7
3.1 Data processing in DataGraft ................................. 12
4.1 General Design ............................................... 15
4.2 Implementation ................................................ 17
4.3 API template .................................................. 20
5.1 Strategy Pattern ............................................... 27
5.2 Transformation Suggestion API ............................... 29
5.3 Transformation Suggestion ................................. 32
6.1 RDF Suggestion ............................................... 36
6.2 Lucene process ............................................... 37
6.3 KS Test ....................................................... 38
6.4 RDF Predicting ............................................... 40
7.1 Ontology Management Design ............................... 44
7.2 Ontology Management API ................................ 47
7.3 RDF Mapping ................................................ 49
7.4 Add Ontology ............................................... 49
8.1 RDF graph example .......................................... 56
8.2 Validation Process ........................................... 58
8.3 RDF validation class diagram ............................... 59
9.1 Pipeline ....................................................... 70
Listings

4.1 JSON example ........................................ 19
4.2 Unit test example ........................................ 20
5.1 Singleton .................................................. 26
5.2 Strategy Pattern ........................................ 27
5.3 Split Suggestion ........................................ 32
5.4 Find common words ...................................... 33
5.5 Unit testing ................................................ 33
6.1 kolmogorovSmirnovTest .............................. 39
7.1 Jena text search example .............................. 46
7.2 Keyword search .......................................... 46
7.3 Rest API example ........................................ 49
7.4 Read Ontology ............................................ 50
8.1 CWA example ............................................. 53
9.1 Evaluation of RDF suggestion ......................... 68
List of Tables

5.1 suggested Transformations .......................... 30
9.1 RDF suggestion testing score ........................ 67
9.2 RDF suggestion testing .............................. 68
Chapter 1

Introduction

1.1 Motivation

With the increasing availability of data, extracting useful knowledge is becoming more important for solving societal challenges. However, the complexity and the heterogeneity of data sources make it more difficult to utilize it. Therefore, it is important to analyze and transform data, thus making it more useful for people.

One important technology for publishing data and enabling easy knowledge extraction is the Semantic Web. The Semantic Web uses ontologies to organize data in a graph structure. This way of representing data preserves the relations of data elements. On the other hand, the most commonly used formats for data publication are based on tables. Grafterizer, as a part of DataGraft, is a framework that can be used to transform tabular data to RDF [35] data. This thesis is applied in the context of Grafterizer’s tabular data transformation capabilities.

An essential part of data transformation is data cleaning. In order to clean tabular data, different kinds of operations are applied. Currently, there is a wide variety of tools specialized in tabular data cleaning, such as Open Refine [15], Wrangler[20], Potter’s Wheel [26]. One important feature for such tools is that they can help users clean data by analysing the content of the dataset. Based on that they can provide useful suggestions for use during the process of data cleaning to guide users to clean data efficiently. However, this functionality is not applicable directly in the context of RDF data transformations.

During the process of transforming tabular data to linked data, it is a non-trivial task to find the appropriate resources from an ontology to map to. Firstly, the process requires users to be familiar with ontologies and linked data technology, which are currently not mainstream. Furthermore, it is common for an ontology to have a large number of resources, which make it even more complex to navigate.

Due to the manual nature of the RDF mapping process and inherent complexity RDF ontologies, the outcome of the semantic annotation is highly error-prone. Based on ontology structure and descriptions of constraints and relationships between concepts, it is possible to perform
checks on RDF mappings for errors, thus helping to eliminate them, but this is not typically implemented in RDF transformation tools.

1.2 Goal

The goal of this thesis is to present the implementation of the relevant functions, including suggestions during data cleaning, predicting and the validation of RDF triple.

The implementation of these functions reuses some ideas and code from previous work and combines them for the Grafterizer. Previous work focuses on different problem areas. Among them, there are some that focus on data cleaning, and others that focus on RDF triple validation, etc. In this thesis we integrate their ideas together and produce a solution that achieves the set goals. The way of integrating the previous work with the current project will also be discussed.

This work will also provide a series of APIs to encapsulate the functions and provide them as a web service.

This thesis will examine the state of the art of providing suggestions for operations for data cleaning and transformation and will provide a tool-independent practical solution to these issues. We also describe functionality to suggest relevant resources to be used given the context of the RDF annotation. Moreover This thesis analyses existing tools and algorithms and implements a practical solution for RDF mapping validation in the context of the Grafterizer tool. To enable the aforementioned set of functionalities, a system for hosting and managing ontologies is implemented. To assure the easy integration of the ontology management system, as part of the practical work in the thesis, we provide the necessary Create, Read, Update, Delete (CRUD) functions in the form of a web service.

1.3 Research Question

This thesis gives an answer to the following research questions.

1. How can we provide relevant data cleaning operation suggestions to the users based on given input data?

2. How to find a proper RDF triple to describe each column in tabular data?

3. How to manage the uploaded ontology on the server.

4. How to verify the correctness of RDF mapping.

5. How to integrate the ideas above in a linked data transformation process.
1.4 Structure of Thesis

The thesis will briefly discuss about the background and relevant knowledge, and introduce each component of the solution in separate chapters.

The thesis will introduce the background knowledge in the second chapter, including explanations of commonly used terms and technologies, and in chapter 3, DaPaaS project will be briefly introduced, all the work in this thesis is implemented in DaPaaS project. Chapter 4 will provide an overview of the software system. Chapter 5 gives an introduction to the suggestion functions during data cleaning. Chapter 6 introduces another suggestion function for RDF mapping. Chapter 7 introduce the ontology management system for the system, Chapter 8 provide a validation function for RDF triples, Chapter 9 is an evaluation of the suggestion functions. And the conclusion and future work are presented in Chapter 10.
Chapter 2

Background

This chapter briefly introduces the relevant background technologies. The first section gives a brief introduction of the semantic web, linked open data and Ontology. The rest of this chapter discusses the related programming techniques and libraries.

2.1 Semantic Web

Semantic Web was first introduced in a paper published by Tim Berners-Lee at Scientific American 2001, "The Semantic Web" [7], semantic Web use ontology for organizing and hosting data, semantic web also identify the connections between data and save the connections, semantic web will organize data in a data structure similar to graph, many other data formats such as tabular data, data in relational database is easy for human to understand, but difficult for machines to understand and process, semantic web makes data easier for machines to process.

The term Semantic Web refers to W3C’s vision of the Web of linked data. Semantic Web technologies enable people to create data stores on the Web, build vocabularies and write rules for handling data. [36]

There are many practical applications using semantic web technology now, British Broadcasting Corporation (BBC) is using the semantic web on their web service and providing ontology for sport, education, music etc. Some big search engines such as google and yahool also use semantic web for improving the search result.

2.1.1 Linked Open Data

Linked Data is a core concept of the semantic web, It is a method of publishing structured data so that it can be interlinked and become more useful through semantic queries. [41].

Open data is another concept to describe data which is freely available for public, the major source of open data can be data from scientific research, government etc. Many countries have already created data portals for open data.
The amount of open data is so big that makes it difficult to query and organize, linked open data adds interlinks in open data and also link different data source together. If we apply semantic web technology to these data, the data can be easier to query and better for machines to understand.

2.1.2 RDF Triple

Now we have a basic understanding of what is the semantic web and linked open data, the next step is to have a way to represent linked data, we will talk about the a basic component of linked open data, that is RDF triples. The Resource Description Framework (RDF) is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources [42]. Here is an example of RDF triple.

$$Martin \quad live \quad in \quad Oslo$$

An RDF triple consists of three components (we can also use the term resources to describe it): the subject, predicate, and object. In this example "Martin" is the subject, "lives in" is the predicate, and "Oslo" is the object. The meaning of this triple is fairly straight forward, the information described is, martin lives in Oslo. The resource of an RDF triple can be URI reference, literal or blank node, but a literal and blank node can only be an object of RDF triple.

A RDF graph is a collection of RDF triples, RDF graph uses graph representation to show the relation of resources, in addition to the RDF triple above, we can add more RDF triples to an RDF graph.

$$Martin \quad study \quad in \quad University \quad of \quad Oslo$$

$$University \quad of \quad Oslo \quad locate \quad at \quad Oslo$$

After we add the two RDF triples above to RDF graph, the graph contains three RDF triples now, we can use a graph figure to visualize RDF graph. As shown in figure ??, the information described in figure ?? is Martin lives in Oslo, and study at the University of Oslo, university of Oslo located in Oslo.

One of the most commonly used resource used in RDF triples is a URI (Uniform Resource Identifier) reference, URI identify resources on the internet, we can represent resources in RDF triples using URIs, so that resources can be used on the internet and interact with resources from other data source, for example, http://dbpedia.org/resource/Oslo can be used to describe Oslo in figure ??.

Literal node is only used for representing literal data resources, one example of literal node is the column header, we will talk about the column header later in the thesis. A blank node is a kind of node without URI, it is used for connecting two RDF triples.

Another important concept is "prefix", since URI is often represented by a long string, URLs are difficult for reading and memorizing. So we can
use a prefix to represent a URI. The prefix is normally a short word, using prefix can make the RDF graph easier to read by the human. An example of the prefix is shown below.

@prefix dbp: <http://dbpedia.org/resource/>

RDF triples and graphs are important components of the Semantic Web, and will be used many times in the remaining chapters of the thesis.

2.1.3 Ontology

In Semantic Web, Ontology defines the concepts and relationships (also referred to as "terms") used to describe and represent an area of concern. Ontology is used to classify the terms that can be used in a particular application, characterize possible relationships, and define possible constraints on using those terms. [44]

One of the most common used functions of Ontology is that Ontology can be used to describe and organize knowledge, for instance, ontology can be used in domains like the social network, health care etc.

Some ontology are often used as a schema for other ontology, such as RDFS(Resource Description Framework Schema)[37], OWL(Web Ontology Language) [34], both of them will be introduced in the following section and used frequently in the thesis.

There are two kinds of special resource named class and property in the ontology. This is not a formal definition, we use this definition in RDFS ontology to simplify the problems we want to solve.

• Class, a class is a kind of resource which is a rdfs:class or owl:class in the thesis. Class is often represented by following RDF triple.

  :classexample1 rdf:type rdfs:class
  :classexample2 rdf:type owl:class

  Class is usually used as subject or predicate of an RDF triple.

• property, property is a kind of resource that is a rdfs:property, owl:ObjectProperty, owl:DatatypeProperty.
Property is often used as predicate of an RDF triple.

Ontology is normally a collection of RDF triples, many of these RDF triples will describe details about class and property, such as constraints, relations between them. And these classes and properties are ready to be used for describing data in various domains.

Ontology is commonly used in semantic web, there are many popular ontologies already created, for instance, One of the most popular Ontology is FOAF (Friend of a friend), FOAF is a project devoted to linking people and information using the Web.[12], we can use FOAF to represent personal information and also the social connection between people, the vocabulary can be used in social network domain to organize knowledge.

2.1.4 RDFS and OWL

RDFS (Resource Description Framework Schema) is an ontology provides resources to describe other ontology, The final version is published in February 2004. There are many resources such as rdfs:class, rdfs:property, rdfs:domain etc, they are widely used in other ontology.

Web Ontology Language (OWL) is a kind of knowledge representation language used for describing other ontology. OWL is published in 2004, the newest version of OWL is OWL2 which is published in 2009. Both of them are based on RDF and RDFS (Resource Description Framework Schema).

OWL is another commonly used schema for describing other ontology, the purpose of OWL and RDFS is similar.

2.1.5 SPARQL

SPARQL is an RDF query language, that is, a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework (RDF) format [45]. The latest version of SPARQL is SPARQL1.1 which is released on 26 March 2013.

SPARQL is similar to SQL, both of them are used for query data, unlike SQL, SPARQL is created for querying RDF data, and used in a way like pattern match to retrieve information.

2.2 Related Techniques

The implementation of this project uses some external libraries, using existing libraries can significantly reduce duplicated work. For example, in the implementations, we use a library named Jena to process RDF Triples, and interact with the database. And another Java library Jersey is also used for providing restful service.
2.2.1 Programming Languages

The programming language used in the thesis is Java and JavaScript. The majority of the programming is done in Java, the web service is provided by Java, and GUI (Graphical user interface) is implemented in JavaScript.

Java is one of the most popular programming languages in the world, it first appeared on May 23, 1995. Java is an objective oriented programming language. In the thesis, the server-side program is written in Java with various libraries and open source code.

JavaScript is commonly used to program in the web browser, it is appearing as early as Java, and becoming popular during the last few years. In the thesis, the front-end code is programmed using JavaScript and related frameworks.

2.2.2 Apache Jena

Apache Jena (or Jena in short) is a free and open source Java framework for building semantic web and Linked Data applications. The framework is composed of different APIs interacting together to process RDF data.[5]. Jena is originally developed by HP labs in 2000, and adopted by Apache in 2010.

Jena has many convenient features and APIs, here are some package used in the thesis.

- RDF API: A series of APIs used for processing RDF graphs and RDF triples, these APIs use concepts like "Model", "Graph", "Resource" to organize RDF triples. There are many Java classes in RDF API, one of the most important one is Model class, a model is created by Jena ModelFactory, we can add triples to model, query and retrieve triple from a model, we can also change the resources in a triple using model. RDF API also includes APIs to read RDF triples from files, and write RDF triples to file system.

- ARQ: ARQ provides a SPARQL query engine for the user, SPARQL is a query language for RDF triples, we can use SPARQL to get the triples from a data triple store by writing query statements.

- Inference API: inference or reasoning of triples is an important function in linked data, if we have some RDF triples, based on the content of these RDF triples, we can use inference API for reasoning, and infer new data from these RDF triples.

There are also many other APIs like Ontology API, which is used to process OWL ontology.

2.2.3 Rest API and Jersey

The term REST(representational state transfer) was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine. REST has
been applied to describe the desired web architecture, to identify existing problems.[43].

Rest API can separate the Front-end and Back-end of a website, and it is also a good way for providing web services, so Back-end APIs designed in a Rest style is becoming popular nowadays.

In this thesis, most of the work is implemented in back-end, and providing web service using REST API.

There are lots of frameworks and libraries provides REST API support, Jersey is relatively light weighted and easy to integrate, so the implementation of this thesis use Jersey to provide REST web service.
Chapter 3

DaPaaS

DaPaaS(Data and Platform as-a-Service) is a research project co-funded by the European Commission, and aims at providing a solution for managing and transforming linked open data. The research project developed a tool named DataGraft to accelerate and simplifies the linked open data publication, consumption, and reuse cycle [10].

3.1 DataGraft

One of the central projects of DaPaaS is DataGraft, DataGraft is an online tool for hosting, transforming and publishing data, DataGraft takes tabular data as input, and transform tabular data to linked data format, the tool also provides functions for hosting data on the cloud, which makes data easily shared between users.

DataGraft consists of four parts:

• Grafter [16], Grafter is an open source tool used for converting tabular data to linked data. Besides transformation, Grafter also provides APIs for processing tabular data.

• Grafterizer, it is the front-end framework of DataGraft, Grafterizer provides online service for a user to process data, and transforming tabular data to linked data. The framework is based on the APIs provided by Grafter.

• The Database-as-a-Service structure. The layer turns the RDF-ised (RDF is one of the key components of Linked Data, providing a graph-based data model for describing things) legacy data into live data services, easily accessible and queryable by developers and applications. The data layer is both scalable and reliable [10].

• Data Portal, This provides a catalogue of various datasets (data services) and reusable data transformation services [10].

The process of processing data and transforming data in DataGraft is shown in [10] figure ??.
The input data format is tabular data, the first step is to put tabular data into a data cleaning process, tabular data will be cleaned interactively in this process, eliminate the noise in tabular data, and get a cleaner tabular data.

In the next step, we can specify "RDF Mapping", that is a function for creating an RDF graph, create related RDF triples in the RDF graph, these triples needs to contain the column header name from tabular data and resources from imported vocabulary. "RDF Mapping" will link the columns in tabular data using resources from the vocabulary. After we linked data together, we transform tabular data to linked data format, and generate the RDF format data for tabular data.

There are also some other functions, including publishing, hosting, sharing, and downloading data. But the thesis will focus on adding functions to data cleaning and RDF mapping, DataGraft is available online, and have been used for some practical project.

One of them is Personalized and Localized Urban Quality Index (PLUQI) app which can be found on the DataGraft platform. The app uses DaPaaS technologies to integrate various open data sets like transportation and crime statistics with various indexes on well-being and sustainability of cities to create visualizations of the results.[11]

3.1.1 Grafterizer

Grafterizer [9] is the Graphical user interface of DataGraft, Grafterizer provides a web portal for importing tabular data, and cleaning the imported data. The user can also transform tabular data to RDF data format using Grafterizer. Grafterizer relies on Grafter, which is a library programmed in Closure and used for converting to RDF data.

Grafterizer is implemented using JavaScript; the work in this thesis is based on Grafterizer. The thesis adds prediction module in data cleaning process, and an RDF prediction module in RDF mapping process.
The work in this thesis is implemented in Grafterizer or providing web service support for Grafterizer. All the work in this thesis is adding functions to data cleaning and RDF mapping in Grafterizer.
4.1 Software Components Overview

In this section, an overview of software components in this project will be presented. A brief introduction of each software module will be given first, and the rest of this section will explain how and why we need to combine these software components. Figure 4.1 shows the process to clean data and convert tabular data to RDF data, the details of these software components will be demonstrated in the remaining chapters.

As figure 4.1 showed. The basic process of data transformation includes data cleaning, RDF mapping and RDF publishing etc. The focus will be put on data cleaning and RDF mapping in the thesis.

The software components implemented in this thesis are "Data cleaning suggestion", "RDF Mapping suggestion", "RDF validation" and "Ontology management", a brief introduction of these software components will be presented in the following sections.
4.1.1 Data Cleaning Suggestion

The purpose of data cleaning in Grafterizer is to make tabular data well organized and more tidy. In order to clean tabular data, various kinds of transformations can be applied to tabular data. For instance, we can change the text of the column header, remove rows or columns in a table, change the content of tabular data, shift columns and rows, merge different columns etc.

The purpose of this thesis is to give suggestion for users during the data cleaning phase. The function will give suggestions based on the selection of user. For instance, when the user select an empty row, the prediction algorithm will give a list of suggestions for the user, one of the suggested item in the list could be “delete empty row”, some other suggestions such as ”delete current row” will also appear in that list.

When the user chooses one suggested item from the list, the algorithm will keep a record of the user’s selection, when the user need suggestions again, the user can get better suggestions.

4.1.2 RDF Mapping Suggestion

Data cleaning is an important process, when the data cleaning process is finished. The next step is creating an RDF mapping for table. RDF mapping is a collection of RDF triples. These RDF triples describe the content of each column in tabular data, and give connections to these columns.

To create RDF mapping, The relevant ontology needs to be imported, each ontology contains lots of resources, RDF mapping use these resources to describe the tabular data.

Due to the large amounts of resources in ontology, it is complicated to create a RDF mapping, the thesis present a way to simplify this process by predicting which resource maybe needed in RDF mapping.

The function will give suggestion to the user during the process of creating RDF graph, if a user wants to assign a resource to a column in tabular data, the suggestion algorithm will give a list of alternative resources for the user. Compare with select resource directly from ontology, select resource from a few alternatives is more efficient. The algorithm can also learn from the history selection of user and improve accuracy of suggestion.

4.1.3 Ontology Management

Another function in this thesis is Ontology Management. Ontology management is also used during the RDF mapping process, the function provides APIs and GUI (Graphical user interface) for user to upload their own ontology, in addition, user can change and delete their uploaded ontology.

Ontology Management also provides functions for searching resource in uploaded ontology. The search function is based on the text search APIs in Jena. The search function will help user to find the resources they need
based on a key word search, with the help of the search function, and RDF mapping suggestion component, user can create their RDF triples and link the column in tabular data easier.

4.1.4 RDF Triple Validation

The last function in this thesis is RDF triple validation, which can help user to validate their RDF triples. During the process of creating RDF mapping, a lot of RDF triples needs to be created. The purpose of this function is to validate the correctness of RDF triples.

The function will extract constraints from these uploaded ontology and validate RDF triples based on the constraints. If the RDF triple does not obey the constraints in ontology, the system will give a warning message to the user. For example, one constraint is rdfs:domain in RDFS ontology, the constraints describe which subjects can be used for a given predicate in RDF triple.

4.2 Software Techniques and Implementation

This section will explain the technology and frameworks used during the implementation phase, and how to apply these technologies in the thesis, figure ?? will show the basic structure of related functions of the thesis.

![Diagram](Figure 4.2: general implementation)

Front-end is the graphical user interface (GUI) of the project. Front-end need to be made clean and efficient to use, and also need to be user friendly. The techniques used in the front-end include AngularJS [2], Hyper Text Markup Language (HTML), Cascading Style Sheets (CSS), and a few related configuration tools. As a modern JavaScript framework, AngularJS
is simple and have plenty of functions, there are also a few configuration
tools which can help create a project from scratch, they can also manage
the relevant JavaScript libraries. The combination of these tools can make
implementation much efficient.

The restful API component is directly visited by the front-end, the APIs
 hides the complex back-end code and decouple the code on the front-end
from back-end. All the programs and algorithms in the thesis need to
provide APIs in the restful API layer. The restful API layer is programmed
in Java, and using Jersey library to create restful APIs. There are plenty of
other techniques for restful API, Jersey is one of the commonly used and
can easily integrate with maven.

There are several libraries and algorithms used in this project, the
majority of application implementation is in the back-end. The techniques
in this part include maven which is a powerful tool for managing Java
libraries, and we use Jena here for RDF processing.

The Database component is using Jena TDB [6] for hosting the ontology
uploaded by users. We will discuss more about database chapter 5.

4.3 Software API

A set of well-designed APIs is important for a program. In the thesis, we
have created a series of Rest API to represent the function at the back-end.

There are several reasons that implement most functions at the back-
end is preferred instead of the front-end, generally, Java is an objective
oriented programming language, and the compiler of Java also supports
type checking, so Java is more suitable for complex tasks compare with
JavaScript, so we choose to use Java for implementing functions at the back-
end. And the performance of server is normally better than computers, so
complex task is usually implemented on the server side.

So the design of this solution is to encapsulate most functions at the
back-end, and use APIs to expose them to the front-end.

In this section, the principle of API design will be illustrated, and the
tools used for designing the APIs will also be demonstrated.

4.3.1 Rest API

Web services are purpose-built web servers that support the needs of a site
or any other application. Client programs use application programming
interfaces (APIs) to communicate with web services. Generally speaking,
an API exposes a set of data and functions to facilitate interactions between
computer programs and allow them to exchange information.[24]

REST APIs use URI (Uniform Resource Identifiers) to address resource,
clients can access the resource on the server by visiting these URIs. Below
is an example of Rest API.

http://api.com/resource/

There are several kinds of REST API.
• Document, A document resource is a singular concept that is akin to an object instance or database record. [24]

http://api.com/resource/students/{1}

Client can visit the resource and performing CRUD operations to that resource only.

• Collection, A collection resource indicates a directory of multiple resources on the server.

http://api.com/resource/students

The resource URI indicates a collection of students, clients can perform CRUD operations to that collection.

• Store, A store is a client-managed resource repository. A store resource lets an API client put resources in, get them back out, and decide when to delete them.

PUT /users/1234/favorites/dog

• Controller, controller resource indicates a resource which is a procedure instead of single resources.

4.3.2 JSON

JSON is an open-standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs. It is the most common data format used for asynchronous browser/server communication. [40]

Listing 4.1: JSON example

```json
{
    "name": "University of Oslo",
    "country": "Norway",
    "address": {
        "streetAddress": "Boks 1072 Blindern",
        "city": "Oslo",
        "postalCode": "0316"
    }
}
```

Here is an example of JSON which is describing the basic information about University of Oslo. The data in JSON file are organized as multiple key-value pairs. In this example, one of the keys is "name" and the value is "University of Oslo".

JSON format is commonly used for data communication, including client and server communication. When the client and server use REST API for communication, passing parameters in JSON format is one of the most commonly used methods. In the thesis, the APIs will pass parameters between client and server using JSON.
4.3.3 API Template

During the process of designing APIs, except for giving an easy understandable name to each APIs, another important thing is to create a document for presenting those APIs, so that other people can understand it easily. In the thesis, we use an online application named swagger [1] to document the APIs. We choose this tool, because it is easy to explain the APIs clearly in a template, and we can also perform a simple test on APIs if we deploy these APIs on the server.

![API template](image)

This is a template of the REST API used in our system. Including API resource URI, HTTP method, parameters and return values.

4.3.4 Unit Testing

Since most of the functions are encapsulated in APIs, before we create graphical user interface in front-end, some techniques will be taken to make sure the correctness of our APIs. In this situation, we need unit testing to make sure the APIs is working correctly.

The principle of unit testing is to create a series of test case for each function. In our situation, a few test cases should be create for testing REST APIs. The test case is usually created by writing code for testing.

In the thesis, JUnit [19] framework is used for testing. JUnit is a unit testing framework for Java, it is commonly used for testing APIs.

```java
public class ExampleUnitTest {
    @Before
    public void setUp() throws Exception {
```

Listing 4.2: Unit test example
New Java class need to be created for testing, there are also some functions annotated by @Before, and @After, and these functions are used for preparing resources before and after each test. More functions with annotation @Test, can also be added, this kind of functions can be applied for testing specific functions in Java code.
Chapter 5

Tabular Data Cleaning Suggestions

The amount of data is growing rapidly nowadays, the large amount of data is one challenge for linked open data, another challenge is the quality of data. In order to convert tabular data to linked data, tabular data with a good quality is needed. The input data of Datagraft may have a poor quality, so a data cleaning process should be conduct for input data first.

Data cleaning is normally a time-consuming work, and this process also needs someone with professional knowledge to do, so it is important to improve the efficiency of data cleaning. Imagine the situation that a data scientist trying to clean a tabular data, if the system can guess what the data scientist want to do during the data cleaning process, and give suggestions to the data scientist. The efficiency of the data cleaning process will improve.

In this section, we get the idea of predictive interaction from previous work, and implement it in our project, the previous work will be discussed in the following sections.

5.1 Problems Description

Data cleaning, also called data cleansing or scrubbing, deals with detecting and removing errors and inconsistencies from data in order to improve the quality of data. Data quality problems are present in single data collections, such as files and databases, e.g., due to misspellings during data entry, missing information or other invalid data.\cite{rahm2000data}

Specifically, in tabular data, some data quality problem can also happen, data cleaning is a necessary step here. The following list discusses about some of the most commonly appeared data quality issues.

- Missing values

People may forget to input some data when they are creating a spreadsheet. In a spreadsheet, there could be missing values, such as empty cells, empty rows or empty columns.
To fix this kind of data quality issue, we can fill the empty values with values from other cells, or fill them by hand, another option is to delete the empty cells, rows or columns.

- **Noisy data**
  Incorrect data can also appear in data cells, such as meaningless data or data which is not suitable for converting to RDF format.
  In order to clean these data, some related transformations, such as removing them from table, correct the data can be apply to these data.

- **Reformatting data**
  Another job need to be done is reformatting the spreadsheet, for example, if the data which should be in two columns are put in a single column.
  In this situation, a split transformation could be taken to split the column into two columns. Sometimes, merge two columns, extract data from one column and put them into another column can also be used.

### 5.2 Related Work

Research about data cleaning has begun for a long time, many other technologies also require data cleaning, one example is data mining, data cleaning is an important phase of data mining. There are plenty of tools for data cleaning, many tabular data cleaning tools has also been developed.

There are different kinds of data format which can be cleaned, in this thesis, the focus will be put on tabular data cleaning. The following list shows some good tools for tabular data cleaning.

- **OpenRefine** [15] used to be supported by Google until 2013, it has become a mature data cleaning, transformation tool.

- **Potter’s Wheel** [26] is an interactive data cleaning tool developed by the University of California.

- **Wangler** [20], is another interactive data cleaning and transformation tools

The thesis gets the interactive data cleaning idea from the Potter’s Wheel and Wrangler, the idea of these interactive data cleaning tool is giving suggestions to the user during the cleaning process, the user can perform one transformation each step, such as spit column or remove rows etc. When the user need to apply one transformation to the table, the system need to illustrate the content of the table and what the user has done before, the system will make a guess about what kind of transformation the user may take based on the result of illustration, and give the suggestion to the user.
It is difficult to predict how the user wants to clean data, so the idea is to give a list of ranked suggestions items to the user, when the user chooses one suggestion, the system learns from the user’s selection and improve the accuracy of suggestion.

The thesis gets the idea of interactive data cleaning from these tools and integrate the idea in the linked data transformation process. The following sections will introduce the implementation of the function.

5.3 Design Pattern

Software design patterns are an important notion in modern software engineering, the software system is generally growing bigger and bigger recently, with the amount of code growing in software systems, the effort needs to maintain and improve the software also grows. Software engineering has raised lots of solutions to make software development and maintain easier. One of the solutions is design patterns.

A software design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations. [38]

One advantage of software design patterns is, design pattern makes it easier to decouple different software modules, in this way, it will be easier to reuse the code of the completed software module. Reuse existing code can improve the efficiency of software development. Another advantage is reducing the effort for maintaining the software. Since different software modules are decoupled, if we change the code in one module, it will unlikely to have a bad effect on another module, this reduces the possibility of having new defects after fixing one.

Usually, applying a software design pattern may make a program relatively complicated, since the simplest way to program is to implement the function needed directly. Design pattern may increase or decrease the readability of a program. If a design pattern is used in a proper way, and other programmers know about the design pattern, in this situation, the readability will be increased. But if a design pattern is not used properly, or poorly documented, it will be more difficult for other programmers to understand the code. Design pattern usually needs more code than implementing a program directly, so it should not be abused.

5.3.1 Singleton

It’s important for some classes to have exactly one instance. Although there can be many printers in a system, there should be only one printer spooler. There should be only one file system and one window manager. A digital filter will have one A/D converter. An accounting system will be dedicated to serving one company. [14]
Singleton design pattern is created for the above situation. Singleton is one of the most commonly used design pattern, and probably the simplest design pattern. For some Java classes in a program, creating only one instance is enough, such as logging, window manager. A global singleton class can be created in this situation, the class will only create one instance in the memory. The class can be a Java class or class in any other object oriented programming language.

The advantage of using the singleton design pattern is to avoid creating multiple instances for one class, for example, if only one printer exist in the system, the printer should not be accessed by multiple users, printer is a shared resource in this situation, this kind of shared resource should only be accessed by one user each time.

Singleton is similar with global variables, but singleton can provide more functions. The singleton design pattern can be implemented in a lazy-initialized way, that will reduce the resource needed for the program. Synchronize functions can also be added to the code in the singleton class so that the program can also be used in a multi-thread environment.

Listing 5.1: Singleton

```java
public class Singleton {
    private static class SingletonInner {
        private static final Singleton INSTANCE = new Singleton();
    }

    private Singleton (){}

    public static final Singleton getInstance() {
        return SingletonInner.INSTANCE;
    }
}
```

The Java code above is an example of singleton implementation. The constructor is a private method, so constructor in the Singleton class can not be used for creating a new instance. The "getInstance()" method can be used for getting an instance. When the method "getInstance()" is using for the first time, The Java inner class will create a new Singleton instance, and if "getInstance()" method is invoked again. The Java inner class will not create a new instance, but simply return the existing one.

The Singleton class will only create a new instance when "getInstance()" is being used, instead of creating an instance using constructor, this is a lazy initialization implementation, which will make the program more efficient.

Since an inner class is used for creating a new instance, Java Virtual Machine (JVM) will ensure the inner class is synchronized, which means it is thread safe for using the singleton class in the multi-thread situation.

5.3.2 Strategy Pattern

Strategy pattern defines a family of algorithms, encapsulate each one, and make them Interchangeable. Strategy lets the algorithm vary independ-
ently from clients that use it.[14]. Strategy pattern can be used for encapsulanting algorithms or strategies into separate class so that we can use these strategies and interchange to another one easily.

A simple example is, there are many ways to write a master thesis, different tools such as Office Word, Latex, maybe even notepad can be used, these are different ways or strategies for writing a thesis, if a Java program is implemented in this case, each tool can be encapsulated in one class, Office word in a Java class for writing thesis in Office world, latex in another Java class and so on. Designing program in this way can make us change the strategy (tools) easily.

The strategy pattern is not as commonly used as Singleton, and it is also more complicated than Singleton. But this design pattern is a good option for some special situations. To illustrate the strategy pattern, a Java program is implemented to demonstrate the situation.

Figure 5.1: Strategy Pattern

Figure ?? is a class diagram describe the example above. The class Thesis Editor is the super-class, and it has an abstract method for writing a thesis. An abstract method means a method which is not implemented, but need to be implemented in subclasses. There are three sub-classes, "Office word", "Latex", "Notepad" are sub-classes of "Thesis Editor". All of the three sub-classes will override the writingThesis() method which is shown in the super-class, but they will override it in different ways, they will use different strategies to achieve their goal. In simple words, they will use different editors for writing thesis, Context class can invoke ThesisEditor class multiple times.

Listing 5.2: Strategy Pattern

```java
public interface ThesisEditor {
    public abstract void writingThesis();
}

public class OfficeWord extends ThesisEditor {
    public OfficeWord(){}
}
```

@Override
The code in the above list implements part of the class diagram in Figure ??, we can interchange the strategies for writing thesis between Latex and OfficeWord. By passing the desired strategy to the constructor of Context class, Strategies can be changed easily.

As mentioned above, the advantage of strategy pattern is switching strategies easily, and conditional statements can also be eliminated in this way. Another advantage is, new strategies can be added without much interference of the origin code. The strategy pattern can also be used for organizing a family of algorithms.

There are also some disadvantages of using the strategy pattern. To implement the strategy pattern, many additional classes need to be implemented. The code may be difficult to read if the reader does not familiar with the design pattern. And to use the strategies, the clients must be aware different strategies. This is potentially a problem for clients.

5.4 Data Cleaning Suggestions Component Design

This section present the design of data cleaning suggestions component, including the APIs, the supported transformations, and the algorithms used for suggestion.
5.4.1 Back-end API

This section will discuss the REST APIs provided by our web service. A list of these APIs will be given first, followed by a brief introduction of them.

As figure ??, the first four APIs are used for generating suggestion items. These APIs will consume the information about user selection, such as indexes of selected rows, content in rows or columns etc, and the function will generate a list of suggestions.

Among them, singleRowPrediction, multipleRowPrediction will generate suggest operations when the user selects a row or multiple rows, and singleColumnPrediction, multipleColumnPrediction are used for generate suggestions for column selection.

When the suggested items are presented to the user, and they choose an item, then we can use the API "chooseItem" to send the users choice back to the system, system will learn from the choice of users and make better suggestion next time. The API will help the system learning the behavior of the user, and help improve the accuracy of suggestion.

5.4.2 Overview of Transformations

Various kinds of transformations can be applied or suggested for tabular data, for instance, Remove columns or rows can be applied for tabular data, merge several columns into one column and some other kinds of transformations could also be used. In this section, a summary about some of the most common used transformations is presented, and the system will select the suggested transformations from those commonly used transformations.

The following table shows the transformations supported in the thesis. There are a few kinds of transformations listed, and the table also presented the supported user operations, for instance, we cannot select one column
Table 5.1: suggested Transformations

<table>
<thead>
<tr>
<th>Transformations</th>
<th>Single Row</th>
<th>Multiple Row</th>
<th>Single Column</th>
<th>Multiple Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Extract</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shift</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Cut</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fill</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Fold</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Merge</td>
<td></td>
<td>Y</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>Rename</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>Make data set</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

and try to use merge transformation, the merge should only be used for multiple rows or multiple columns, so single row transformation does not support merge transformation.

For some transformations, these transformations should only be used in special situations, for example, the "make data set" transformation, that is used for setting table header with the selected row, for now, this transformation is only used when the user chooses the first row.

5.4.3 Data Cleaning Suggestions

The process of giving suggestions for tabular data cleaning will be presented in this section first and a simple system which can learn from the transformations done by user, and improve the accuracy of suggestion will also be presented afterwards.

The basic procedure for making suggestions is simple, the process is listed below.

1. The first step is, user selects rows or columns of the input table.

2. If the selected rows or columns is special, such as empty, or most cells contain special characters, common words. Then the system will give suggestions for these kind of special cases.

3. If nothing special with the selected rows or columns, system give a list of suggestions based on the history transformations of the user.

4. The user chooses one suggestion from the list given by the system.

5. System learns from the choice of the user and improve the accuracy of suggestion next time.

The system will need to improve itself by learning from the transformations made by the user. The way used for learning is a frequency based learning method, if the user selects one transformations more frequently than others, then this transformation has a higher priority. For example,
for the row selection, if the user has used delete operation, and move operation more times than other transformations for row selection, then the system will give a higher priority for delete and move operation.

In order to improve the accuracy of the suggestions, There are some other factors considered by the system.

- Empty rows or columns, the system will give special suggestions, such as “delete all empty rows”.

- The system should not give more than 3 suggestions for one category of transformation. For example, there should be no more than 3 suggestions for deletion in the suggestion list, this rule also applies to other categories like split, rename etc.

- Columns contain special characters, for instance, if comma appears in more than 50% of cells in the selected column, then the system should give suggestions for split the selected column.

- If more than 50% of the selected column have common words, then the system should give a suggestion to extract or cut the word and create a new column for showing the common word.

### 5.5 Component Implementation

The implementation of the system will be illustrated in this section and a class diagram for presenting the solution will be demonstrated first, some code samples will also be presented in this section.

Figure ?? is the class diagram for implementation, the implementation is using the strategy design pattern, the advantage of using the strategy pattern here is this design pattern will encapsulate each transformation into different classes, A new transformations can be added without changing the existing code.

There is a series of transformations as the table showed in the last section, including transformations such as cut, split, remove etc. Each transformation will be encapsulated in a Java class, all of these classed will need to inherit functions from the super class, the super class is Suggestion.

The class SuggestionMgr is used for managing the suggestion classes, the prediction is the client or a context clues in strategy pattern, that is used for deciding which kind of suggestion will be used.

Besides the functions above, we also need functions for training the system in order to increase the accuracy of the suggestions. So we have a ProbabilityDAO class, that is used for saving the history user selection data in the database. A relational database is chosen in this case, the database is easy for saving data for distributed system, even though the system is not a distributed system now, however, there is a chance that the system will be used in a distributed environment in future.

At the end, we need to encapsulate all the functions in REST API, adding these APIs to existing APIs in other chapters, so that the client can invoke the service easily.
5.5.1 Code Sample

This section will give code samples for some important parts of the system, we will give an example to find frequently appeared words in a column. As illustrated in the last section, if most of the cells in a column contains common words, a suggestion to extract or cut the words should be given.

Listing 5.3: Split Suggestion

```java
List<List<String>> data = new ArrayList<List<String>>();

// put all word in data array
for(int i = 0; i < selectedColumnData.length; i++){
    String current = selectedColumnData[i];
    String[] ret = current.split(" |\.|@|,");
    List<String> innerList = new ArrayList<String>);
    for(int j = 0; j < ret.length; j++){
        innerList.add(ret[j].trim());
    }
    data.add(innerList);
}
List<String> commenWord = findCommenWords(data);
```

When user selects a column, the code above will extract the column
data of input tabular data, split all the text data in a column into words.
and invoke another function to find common words.

Listing 5.4: Find common words

```java
Set<String> wordsSet = new HashSet<String>;
Iterator<String> wordIt = wordsSet.iterator();

// check whether list contains a word
while(wordIt.hasNext()){
    String word = wordIt.next();
    int containsNumber = 0;
    Iterator<List<String>> it = data.iterator();

    while(it.hasNext()){
        if(it.next().contains(word)){
            containsNumber++;
        }
    }

    if(containsNumber > listLength/2){  // common words found
        //
    }
}
```

The code sample above will find the most common words in a column. If a word appears in half of the cells in the selected column, then this word is frequently appearing in the column.

Listing 5.5: Unit testing

```java
String[][] data = {
    {"2014", "34", "place is Oslo"},
    {"2015", "3534", "place is Bergen"},
    {"2016", "65", "place is Stavanger"},
    {"", "", ""},
    {"2017", "2356.2", "place is Sweden"},
    {"2018", "432.1", "place is Finland"}
};

String[] columnHead = {
    "year",
    "randomNumber",
    "place"
};

Selection s = new Selection();
s.setSelectedRow(1);
s.setType(EnumType.rowSingle);

Iterator<PredictionProbability> it = p.generateOperations(data[1],
columnData, s, columnHead).iterator();
while(it.hasNext()){
    PredictionProbability p = it.next();
    System.out.println(p.getStrOp());
}
```
The sample code above is a unit testing for the APIs, A sample table is created in the unit test, the data in the table is irrelevant, Some random data is used for testing the APIs.

The function we need to use is "generateOperation", that will invoke various of suggestion methods, and give a list of suggestion, the unit test program just prints the result in the console.
Chapter 6

RDF Mapping Suggestions

In this chapter, the software component design and implementation of RDF Mapping suggestion will be presented. This chapter will also discuss the related technology and previous work. The project reuse open source code from previous work, the way of integrating DataGraft with the existing open source code will be presented in the last section.

6.1 Previous Work

There are a few previous works about adding tags to the column of a table, Karma [21]- is a tool used for transform tabular data to RDF format, developed by the University of Southern California. This chapter will explain the algorithms in Karma and how to integrate the algorithm with DataGraft.

6.2 Assign RDF triple to column

6.2.1 Problems Description

In order to create RDF mapping for tabular data, it is necessary to assign RDF triples for each column in the table. Predict RDF triples for each columns will be a useful function.

Suppose there is a column in tabular data, we can get the header of this column and the content of the column.

One option to predict RDF triple is to make prediction based on the header of column, the header of column usually contains the description of the column. But this is not a reliable option, since header contains very little information, and the header of a column is usually input by people, sometimes, header may not describe the content of column accurately.

In the thesis, we predict the RDF mapping based on the content of the column. Normally, there is a large amount of rows in each column, suggestions made based on content of columns could be more accurate than column header. We can pick a few samples from the column and make prediction based on these data.
There are some other information which can be taken into consideration, one of them is the history selection of a user. If the user has assigned RDF triples to a few columns before, we can take the behavior of that user into consideration.

To make prediction in this way, we need to take a record of the previous selection of the user as shown in figure ??, we can use a data structure which can save data using key-value pairs, the key is the selected RDF triple and the value is the content of a column. With many key-value pairs saved as at the left side of figure ??, If we need to assign RDF triple to a new column, we can simply compare the content of the new column with the saved columns in history key-value pairs, and find the closest key-value pair.

Based on the content of each column, columns can be divided in two categories, numeric column or text column. To compare the similarity of two sample columns, if the data is text data, we can evaluate based on how many common words occur in both columns. An simple solution for comparing two columns is by using the search function in Apache Lucene, the details about this solution will be presented in the next section.

The content of a column can also be numerical data, we cannot compare two numerical columns using Apache Lucene, because we cannot expect the number in two columns is exactly the same. But we can use algorithm to evaluate whether two group of numbers are similar or not. In this thesis, a statistic based algorithm named KS test is used to solve the problem, the detail of KS test will be demonstrated later.

![RDF Suggestion](image)

Figure 6.1: RDF Suggestion

### 6.3 Apache Lucene Introduction

A place to save the history key-value pairs is needed, and the efficiency of program is also important. Since the history data can be a large dataset, if we want to get the result quickly, we need solutions to ensure the search speed. Apache Lucene [23] is used in this case due to the efficiency in
searching.

Apache Lucene is an open source library for retrieving data from text, and it is originally written by Doug Cutting in 1999, and Lucene becomes a part of Apache software foundation in 2001. Lucene is originally written in Java and it has been ported to many other programming languages.

While suitable for any application that requires full-text indexing and searching capability, Lucene has been widely recognized for its utility in the implementation of Internet search engines and local, single-site searching. [39] And Lucene also supported many text formats, such as PDS, HTML etc.

There are also other text retrieving tool such as Apache Solr [3], but that is more like a text retrieving framework, and the core of Apache Solr is also Lucene, Apache Solr also does not have the flexibility of Lucene, So Lucene is a better choice in the situation.

At the rest of this section, we will illustrate how to use Lucene, and how to integrate Lucene into the solution.

![Figure 6.2: Lucene process](image)

Figure 6.2: Lucene process

The basic process of indexing data and searching is demonstrated in Figure 6.2. The process of applying Lucene to our problems mainly contains two parts.

One of them is indexing data, that means processing input data, indexing them, and put them in documents. The first step of this process is using Lucene Analyzer to split text into words, like the following sentence, I study in University of Oslo

Analyzer will split it as,

```
["I", "study", "in", "university", "of", "Oslo"]
```

Some analyzer will also translate these words to other languages for search. Another step is eliminated useless words, and frequently used words, in this case, "I", "in", "of" should be removed, the result will be,
Afterward, a directory can be specified to save the data, the next step is creating a Lucene document object and put the data in the object, we can use another Java object named IndexWriter to write the document in an index file.

The second process we can do is searching the indexed data. We can use the Lucene QueryParser object to parse the words we want to search, parsing them to a Lucene Query, and use the Lucene search method for searching. The result should be a list of documents which match our search condition and a score for evaluating the degree of matching.

Lucene is good for text searching, if the format of sample column data is text, we can simply use the search function in Lucene. During the training phase, we can input one RDF triple and the related sample column data, and have a key-value pairs data structure.

RDF triple - Document(Column data)

Using the above structure and save a list of these pairs.

After training the system, when there is a new input column , we can use the Lucene search function to find similar "values" in the history key-value pairs, and the related "key" is the RDF triple we want.

### 6.3.1 KS Test Introduction

If the content of column are numerical data, and we also have history key-value pairs, the "value" in these pairs are also numerical data, the next thing is to find an algorithm to calculate the similarity of two numerical columns.

In statistics, the Kolmogorov-Smirnov test (KS test) is used for testing the equality of two sample probability distributions, KS test is usually used for testing one-dimensional data.

![KS Test](image)

Figure 6.3: KS Test

Figure ?? is an example of KS test, the red line and the blue line are two probability distributions, and the figure uses the black arrow to indicate
the KS-test. By using KS test, we can find out whether two distributions are similar.

The KS test requires lots of knowledge about statistics to use, that is not convenient for nonprofessionals, To use this algorithm, we can also get support from third party libraries.

The Apache Commons Mathematics Library(common math)[4] is a library of lightweight, self-contained mathematics and statistics components addressing the most common problems not available in the Java programming language or Commons Lang, it is widely used for easy implementation of some algorithms.

The list below is the commonly used functions in common math.

- linear algebra, provide different kinds of operation for the matrix.
- Mathematical analysis, including Polynomial, Interpolation and many kinds of mathematical functions.
- Probability and Statistics, such as distribution, cluster and regressions and other statistical methods.

In this case, we will use an API for KS-test, in this way, we do not need to be professional in statistics. In common math, KS test is implemented in the class below [22]

org.apache.commons.math3.stat.inference.KolmogorovSmirnovTest

```java
Listing 6.1: kolmogorovSmirnovTest
public double kolmogorovSmirnovTest(double[] x, double[] y)
Parameters:
x - first sample dataset
y - second sample dataset
Returns:
p-value associated with the null hypothesis that x and y represent samples from the same distribution
Throws:
InsufficientDataException - if either x or y does not have length at least 2
NullArgumentException - if either x or y is null
```

Since we have the history records of column data and the relative RDF triple, when we get a new column, we can select a few sample data from the new data, and use the API mentioned above to compare the history data with new samples. If the history samples are the data we want, we can get the relative RDF triple and recommend it for the user.

### 6.4 Intergration and Implementation

In this section, we will discuss how to integrate the open source solution with our project.

The requirement we have is listed.
• Training, that means adding RDF triple and the related sample column data to Lucene index file.

• Searching, this step is getting the input sample column data, and get the related RDF triple from Lucene index file.

As we have mentioned above, based on the data type in the column, column sample data can be divided into two categories, numerical column, and the text column. In the referred open source code, the process of training is similar to the numerical column and text column. Both of them is using Lucene to index data and save the index data in files for search.

The process of searching is different, for text data, the system simply takes the sample column data and use Lucene searching method to get a list of RDF triples and scores of matching. But for numerical data, the system is not using Lucene search method, the system get the RDF triple and Sample column data from Lucene index file, and using KS test algorithm to compare the input sample column data with the data in index file, if the probability distribution of some data in the index file is similar to input data, then the result will be the related RDF triple.

To utilize the functions in the open source code, we add new APIs to DataGraft.

The first one is used for the training of system, with the RDF triple and column data as input parameter. The second one is used for predict RDF triple based on input column data.

The implementation of this module is relatively simple, since we have the open source code for reference, but still, we need to understand how the code working in order to utilize it.
As in figure ?? the class KSTest is using KS test algorithm for evaluation columns which is composed of numbers. Searcher and IndexSearch class is used for processing text data column.

Since most of the code above is open source code in another project, after extracting these Java classed, what we need to do is adding a RestAPI class, that is the API layer for creating web service for the client.
Chapter 7

Ontology Management

Ontology is the basics for create RDF triples in RDF mapping process. Users need to upload ontology to server before creating RDF triples or searching resources in ontology. RDF mapping suggestion also needs to get resources from ontology. Another important function, RDF triple validation, needs to get constraints from ontology. For the above reasons, creating a module for managing ontology is essential for the project.

In this chapter, we will introduce the details about the ontology management, including the database for saving ontology, the CRUD operations for ontology, and a searching function to search resources in ontology.

7.1 Software Component Requirement

The purpose of implementing the ontology management is to support other components, since some other components, such as RDF mapping suggestion and RDF triple validation, needs to get resources from ontology. Another reason we need Ontology management is to provide a keyword search function to make users find the resource quickly, so we can list the requirement in the list below.

1. Ontology management functions need to be used by front-end, so we need a series of REST APIs to perform CRUD operations to ontology on the server.

2. The APIs need to support users adding their own ontology, updating ontology, deleting ontology. The server needs to persist the name and the content of the ontology uploaded by the user.

3. User can search the resource in ontology based on input keyword, and resource to be searched is either a class or property.

4. The efficiency of system should be ensured, Adding, updating, deleting and searching functions should not take too long time.
7.2 Jena TDB

TDB is a graph based database provided by Jena, it is a triple store used for hosting RDF triples, we can use Jena API to access and query TDB, the CRUD(create, read, update and delete) operation of TDB can also be easily achieved by using Jena. TDB not only provide APIs for CRUD of RDF triples, it also has a web portal for SPARQL statements. We can get the data using SPARQL query in the web portal, and also writing programs for using TDB.

If we want to host ontology in the server, the first step is to choose a database. We can choose relational database, like MySQL, PostgreSQL etc., other options include noSQL database, like MongoDB. But in this thesis we choose TDB, a triple store. One reason for choosing TDB is that TDB is designed for linked data, and it is a widely used triple store in the semantic web. Another reason is TDB closely integrated with Jena, and we can use Jena APIs to access TDB easily.

7.3 Software Component Design

This section describes the design of ontology management function. Besides the CRUD operation of TDB, another challenge we need to consider is the efficiency of keyword search function. Suppose user have uploaded large amount of ontology, with the search space expanding, we need to improve the efficiency of search function.

![Ontology Management Design](image)

Figure 7.1: Ontology Management Design

Figure ?? is the general design of the ontology management module, Ontology dataset and Index dataset are two datasets in TDB, Ontology dataset is used for saving all the uploaded ontology, in order to improve the efficiency of search function, another dataset named index dataset is created, when a user upload a new ontology, the program will extract
some resources from an ontology and put them in the index dataset. If the resource is identified by the following type, then it will be extracted.

- rdfs:Class
- owl:Class
- rdf:Property
- owl:ObjectProperty
- owl:DatatypeProperty

There are two kinds of resources we want to extract, one of them is classes that is identified by rdfs:Class and owl:Class. The other one is properties which is identified by rdf:Property, owl:ObjectProperty and owl:DatatypeProperty.

For the resource search function, we only need to search classes and properties, so it is not necessary to go through all the triples in ontology dataset, because an ontology may contains large amount of triples, and only a few of them is used for identifying class or property. If we only search the class and property of ontology instead of searching the whole ontology the performance will be better.

The add operation will create triples in index dataset, update and delete operation will update and delete the specified RDF triple in the ontology dataset.

### 7.4 Keyword Search

With more ontology uploaded, the size of index dataset will also grow, that will make the search operation takes more time. In order to improve the efficiency of search operation, we use the Jena text search function, it is a new function released in Jena 2.11.0, and the function is based on Apache Lucene.

#### 7.4.1 Jena Text Search

Jena Text Search gives applications the ability to perform free text searches within SPARQL queries. Text indexes are additional information for accessing the RDF graph.

The text search function is a component in Jena, so we can easily use it with Jena APIs, Jena text search function requires SPARQL, we can perform ambiguous text search by using the function.

An example of search by using SPARQL is the code below.
Listing 7.1: Jena text search example

SELECT ?s
{ ?s text:query (rdfs:label 'word' 10) ;
  rdfs:label ?label ;
  rdf:type ?type
}

The above query tries to find the subject with rdfs:label match the literal 'word'. We can use the text search function, and put use the searching keyword to compose a SPARQL query for searching.

7.4.2 Query using Jena

We have briefly introduced Jena text search in the previous section, in this section, we will try to use the text search function in our project.

The user will input the keyword from a client, the search function will get the keyword and construct a SPARQL query, using Jena text search to query TDB, and send the result back to the client.

The following code explained how to use Jena text search in the thesis.

Listing 7.2: Keyword search

```java
String pre = StrUtils.strjoinNL(
    "PREFIX text: <http://jena.apache.org/text#>",
    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>",
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>");
String qsclass = StrUtils.strjoinNL("SELECT ?label",
    "{{ ?s text:query (rdfs:label '*" + name + "*').",
    "?s rdfs:label ?label",
    "?s rdf:type 'class'.} UNION",
    "{ GRAPH ?g {
        "?s text:query (rdfs:label '*" + name + "*').",
        "?s rdfs:label ?label",
        "?s rdf:type 'class'.",
    }"})
    ");
```

In order to identify the resource name in index dataset, we add a new RDF triple for all classes and properties. In the new RDF triple, we use rdfs:label to identify the name of the resource. So in the SPARQL above, that query will try to match the key word with the object of rdfs:label, and get the label of class or property.

7.5 API Design

In this section, the APIs for add, update, delete, search ontology will be demonstrated and explained. As in chapter 4, by using tools for document API, we have a list of APIs listed in figure ??.

As the list is shown in Figure ??, we use POST, GET, DELETE, -ref- to identify the type of REST API. In order to communicate between client and server, we need network protocols, including RPC(Remote Procedure Call), HTTP(Hypertext Transfer Protocol) etc. HTTP method is used in the
thesis, there are some commonly used HTTP methods such as POST, GET, DELETE.

- **GET**: The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

- **POST**: The POST method is commonly used for update the information in the server. By posting the new information from the client, the server will update the old information with new one.

- **PUT**: The PUT method is similar to POST method, PUT method is also used when the client sent information to the server for creating or updating information. But the PUT method is idempotent, when the PUT method is used multiple times for adding one object, only one object will be added.

- **DELETE**: DELETE method should be used when the client wants to delete the resource specified by the following URL.

There are a few kinds of protocols already implemented by some frameworks. We use an external framework named Jersey. Jersey framework implements HTTP protocol and provide REST API support. So we do not need to care about the detail of HTTP protocol, this kind of modern frameworks simplify the web development.

All the APIs have a common path start with "vocabulary", this path identifies the category of API, meaning that these APIs are used for process ontology.

In Figure 7.2, we have the following APIs.

- **Add**: This API is used for adding ontology to the database, by sending the content or URL path of ontology, the server will save
the ontology in the triple store, the namespace and ontology name also need to be specified. There will be some default ontology in the server, all of them is the common used ontology, the users can add their ontology by using the API.

- GetClassAndPropertyFromOntology: The API is used for processing an ontology, and extracting the classes and properties from the ontology. The API will also update the index dataset for search and put the result got from the last step in the index dataset.

- GetAll: This API will get the name of all saved ontology in the triple store, The API is used when the client needs to show a user the name of all ontology saved in the triple store.

- Search: Search method will take a keyword as input, and search through index dataset, get a list of classes and properties. The client will receive the result and show them in GUI.

- Delete: delete method is used for delete one ontology from the server. The operation will also update index dataset

- Update: update method get the name, namespace and content of a new ontology from client and update the old ontology on the server, user can change the name and content of ontology

7.6 Implementation

This section will present the graphical user interface at first, to demonstrate the implemented function. The second section demonstrates some of the important code logic and give an explanation for them.

7.6.1 Graphical User Interface

To make the APIs accessible by the non-technical user, normally we need to create a GUI(Graphical User Interface) for a user, the functions in API will be easier accessible in this way. A few sample GUI is shown in Figure ?? and Figure ??.

The GUI is developed using JavaScript, the JavaScript code invokes the REST APIs and provide a GUI to show the result of APIs. This is a good way to test APIs.

The GUI is shown in Figure ?? is the main interface for ontology management, all saved ontology can be listed there, we can also access the function of show classes and properties name by choose ontology name.

Figure ?? is a sample interface for adding ontology, two different ways of adding ontology can be used, the user can either upload ontology by dragging the local file to the dialog or specify a remote URL for the ontology.
7.6.2 Sample implementation

This section will give some sample implementation of ontology management, the sample code in this section will only demonstrate the essential part of design discussed in previous sections.

We have created many REST APIs by now, the next step is to show how our client use these API, the following list is a code sample for using the search API. The code is implemented using JavaScript on the front-end, and try to use REST API provided by our web service. To make the code sample simple, the process of the process the response data and show them in GUI is not demonstrated here.

Listing 7.3: Rest API example

```javascript
//get the url address of server URI.
var connection = leObject.serveraddress;

$scope.search = function(Para) {
    //get search result from server
```
To create the index dataset for the following searching function, we need to extract the class and property from uploaded ontology, so we have a code sample to show how to extract class using rdfs:class.

The following code use Jena model to read the ontology uploaded by the user first, and using SPARQL query to find the resource identified by rdfs:class, and extract them. The way to extract class and properties identified by rdfs:property, and OWL ontology is similar.

In order to get the resources we want from the triple store, it is convenient to use SPARQL for query the triple store.

Listing 7.4: Read Ontology

```java
// read online ontology path
Model model = ModelFactory.createDefaultModel();
FileManager.get().readModel(model, location,
                          getFileExtension(location));

ResultSet result;
String pre = StrUtils.strjoinNL(
    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>",
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>",
    "PREFIX owl: <http://www.w3.org/2002/07/owl#>");
// get rdfs classes
String qs = StrUtils.strjoinNL("SELECT DISTINCT ?classname
HERE{ ",
    " ?classname a rdfs:Class. }");

Query query = QueryFactory.create(pre + "\n" + qs);
QueryExecution queryExecution =
    QueryExecutionFactory.create(query, model);

// the result of get rdfs class
result = queryExecution.execSelect();
QuerySolution solution = res.next();
RDFNode a = solution.get("?classname");
```
Chapter 8

RDF Validation

In this Chapter, the design and implementation of RDF validation function will be presented. At first, we will briefly explain Close World Assumption (CWA) and Open World Assumption (OWA), because in the RDF validation chapter, we need to take the Close World Assumption. The next section will discuss the constraints in RDF and OWL, The constraints defined by RDFS and OWL can be used for RDF validation. At last, We will discuss how to implement the functions.

During the process of converting data from other format to linked data format, RDF mapping need to be created. Usually, There are lots of RDF triples in a RDF mapping. With the total number of ontology increasing rapidly, it became easier for people to make mistakes during the process of creating RDF triples.

One way to find potential mistakes is validate RDF triples by using constraints in ontology. constraints is defined using some RDF schema such OWL or RDFS, There are some resources in OWL and RDFS, such as "rdfs:domain", "rdfs:range" or carnality properties. These resources give a restriction to other ontology, for example, in an ontology, the "rdfs:domain" of a class may be defined, when this class is used in RDF mapping, the restriction needs to be considered. These kinds of restrictions may easy to be neglected by users due to the large amounts of ontology. This will result in mistakes when converting data from other format to RDF format.

To improve the quality of output data, it is a good choice to add a checking mechanism to make sure users are using ontology in a correct way when adding new triples to RDF mapping. If a user violates constraints in ontology, Warning or error messages should be given to inform the user about the mistake. And we may also add a validation function for the output file to ensure the data quality of output RDF file.

8.1 Related Logic Terms

8.1.1 Open World Assumption

In a formal system of logic used for knowledge representation, the open-world assumption is the assumption that the truth value of a statement
may be true irrespective of whether or not it is known to be true.[25]

For Example, suppose we have the following statements and reasoning these statements.

"I live in Oslo"
"I can live only in one city"
"I live in Bergen"

If we take Open World Assumption, then new knowledge will be inferred, the new knowledge we got is

"Oslo and Bergen is the same city"

Semantic web takes Open World Assumption by default, one advantage of OWA is, suppose we have some knowledge, it is easier for a reasoner to get new knowledge by inferring from old knowledge. The semantic web is designed in a way that it can get more new knowledge.

8.1.2 Close World Assumption

The closed-world assumption (CWA), in a formal system of logic used for knowledge representation, is the presumption that a statement that is true is also known to be true. Therefore, conversely, what is not currently known to be true, is false.[8] This is the opposite of OWA.

CWA is widely used in relational database. The design of relational database decides that the user should get precise information from it. So OWA is not a good option in the relational database.

If we continue to take the example in the last section, OWA will assume that the example is invalid, because the statement in the last example did not say "Oslo and Bergen is the same city".

Even though semantic web takes OWA, but if we want to validate RDF triples, it is difficult to use OWA. For example, if we have defined

```
:person1 rdf:type foaf:Person
:driving rdf:type rdfs:property
:driving rdfs:domain foaf:Person
```

Then we create new RDF triple using ":driving", and we set the subject of the triple as ":fish".

```
:car rdf:type rdf:Class
:person1 :driving :car
```

The above RDF triple is correct, A person is driving car. But if we have the following RDF triple.

```
:fish :driving :cars
```

OWA will assume ":fish is a person". But this does not make sense. At least when a person creates the triple, this may be a mistake. So to make the validation working, we should take CWA in validation.
8.1.3 Unique Name Assumption

The unique name assumption is a simplifying assumption made in some ontology languages and description logics. In logics with the unique name assumption, different names always refer to different entities in the world.[33]

Semantic web does not take the assumption, instead, it takes a Non-unique Name Assumption. For example, if we define two cities in the semantic web, that is "Bergen" and "Oslo", and again we use the example in the OWA section. Then "Bergen" and "Oslo" will be inferred as the same city by the reasoner. To make the validation function find as many mistakes as possible, we should use Unique Name Assumption for validating RDF triples.

Here is an example in semantic web. Suppose we have the following RDF in one ontology.

```r
Listing 8.1: CWA example
:Worker rdf:type owl:Class
:Worker rdfs:subClassOf
    [ rdf:type owl:restriction;
      owl:onProperty :WorkFullTimeIn;
      owl:maxCardinality 1"^^xsd:integer
    ];
:WorkFullTimeIn rdf:type owl:Class
:WorkFullTimeIn rdfs:range :WorkPlace
```

Meaning a worker can work full time in no more than one place, and assume we want to use ":WorkFullTimeIn" in the ontology defined above. And create the following RDF triples.

```r
:Mike rdf:type :Worker;
:WorkFullTimeIn :Supermarket;
:WorkFullTimeIn :Factory;
:Supermarket rdfs:subClassOf :WorkPlace
:Factory rdfs:subClassOf :WorkPlace
```

If we take Non-unique name assumption, this mapping is still correct, because it can be inferred that ":Supermarket" and ":Factory" are different names of the same place unless we declare ":Supermarket" is not the same with ":Factory" explicitly using "owl:disJointWith". However, in this case, the user may forget ":Mike" is working in a supermarket, and try to make ":Mike" working in two places by mistake. If we take unique name assumption here, these triples above will be invalid, and will get a warning about the mistake.

For lots of applications in semantic web, the reasoner can use OWA and none unique name assumption and make the proper inference get new knowledge and enrich the dataset, but for an application used for
validating RDF graph, we can assume CWA and UNA to identify the possible mistakes. More potential mistakes in RDF triples could be detected.

8.2 Data Quality Problem in RDF

In this section, we are going to talk about some basic kinds of data quality problem in RDF triples, we will also raise some approaches to identify them. Most of the data quality problem are mistakes during people use literal values. We can validate some of these cases, such as out ranged value, outdated value, for example, We can validate out ranged value by specifying a range manually and use this range to validate the converted value.

The list below shows some example of data quality problems we may encounter, and this list is summarized from [13]

- Word transposition/Syntax violation
- Outdated values
- Misfiled values
- Out of range values
- Unique value violation
- False values
- Referential integrity violation

One approach to identify the error is using SPARQL to query the output RDF file, by specifying some attributes in SPARQL, we can identify the bad quality data, the following table [13] shows some examples.

<table>
<thead>
<tr>
<th>Data Quality Problem</th>
<th>Generalized SPARQL Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing values</td>
<td>ASK WHERE{ ?this ?arg1 &quot;&quot; }</td>
</tr>
<tr>
<td>Syntax violation(only letters allowed)</td>
<td>ASK WHERE {?this ?arg1 ?value. FILTER(!regex(str(?value), &quot;^[A-Za-z]*$&quot;) ) }</td>
</tr>
</tbody>
</table>

This is a kind of data quality problem that often happens, another is the violation of constraints in RDFS and OWL. We are going to implement the latter in the following sections.
8.3  Reasoning for Constraints in RDFS and OWL

Besides the data quality problem in the last section, another data quality problem is a violation of constraints in the ontology. As we have discussed in the previous chapter. This kind of validation requires taking Close World Assumption and Unique Name Assumption.

To use these constraints in validating RDF triples, we also need to use semantic web reasoning. If we have a large amount of RDF triples, it is difficult to analyze these triples, one easy way is to use the help of semantic web reasoner.

8.3.1  Reasoner

A semantic reasoner, reasoning engine, rules engine, or simply a reasoner, is a piece of software able to infer logical consequences from a set of asserted facts or axioms [27]. Reasoner can also be used for getting new knowledge.

It is difficult and error prone to consider all the classes and properties in owl, so we may use a reasoner here. There are lots of famous and useful reasoners now, including Pellet [29], Racer [17], Fact++ [32] etc. Most of them are used for reasoning in OWL. Apache Jena, which we have used for search text and manage ontology. also, provide reasoners for RDFS and OWL reasoning.

We choose the reasoner in Jena is mostly because we have used Jena for processing and CRUD of ontology, the reasoner is easy to integrate with other software modules. And Jena also has a framework to use external reasoner.

We can use the reasoning function in Apache Jena, The Jena inference subsystem is designed to allow a range of inference engines or reasoners to be plugged into Jena. Such engines are used to derive additional RDF assertions which are entailed from some base RDF together with any optional ontology information and the axioms and rules associated with the reasoner. The primary concern for of this mechanism is to support the use of languages such as RDFS and OWL which allow additional facts to be inferred from RDF triples.

8.3.2  Using Jena for reasoning

In this section, we will give a brief introduction of the reasoner in Jena, and explain how to integrate with Jena.

In figure ?? [18] Jena is using ModelFactory for associating dataset and creating a model, the model is a basic structure in Jena, a model can contain many RDF triples, we can also add RDF triples to model. Jena also provides reasoner which can infer from existing model and generate new RDF triples.

Jena OWL reasoner is a reasoner used for inferring ontology which contains constraints and resources from OWL and RDFS since most of
the constraints we need to consider is in OWL and RDFS, the Jena OWL reasoner is a good option for reasoning in our situation.

Jena OWL reasoner provide three implementations, a default "full" version, a cut down "mini" version, and the smallest and most efficient version "micro". OWL provide lots of resources, the table below describes the supported resources, so that we can decide which version we should choose. [18]

<table>
<thead>
<tr>
<th>Constructs</th>
<th>Supported by</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:subClassOf, rdfs:subPropertyOf, rdf:type</td>
<td>full, mini, micro</td>
</tr>
<tr>
<td>rdfs:domain, rdfs:range</td>
<td>full, mini, micro</td>
</tr>
<tr>
<td>owl:intersectionOf</td>
<td>full, mini, micro</td>
</tr>
<tr>
<td>owl:unionOf</td>
<td>full, mini, micro</td>
</tr>
<tr>
<td>owl:equivalentClass</td>
<td>full, mini, micro</td>
</tr>
<tr>
<td>owl:disjointWith, owl:sameAs,</td>
<td>full, mini</td>
</tr>
<tr>
<td>owl:differentFrom, owl:distinctMembers,</td>
<td>full, mini</td>
</tr>
<tr>
<td>Owl:Thing, owl:equivalentProperty, owl:inversOf</td>
<td>full, mini, micro</td>
</tr>
<tr>
<td>owl:FunctionalProperty</td>
<td>full, mini, micro</td>
</tr>
<tr>
<td>owl:SymmetricProperty,owl:TransitiveProperty</td>
<td>full, mini, micro</td>
</tr>
<tr>
<td>owl:someValuesFrom, owl:allValuesFrom,</td>
<td>full, mini</td>
</tr>
<tr>
<td>owl:minCardinality, owl:maxCardinality, owl:cardinality</td>
<td>full, mini</td>
</tr>
<tr>
<td>owl:hasValue</td>
<td>full, mini, micro</td>
</tr>
<tr>
<td>owl:InverseFunctionalProperty</td>
<td>full, mini, micro</td>
</tr>
</tbody>
</table>

8.3.3 Constraints in OWL and RDFS

As illustrated in the previous sections, there are lots of constraints in OWL and RDFS, we have given examples about rdfs:domain, rdfs:range, owl:cardinality. In this section, more constraints will be given, not all of them is going to be considered in our project, but if we want to extend this RDF
validation function, we need to take these constraints into consideration. Here is a list of the constraints need to be considered for the project. All of them normally play as the predicate in a RDF triple, and defined as properties in OWL and RDFS ontology.

- rdfs:domain, rdfs:domain used to describe the subject of the specified predicate in RDF triple.
- rdfs:range, rdfs:range used to describe the object of the specified predicate in RDF triple.
- owl:minQualifiedCardinality, if the minQualifiedCardinality is defined for a property, and we use the property in RDF triples as predicate, then the minimum number of object should satisfy owl:minQualifiedCardinality.
- owl:maxQualifiedCardinality, the opposite of owl:minQualifiedCardinality.
- owl:allValuesFrom. if owl:allValuesFrom is defined for a property, and we use that property as in RDF triples as predicate, then object can only select from the value defined by owl:allValuesFrom.
- owl:disjointWith. We can use owl:disjointWith to make two class disjoint with each other.
- owl:sameAs, We can use owl:sameAs to identify two classes are the same for the none unique name assumption case.
- owl:differentFrom, opposite of owl:sameAs, this property is used for identify two class as different class.

8.4 Problem Definition

We have discussed CWA, OWA, and UNA in the previous sections, to validate RDF triple using constraints from ontology, we need to use CWA and none unique name assumption in our situation. We can find potential mistakes in this way. As the examples in the sections 6.1, if we choose to use OWA, we can get the conclusion that Bergen and Oslo are the same cities, but based on the common sense of human, this is not a correct statement, people may add those RDF triples by mistake, by taking CWA, the system can find out that error. The purpose of the system is to find out this kind of potential errors.

8.5 Related Work

Research related to this problem has been conducted by many other researchers. Pellet ICV [31], that is a subsystem of Pellet reasoner, has done research about this area. And another commercial product named Stardog [30] also working on this topic.
These tools are powerful tools, but we cannot integrate them into our system, provide a RDF validation function for our system, we need to implement a similar function with them, but we have a more integrated solution.

8.6 RDF Validation Process

In this section, the implementation of the RDF triple validation will be presented. This function relies on lots of external libraries, we have used Jena owl reasoner for reasoning, and use Jersey framework for providing Rest API service.

Figure 8.2: Validation Process

Figure ?? describe the basic procedure of the RDF validation function.

The procedure start by uploading ontology to the server, we also have a few common used ontology such as FOAF [12], SIOC [28], already saved on the server. These ontology contains constraints we need to use in the following step.

The next step is adding RDF triples to RDF mapping, the server will take the triples added as input and search the uploaded ontology, finding related constraints. The next step is using Jena OWL reasoner to infer these constraints, for example, the predicate of newly added triple is "foaf:name", we can get constraints in FOAF ontology

\[
\text{foaf:name} \quad \text{rdfs:domain} \quad \text{foaf:Person}
\]

And check whether the subject of input triple is a(rdf:type) person or not. System will give a warning is the new added triple violate the constraints.

The system implement the following functions
• RDFS:domain
• RDFS:range
• misplaced property
• Not associated data column.

8.7 Software Component Implementation

Figure r?? briefly describes the implementation of the system. We have created related functions for validating the above ontology constraints violation.

Besides validating domain, range and misplace property, the detection of not associated column is specially for tabular data. Since our purpose is to use resources from ontology to associate columns in tabular data. If we do not associate column to a class, or using a RDF triple to describe it. And we try to use the column in another RDF triple, in this case, the system will be not able to detect the column name. So we can not simply pass the column name to Jena reasoner.

One example is the case below, suppose :person1, :person2, :person3 is three columns in tabular data. and foaf is the prefix for FOAF(friend of a friend ontology). We have RDF triple in FOAF that defines the domain and range of foaf:knows.

```
foaf:knows rdfs:domain foaf:person
foaf:knows rdfs:range foaf:person
```

So in the following test case, :person1 and :person2, this two column is has been assigned a type, which is foaf:Person, foaf:knows can be used for these two columns. However, :person3 has not been assigned a type. We do not know what is :person3, the system will give a warning for ":person1 foaf:knows :person3".
8.7.1 Code sample

There are lots of Jena APIs we can use to implement our functions. The most frequently used one in the thesis is the Jena reasoner, Jena will do the reasoning task by creating an OntModel, the OntModel will contain the result of reasoning task. Compare with Jena model, OntModel will have lots of extra RDF triples, these triples are generated by reasoning.

```java
Model model = ModelFactory.createDefaultModel();
// using Jena reasoner.
OntModelSpec ontSpec = new OntModelSpec(OntModelSpec.OWL_DL_MEM_RULE_INF);
OntModel ontModel = ModelFactory.createOntologyModel(ontSpec, model);
List<String> list = new ArrayList<String>();
OntProperty prop = ontModel.getOntProperty(p.toString());
// get domain of specific property.
ExtendedIterator<OntClass> opDomains = (ExtendedIterator<OntClass>) prop.listDomain();
while (opDomains.hasNext()) {
    OntClass domain = opDomains.next();
    list.add(domain.toString());
}
```

Besides the Jena reasoning function, we can also use some Jena APIs of OntModel, in the code sample below, we use the listDomain API for getting the rdfs:domain of a property.

One of the main validation function in the thesis is to validate rdfs:domain for a property. The sample code below gives a simplified implementation of validation rdfs:domain of a property.

```java
// get all the possible domains for a property.
Iterator<String> domainIter = getDomainList(ontModel, predicate);
if (domainIter == null) {
    return false;
}
```
while(domainIter.hasNext()){
    String str = domainIter.next();
    OntClass ontclass = ontModel.getOntClass(str);

    Property typeproperty =
        model.createProperty("http://www.w3.org/1999/02/22-rdf-syntax-ns#type");
    Statement statement = model.getProperty(subject, typeproperty);

    // consider the class hierarchy
    if( statement != null ){
        if(ontclass.hasSubClass(statement.getResource())){
            return true;
        }
    }
}
Chapter 9
Evaluation

This chapter will perform several test cases for evaluating the solution in the thesis, the evaluation process is divided into two sections, at first we need to evaluate the function of assigning RDF triples for table columns, and we also need to evaluate the performance of suggestions during tabular data transformation.

9.1 Test Data Description

In order to evaluate the solution, we have collected 11 CSV files as a testing dataset, these files contain data from different domains, including data about buildings, persons, risk etc. Besides these tabular data, we also have the history records of how people transform these data, and what kind RDF mappings have been used in the real world for transforming these files.

Grafterizer has a pipeline function for taking records of the transformations which have been applied for tabular data, users can put the transformations they made for a table in a pipeline, each pipeline can save the transformation history for one or more tables. In other words, a pipeline describes the entire procedure of transforming a table. With this function, it is easier to reuse the transformations.

In this case, we get the 10 pipeline for the 11 CSV files, we have created test code for simulating these transformations.

And the RDF Mapping function in Grafterizer will also save the history of RDF Mapping, we can get the mapping for transforming data, and also implement test code using these mappings to simulate the user operation in the real world.

Since all the CSV files, transformations, RDF mapping is coming from the real world, so we can use these data as test data to evaluate our solutions.

The 11 dataset contains the following information.

- test dataset 1 - Persons data
- test dataset 2 - Claylab employed persons
- test dataset 3 - Infrarisk floods
Among the test datasets, test dataset 5, 6, 7-1, 7-2, 8 is in the same domain, all of them are describing information about buildings, each test dataset has a corresponding pipeline, test dataset 7-1 and 7-2 have the same pipeline. We also have test dataset 1 and 2 describe person data. The other datasets describe a few other domains such as floods, sensors etc.

In this case, we create Java programs to simulate the testing process, the dataset, transformations, RDF mapping are from the real world. And we will also use programs to output test result. The advantage of testing the solutions in this way is to avoid the time-consuming manual work. And since we may perform the testing many times, to automate the testing process is more important.

### 9.2 Evaluation of RDF Suggestion

For testing the RDF suggestion case, we will also take the 11 test datasets as input tabular data, and write code to simulate the process of assigning RDF triples for each column. The input dataset is a bit different from the previous section, all the datasets in this section are cleaned using the functions in the previous section.

Since the dataset 5-8 is about building information, they belong to the same domain, this could be a good test case for evaluating the performance of RDF suggestion in a specific domain.

#### 9.2.1 Test Case 1

The first test case is a simple test case, after users have assigned RDF triples to one table, there is a chance that users will try to process the table again in future. So we will test the performance of RDF suggesting in this situation.

- **Training Phase**

  The input data is the 11 test datasets, and we will assign RDF triple for each column in these datasets. The system will save the RDF triple and columns in Lucene index file.
• Testing Phase

After the training of system, we will randomly select 20 columns from 11 test datasets, using the system to suggest RDF triples for the 20 columns, since we already know the corresponding RDF triples for each column, we can compare the suggestion result with the RDF triple we know.

We can configure how many suggestions we can get for each column, in this test case, we will generate 5 suggestions, and if the 5 suggestions contain the result we want, then the testing program will give a "match" result.

• Result

In the 20 test results, 19 results match the correct RDF triple.

• Discussion

Based on the result of testing, If we have assigned RDF triples for columns, and try to suggest RDF triples for the same column again, then there is a great chance, we can get a good result.

9.2.2 Test Case 2

The second test case is used to evaluate RDF suggestion for a specific domain, the domain in this test case is data about buildings. Among the 11 test datasets, dataset 5 - 8 is about building information, these data use the same ontology.

The intention of this test case is to evaluate RDF suggestion in one domain since one user will likely to familiar with knowledge in one specific domain, and create many RDF mappings just in one specific domain. So we should create a test case to evaluate RDF suggestion in this situation.

• Training Phase

The dataset 5 - 8 contains 5 test datasets and 4 RDF mappings since test dataset 7 is split into two parts. In this case, we use dataset 5 - 7 and the corresponding transformations for training and predict RDF triples for dataset 8. Since the data in the above datasets are all about buildings, so we can use them for testing the solution in one specific domain.

In this test case, we use a similar method with test case 1 for training. We collect the dataset 5, 6, 7-1, 7-2, and the corresponding RDF Mappings, feed them for the training function.

• Testing Phase

In the Testing phase, we use test dataset 8 for testing, since we already have the corresponding RDF mappings, we compare the suggestion result with the RDF mapping for validating the correctness of the result. We will suggest RDF triples for all the columns in dataset 8, generate 5 suggestions for each column, and evaluate whether the suggestions containing the correct result.
• Result
In the table above "_" means a blank node in RDF triples. The first column is the expected result, the suggestion column is the 5 suggestions generated by the system, and the system will also give a score for ranking the suggestions. We can evaluate whether the suggestion column contains the expected result.

• Discussion
Based on the result in the table above, among the all the cases, 5 out of 7 cases get a good result. This is because "prodm" vocabulary has been used many times in test dataset 5, 6, 7 - 1, 7 - 2. So the suggestions about "prodm" vocabulary go well, but "sbig-complexes" does not get good suggestions. The scores related with "sbig-complexes" is also very small.

So the suggestions in a specific domain are generally good for this solution. But to suggest RDF triples for the different domain will not get good suggestions.

9.2.3 Test Case 3
We have evaluated the performance for the suggestion in a specific domain. We can also have a test case which will train the system using several random datasets, and make suggestions for columns of other datasets. This is a general evaluation of the solution.

• Training Phase
We select test dataset 2, 4, 6, 8, 10 for training, and test dataset 1, 3, 5, 7 - 1, 7 - 2, 9 for testing. We get the columns from the training dataset, and train the system using columns and corresponding RDF triples.

• Testing Phase
The dataset used for testing is dataset 1, 3, 5, 7 - 1, 7 - 2, 9. We select two columns from each dataset randomly and suggest RDF triples for each column. Since we already have the correct RDF triples. We can use the correct RDF triples as expected result and compare it with the suggested RDF triples.

• Result
We create a table to show the result of testing, the second column is the expected RDF triples for a column. In this table, we did not show the suggestions and scores and make a summary of these two instead. The third column indicates whether the 5 suggested RDF triples contains the expected result. If the suggested RDF triples match the expected result, we mark the column with "Yes".

• Discussion
Based on the testing result, among the 11 test columns, 3 columns get the correct suggestion. This is not a good result for the suggestion.
Table 9.1: RDF suggestion testing score

<table>
<thead>
<tr>
<th>Correct Result</th>
<th>Suggestions</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ prodm:</td>
<td>matrikkel-uri prodm:hasBuildingType</td>
<td>2.3482955</td>
</tr>
<tr>
<td>hasPostLocation</td>
<td>_ prodm:hasPostLocation</td>
<td>2.107153</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasDistrict</td>
<td>0.99408865</td>
</tr>
<tr>
<td></td>
<td>matrikkel-uri prodm:hasNumber</td>
<td>0.13320494</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasAddress</td>
<td>0.05598068</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasArea</td>
<td>1.3756707</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasZipCode</td>
<td>1.0216188</td>
</tr>
<tr>
<td></td>
<td>matrikkel-uri prodm:hasName</td>
<td>0.22022334</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasPostLocation</td>
<td>0.14857337</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasCommercialAddress</td>
<td>0.1102684</td>
</tr>
<tr>
<td>sbig-complexes</td>
<td>_ prodm:hasAddress</td>
<td>0.016995274</td>
</tr>
<tr>
<td>prodm:hasNumber</td>
<td>_ prodm:hasPostLocation</td>
<td>0.011756759</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasZipCode</td>
<td>0.011576572</td>
</tr>
<tr>
<td></td>
<td>matrikkel-uri prodm:hasBuildingType</td>
<td>0.008568285</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasDistrict</td>
<td>0.008364697</td>
</tr>
<tr>
<td>sbig-complexes</td>
<td>_ prodm:hasCommercialAddress</td>
<td>0.22175165</td>
</tr>
<tr>
<td>prodm:hasBuildingComplexType</td>
<td>_ prodm:hasPostAddress</td>
<td>0.22175165</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasAddress</td>
<td>0.15687957</td>
</tr>
<tr>
<td></td>
<td>matrikkel-uri prodm:hasName</td>
<td>0.14286828</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasCommercialAddress</td>
<td>0.107926379</td>
</tr>
<tr>
<td>_ prodm:</td>
<td>matrikkel-uri prodm:hasDistrict</td>
<td>2.7735083</td>
</tr>
<tr>
<td>hasDistrict</td>
<td>_ prodm:hasPostLocation</td>
<td>2.2343202</td>
</tr>
<tr>
<td></td>
<td>matrikkel-uri prodm:hasArea</td>
<td>1.3795675</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasPostLocation</td>
<td>0.13596761</td>
</tr>
<tr>
<td></td>
<td>matrikkel-uri prodm:hasNumber</td>
<td>0.05003472</td>
</tr>
<tr>
<td>matrikkel-uri</td>
<td>_ prodm:hasCommercialAddress</td>
<td>1.6326188</td>
</tr>
<tr>
<td>prodm:hasName</td>
<td>_ prodm:hasAddress</td>
<td>1.6326188</td>
</tr>
<tr>
<td></td>
<td>matrikkel-uri prodm:hasName</td>
<td>1.314313</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasCommercialAddress</td>
<td>1.1269786</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasAddress</td>
<td>0.03014524</td>
</tr>
<tr>
<td>_ prodm:</td>
<td>_ prodm:hasPostLocation</td>
<td>0.018933255</td>
</tr>
<tr>
<td>hasZipCode</td>
<td>_ prodm:hasAddress</td>
<td>0.017306264</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasDistrict</td>
<td>0.012944511</td>
</tr>
<tr>
<td></td>
<td>_ prodm:hasZipCode</td>
<td>0.012751159</td>
</tr>
<tr>
<td></td>
<td>matrikkel-uri prodm:hasName</td>
<td>0.011582407</td>
</tr>
</tbody>
</table>
Table 9.2: RDF suggestion testing

<table>
<thead>
<tr>
<th>Case Nr</th>
<th>Expected Result</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>_ prodm:hasAddress</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>_ prodm:hasPostLocation</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>mkk:matrikkel-nr prodm:hasArea</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>foaf:Person foaf:name</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>obs-uri sdmx-dimens:refTime</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>foaf:Person foaf:gender</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>_ prodm:hasZipCode</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>obs-uri rdfs:label</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>mkk:matrikkel-nr bygg:hasDisabilityFriendlySideEntrance</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Event:Number infrarisk:hasDuration</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Point:Number geo:lat</td>
<td></td>
</tr>
</tbody>
</table>

The 3 columns which have got the correct suggestions are all using "prodm" vocabulary. If we check the data in columns, we can find out there are 4 datasets using "prodm" vocabulary for RDF mapping, and the other 6 datasets describe information about other domains. So the system can make correct suggestions for a specific domain if we have used enough datasets for training.

9.2.4 Sample Code

The sample code below demonstrate the process of comparing the suggestion result with the expected result. And also print the scores for suggestion result.

```java
List<List<String>> examples = importCsv("test 10 - Turin census sections and coordinates.csv");

sampleMap.put("sez:joinedld geosparql:asWKT", examples.get(3));
sampleMap.put("obs-uri ter:haSuperficie", examples.get(4));

for (Map.Entry<String, List<String>> entry : sampleMap.entrySet())
{
    int numPrediction = 5;
    ret = hybridPredict.predictType(entry.getValue(), numPrediction);
    System.out.println("---------------------------------------------------");
    System.out.println(entry.getKey());
    List<String> tempList = new ArrayList<String>();
    for (SemanticTypeLabel s : ret){
        tempList.add(s.getLabel());
        System.out.println("--------------------");
    }
    System.out.println("------------------");
    List<String> tempList = new ArrayList<String>();
    for (SemanticTypeLabel s : ret){
        tempList.add(s.getLabel());
        System.out.println("--------------------");
    }
    System.out.println("------------------");
}
```

Listing 9.1: Evaluation of RDF suggestion
9.3 Evaluation of Data cleaning Prediction

9.3.1 Test Data Source

For testing the suggestions during tabular data transformation, same as the previous section, we also use 11 test datasets as input tabular data, and use the code for implementing functions to simulate the transformations in a pipeline. This kind of testing automates the process of evaluation.

9.3.2 Data Preparation

In order to test the tabular transformation functions, we need the history record of transforming data. As explained at the beginning of this chapter, we will use pipelines in Grafterizer, similar with RDF mapping, the pipelines save the whole procedure of transforming tabular data.

The algorithms used in this transformation prediction is relatively simple, so we create one test case for evaluation. Similar to the testing in RDF suggestion, the intention of the test case is to find out whether the suggestion is useful for the user or not. So we split the 10 transformation pipelines into two parts, we select test dataset 1, 2, 3, 7, 8, 9, 10 for training, and test dataset 4, 5, 6 for testing.

9.3.3 Training

In order to train the system, we use the transformation pipeline of test dataset 1, 2, 3, 7, 8, 9, 10, the process of training is

- Users select the transformation they want.

- System invokes the relevant Java method, the Java method will take the input of users as parameters.

- System processes the input and save them in the database.

To automate the process, we implement code for the second and third step, invoke the relevant Java method directly. Using Java code to simulate the process of selecting transformation.

Since we already have the pipelines, we can simply implement code to simulate the transformations in each pipeline.
9.3.4 Testing

In the testing process, we take the test dataset 4, 5, 6 and the related pipelines. The transformations in the pipeline are the correct transformations to process test dataset 4, 5, 6. So we can take these transformations as the expected output.

The testing process is basically, we can implement one dataset each time.

- Send the data in test dataset to the system.
- Using Java code to simulate the selection of a user, such as user select one column in tabular data.
- Get the suggestions and compare the suggestions with the expected output. Evaluate whether the suggestions contain the expected output.

We can repeat the above process for test dataset 4, 5, 6. Since this is a simple suggestion function, and the testing is also relatively simple. We just implement code to show the suggestion result, and instead of using code to compare suggestion result with pipelines, we compare the result manually. There are not so many test cases here, so this process does not necessarily need too much automated test.

In this test case, we took the first 5 suggestion items and compare it with the expected output. Because in the real world application, there will not be too much space to show all the suggestions.
9.3.5 Result

We can classify the transformations in pipeline into two categories. Single row selection and single column selection. since there are no multiple row or column selections in these pipelines.

• Single row selection

In the pipelines for test dataset 4, 5, 6, there are 4 transformations related with single row selection. Two of them is "make dataset" transformation and the other two is "remove row" transformation.

Among the 4 transformations, 2 of them get a good result, both of them is "remove row" transformation, the suggestions related with "make dataset" transformations did not get correct result

• Single column selection

In these test datasets, there are totally 22 transformations related with single column selection. With 9 "extract" transformation, 8 "mapc" transformation, 4 "grep" transformation and 1 "split" transformation

In these transformations, 9 "extract" transformations get a good result, and we did not implement "mapc", "grep" transformations yet, so we will not get suggestions about that. the "split" transformation did not get correct transformations.

9.3.6 Discussion

Based on the result we got, we can find out the advantage and disadvantage of the system. The system can get a good result for "remove row" and "extract column". This is because "remove row" and "extract column" transformation is frequently used in processing the training datasets. So they have got a much higher priority than others.

On the contrary, some of the transformation which are seldom used in training dataset will not get a good result, one example is "split" transformation. Since "split" did not use much time in the training dataset, so "split" transformation has a lower priority.

As a conclusion, if a user tends to use only a few kinds of transformations many times, and seldom using other transformations, the system can give a relatively good result. But if the user tends to use various kinds of transformations and do not use one or two of them frequently, the system will not generate a good result.
Chapter 10

Conclusion

This chapter will briefly summarize the thesis, demonstrate the contribution of this thesis, and discuss the limitations and future work.

10.1 Summary

As a new emerging technology, linked data are not widely used. Even though there are a few governments and organizations using linked data for publishing or utilizing data, the majority of data is still stored in relational databases and tabular data files. To improve the acceptance of linked data, tools for transforming data to linked data are necessary.

Compared to many other solutions, one advantage of DataGraft is that it is provided as an online service. Thus there is no need to download software to do the transformation task, which improves the effectiveness of data transformation. DataGraft also provide a large number of functions such as data cleaning, RDF mapping, and hosting.

As a summary, the focus of the thesis is listed below.

1. Ontology management—which is a basic functionality provided in the thesis. It can be used the function to host ontologies, managing ontologies using CRUD operations. We have implemented the back-end web service and a front-end web application for using it. This is a basic function in DataGraft, but it can also improve the efficiency of transforming linked data. The key word searching function enable user to find resource faster.

2. RDF triple validation—the function provides a way to validate RDF triple using CWA and non-UNA, helping users find the mistakes in RDF triples. Ontology management is an essential component for the service of this function.

3. RDF suggestion—provide suggestions for helping users create RDF triples, this could be a complicated function, but we have used external open source software to simplify the process.

4. Data cleaning suggestion—provide suggestions during data cleaning, that help users clean the tabular data quicker.
10.2 Contribution

To implement the suggestion functions of data cleaning and RDF mapping, the thesis reused the solution of Wrangler[20] and Karma. However, these two applications only focus on one aspect of the problem. Wrangler focus on data cleansing while Karma focus on RDF suggestion. It is useful to integrate these two solutions for converting tabular data to linked data. Besides these two functions, this thesis also introduces a validation function and an ontology management function.

The idea of this project can be applied in future study as well. Some other contribution of the thesis listed below,

1. The thesis created many back-end API, encapsulate the functions these API, by using these APIs, we can improve the usability of DataGraft.

2. The thesis also create an ontology management system for DataGraft, and both of the front-end and back-end programs. The system host the ontology uploaded by user, and enable user to search them. None professional users can also use the online service for managing ontology.

3. The research of RDF triple suggestion, validation, and data cleaning suggestion has been conducted by some institutions. They focus on a single aspect of the question, such as Stardog focus on RDF triple validation, Wrangler focus on Data cleaning. We put together the ideas and create a solution for cleaning and transforming linked data.

10.3 Improvement and Future work

In case of future work and possible improvements, we have the following list.

1. In the RDF triple validation chapter, we only consider limited number of constraints, in OWL and RDFS, there are more constraints can be considered, one possible future work is implemented more constraints, so that the function can identify more mistakes. For example, we can implement the carnality constraints in OWL ontology, that is a commonly used constraint.

2. RDF mapping suggestion, for now, we only make suggestions for columns in tabular, an improvement could be, when user input parts of a triple, the system can suggest the other parts.

3. As for the suggestion function during data cleaning, a simple frequency based algorithm is used there, we should consider applying more effective machine learning algorithm to improve the accuracy of suggestion.
4. Consider the aspect of implementation, we should implement front-end application for using the APIs, for now, we only have APIs on the back-end, which is not user-friendly. And the APIs is used for DataGraft now, we did not consider to make it public, it will be better if the APIs can be open public and used by external applications.
Appendix A

Code and Application

The project code is available in GitHub, the front-end code of Grafterizer can be downloaded in https://github.com/dapaas/grafterizer.

The code of data cleaning suggestions are in https://github.com/dapaas/grafterizer-backend/tree/prediction/Vocabularys/VocabBackend/VocabBackend/src/main/java/main/java/suggestion

The code of RDF triple validation functions are in https://github.com/dapaas/grafterizer-backend/tree/prediction/Vocabularys/VocabBackend/VocabBackend/src/main/java/main/java/validation

The code of ontology management and RDF mapping suggestion are in https://github.com/dapaas/grafterizer-backend/tree/prediction/Vocabularys/VocabBackend/VocabBackend/src/main/java/main/java

The application is available in https://datagraft.net/pages(transformations/)

Some of the functions has been implemented in the application, for the other functions, the REST API code is available in Github.
Bibliography


[28] SIOC. URL: https://www.w3.org/Submission/sioc-spec/S (visited on 05/04/2016).


[34] W3C. owl. URL: https://www.w3.org/TR/owl-guide/ (visited on 10/04/2016).


