

UNIVERSITÉ DU QUÉBEC À MONTRÉAL

INFERRING MISSING SCHEMA FROM LINKED DATA  
USING FORMAL CONCEPT ANALYSIS (FCA)

MASTER THESIS  
PRESENTED  
AS A PARTIAL REQUIREMENT  
FOR THE MASTER IN COMPUTER SCIENCE

BY  
RAZIEH MEHRI DEHNAVI

FEBRUARY 2014

UNIVERSITÉ DU QUÉBEC À MONTRÉAL  
Service des bibliothèques

Avertissement

La diffusion de ce mémoire se fait dans le respect des droits de son auteur, qui a signé le formulaire *Autorisation de reproduire et de diffuser un travail de recherche de cycles supérieurs* (SDU-522 – Rév.01-2006). Cette autorisation stipule que «conformément à l'article 11 du Règlement no 8 des études de cycles supérieurs, [l'auteur] concède à l'Université du Québec à Montréal une licence non exclusive d'utilisation et de publication de la totalité ou d'une partie importante de [son] travail de recherche pour des fins pédagogiques et non commerciales. Plus précisément, [l'auteur] autorise l'Université du Québec à Montréal à reproduire, diffuser, prêter, distribuer ou vendre des copies de [son] travail de recherche à des fins non commerciales sur quelque support que ce soit, y compris l'Internet. Cette licence et cette autorisation n'entraînent pas une renonciation de [la] part [de l'auteur] à [ses] droits moraux ni à [ses] droits de propriété intellectuelle. Sauf entente contraire, [l'auteur] conserve la liberté de diffuser et de commercialiser ou non ce travail dont [il] possède un exemplaire.»

UNIVERSITÉ DU QUÉBEC À MONTRÉAL

INFÉRENCE DU SCHÉMA MANQUANT À PARTIR DE DONNÉES LIÉES  
À L'AIDE DE L'ANALYSE DE CONCEPTS FORMELS (ACF)

MÉMOIRE  
PRÉSENTÉ  
COMME EXIGENCE PARTIELLE  
DE LA MAÎTRISE EN INFORMATIQUE

PAR  
RAZIEH MEHRI DEHNAVI

FÉVRIER 2014





## ACKNOWLEDGMENTS

I am deeply thankful to my supervisor, Dr. Petko Valtchev, for his guidance and encouragement throughout my Master studies at UQAM. He has been an excellent advisor and a constant source of knowledge, motivation, and encouragement during this dissertation work.

I would like to extend my thanks to Dr. Fatiha Sadat, my co-supervisor, for her guidance throughout this research work.

I am continuously grateful to my family specially my parents for their support, love and encouragement.

Finally, I would like to thank all the staff members of the Computer Science department at UQAM for their direct and indirect helps during my studies at UQAM.



## TABLE OF CONTENTS

|   |      |
|---|------|
| LIST OF FIGURES   | xi   |
| LIST OF TABLES  | xiii |
| ABREVIATIONS  | xv   |
| RÉSUMÉ  | xvii |
| ABSTRACT  | xix  |
| INTRODUCTION  | 1    |
| CHAPTER I   |      |
| MAIN CONCEPTS   | 3    |
| 1.1 RDF . . . . .                                       | 3    |
| 1.2 RDF Schema (RDFS) . . . . .                         | 5    |
| 1.3 FCA . . . . .                                       | 7    |
| 1.3.1 Introduction to Formal Concept Analysis . . . . . | 7    |
| 1.3.2 Concept Lattice . . . . .                         | 8    |
| 1.4 DBpedia . . . . .                                   | 9    |
| 1.4.1 Data Source . . . . .                             | 10   |
| 1.4.2 Data Structure . . . . .                          | 11   |
| 1.4.3 Data Access . . . . .                             | 13   |
| 1.5 Summary . . . . .                                   | 14   |
| CHAPTER II  |      |
| REVIEW OF THE LITERATURE                                | 15   |
| 2.1 Basic Concepts Related to Similarity . . . . .      | 15   |

|                                |   |    |
|--------------------------------|---|----|
| 2.1.1                          | Name-based Techniques . . . . .   | 18 |
| 2.1.2                          | Structure-based Techniques . . . . .                                      | 27 |
| 2.1.3                          | Extensional Techniques . . . . .  | 30 |
| 2.1.4                          | Semantic-based Techniques . . . . .                                       | 32 |
| 2.2                            | RDF and Similarity . . . . .  | 34 |
| 2.2.1                          | Similarity of RDF Graphs on Linked Open Data (Interlinking Tools)         | 34 |
| 2.2.2                          | Finding Similarity between RDF Individuals Using FCA . . . . .            | 45 |
| 2.3                            | FCA and Semantic Web Applications . . . . .                               | 46 |
| 2.4                            | Summary . . . . .   | 47 |
| CHAPTER III                    |   |    |
| METHODOLOGY AND IMPLEMENTATION |   | 49 |
| 3.1                            | Approach . . . . .  | 49 |
| 3.1.1                          | Converting RDF to FCA Input . . . . .                                     | 50 |
| 3.1.2                          | Converting FCA Output to RDFS . . . . .                                   | 52 |
| 3.1.3                          | Choosing Plausible Names for RDFS Classes Using DBpedia . . . . .         | 54 |
| 3.2                            | Implementation . . . . .  | 58 |
| 3.2.1                          | Java Frameworks and APIs . . . . .  | 58 |
| 3.2.1.1                        | Jena . . . . .  | 58 |
| 3.2.1.1.1                      | RDF API . . . . .   | 59 |
| 3.2.1.1.2                      | SPARQL . . . . .  | 60 |
| 3.2.1.2                        | Galicia . . . . .   | 61 |
| 3.2.1.3                        | RDF Gravity . . . . .   | 61 |
| 3.2.2                          | Generating RDFS from RDF data . . . . .                                   | 64 |
| 3.2.2.1                        | Step One: Converting <code>.rdf</code> to <code>.rcf.xml</code> . . . . . | 64 |
| 3.2.2.2                        | Step Two: Converting <code>lat.xml</code> to RDFS . . . . .               | 65 |
| 3.2.2.3                        | Step Three: Naming Classes Using DBpedia . . . . .                        | 66 |
| 3.3                            | Summary . . . . .   | 67 |

|   |    |
|---|----|
| CHAPTER IV                                  |    |
| EXPERIMENTS AND RESULTS                     | 69 |
| 4.1 Dataset . . . . .                       | 69 |
| 4.2 Results . . . . .                       | 70 |
| 4.2.1 Binary Relation Table . . . . .       | 70 |
| 4.2.2 Concept Lattice . . . . .             | 70 |
| 4.2.3 RDFS Graph . . . . .                  | 70 |
| 4.3 Discussion of the Experiments . . . . . | 72 |
| 4.4 Summary . . . . .                       | 77 |
| CONCLUSION                                  | 79 |
| BIBLIOGRAPHY                                | 79 |



## LIST OF FIGURES

| Figure   | Page |
|--|------|
| 1.1 RDF graph example . . . . .  | 4    |
| 1.2 RDFS graph example . . . . .   | 6    |
| 1.3 Formal context example . . . . .   | 7    |
| 1.4 Concept lattice example . . . . .  | 9    |
| 1.5 Reduced labeling diagram of concept lattice example . . . . .  | 10   |
| 1.6 Infobox of Portugal . . . . .  | 11   |
| 1.7 The DBpedia dataset for Barack Obama . . . . .   | 14   |
| 2.1 Similarity techniques . . . . .  | 16   |
| 3.1 RDF graph of music dataset . . . . .   | 51   |
| 3.2 Lattice of music dataset . . . . .   | 52   |
| 3.3 Reduced labeling diagram of music dataset lattice . . . . .  | 53   |
| 3.4 RDFS graph of music dataset . . . . .  | 55   |
| 3.5 RDF Gravity representation of music dataset's RDFS graph . . . . .   | 56   |
| 3.6 Objects of <code>rdf:type</code> predicate with <code>dbpedia-owl</code> prefix of <i>Lake Onega</i><br>and <i>Neva River</i> in DBpedia . . . . . | 56   |
| 3.7 Objects of <code>dbpedia-owl:Wikipagesdirect</code> predicate for <i>Neva River</i> in<br>DBpedia . . . . .  | 57   |
| 3.8 Jena framework . . . . .   | 59   |
| 3.9 SPARQL Example . . . . .   | 60   |
| 3.10 Galicia v.2 beta view . . . . .   | 62   |
| 3.11 RDF Gravity View . . . . .  | 63   |

|  |    |
|--|----|
| 3.12 SNORQL . . . . .  | 66 |
| 4.1 Lattice of Russia dataset by Galicia . . . . .             | 71 |
| 4.2 Full RDFS graph of Russia dataset by RDF Gravity . . . . . | 73 |
| 4.3 Parts of RDFS graph of Russia dataset . . . . .            | 76 |

---



## LIST OF TABLES

| Table  | Page |
|--|------|
| 1.1 Classes in DBpedia ontology . . . . .            | 11   |
| 2.1 Tool comparison . . . . .                        | 44   |
| 3.1 Binary relation table of music dataset . . . . . | 50   |
| 3.2 RDF Gravity notations . . . . .                  | 64   |
| 4.1 Measurement table . . . . .                      | 75   |



## ABBREVIATIONS

|             |   |
|-------------|---|
| FCA         | Formal Concept Analysis                                 |
| RDF         | Resource Description Framework                          |
| RDFS        | RDF Schema  |
| LOD         | Linked Open Data  |
| SW          | Semantic Web  |
| Galicia     | Galois Latticebased Incremental Closed Itemset Approach |
| SPARQL      | Simple Protocol and Rdf Query Language                  |
| RDF Gravity | RDF GRAPh VIualization Tool                             |
| XML         | Extensible Markup Language                              |
| W3C         | World Wide Web Consortium                               |
| NLP         | Natural Language Processing                             |



## RÉSUMÉ

Avec l'augmentation massive de la quantité de données disponibles sur le web, la détection et l'analyse d'information dans le contenu web deviennent très rentables. Le déploiement des données structurées fondé sur les technologies du Web sémantique a augmenté de façon significative en ligne au cours des deux dernières décennies. L'extraction d'information devient donc un problème majeur entre les chercheurs du Web sémantique.

Pour publier des données structurées sur le Web, les sources de données sont décrites avec le Cadre de Description des Ressources (Resource Description Framework ou RDF).

Dans cette mémoire, nous cherchons à extraire la structure conceptuelle du Web de données, c'est à dire, des données RDF dans le Web de documents. L'objectif principal est d'apprendre le niveau du schéma à partir du niveau d'instances, en d'autres termes, nous essayons de convertir les données RDF à RDF Schéma (RDFS) par apprentissage de la structure conceptuelle induite par des individus décrits en RDF.

Pour construire le treillis de concepts à partir de données RDF, les concepts sont identifiés à l'aide de l'Analyse de concepts formels (FCA). Le nombre de concepts est basé sur le nombre de sous-ensembles possibles contenant ressources RDF similaires. Par ressources RDF similaires, on veut dire que l'on considère l'ensemble des ressources RDF qui partagent un ensemble commun d'attributs. Après la construction du treillis de concepts, nous allons tenir compte des propriétés et des propriétés de données déduites à partir de données RDF pour construire le schéma.

Un autre défi pour construire le modèle RDFS est le fait de nommer les classes de RDFS. Pour cela, on utilise DBpedia. DBpedia contient l'information structurée de Wikipédia, qui contient des informations très utiles nous permettant d'apprendre le type d'instances de sortie dans les données RDF.

La méthodologie présentée dans cette thèse extrait le schéma maximum possible à partir du niveau d'instance de données RDF. En adoptant les étapes mentionnées avant, on atteint la capacité d'exploiter la structure conceptuelle à partir du Web de données.

Mots-clés: RDF, RDFS, DBpedia, treillis de galois, données liées



## ABSTRACT

The amount of available data on the web has considerably increased in recent years, thus the detection and analysis of useful information from its content is very profitable. Deployment of structured data based on Semantic Web technologies has grown significantly online in past two decades.

Therefore, information extraction has become a major concern among Semantic Web researchers. To publish structured data on the web, data sources are published using the Resource Description Framework (RDF) data model.

This thesis aims at extracting conceptual structures from Web of Data, i.e., RDF data in Web of documents. The main objective is to learn schema level from instance level in a dataset; in other words, we try to convert the RDF data into a data with the RDF Schema (RDFS) model by learning the conceptual structure between RDF individuals in the instance level.

To construct a concept lattice from the RDF data, concepts are identified via Formal Concept Analysis (FCA). The number of concepts is based on the number of possible subsets containing similar RDF individuals. By similar RDF individuals we mean the set of RDF resources which share a common set of attributes. After detecting concepts of the concept lattice -classes of RDFS- and the hierarchical relations between them, we take into account the properties and the inferred data properties from the RDF data in order to construct the schema level.

Another challenge in building the RDFS model from data is naming the RDFS classes. We overcome this issue by using DBpedia. DBpedia contains the structured information from Wikipedia, which contains very useful information allowing us to learn the type of exiting instances in the RDF data.

The proposed methodology in the thesis extracts the maximum possible schema from the instance level of RDF data. By adopting the aforementioned steps, we achieved the capability to exploit conceptual structure from Web.

Keywords: RDF, RDFS, DBpedia, concept lattice, linked data





## INTRODUCTION

Today, the Web of documents has expanded to the Web of Data since the appearance of Semantic Web. Web of Data is described as graphs of data. It rapidly produces large datasets containing billions of RDF triples from different domains of knowledge. Thus, with high growing availability of structured data on the web, exploiting it becomes ever more interesting.

Compared to RDF data, XML and HTML are more readable by humans than RDF since RDF data doesn't explicitly follow hierarchical and sequential structure formats. Therefore, RDF model lacks the simplicity of human readability and writability for its documents.

We believe that concept extraction from Web of Data provided in RDF helps us for fulfilling user's requirements in having a better understanding of heterogeneous data on the web. Implementing this idea could lead us to improve the readability of RDF statements by ordering and grouping them.

FCA is a key issue for formally discovering and representing concept hierarchies as well as the clustering of knowledge found on the web.

### Motivation

Even though the data sources are structurally defined on Web of Data, the effort for reducing decentralization of data which suffers from the lack of vocabulary in non-conceptualized data is interesting. In other words, extracting schema from data becomes more interesting when it comes to data without explicit conceptualization.

RDF describes resources without considering taxonomies of their classes and properties. The approach of discovering conceptual structure from Web of Data represented as RDF triples is possible by using FCA.

## Objective

Extraction of schema from RDF data could lead to RDFS model construction which contains richer vocabularies for describing the data. RDFS is an extension of RDF model which allows the description of RDF terms in the form of **class** (types of the instances), **subClass** (relation between classes), **property** (properties which describe classes) and **subProperty** (relation between properties) as well as **domain** and **range** of the properties. Obtaining an RDFS model from the RDF data helps us solve the problems of heterogeneity in raw data of the web.

## Structure of this dissertation

This dissertation is organized as follows:

In Chapter 1, we define the basic concepts which are used throughout this thesis. The main concepts that are explained in the chapter include: Resource Description Framework (RDF), RDF Schema (RDFS), Formal Concept Analysis (FCA) and DBpedia.

Chapter 2 presents a review of the literature. It presents related works to our thesis and the comparison of our works to them. First, the chapter discusses the basic similarity methods that exist for ontologies. The similarity methods are used for building interlinking tools which are introduced in continuance briefly. Finally, we present our approach in comparison to the other works for extracting similar RDF individuals.

Chapter 3 describes the full implementation of our methodology in addition to the introduction to some Java platforms and APIs required during implementation.

Finally, the methodology is evaluated by three metric measurements including precision, recall and f-measure in Chapter 4.

## CHAPTER I

### MAIN CONCEPTS

The current chapter provides background information on technologies we benefited from during our approach. In two first sections, brief introductions to RDF and RDFS models are provided. Third section introduces FCA which plays an important role in our methodology. Finally, the DBpedia which contains useful knowledge for generating our final output is proposed.

#### 1.1 RDF

The Resource Description Framework (RDF) is a fundamental data model in Semantic Web technology [MM04]. It is designed to be read and understood by machines. As a generic data model, RDF represents the information on the web in the form of <subject-predicate-object> triples. Each triple is a sentence describing a resource. A resource is an entity which can be a subject, predicate or object in an RDF triple. Each resource on the web is uniquely identified by Uniform Resource Identifier (URI). URI identifies a resource via location or a name or both.

The subject or first part of an RDF triple is a resource which the statement describes. The predicate or second part of a triple is a property or aspect which relates the resource to an object. Therefore, the object is third part of a triple which could be another resource or a literal value defined as a string or a number, a date, etc [LS99].

RDF depicts the information on the web as directed graphs. An RDF graph is composed

of a set of triples where each triple represents an arc. Therefore, each RDF statement is a subgraph where each node is a subject or object whereas arcs are predicates (The arc starts from the subject and it is directed to the object). Further, RDF can use XML based syntax, i.e., RDF/XML to create or modify the RDF graphs [RDF04]. An example of RDF graph is given in the following [Li13].

Suppose that a student with name *James Anderson* has professor *Paul Jones* as his supervisor. The statements related to this information are represented as an RDF graph shown in Figure 1.1.

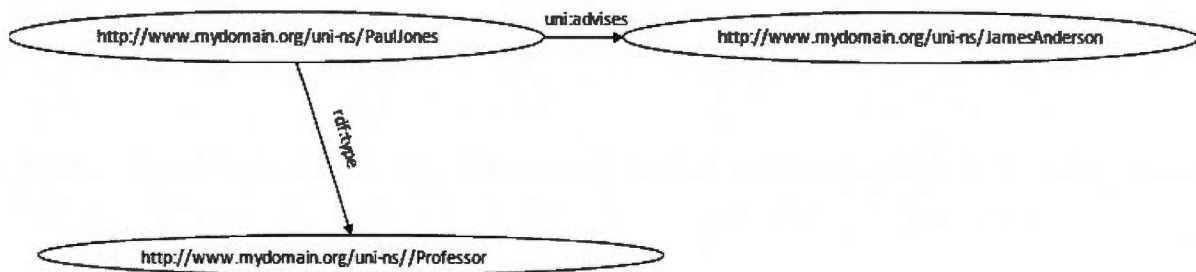


Figure 1.1: RDF graph example [Li13]

The XML syntax of the RDF data is:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:uni="http://www.mydomain.org/uni-ns">
  <rdf:Description rdf:about="http://www.mydomain.org/uni-ns/PaulJones">
    <rdf:type rdf:resource="http://www.mydomain.org/uni-ns/Professor"/>
    <uni:advises rdf:resource="http://www.mydomain.org/uni-ns/JamesAnderson"/>
  </rdf:Description>
</rdf:RDF>
```

`http://www.w3.org/1999/02/22-rdf-syntax-ns#` and `http://www.mydomain.org/uni-ns` are XML namespaces. One uses XML namespace in RDF to show the collection of names

of resources and properties. For example, the `xmlns:rdf` namespace specifies that the elements with `rdf` prefix come from the namespace `http://www.w3.org/1999/02/22-rdf-syntax-ns` which is known as a namespace for RDF vocabularies. Moreover, XML Qualified Name is a shortcut for URI; for example, we could use `uni` instead of the full URI `http://www.mydomain.org/uni-ns`.

In the graph, *Paul Jones* is connected to *James Anderson* by predicate *advises*. Besides, there exists another relation which connects *Paul Jones* to the class *Professor* at the schema level; therefore, *Paul Jones* is connected to class *Professor* by using predicate *type*. Again, the namespace `rdf` is used instead of writing the full URI `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. The full example of the schema level is given in the following section.

## 1.2 RDF Schema (RDFS)

On top of the RDF which doesn't provide significant semantics, RDFS is an extensible knowledge representation language which adds vocabulary to RDF in order to express information about class and subclass and properties (relationship between classes) [BVM04]. These vocabularies contain class, subclass, relationship between classes, property, subproperty, relationship between properties, domains and ranges, etc.

The RDFS level of the example described in the previous section is given in the following graph [Li13] (drawn as Figure 1.2):

The schema part of the RDF/XML syntax from RDFS data is:

```
<rdfs:Class rdf:ID = "Person"/>
<rdfs:Class rdf:ID = "Student"/>
<rdfs:subClassOf rdf:resource = "#Person"/>
</rdfs:Class>

<rdfs:Class rdf:ID = "Professor">
<rdfs:subClassOf rdf:resource = "#Person"/>
```

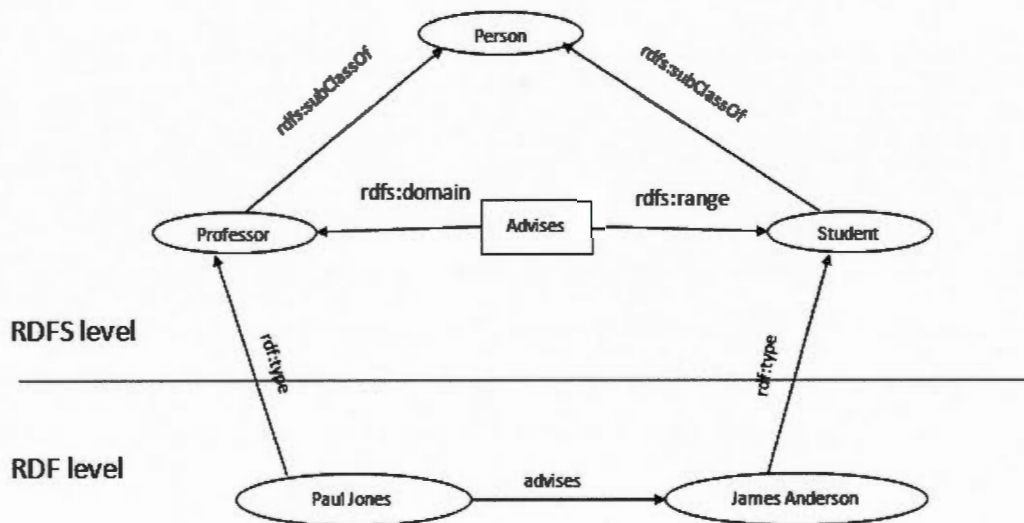


Figure 1.2: RDFS graph example [Li13]

```

</rdfs:Class>
<rdfs:Property rdf:ID = "advices">
<rdfs:domain rdf:resource = "#Professor"/>
<rdfs:range rdf:resource = "#Student"/>
</rdfs:Property>

```

"#" is used instead of writing URI reference. The `rdfs:domain` and `rdfs:range` predicates relate a predicate to the class of instances which can be considered as the subject or object of the predicate, respectively. `rdfs:subClassOf` identifies the hierarchical relationship between classes at the schema level. In the above example, *Professor* and *Student* are subclasses of *Person* class. *advices* is a property which has classes *Professor* and *Student* respectively as its domain and range.

### 1.3 FCA

#### 1.3.1 Introduction to Formal Concept Analysis

FCA stands for Formal Concept Analysis, a formal representation of data that has the potential to be represented as conceptual structure [GW99]. FCA is a data analysis technique that helps to identify the conceptual structure of data using formal contexts and concept lattices. Every dataset which consists of a binary relation between a set of objects and a set of attributes can be introduced as a formal context in FCA [WB04].

**Definition 1 (Formal Context):** A formal context is a triple  $K := (G, M, I)$ , where  $G$  and  $M$  are sets and  $I$  is a relation between  $G$  and  $M$ . The elements of  $G$  and  $M$  are called objects and attributes, respectively, and  $(g, m) \in I$  is read as "an object  $g$  has an attribute  $m$ ".

A set of objects and their corresponding attributes plus the relations that exist between those objects and attributes can be shown in a formal context. Formal context can be represented as a table in which rows are objects and columns are attributes and each cross in the table is a relation between an object and corresponding attribute.

An example of formal context can be seen in Figure 1.3. The example includes four object and four attributes.

|      | Att1                                | Att2                                | Att3                                | Att4                                |
|------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Obj1 | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| Obj2 | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/>            |
| Obj3 | <input checked="" type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/>            | <input checked="" type="checkbox"/> |
| Obj4 | <input checked="" type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/>            | <input checked="" type="checkbox"/> |

Figure 1.3: Formal context example

Further, the formal context can be represented in conceptual structure which will be explained in the next section.



### 1.3.2 Concept Lattice

A formal concept can be represented in a lattice of concepts in which each concept includes a set of objects and related attributes. The definition of formal concept and concept lattice are given in the following [WH06].

**Definition 2 ( ' Operation):**

For a set  $A \subseteq G$  of objects, we define:  $A' = \{m \in M \mid \forall g \in A : (g, m) \in I\}$

Correspondingly, for a set  $B \subseteq M$  of attributes, we define:  $B' = \{g \in G \mid \forall m \in B : (g, m) \in I\}$

The formal concept is defined as:

**Definition 3 (Formal Concept):** A formal concept  $C$  in the formal context  $(G, M, I)$  is a pair  $(A, B)$ , where  $A \subseteq G$ ,  $B \subseteq M$ ,  $A' = B$  and  $B' = A$ . The set  $A$  is called the extent and the set  $B$  the intent of the concept  $C$ .

In other words, each concept is represented by a pair consisting of an extension and an intension which are a set of objects and a set of attributes, respectively. As a general rule, the objects in the extension have all the attributes in their intension in common and have no other attributes in common. Further, all the attributes in the intension are shared by all the objects in the extension and no other object outside of the extension. A concept lattice arises on the top of the concepts derived from formal context.

**Definition 4 (Concept Lattice):** For a formal context  $K := (G, M, I)$  and two concepts  $C_1 = (A_1, B_1)$  and  $C_2 = (A_2, B_2)$ , a hierarchical subconcept-superconcept relation is given by

$$(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2 (\Leftrightarrow B_1 \supseteq B_2)$$

The set of all concepts in  $K$  ordered by the  $\leq$  relation is called the concept lattice of  $K$ . The concept lattice of the above example is shown in Figure 1.4.

The lattice can also be presented using reduced labeling diagram [GW99]. Reduced



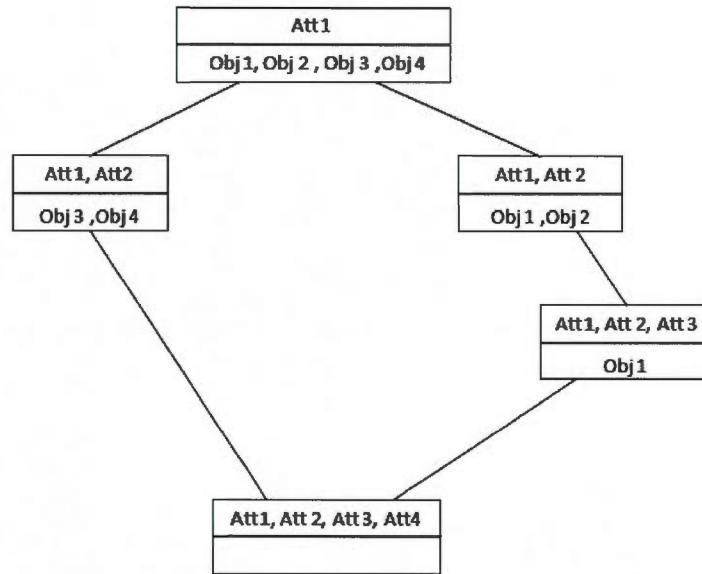


Figure 1.4: Concept lattice example

labeling diagram only shows the attributes and objects once in lattice diagram (Figure 1.5); therefore, it makes data analyzation easier for some applications.

#### 1.4 DBpedia

DBpedia<sup>1</sup> is a project that aims at extracting structured information from Wikipedia content. This open source data set is available on the web as linked data -RDF triples- for human and machine usage. Since DBpedia is provided in structural form, it allows users for much easier querying and exploring against Wikipedia content by using SPARQL endpoint. So far DBpedia is known as a central interlinking hub for published data on the web and it is evolved by any changes in Wikipedia [ABKLC08]. DBpedia includes around 3.5 million instances that belong to different categories. Also, DBpedia is available in 97 different languages. More information is given later in this section.

In the following, the structure of DBpedia and the source of its data will be described. Finally, the methods for accessing DBpedia are discussed.

<sup>1</sup><http://dbpedia.org/About>

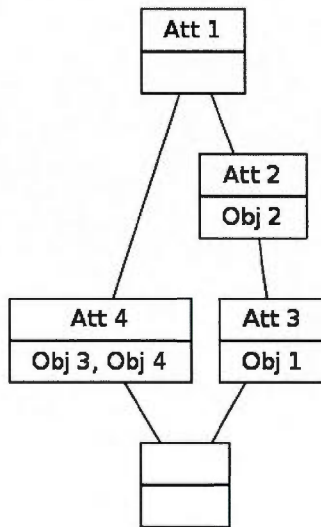


Figure 1.5: Reduced labeling diagram of concept lattice example

#### 1.4.1 Data Source

DBpedia is a cross-domain ontology which has been built manually by the members of DBpedia community. DBpedia uses Wikipedia as its source of knowledge. Wikipedia is one of the fastest-growing and largest collections of human knowledge ever collected. Since some of the information in Wikipedia is unstructured, querying information from it needs a full text search. The DBpedia community found a way to convert the contents of Wikipedia into RDF triples. In addition to free text information, DBpedia also uses the different types of structured information from Wikipedia including infobox templates<sup>1</sup>, title, abstract, categorization information, images, geo information, and external url links and converts them into RDF triples. Figure 1.6 shows an example of extracting semantics from a Wikipedia infobox for Portugal's content in DBpedia. Currently, the DBpedia ontology<sup>2</sup> is created based on several Wikipedia infobox templates and converts them into 359 classes with 1,775 properties.

As mentioned earlier, DBpedia includes 3.5 million instances and 2.35 million of which are classified in the DBpedia Ontology, including persons, places, works (contain music

<sup>1</sup><http://en.wikipedia.org/wiki/Help:Infobox>

<sup>2</sup><http://wiki.dbpedia.org/Ontology>

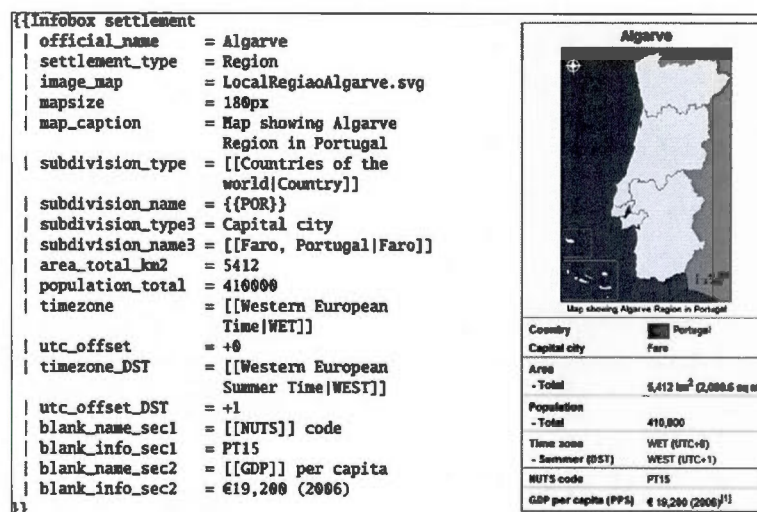


Figure 1.6: Infobox of Portugal

| Class              | Instances |
|--------------------|-----------|
| Resource (overall) | 2,350,000 |
| Place              | 573,000   |
| Person             | 764,000   |
| Work               | 333,000   |
| Species            | 202,000   |
| Organisation       | 192,000   |

Table 1.1: Classes in DBpedia ontology

albums, films and video games), organizations (contain companies and educational institutions), species. Table 1.1 shows the number of the instances per class in the DBpedia ontology.

#### 1.4.2 Data Structure

In contrast to Wikipedia, which lacks a structural representation of data, DBpedia uses Semantic Web technologies for extracting structured information from Wikipedia to facilitate querying and searching tasks. DBpedia ontology is based on the OWL language [PHH04].

The DBpedia extractor framework is used to extract intended data from Wikipedia

[MLASH12]. Some of the main information extracted from Wikipedia to be used in DBpedia is given below:

**Labels:** Every Wikipedia page defines a resource and has a title which is specified by `rdf:label` in the DBpedia dataset.

**Abstracts:**

**Short Abstract:** The first paragraph in each Wikipedia is considered as a short abstract of a corresponding resource specified by `rdfs:comment`.

**Long Abstract:** The text before a table of contents in each Wikipedia article is considered as a long abstract of a resource specified by `dbpedia-owl:abstract`.

**Interlanguage Links:** For each resource, the DBpedia dataset includes the links which connect articles about the same topic in different language editions of Wikipedia and uses them for assigning labels and abstracts in different languages to the resource.

**Images:** Each image in Wikipedia article related to a resource refers to Wikimedia Commons and it is specified by `foaf:depiction` in the DBpedia dataset.

**Redirects:** Since the synonym of each resource can be resources dedicated to other articles in Wikipedia, DBpedia redirects the page of the resource to those pages as references to that article and specify it by `dbpedia-owl:wikiPageRedirects`.

**Disambiguation:** The disambiguation links of a Wikipedia page corresponding to a resource are specified by `dbpedia-owl:wikiPageDisambiguates`.

**Infobox:** As mentioned earlier, each infobox contains properties which are represented by `http://dbpedia.org/property/` namespace in DBpedia. The name of a property is the same as the name used in infobox. Moreover, the Ontology Infobox Types dataset contains the `rdf:types` of the instances which have been extracted from the infoboxes.

**Geo-coordinates:** The latitude and longitude of a resources are specified by `geo:lat` and `geo:long` in DBpedia and express coordinates using Basic Geo (WGS84 lat/long) Vocabulary and the GeoRSS Simple encoding of the W3C Geospatial Vocabulary.

**External links:** Specified by `dbpedia:reference` in DBpedia, External links contain the references to external web resources.

**Pagelinks:** Pagelinks include all links between Wikipedia articles specified by `dbpedia-owl-`

:wikilink.

**Homepages:** The homepages of resources such as companies and organizations are specified by `foaf:homepage` in DBpedia.

**Person Data:** It include the resources about persons containing personal information such as surname, and birth date which are respectively specified by `foaf:surname`, and `dbpedia:birthDate`.

**Categories:** Categories are specified by `skos:concepts` and `skos:broader` and they includes categories using `skos` .

Moreover, DBpedia also includes links to other knowledge bases such as EuroStat, The World Factbook, Freebase, OpenCyc ,YAGO and Umbel.

### 1.4.3 Data Access

Three mechanisms exist for accessing DBpedia: Linked Data, the SPARQL protocol, and downloadable RDF dumps [ABKLC08].

**Linked Data:** DBpedia presents the structural format of Wikipedia content as Linked Data. Linked data publishes data on the web as RDF triples. Linked data principles include: URIs for identifying things (resources) in the world, RDF model for structuring and linking descriptions of things, HTTP for retrieving descriptions of those resources. The URIs give information about the resource, i.e., every information related to the resource in the form of RDF triples. For example, part of information about *Barack Obama* identified as URI `http://dbpedia.org/page/Barack_Obama` can be found in Figure 1.7. The Figure shows a HTML view of the information about resource accessible by typing the resource address in browsers.

**SPARQL Protocol:** The DBpedia dataset allows users to ask complex queries against Wikipedia. Looking for the intended triple by looking on linked data is not practical; therefore, developers came up with frameworks where users can ask their queries over the SPARQL protocol to this endpoint at `http://dbpedia.org/sparql`. More information on SPARQL and instructions are given later in chapter 3.



## About: Barack Obama

An Entity of Type : `agent`, from Named Graph : `http://dbpedia.org`, within Data Space : `dbpedia.org`



Barack Hussein Obama II is the 44th and current President of the United States, in office since 2009. He is the first African American to hold the office. Born in Honolulu, Hawaii, Obama is a graduate of Columbia University and Harvard Law School, where he was president of the Harvard Law Review. He was a community organizer in Chicago before earning his law degree.

| Property             | Value   |
|----------------------|---|
| dbpedia-owl:abstract | <ul style="list-style-type: none"> <li>Barack Hussein Obama II [bəˈjoʊk huːˈseɪn ouˈbɑːmə] ist ein US-amerikanischer Politiker und seit dem 20. Januar 2009 der 44. Präsident der Vereinigten Staaten. Er wurde bei der Präsidentschaftswahl 2008 in dieses Amt gewählt und am 6. November 2012 für eine zweite Amtsperiode als US-Präsident bestätigt. Obama, Sohn einer weißen US-Amerikanerin und eines Kenianers, ist der erste Afroamerikaner in diesem Amt. Obama ist ausgebildeter Rechtsanwalt für US-Verfassungsrecht und seit 1992 Politiker der Demokratischen Partei. Von 2005 bis 2008 gehörte er als Junior Senator für den US-Bundesstaat Illinois dem Senat der Vereinigten Staaten an. Am 10. Dezember 2009 wurde ihm der Friedensnobelpreis verliehen. Im Mai 2012 sprach er sich als erster US-Präsident im Amt öffentlich für die Legalisierung von gleichgeschlechtlichen Ehen aus.</li> <li>Barack Hussein Obama II is the 44th and current President of the United States, in office since 2009. He is the first African American to hold the office. Born in Honolulu, Hawaii, Obama is a graduate of Columbia University and Harvard Law School, where he was president of the Harvard Law Review. He was a community organizer in Chicago before earning his law degree. He worked as a civil rights attorney in Chicago and taught constitutional law at the University of Chicago Law School from 1992 to 2004. He served three terms representing the 13th District in the Illinois Senate from 1997 to 2004, running unsuccessfully for the United States House of Representatives in 2000. In 2004, Obama received national attention during his campaign to represent Illinois in the United States Senate with his victory in the March Democratic Party primary, his keynote address at the Democratic National Convention in July, and his election to the Senate in November. He began his presidential campaign in 2007, and in 2008, after a close primary campaign against Hillary Rodham Clinton, he won sufficient delegates in the Democratic party primaries to receive the presidential nomination. He then defeated Republican nominee John McCain in the general election, and was inaugurated as president on January 20, 2009. Nine months later, Obama was named the 2009 Nobel Peace Prize laureate. He was re-elected president in November 2012, defeating Republican nominee Mitt Romney, and was sworn in for a second term on January 20, 2013. Early in his first term in office, Obama signed into law economic stimulus legislation in response to the Great Recession in the form of the American Recovery and Reinvestment Act of 2009 and the Tax Relief, Unemployment Insurance Reauthorization, and Job Creation Act of 2010. Other major domestic initiatives in his presidency include the Patient Protection and Affordable Care Act; the Dodd–Frank Wall Street Reform and Consumer Protection Act; the Don't Ask, Don't Tell Repeal Act of 2010; the Budget Control Act of 2011; and the American Taxpayer Relief Act of 2012. In May 2012, he became the first sitting U.S. president to publicly support the rights of same-sex couples to legally marry, and in 2013 his administration filed briefs which urged the Supreme Court to rule in favor of same-sex couples in both the cases of <i>Hollingsworth v. Perry</i>.</li> </ul> |

Figure 1.7: The DBpedia dataset for Barack Obama

**RDF Dumps:** However, SPARQL protocol accessible at `http://dbpedia.org/sparql` allows the online queries for user, some users want to do the offline queries from DBpedia by downloading RDF Dumps (or N-Triple serializations) of DBpedia.

### 1.5 Summary

This chapter introduced the main concepts we need to know before diving into the details of the methodology steps. We first introduced RDF and RDFS to specify our input and output data. Then, FCA theory and DBpedia as a knowledge tool are proposed. In the next chapter, the related works to our approach are introduced.

## CHAPTER II

### REVIEW OF THE LITERATURE

With the rise of data on the web, large numbers of data sources from a wide range of domains have been produced recently. In order to publish structured data on the web, these data sources are published using the RDF data model. Due to the RDF graph topology, linked data-based applications can navigate throughout a data source and discover new data sources by following RDF links.

This thesis aims to look for schema out of the concrete data, i.e., grouping individual resources into clusters to become RDFS classes. Since grouping typically uses similarities between the individuals in a dataset, examining different ways of defining the similarity in the semantic web context is necessary.

The chapter studies similarity in interlinking tools. Before discussing interlinking tools, some similarity measurements used in those tools are introduced. Then, the similarity methodology used in our research, i.e. FCA, is discussed and compared with other work on RDF data. Finally, the usage of FCA in different semantic web applications is described.

#### 2.1 Basic Concepts Related to Similarity

The following introduces the basic methods for assessing the similarity and relations between entities of ontologies. Interlinking tools use the combination of these similarity methods in an adequate way to assess the similarity between RDF individuals in LOD.

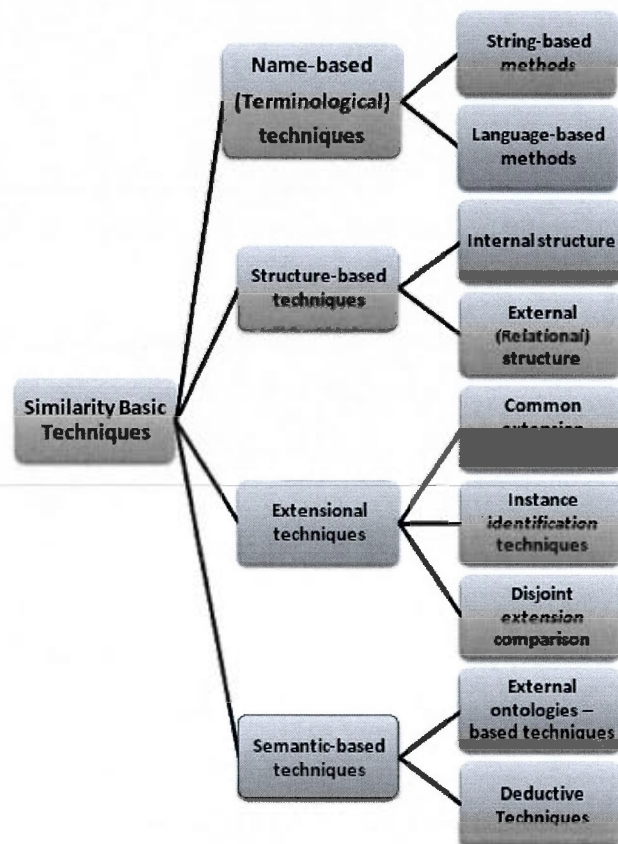


Figure 2.1: Similarity techniques



Four basic techniques for assessing the similarity between ontology entities exist: Name-based, Structure-based, Extensional and Semantic-based. They are respectively described in the following sections [ES07].

First, some basic definitions related to the similarity measurement between two entities are introduced.

**Definition 5 (Similarity):** A similarity  $\sigma : O \times O \rightarrow \mathbb{R}$  is a function from pair of entities to a real number expressing the similarity between two objects such that:

$$\forall x, y \in O, \sigma(x, y) \geq 0 \quad (\text{positiveness})$$

$$\forall x \in O, \forall y, z \in O, \sigma(x, x) \geq \sigma(y, z) \quad (\text{maximality})$$

$$\forall x, y \in O, \sigma(x, y) = \sigma(y, x) \quad (\text{symmetry})$$

**Definition 6 (Dissimilarity):** Given a set  $O$  of entities, a dissimilarity  $\delta : O \times O \rightarrow \mathbb{R}$  is a function from a pair of entities to a real number such that:

$$\forall x, y \in O, \delta(x, y) \geq 0 \quad (\text{positiveness})$$

$$\forall x \in O, \delta(x, x) = 0 \quad (\text{minimality})$$

$$\forall x, y \in O, \delta(x, y) = \delta(y, x) \quad (\text{symmetry})$$

For dissimilarity, the distance and ultrametric notions could be considered:

**Definition 7 (Distance):** A distance (or metric)  $\delta : O \times O \rightarrow \mathbb{R}$  is a dissimilarity function satisfying the definiteness and triangular inequality:

$$\forall x, y \in O, \delta(x, y) = 0 \text{ if and only if } x = y \quad (\text{definiteness})$$

$$\forall x, y, z \in O, \delta(x, y) + \delta(y, z) \geq \delta(x, z) \quad (\text{triangular inequality})$$

**Definition 8 (Ultrametric):** Given a set  $O$  of entities, an ultrametric is a metric such that:

$$\forall x, y, z \in O, \delta(x, y) \leq \max(\delta(x, z), \delta(y, z)) \quad (\text{ultrametric inequality})$$

To simplify the process of comparing measures with each other and improving the accuracy of measurements, the measures shall be normalized. To do that, for all (dis)similarity methods in this report the normalized versions are given.

**Definition 9 (Normalised (dis)similarity):** A (dis)similarity is said to be normalised if it ranges over the unit interval of real numbers  $[0, 1]$ . A normalised version of a (dis)similarity  $\sigma$  (respectively,  $\delta$ ) is denoted as  $\bar{\sigma}$  (respectively,  $\bar{\delta}$ ).

### 2.1.1 Name-based Techniques

Name-based (Terminological) techniques are the similarity methods based on terminology and they can be used to compare class names, URLs, label and comments of different entities.

Due to Synonymy (entities with the different names but the same meanings) and Homonymy (entities with the same name but different meanings) problems, we cannot simply compare the entities just by their names. In addition to Synonymy and Homonymy problems, words from different languages and syntactic variations (For example, an abbreviation with two different usage expanded) of the same words could also cause difficulties in comparing terms with each other according to their similar names.

Name-based techniques compare the terms by considering strings only as sequences of characters (String-based methods) or by taking into account some linguistic knowledge interpreting these strings (Language-based methods).

### String-based methods

Depending on the structure view of strings, many string-based methods have been created. Some of these methods used frequently in similarity techniques are discussed below.

#### a. Normalisation:

To enhance the results of string comparison, the initial strings could be normalized by some normalization procedures:

*Case normalization* converts each alphabetic character of string to its lower case.

*Diacritics and Digit suppression* replaces character with diacritic signs with their most frequent replacements and removes the numbers from strings.

*Blank normalization* converts all blank characters to a single blank.

*Link stripping* replaces the links between words such as apostrophe and underline with blanks or dashes.

*Punctuation elimination* removes punctuation signs.

All these normalizations must be applied with some caution. For example, in diacritics suppression which converts *livré* to *livre*, we should consider in French there are differences between the meanings of these two words; therefore, diacritics suppression shouldn't be applied in this case.

#### b. String equality and Substring techniques:

String equality returns 0 when the input strings are different and returns 1 when they are the same (when the result is 0 it doesn't consider how much these two strings are differ from each other).

**Definition 10 (String equality):** String equality is a similarity  $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0 \ 1]$

such that  $\forall x, y \in \mathbb{S}, \sigma(x, x) = 1$ , and if  $x \neq y, \sigma(x, y) = 0$

*Hamming distance* is another metric which calculates the dissimilarity between two strings by counting the number of positions in which the characters of two strings are different.

Substring test examines if one string is a substring of another; thus, they become very similar.

**Definition 11 (Substring test):** Substring test is a similarity  $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0 \ 1]$  such that  $\forall x, y \in \mathbb{S}$ , if there exist  $p, s \in \mathbb{S}$  where  $x = p + y + s$  or  $y = p + x + s$ , then  $\sigma(x, y) = 1$ , otherwise  $\sigma(x, y) = 0$ .

*Substring similarity* computes the ratio of the common subpart between two strings (this method is also useful for examining if one string is prefix or suffix of another string or finding the longest common suffix or prefix among strings; it is also used to compare specific and general strings and compare strings and similar abbreviations to those strings).

**Definition 12 (Substring Similarity):** Substring similarity is a similarity  $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0 \ 1]$  such that  $\forall x, y \in \mathbb{S}$  and  $t$  is the longest common substring of  $x$  and  $y$ :

$$\sigma(x, y) = \frac{2|t|}{|x| + |y|}$$

The  $n$ -gram as another substring test method counts the number of common  $n$ -grams (sequence of  $n$  characters) between two strings. For example, 3-grams of word *paper* are *pap*, *ape* and *per*. This function is efficient when only few characters of string are missed in another string.

**Definition 13 ( $n$ -gram similarity):** Let  $ngram(s, n)$  be the set of substrings of  $s$

of length  $n$ . The  $n$ -gram similarity is a similarity  $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}$  such that:

$$\sigma(s, t) = |ngram(s, n) \cap ngram(t, n)|$$

The normalised version of this function is as follows:

$$\bar{\sigma}(s, t) = \frac{ngram(s, n) \cap ngram(t, n)}{\min(|s|, |t|) - n + 1}$$

c. *Edit distance*:

Edit distance between two strings consists of the sequence of operations with minimal cost (each operation has a cost) to obtain one string from another (might be used for the words with mistakes in their spelling).

**Definition 14 (Edit distance):** Given a set  $Op$  of string operations ( $op : \mathbb{S} \rightarrow \mathbb{S}$ ), and a cost function  $w : Op \rightarrow \mathbb{R}$ , such that for any pair of strings there exists a sequence of operations that transform the first one into the second one (and vice versa), the edit distance is a dissimilarity  $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0, 1]$  where  $\delta(s, t)$ , is the cost of the less costly sequence of operations that transform  $s$  into  $t$ .

$$\delta(s, t) = \min_{(op_i)_{i \in I}; op_n(\dots op_1(s))=t} \left( \sum_{i \in I} w_{op_i} \right)$$

$(op_i)_{i \in I}$  indicates the set of operations that convert string  $s$  to  $t$  and  $I$  consists of a variety of sets of operation numbers. For each set of  $I$  the cost of sequence of operations are calculated to examine the one with the lowest cost.

Three main operations of edit distance consist of: *Insertion of a character*, *Replacement of a character with another one* and *Deletion of a character*.

The Edit distance was first introduced by Levenshtein and in its simplified definition all operations have the same cost equal to 1 [Lev65].

*Jaro* measure is another measure based on the common characters between two strings (since the *Jaro* is not symmetric, it's not similarity).

**Definition 15 (Jaro measure):** The *Jaro* measure is a non symmetric measure  $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0 \ 1]$  such that

$$\sigma(s, t) = \frac{1}{3} \times \left( \frac{|com(s, t)|}{|s|} + \frac{|com(t, s)|}{|t|} + \frac{|com(s, t)| - |transp(s, t)|}{|com(s, t)|} \right)$$

$|com(s, t)|$  stands for the number of common characters between  $s$  and  $t$ .

Notice that two characters of strings  $s$  and  $t$  are common if they are the same ( $s[i] = t[j]$ ) and  $j$  and  $i$  satisfy  $\exists j \in [i - (\min(|s|, |t|)/2, i) + \min(|s|, |t|)/2]$  ( $i$  and  $j$  are the positions of common characters in  $s$  and  $t$ ).

$|transp(s, t)|$  indicates the number of cases where the  $i$ -th common character of  $s$  is not equal to the  $i$ -th common character of  $t$ . For instance, the  $transp('MARTHA', 'MARHTA')$  is equal to 2 because between all common characters of  $s$  and  $t$  only two characters  $T$  and  $H$  are placed disorderly in  $s$  and  $t$ .

*Jaro – Winkler* measure is a variant of *Jaro* measure and it focuses on the longest common prefix between two strings.

**Definition 16 (Jaro-Winkler measure):** The *Jaro – Winkler* measure  $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0 \ 1]$  is as follows:

$$\sigma(s, t) = \sigma_{Jaro}(s, t) + P \times Q \times \frac{1 - \sigma_{Jaro}(s, t)}{10}$$

such that  $P$  is the length of the common prefix and  $Q$  is a constant and  $\sigma_{Jaro}(s, t)$  stands for the *Jaro* measure in the previous definition.

*Smoa* is another similarity measurement introduced in [SSK05]. *Smoa* depends on both matched and unmatched substring lengths and the result value of the measure is in  $[-1, 1]$ .



d. *Token-based distance*:

In Token-based technique, each string is considered as a (multi)set of words (bag of words). In comparison to set, multi set's elements could appear several times in multi set. This technique works out for long texts by aggregating different sources of strings or by splitting strings into independent tokens.

Each multi set or bag of words could be a vector in which each dimension is a term (token) and each position in the vector is the number of occurrences of the token in its related multi set. Many existing measures can be cited related to the token-based techniques. Two of these measures (*Cosine similarity* and *TF - IDF*) are discussed here:

*Cosin* Similarity as a usual metric distance calculates the cosine of the angles between two vectors.

**Definition 17 (Cosine similarity):** Given  $\vec{s}$  and  $\vec{t}$ , the vectors corresponding to two strings  $s$  and  $t$  in a vector space  $V$ , the cosine similarity is the function  $\sigma_V : V \times V \rightarrow [0, 1]$  such that:

$$\sigma_V(s, t) = \frac{\sum_{i \in |V|} \vec{s}_i \times \vec{t}_i}{\sqrt{\sum_{i \in |V|} \vec{s}_i^2 \times \sum_{i \in |V|} \vec{t}_i^2}}$$

For instance, for two sentences "I have to be there" and "I have to go to" we would have (For each sentence, the vector including all words of both sentences is created and the values of each dimension would be the frequency of each word in respect sentences.):

Sentence 1:  $[I, have, to, be, there, go] = [1, 1, 1, 1, 1, 0]$

Sentence 2:  $[I, have, to, be, there, go] = [1, 1, 2, 0, 0, 1]$

and their *Cosin* similarity would be:

$$\sigma_v(I\ have\ to\ be\ there, I\ have\ to\ go\ to)$$

$$= \frac{(1 \times 1) + (1 \times 1) + (1 \times 2) + (1 \times 0) + (1 \times 0) + (0 \times 1)}{\sqrt{((1^2) + (1^2) + (1^2) + (1^2) + (1^2) + (0^2)) * ((1^2) + (1^2) + (2^2) + (0^2) + (0^2) + (1^2))}} = \frac{4}{\sqrt{35}}$$

There exists several similarity measures that attempt to use lower dimensions. Latent Semantic Indexing (*LSI*) as a dimensionality reduction technique, using matrix-computation methods to reduce the dimension space of each vector. *LSI* is based on the idea that words that occur in the same context have identical meanings. *LSI* assigns a column of a matrix to each document; then, all columns of the matrix are decomposed by the singular-value decomposition (*SVD*) method and at the meantime the factors of matrix with less influence on the rest information would be removed; thus, the dimensions of vectors became smaller [DDFLH90].

Another common measure is *TF-IDF* (Term frequency-inverse document frequency). This measure is used to assess the importance of each word in a document to whole corpus. The importance is increased according to the number of times that the word appears in the document (*tf*) by considering the inverse proportion of the word's occurrence in the entire corpus (*idf*). Therefore, for measuring *TFIDF*, both *tf* and *idf* become important.

**Definition 18 (Term frequency-Inverse document frequency):** Given a corpus *C* of multisets, we define the following measures:

$$\forall t \in \mathbb{S}, \forall s \in C, tf(t, s) = t\#s \quad (\text{term frequency})$$

$$\forall t \in \mathbb{S}, idf(t) = \log\left(\frac{|C|}{|\{s \in C; t \in s\}|}\right) \quad (\text{inverse document frequency})$$

$$TFIDF(s, t) = tf(t, s) \times idf(t) \quad (TFIDF)$$

In the above formula, *s* could be assumed as a document therefore *t*#*s* is the number of occurrences of term *t* in document *s* and *idf*(*t*) is the inverse document frequency of term *t* explained before.

e. *Path comparison:*



Path comparison considers the distance of two compared entities from their superest class as well as comparing the label of those entities and the entities in considered paths (the path from their superest class).

### Language-based methods

In language-based methods, a string is considered as a text composed of words occurring in sequence with a grammatical structure. This method uses *NLP* techniques to extract the meaningful terms from texts; by comparing these terms and their relations the similarity between entities are assessed. There exist two types of linguistic methods: the ones using only the internal linguistic properties of the instances (Intrinsic methods) and those using external resources like dictionaries and lexicons (Extrinsic methods).

#### *Intrinsic methods-Linguistic normalisation:*

Each term can appear in various forms. The main kinds of terms' forms are: **Morphological** (foundation form of a term based on some roots) divided to inflection and derivational or combination of them, **Syntactic** (Grammatical Structure of term) divided into coordinate, permutation and insertion, **Semantic** (usually using **Hypernymy** (general meaning) or **Hyponymy** (specific meaning) or **Synonymy** (same meaning)), **Multi-lingual** and **Morphosyntactic** combination of morphological (derivational) and syntactic variants.

To obtain the standardized form of the term (normalisation), the combination of these functions are used: **Tokenizer** (separate a string into its tokens by recognizing them by punctuation, cases, blank characters, digits, etc.), **Lemmatisation** (Stemming), **Term extraction** (extracting terms from text by applying syntactic and morphological transformations and using patterns -multi rules- on multi terms for extracting terms), **Stop word elimination** (eliminating common words)

#### *Extrinsic methods:*

Various kinds of linguistic resources are useful for finding the similarities between terms. The list of linguistic recourses is proposed in the following:

*Lexicons*: Lexicons or dictionaries are composed of sets of words with their definitions. (For the words with several synonyms gloss-based distance could be used)

*Multi-lingual lexicons*: dictionaries or lexicons in which words have their equivalent terms in other languages as their definitions, for example, the definition of word *Paper* is *Article* which is a French word. Multi-lingual lexicons are useful for ontologies which include words in different languages.

*Semantico-syntactic lexicons*: lexicons which record names, their categories and the types of arguments taken by verbs and adjectives.

*Thesauri*: A thesaurus is a kind of lexicon with some relational information that includes hypernyms, synonyms, and antonyms. (WordNet is an example of Thesaurus which distinguishes between words by making synsets (sets of synonyms)).

*Terminologies*: Treasures which contains phrases rather than single words (in comparison to dictionaries it has less semantic ambiguity).

If these lexicons focus on specific domain the results would be more efficient; because in this way the ontologies only concern on specialized senses for a word and not every senses existing for a word. This would also be useful in common abbreviations. For example, in company domain it would recognize the abbreviation PO as Purchase Order instead of Post Office.

Some similarity measures based on lexicon resources are defined in the following:

**Definition 19 (Synonymy similarity)**: Given two terms  $s$  and  $t$  and a synonym resource  $\Sigma$ , the synonymy is a similarity  $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0 \ 1]$  such that:

$$\sigma(s, t) = \begin{cases} 1 & \text{if } \Sigma(s) \cap \Sigma(t) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

**Definition 20 (Cosynonymy similarity)**: Given two terms  $s$  and  $t$  and a synonym

resource  $\Sigma$ , the cosynonymy is a similarity  $\sigma : \mathbb{S} \times \mathbb{S} \rightarrow [0\ 1]$  such that:

$$\sigma(s, t) = \frac{|\Sigma(s) \cap \Sigma(t)|}{|\Sigma(s) \cup \Sigma(t)|}$$

Some other similarity measures consider hyponymy/ hypernymy between synstes (these methods are discussed in Relational structure of structure-based techniques).

The rest of the methods (usually not normalized) rely on information theoretic perspective. For example, *Resnik* method uses the hierarchical relation between synsets of terms for measuring the similarity between two terms and it is based on information theory. Therefore, the information content of a concept is the inverses of its occurrence probability.

Another similarity measurement is *Information Theoretic Similarity* method. it considers information content of common synset information of two terms to the information content of both terms.

For comparing two strings through lexicon resource *Gloss overlap* method is used. The Gloss overlap of two terms is based on the number of shared words (overlaps) in their definitions (glosses).

### 2.1.2 Structure-based Techniques

Structural-based techniques based on conceptual relations which compare the structure of entities. This comparison is categorized into: 1- Internal structure which considers the properties of entities in addition to their names and labels and 2- Relational structure which compares the entities related to each other.

### Internal structure

Comparing entities according to their internal structure (e.g. the domain and range of their properties) is another way to find the similarity between entities, but these methods try to cluster entities according to their similar structure rather than finding the accurate similarity between these entities. Since the internal structure couldn't provide much information, these methods should be used in combination with other techniques. Some internal structure comparisons are mentioned in the following:

#### *Property comparison and keys:*

Keys in classes have a main role in identifying individuals. Two classes identified in the same way (each class has a primary key and the main keys of two classes have the one-to-one relationship with each other) represent the same set of individuals.

#### *Datatype comparison:*

Property comparison may include property datatype comparison. One can say the proximity between two datatypes are maximal when their types are the same (string and string), low when compatible (string and character) and lower when non compatible (string and integer). Since comparing datatypes technique is not complete and might have some incorrect results, the methods using such techniques should use other techniques in combination with datatype comparison.

#### *Domain comparison:*

In comparison to individuals, classes have domains instead of values. Comparing domains with each other is based on both intersection and union of intervals.

### Relational structure (external)

The similarity measure between two entities could be based on their positions hierarchies. When considering ontologies as graphs, relations between entities (edges between nodes) are obtained. Contrary to Extensional methods in which entities mean individuals, in this approach, entities mean classes and properties. For finding similarity

between graphs, graph homomorphism problem and maximum common directed sub-graph (*MCS*) definitions are considered.

There exist three types of structural relations between entities, all based on their hierarchies: Taxonomic relations (*subClassOf*), Mereologic relations (*a-part-of*) and all involved relations.

*Taxonomic structure:*

Taxonomy relation (*SubClassOf*) is a very important factor in comparing ontologies structures; therefore, many measures have been discovered in finding such relation between classes of ontologies. In taxonomy, a super-entity could have relation with one or more sub-entities while a sub-entity could have relation with one or more super-entities. Three popular similarity measures are: *Structural Topological Dissimilarity on Hierarchies* which is used to discover the shortest paths, *Wu-Palmer* counts the number of edges in the taxonomy between two classes and *Upward Cotopic* which counts the number of common super classes.

In addition to global methods mentioned so far, there exist some non-global measures, such as super or subclass rules and bounded path matching.

*Mereologic:* In mereology, the relations between entities are whole part relations. The sub-entity is a part of the super-entity and the super-entity can be composed of different sub-entities. Therefore, the classes are considered more similar if they share similar parts (we cannot have the same criteria as we had for Taxonomy -if all or most of leaves of two entities are the same then the entities are the same to-because two super-entities don't have the same leaves). Besides, the extension of classes could be compared for finding more similar objects (might share the same set of parts).

*All relations:* The similarity between entities could also be based on their all relations. This can be extended to a set of classes and a set of relations. It means that if we have a set of relations  $r_1 \dots r_n$  in the first ontology which is similar to another set of relations  $r_1 \dots r_n$  in the second ontology, it is possible to say that two classes are similar too.

For having more similarity we have to consider two other extension solutions:

*Children:* Two non-leaf entities are structurally similar if their immediate children sets



are highly similar.

Leaves: Two non-leaf elements are structurally similar if their leaf sets are highly similar, even if their immediate children are not.

### 2.1.3 Extensional Techniques

Extensional techniques are instance-based techniques and they are really useful for comparing classes with their available individuals (elements). Extensional methods are divided into: 1- the methods applied to classes (of ontologies) with common instances (common extension comparison), 2- the methods proposing instance identification techniques and 3- the ones without identification (disjoint extension comparison). In the mentioned extensional techniques the similarities between the sets of class instances are measured.

#### Common extension comparison

One way to compare the classes is to find the intersection of their instances (extensions). The relations between entity sets are considered based on set theory: equal ( $A \cap B = A = B$ ), contains ( $A \cap B = A$ ), contained-in ( $A \cap B = B$ ), disjoint ( $A \cap B = \emptyset$ ) and overlap.

Even though, the amount of incorrect data is small, it may lead to a huge fault in whole results. Besides, the dissimilarity is one if two classes don't have any of their instances in common. For solving these problems the Hamming distance is used.

**Definition 21 (Hamming distance):** The Hamming distance between two sets is a dissimilarity function  $\delta : 2^E \times 2^E \rightarrow \mathbb{R}$ , such that  $\forall x, y \subseteq E$

$$\delta(x, y) = \frac{|x \cup y - x \cap y|}{|x \cup y|}$$

Another way to compute the similarity is based on the probabilistic of instances in sets.

**Definition 22 (Jaccard similarity):** Given two sets  $A$  and  $B$ , let  $P(X)$  be the probabilities of a random instance to be in the set  $X$ . The *Jaccard* similarity is defined by:

$$\sigma(A, B) = \frac{P(A \cap B)}{P(A \cup B)}$$

$P(X)$  is a probability of random instance belongs to set  $X$ .

This measure is normalized and reaches 0 when  $A \cap B = \emptyset$ ; and 1 when  $A = B$ .

One tool from formal concept analysis (FCA) is to compute concept lattice. The concept lattice is composed of set of objects (instances) and their properties. This operation is not accurate but it could be optimized by removing redundant relations and it starts with the complete lattice of the set of instances and preserves the nodes if they are closed otherwise the nodes are discarded [GW99].

#### Instance identification techniques

If the sets don't have any instances in common; then, one way could be the identification of the relations between instances. One solution is to use keys in their external identification. If the keys are not available other approaches with string-based and structure-based techniques might be used to compare property values.

#### Disjoint extension comparison

When comparing instances is not directly possible some approximate techniques should be used:

##### *Statistical approach:*

This approach is computing some statistics about the property values found in instances

(such as maximum, minimum,...); therefore, this may help to characterize some internal structure of entities (domain of class properties). Moreover, since the data patterns and distribution have lower time-consumption and need less data, data values can be replaced with them.

*Similarity-based extension comparison:*

The method is based on the computed (dis)similarities between instances for comparing the set of instances between classes (There is no any expectation from classes to share the same instances).

*Matching-based comparison:*

In matching-based comparison, only the related elements(instances) would be compared with each other (not all instances).

**Definition 23 (Match-based similarity):** Given a dissimilarity function  $\sigma : E \times E \rightarrow \mathbb{R}$ , the *Match – based* similarity between two subsets of  $E$  is a similarity function:  $MSim : 2^E \times 2^E \rightarrow \mathbb{R}$  such that  $\forall x, y \subseteq E$

$$MSim(x, y) = \frac{\max_{p \in \text{pairing}(x, y)} (\sum_{\langle e, e' \rangle \in p} \sigma(e, e'))}{\max(|x|, |y|)}$$

with  $\text{Pairings}(x, y)$  being the set of mappings of elements of  $x$  to elements of  $y$ .

#### 2.1.4 Semantic-based Techniques

Semantic-based techniques compare the interpretation of entities. They have the ability to ensure the completeness (finding all correspondences that must hold) and consistency (finding all correspondences leading to inconsistency). Before applying deductive techniques, the alignments on entities (ontology anchoring) are initiated and then extended by deductive methods.



### External ontologies -based techniques

One way to find the similarity between two ontologies is to find the external resources related to those ontologies. Three characteristics make these resources different (in all of them, there is the possibility to match terms):

**Breadth** (whether the ontologies have a general purpose resources or specific purpose resources), **Formality** (whether the ontologies are pure, i.e., informal resources such as WordNet or not, i.e., formal resources, and **Status** (whether the resources are sets of instances or ontologies).

Two steps are needed for initiating an alignment between ontologies :

1. Anchoring: Matching two ontologies (non-reference) to the background ontology (reference)
2. Deriving relations: Combining the anchor relations with the relations coming from the reference ontology and use them to make new relations between anchor ontologies (non-reference) by reasoning methods.

### Deductive Techniques

To search more correspondences between entities of ontologies, some deductive techniques such as Propositional techniques and Description logic techniques are used.

#### *Propositional techniques:*

Propositional satisfiability for matching two classes from two different ontologies includes three steps:

1. Building a theory or domain knowledge which might use WordNet to discover identical words and then convert the relations between classes into the language of propositional logic.
2. Creating matching formulas which have axioms as premises entailing the relations between two classes ( $=, \sqsubseteq, \sqsupseteq, \perp$ ).
3. In the last step, the created formula from step two should be checked for validity.

Therefore, if the negation of formula is unsatisfiable, then we are allowed to say the formula is satisfiable.

*Description logic techniques:*

Compared to propositional logic, description logic is more expressive. We can match terms with the same interpretation by merging ontologies and looking for equivalence between pairs of ontology classes and subsumption roles.

## 2.2 RDF and Similarity

The aforementioned basic similarity methods can be used to find similarity between individuals in LOD (RDF datasets). Most of the interlinking tools proposed in section 2.2.1 use the combination of these basic methods along with the methodologies they use to find similarity between RDF graphs. Even though, we may use very common and widely used string matching techniques in the whole process of our application while converting RDF dataset to RDFS model, the main purpose is to use FGA to detect similar RDF individuals according to the properties they share. In this section, the studies of similarity method in interlinking tools as well as the methodology used to find similarity in RDF dataset by the help of FCA are discussed.

### 2.2.1 Similarity of RDF Graphs on Linked Open Data (Interlinking Tools)

With large amount of published data on the web, discovering explicit links between entities in different data sources, i.e., interlinking the Web of Data becomes more essential. Many interlinking tools exist which can be applied to different domains (e.g., music ontology, publications, etc.) or data types (e.g., multimedia data).

In the following, we introduce each interlinking tool briefly. More details about each tool can be seen in Table 2.1.

LD-Mapper

LD-Mapper [RSS08] is an interlinking framework in music domain. LD-Mapper provides two matching approaches. The first one is naïve approach consisting of simple literal lookup and extended literal lookup methods and the second is graph matching approach which has better performance since it takes into account both the similarity between resources and their neighbors. The interlinking experiments include linking a creative common music dataset (<http://www.jamendo.com/de/>) to an editorial one (<http://musicbrainz.org/>) and linking personal music collection to corresponding web identifiers (<http://musicbrainz.org/>). The latter tries to link audio files (using ID3 metadata of files) in personal music collection to editorial music dataset and it is known as GNAT tool.

#### RDF-AI

RDF-AI [SLZ09] is a tool for integrating RDF datasets through merging and interlinking. RDF-AI finds alignment between two given data sets in order to merge or interlink them. RDF-AI includes five steps: 1) Preprocessing prepares two data sets for matching by doing some operation on them including: Checking for any inconsistency of input datasets, Materialization of RDF triples, Translation of properties from one language to another language, Ontology evolution and Properties transformation such as name reordering, 2) Matching returns alignment between two data sets, 3) Interlinking generates linkset between two datasets from alignment, 4) Fusion merges two datasets and 5) Post-processing checks the inconsistencies of fusion results.

#### RKB-CRS

RKB-CRS [JGM08] manages the co-reference between URIs (Two different URIs referring to one entity) by using Consistent Reference Service (CRS) [JGM07]. The synonym URIs can be originated from the same dataset or different datasets. Co-reference is provided with bundles where each bundle stores the resources refer to the same entity. Therefore, at first a bundle is dedicated to each URI, i.e., the number of URIs is equal to the number of bundles. Then the different bundles with the resources

refer to the same entity are merged gradually in order to generate the new and bigger bundles.

#### ODD-Linker

ODD-Linker [HKLM09] or LinQuer [HXML09] is an interlinking tool for discovering relations between data items in different relational data sets stored as RDBMS. ODD-Linker links the equivalent records within relational datasets represented in LOD.

The equivalent records are determined by two matching methods: 1) An approximate string matching method tries to find approximate or similar strings to an input string using Jaccard weighted method (Definition 22) with q-grams (Definition 13). 2) Semantic Matching method uses semantic knowledge (ontology) which encapsulates synonymy, hyponymy/hypernymy relation types between records of relational datasets. Finally, LinQL languages is proposed to specify the linkage within relational datasets.

#### Knofuss

Knofuss [NUM07] merges two different RDF datasets by integrating ontologies in instance level, i.e. knowledge fusion. In order to compare two datasets, first an ontology is dedicated to each dataset specifying which resources to compare. If two ontologies are different, ontology alignment is used. An application context for each resource type is defined and for each application context it is specified which similarity method should be used. The application contexts with the same similarity method are given the same ID in both ontologies of datasets. Knofuss also checks the results of two datasets fusion for any inconsistencies.

Knofuss knowledge fusion is composed of two main subtasks: 1) ontology integration (schema level) and 2) knowledge integration (instance level). Co-referencing step of integration process in Knofuss specifies 1) the tasks which have to be accomplished, 2) library of methods for solving the problem and 3) the appropriate methods to be selected.

Three methods exist in order to solve co-referencing problem: manually constructed

using primary key concept for each object, supervised method including machine learning methods and unsupervised methods containing similarity measures such as string similarity -Edit distance (Definition 14), Jaro-Winkler (Definition 16), Leveshtein- and set similarity -Cosin (Definition 17), Jaccard (Definition 22), TDF-IDF (Definition 18)-metrics. These matching algorithms could be combined in order to make an efficient matching algorithm.

### Guess What!?

Guess What [MV10] is a semantic game using human intelligence in order to create formal domain ontology by mining linked open data. Guess what provides players with a described concept (class expression) from LOD and asks them to guess a suitable class name related to the class expression, i.e., extraction of ontologies from unstructured text.

The system's architecture includes three layers: Data, Data Access, Business logic. Data layer composed of a Sesame RDF store and a MySQL database. Data access layer gathers RDF triples from semantic web resources. The business layer accesses to the data from data access layer and generates class expression from.

### Pool Party

Pool Party [SB10] is a thesaurus management tool which enriches thesaurus by gathering relevant information from LOD along with text analyzing since Pool Party has natural language processing capabilities. Pool Party represents thesauri in RDF and Simple Knowledge Organization System (SKOS) form. In addition to publishing its data on LOD, Pool Party consumes LOD for enhancing its thesauri. It also provides Personal Information Management tool which allows users to create categories from LOD (e.g. A movie expert can create knowledge model of filmmakers and the countries they lived in).

The possible matchings from DBpedia are returned for each concepts in Pool Party us-



ing DBpedia lookup service. User can select the DBpedia resource which matches the concept and therefore link the URI in thesauri and DBpedia through owl:sameAs links. Pool Party probably uses string matching algorithm.

### Deriving Similarity Graphs from Linked Data

Semantic Similarity Transition(SST) method finds similar herbs in TCM (Traditional Chinese Medicine) linked data set [MCLYP09]. Finding similar herbs to a given herb helps medicine researchers and physicians in making prescription or other research activities.

In order to calculate the similarity between two resources, SST transits similarity in the graph iteratively by following link data in the graph. The main idea of transitivity comes from classic Page Rank algorithm where the importance of each page is calculated by the importance of other pages related to that page.

Since SST calculates the semantic similarity between nodes, two approaches could be used for calculating the semantic similarity: taxonomy-based approach which uses is-a hierarchical relation between concepts, and relationship-based approach which considers the common information between two concepts.

### Interlinking Distributed Social Graph

Modern web users have their own profile over many social network web sites. Integrating these pieces of information distributed over multiple web sites helps us to identify a real world person [Row09].

Three methods of computing similarity help graphs with their linkage process including: 1) Node/Edge Overlap which is used to derive the similarity measure between two graphs. The method uses Jaccard (Definition 22) distance to match two graph by overlapping nodes and edges from them. 2) Node Mapping matches two graphs by mapping all possible nodes from two graphs; therefore, the similarity measure between two graphs is derived by measuring the similarity between every possible combination of object nodes and also between every possible combination of subject nodes. 3) Graph

Reasoning is a low level basic reasoning which matches graphs with each other. In order to compare two graphs, Levenshtein similarity is used for comparing literals from graphs.

### An Approach for Entity Linkage

Name Feature Matching (*nfm*) is a new approach of entity matching in large entity repositories by data linkage [SBO09].

Each entity is represented as a feature consists of name/value pair  $\langle n, v \rangle$ . First, the closeness between two entities is calculated using Levenshtein distance. Afterward, a matrix represents the similarity between every possible combination of two entities' features is created. Finally, Name Feature Score (*nfs*) returns the sum of maximum amount of each row of matrix as a result of similarity between two entities.

### Silk: A Link Discovery Framework

Silk framework discovers semantic relationships between entities through different sources. Silk makes easier for data publishers to set RDF links from their data sources to other data sources on the Web [VBGK09]. First, the stream of data items are generated from Data Source. The data items generated from data source could be clustered in the optional step called Blocking. The Link Generation step dedicates a similarity value to each pair of data items. Then pairs of data items are generated from an internal cache which all data items were written into before. If blocking section exists, cache returns the complete cartesian product of two data sets. If blocking doesn't exist, only data items from the same cluster are compared with each other. Then, for each pair of data items, their link (similarity) is evaluated. In this section some similarity metrics are used in order to compare property values or sets of entities. These similarity metrics are: string comparison techniques, numerical and date similarity measures, concept distances in a taxonomy, and set similarities. These similarity metrics could be combined using aggregation functions: AVG (weighted average of similarity value set), MAX (choose highest similarity value in set), MIN (choose lowest similarity value in set), EUCLID

(Euclidian distance aggregation) and PRODUCT (weighted product of similarity value set). Afterward, the Filtering step removes the links with a lower confidence than the threshold. At last, in output step the generated and filtered links are extracted as a result.

#### LinksB2N: Automatic Data Integration

A UK B2B Marketing framework discovers information overlaps in different RDF data sources by using clustering methods [SCRGDS09].

The LinkB2N algorithm is composed of four steps (Figure 7): 1) **Single Data Source Analysis (SDSA)** which collects graph statistics and creates clusters of similar values (objects) for each RDF predicate, 2) **RDF Predicate selection (RPS)** which finds suitable pairs of RDF predicates to be compared using clusters produced from previous step, 3) **Predicate Value Evaluation (PVE)** which for each pair of predicates calculates confidence ratio by evaluating the equivalences between RDF objects and 4) **Filter of Non-confidence Matching (FNC)** in which iterations of the previous step are applied in order to find more matches between instances.

All proposed tools in this section are summarized and compared in Table 2.1 based on their main criteria.



| Tools                                  | Automatic | Goal   | Matching Techniques  | Input  | Output  | Ontologies |
|--|-----------|--|--|--|---|------------|
| LD-Mapper                              | Automatic | Interlinking music datasets (Jamendo and Musicbrainz), Interlinking personal music collection to corresponding web identifiers (Musicbrainz) | String matching, Similarity propagation using graph matching               | Music datasets, Personal music collection            | owl:sameAs linkset, mo:available linkset(Interlinking personal music collection to Musicbrainz) | Multiple   |
| Interlinking Distributed Social Graphs | Automatic | Identify a real world person by integrating data from Facebook, Myspace and Twitter profiles   | Graph matching, String matching  | Users' profile distributed over many social websites | Provide owl:sameAs links between matching foaf:Person instances in separate graphs              | Multiple   |
| RDF-AI                                 | Semi      | Integrating RDF datasets plus fusion and interlinking (any domain)   | String matching, Word relation matching (synonyms, taxonomical similarity) | Two datasets   | Merged data set or owl:sameAs linkset   | Single     |

| RKB-CRS    | Semi    | Managing URI synonymy problems for publications  | String matching  | Multiple publication sources         | owl:sameAs linkset       | Multiple |
|------------|---------|--|--|--------------------------------------|--------------------------|----------|
| Silk       | Semi    | A link discovery framework (any domain)  | String matching, Similarity metrics (numerical, dates, concept distance, sets) | Links specification alignment method | Alignment format linkset | Single   |
| ODD-Linker | Semi    | Link discovery from relational data (any SQL access)   | synonym, hyponym and string matching, link clause expressions                  | SQL database                         | Linkset                  | Single   |
| LinksB2N   | Semi(?) | Integrating data from different data sets in domain of business to business (B2B) marketing analysis | Clustering, String matching  | Two data sets                        | Linkset                  | Multiple |

|   |           |   |   |   |                                    |          |
|---|-----------|---|---|---|------------------------------------|----------|
| Deriving Similarity from Graphs Linked Data | Automatic | Transitive Semantic Similarity (SST)<br>method in order to find similar herbs in Traditional Chinese Medicine | Transitive semantic similarity, i.e, word relation matching (synonyms, taxonomical similarity), String matching | Two datasets                                      | Linkset                            | Single   |
| Entity Linkage                              | Semi(?)   | A novel, proof-of-concept approach for entity matching in a large entity repository                           | String matching   | Two data sets                                     | Linkset                            | Multiple |
| Knofuss                                     | Semi      | Handling instance coreferencing   | String matching, Adaptive learning  | Source and target knowledge base, fusion ontology | Alignment format of merged dataset | Multiple |

|            |      |   |  |                               |                                       |                                       |          |
|------------|------|---|--|-------------------------------|---------------------------------------|---------------------------------------|----------|
| Guess What | Semi | Use of URIs from DBpedia, Freebase and OpenCyc  | Natural language mining, guessing and evaluation | lan- process- Graph and Human | RDF triples from Semantics web source | could be formalized as OWL            | n/a      |
| Pool Party | Semi | Web base thesaurus management system Using SKOS | probably string matching                         |                               | RDF triples from LOD                  | RDF triples with owl:sameAs relations | Multiple |

Table 2.1: Tool comparison

### 2.2.2 Finding Similarity between RDF Individuals Using FCA

With rapid growth in the availability of data on the web, analyzing the Web of Data becomes ever more interesting. Linked data on the web is represented in RDF model. As mentioned in the previous chapter, RDF represents the information on the web as directed graphs. In RDF graphs, each node is an object whereas arcs are relations between objects. Therefore, RDF graphs or simply graphs constitute a simple yet powerful way to represent conceptual description of data of web sources. Representation of data on the web as such standard format makes it easier for exploiting, managing, etc.

Building concept lattice from RDF data can construct an ontology model. One way to do this is to view the set of RDF statements as a directed labeled graph. Each RDF statement or combination of RDF statements builds sub graphs of a entire RDF graph. Afterwards, based on common predicate-object paths, resources are clustered into concepts in lattices to construct the desired conceptual model.

Delteil et.al. represented a method for extracting knowledge from Web of documents by learning new concepts from RDF graphs [DFD02]. To build a concept lattice, authors have built a concept hierarchy (lattice). The extension contains a set of resources (objects) and intension contains a set of descriptions (attributes) shared by those resources. The resource description is defined as description of a specific length of a resource, i.e., the largest connected sub graph which contains all possible paths of the specific length started from and ended to the resource. The full definition could be found in [DFD02]. First, the lattice (or concept hierarchy) is built of resources' descriptions with length one, i.e., concepts of lattice have intensions of length 1. Then, concept hierarchy is completed incrementally by incrementing the length of resource description till we reach the complete concept hierarchy. Moreover, for the resources without any name we would consider their classes from RDFS level, i.e., adding type path and including their classes while building the intensions of each concept in lattice.

The methodology for building concept lattice from RDF data we use in this thesis is different from above-mentioned technique. First, we only consider instance level as an input

for building the lattice; then, the ontology model or RDFS (schema level) are built based on the information in lattice. The built lattice includes concepts which are constructed based on triples in RDF data. Each concept in lattice includes extension containing a set of resources (objects) and intension containing a set of predicates (attributes) shared by those resources in RDF statements. More information of the methodology with an example is given later in chapter 3.

### 2.3 FCA and Semantic Web Applications

Some ontology-related technologies such as ontology alignment (ontology matching), ontology learning and engineering can be done by knowing the conceptual structure of ontologies. FCA helps those ontology technologies for discovering patterns, regularities, etc., in ontologies. Besides, a large amount of applications have been carried out on the usage of FCA along with Semantic Web tasks [KL12]. All related applications are discussed in [Zha07]. Besides, FCA has been applied to many other Semantic Web tasks such as querying, visualization and so on [KL12].

FCA is also useful in information retrieval applications using Semantic Web. The retrieved results from structured data on the web can be improved by the usage of FCA. In the following, some of those applications related to RDF are explained. d'Aquin et.al. introduced a method to extract relevant questions on a input RDF dataset using FCA [AM11]. The method transforms the hierarchy of meaningful sets (concepts containing entities) into natural language questions. The sets of entities represent the clauses of generated questions. In [Fer10], authors provided a navigation mechanism for RDF graphs using FCA, i.e., accessing concepts through SPARQL-like queries.

The other work related to information retrieval from web of data is found in [ROH05] in which Topia answers questions posed by users. Therefore, in return of users' questions, the system converts the structural data in a form of RDF to hierarchical structure of documents which can be easily analyzed by users. In order to build a hierarchical structure, RDF properties of annotations are considered as attributes of their subjects.

Moreover, if input data is also provided with schema level -RDFS- `rdfs:subclass` could help us for identifying more groups, i.e., better hierarchical structure of documents. FCA is also used to build ontology from scientific corpus by mapping a concept lattice to a formal ontology [JN07]. In [MA13], authors used concept lattices to reveal hidden semantics in the content of query answers. By adding a formal concept layer to the Semantic Web, the exploration of LOD datasets is possible. This also supports query refinement, data cleaning, concept clustering, and more [KL11].

## 2.4 Summary

Extracting useful information from web of data is always a big concern among Semantic Web researchers. This information could be the semantic similarity that exists between individuals in data. The extracted information could be used to construct a concept lattice for other usages. In this chapter, after introducing some similarity measurements which are used for discovering similarity between entities of ontologies, we discussed the study of similarity in LOD tools. Further we talked about the usage of FCA in finding similar RDF individuals, i.e., converting RDF to RDFS. Finally, the usage of FCA in different semantic web applications is described.





## CHAPTER III

### METHODOLOGY AND IMPLEMENTATION

Chapter 2 introduced works related to the usage of FCA in RDF applications as well as the study of similarity in LOD tools. FCA as a mathematical tool represents ontologies in concept lattices. The approach proposed in this thesis is to find similar RDF individuals based on their common properties using FCA. Then, an FCA tool creates a lattice that contains concepts while each concept includes similar individuals. Finally, each concept is assigned a symbolic named according to the names of its individuals. To that end, we use DBpedia.

This chapter presents our research methodology with a small RDF music dataset example. The chapter also covers the implementation of the methodology including the introduction to Java platforms and APIs used for developing our product.

#### 3.1 Approach

A discussion in this section covers the methodology used to extract concepts from RDF dataset and generate an RDFS graph from them. To that end, the approach includes three steps: 1) Converting RDF to FCA input: In this step, the resources and their properties are extracted from RDF data. The extracted information is used to construct the formal context table; then, the FCA tool converts the formal context into a lattice. 2) Converting FCA output to RDFS: The step converts the concept lattice into an RDFS graph according to the rules described later. 3) Choosing plausible names

for RDFS classes: The most important step is to choose appropriate names for classes in the resulting RDFS graph. To do this, the types of objects of each node are extracted from DBpedia. A proper name for the node is selected from the intersection between the types of all objects belong to the node.

In the following, each step is described in detail.

### 3.1.1 Converting RDF to FCA Input

Each concept in the concept lattice contains resources (called objects in FCA) with common properties (called attributes in FCA). In the following, an example of an RDF music store is given. This example includes partial information about four music bands shown in Figure 3.1.

Table 3.1 shows resources (objects in binary table) as well as properties (attributes in binary table) that belong to each resource. For ease of reading, the abbreviation has been applied for each resource's name, e.g., *897* stands for URI <http://www.music.fake/band/897>.

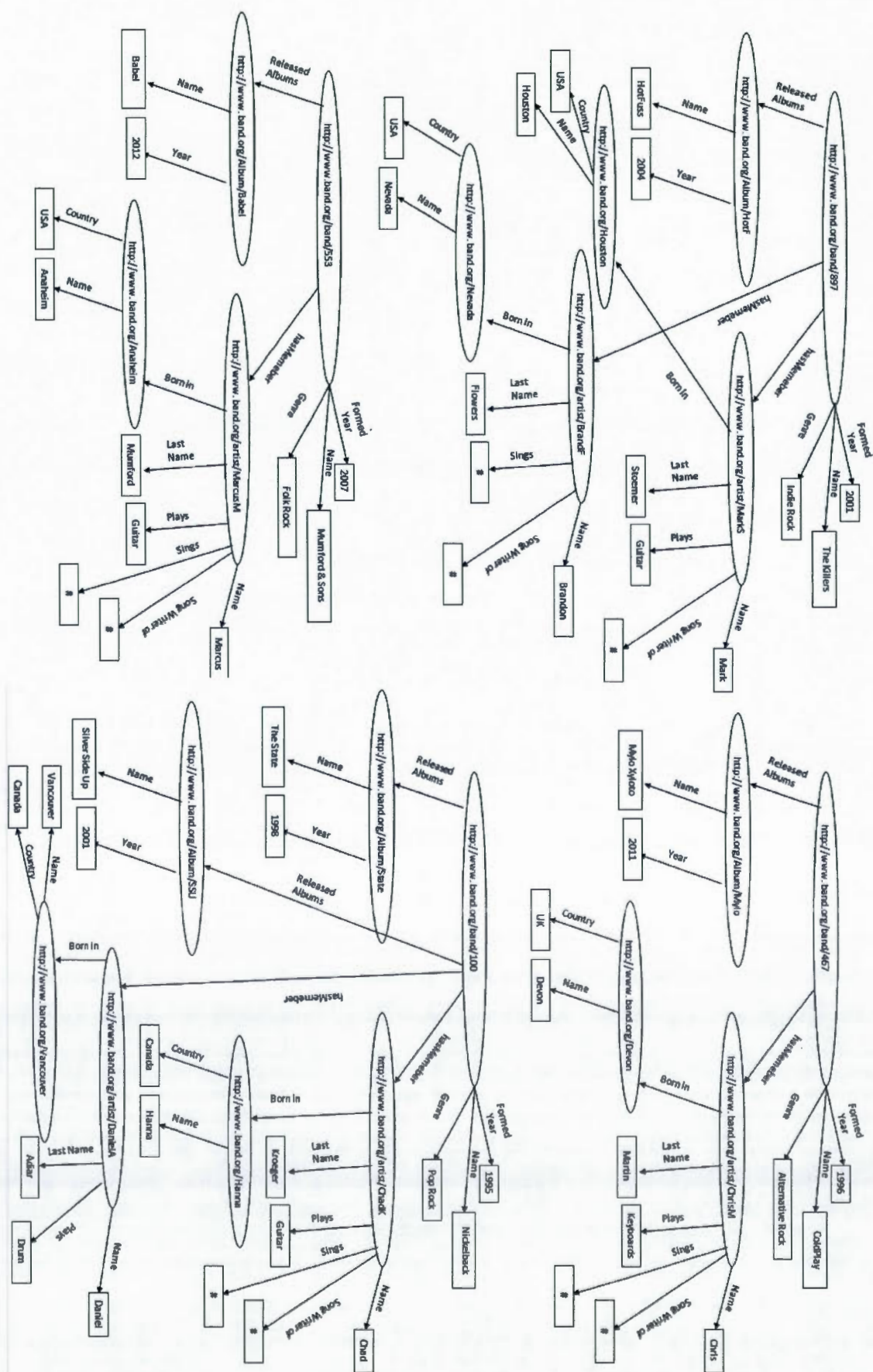
The lattice of the aforementioned RDF example is drawn as Figure 3.2 (Here, the lattice

| A          | B        | C          | D     | E    | F        | G    | H      | I     | J          | K       | L     | M          |
|------------|----------|------------|-------|------|----------|------|--------|-------|------------|---------|-------|------------|
| Default Na | Released | Has Member | Genre | Name | LastName | Year | BornIn | Plays | SongWriter | Country | Sings | FormedYear |
| 897        | X        | X          | X     | X    | 0        | 0    | 0      | 0     | 0          | 0       | 0     | X          |
| HoltF      | 0        | 0          | 0     | X    | 0        | 0    | 0      | 0     | 0          | 0       | 0     | 0          |
| MarkS      | 0        | 0          | 0     | X    | X        | 0    | X      | X     | X          | 0       | 0     | 0          |
| BrandF     | 0        | 0          | 0     | X    | X        | 0    | 0      | 0     | 0          | 0       | X     | 0          |
| Huston     | 0        | 0          | 0     | X    | 0        | 0    | 0      | 0     | X          | X       | 0     | 0          |
| Nevada     | 0        | 0          | 0     | X    | 0        | 0    | 0      | 0     | 0          | X       | 0     | 0          |
| 467        | X        | X          | X     | X    | 0        | 0    | 0      | 0     | 0          | 0       | 0     | X          |
| Mito       | 0        | 0          | 0     | X    | 0        | X    | 0      | 0     | 0          | 0       | 0     | 0          |
| ChrisM     | 0        | 0          | 0     | X    | X        | 0    | X      | X     | X          | 0       | X     | 0          |
| Devon      | 0        | 0          | 0     | X    | 0        | 0    | 0      | 0     | 0          | X       | 0     | 0          |
| 110        | X        | X          | X     | X    | 0        | 0    | 0      | 0     | 0          | 0       | 0     | X          |
| State      | 0        | 0          | 0     | X    | 0        | X    | 0      | 0     | 0          | 0       | 0     | 0          |
| SSU        | 0        | 0          | 0     | X    | 0        | X    | 0      | 0     | 0          | 0       | 0     | 0          |
| Chook      | 0        | 0          | 0     | X    | X        | 0    | X      | X     | X          | 0       | X     | 0          |
| DanielA    | 0        | 0          | 0     | X    | 0        | 0    | X      | X     | 0          | 0       | 0     | 0          |
| Hanna      | 0        | 0          | 0     | X    | 0        | 0    | 0      | 0     | 0          | X       | 0     | 0          |
| Vancouver  | 0        | 0          | 0     | X    | 0        | 0    | 0      | 0     | 0          | X       | 0     | 0          |
| 553        | X        | X          | X     | X    | 0        | 0    | 0      | 0     | 0          | 0       | 0     | X          |
| Baker      | 0        | 0          | 0     | X    | 0        | X    | 0      | 0     | 0          | 0       | 0     | 0          |
| MarcusM    | 0        | 0          | 0     | X    | X        | 0    | X      | X     | X          | 0       | X     | 0          |
| Anaheim    | 0        | 0          | 0     | X    | 0        | 0    | 0      | 0     | 0          | X       | 0     | 0          |

Table 3.1: Binary relation table of music dataset

is drawn by **Lattice Miner 1.4**<sup>1</sup> tool): The attributes and objects are respectively

<sup>1</sup><http://sourceforge.net/projects/lattice-miner/>



written in blue and red colors.

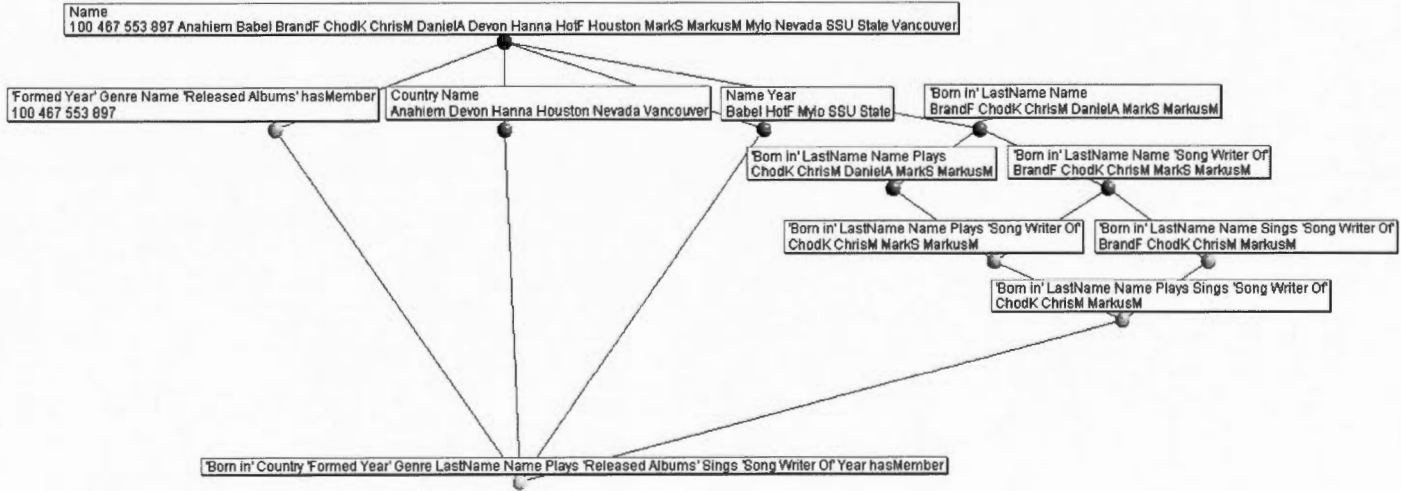


Figure 3.2: Lattice of music dataset

The lattice is also presented as a reduced labeling diagram depicted in Figure 3.3. Each concept of lattice contains objects with the same attributes. For example, *MarkS*, *ChrisM*, *BrandF*, *ChodK* and *MarcusM* as members all write song lyrics for their own bands. Therefore, they belong to the same concept but toward bottom they split into two different concepts. *ChrisM*, *ChodK* and *MarcusM* belong to both concepts, since they can play instruments and also sing while *BrandF* only sings and *MarkS* only plays an instrument.

### 3.1.2 Converting FCA Output to RDFS

The translation of the above lattice to RDFS graph is shown in Figure 3.4. (The number in each RDFS node is the number of corresponding concept in the reduced labeling diagram). Three rules have been applied to the lattice in order to create RDFS graph.

**Node Rule:** All nodes except the one including all objects, i.e., node number 0 in Figure 3.2, as well as the one which doesn't include any object, i.e., node number 11 in the same figure should be used as classes to create the RDFS graph.

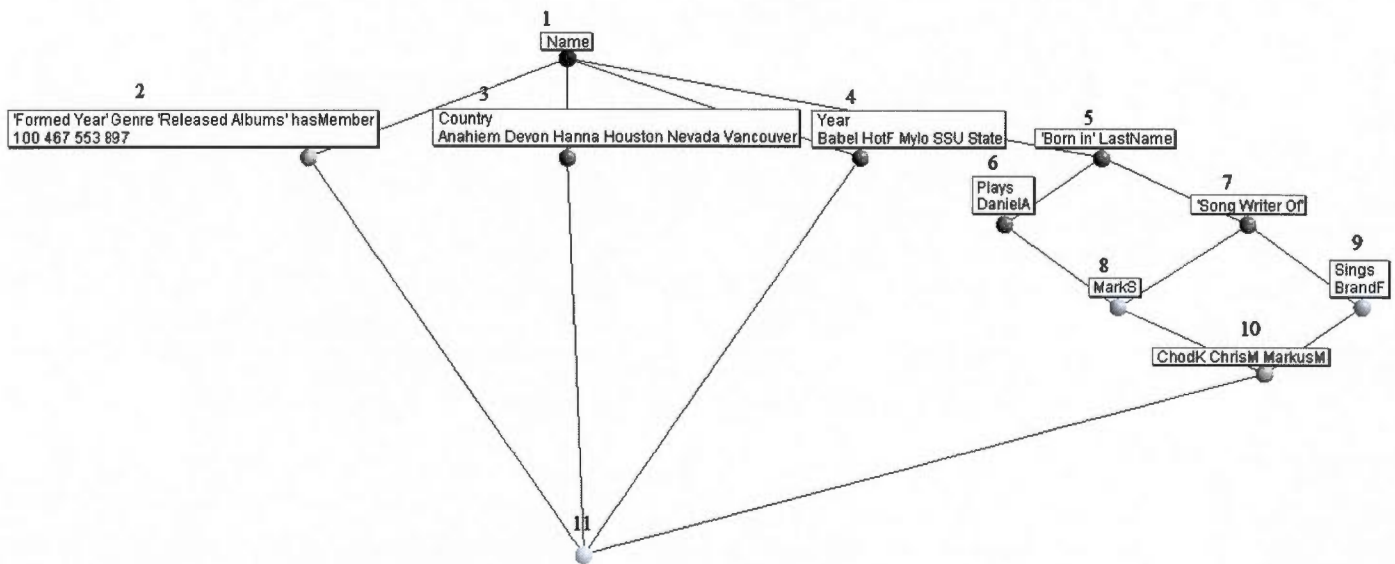


Figure 3.3: Reduced labeling diagram of music dataset lattice

**rdfs:subclass Rule:** `rdfs:subclass` relation between classes is built according to the hierarchy between concepts in lattice. Notice that even though the last step of methodology called *choosing plausible names for RDFS classes* explains how to choose name for each node, for easiness of analyzing predicate rule, we considered proper names for some of the nodes in advance, i.e. node number 2,3,4,5,6,7 and 9 are respectively called *Musician*, *Country*, *Album*, *Artist*, *Band*, *SongWriter* and *Singer*.

**Predicate Rule:** For relating classes with properties, we shall survey all properties related to those classes. Among all of the properties that belong to a class those who go to resources in the RDF graph can be related to the other classes in RDFS graph. For example, class *Musician* has four properties *Name*, *LastName*, *Born in* and *Plays*. Among all properties of class *Musician*, only *Born in* property goes to resource *Country* in the RDF graph but other properties go to literals. Therefore, *Born in* predicate is considered for relating two classes, i.e., *Born in* should relate class *Musician* to class *Country*. Besides, the other properties (*Name*, *LastName* and *Plays*) which go to literals cannot be ignored while building RDFS graph. Therefore, their data types are taken into account and based on their respective types in RDF file, they are related accordingly.



Below, a part of the RDF file for music dataset is illustrated. As it is seen, the predicates of resource *897* which go to literals have explicit datatypes in the RDF file:

```
<rdf:Description rdf:about="http://bands.org/musicband#897">
  <band:ReleasedAlbum rdf:resource='http://bands.org/musicband#HotF' />
  <band:HasMember rdf:resource='http://bands.org/musicband#MarkS' />
  <band:HasMember rdf:resource='http://bands.org/musicband#BrandF' />
  <band:Genre rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Indie Rock</band:Genre>
  <band:Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The Killers</band:Name>
  <band:FormedYear rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2001</band:FormedYear>
</rdf:Description>
```

Moreover, for each property whose range or domain is related to a class, we should remove the duplicated relations to those classes where their super classes have the same relations. For example, *Born in* has all classes *Artist*, *Musician*, *SongWriter* and *Singer* as its domain. Since *Artist* is superclass of all three classes *Musician*, *SongWriter* and *Singer*, it would be enough for *Born in* property to only have *Artist* as its domain.

In Figure 3.4, all classes with their properties and all relations are declared in an RDFS graph.

Other properties which go to datatypes are shown in Figure 3.5 (Notice that the node numbers in this figure are different from node numbers in the previous figure but locations are the same). **RDF Gravity 1.0**<sup>1</sup> tool is used to show the representation of music dataset's RDFS graph.

### 3.1.3 Choosing Plausible Names for RDFS Classes Using DBpedia

A Key part of the algorithm consists of naming classes built with our tool. In addition to the previous example, the examples in this section also contain the Russia dataset used in chapter 4.

In order to name the nodes in the RDFS graph, for each node we shall survey common types (objects of `rdf:type` predicate in DBpedia) of resources of DBpedia which match

<sup>1</sup><http://semweb.salzburgresearch.at/apps/rdf-gravity/index.html>

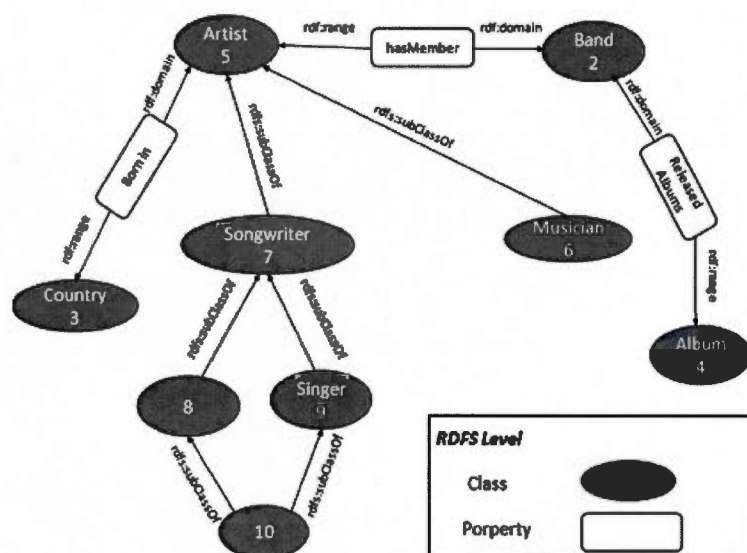


Figure 3.4: RDFS graph of music dataset

with the resources in the node. The purpose is to find the most plausible name for a node; therefore, among all the common types extracted from DBpedia for the resources of one node, the more specific one is selected. For example, for a node with *Lake Onega* and *Neva River* as its members, among the objects of `rdf:type` predicate with `dbpedia-owl` prefix of those resources, *Place*, *BodyOfWater* and *NaturalPlace* are in common. Since the second one (*BodyOfWater*) is more specific than others, we choose *BodyOfWater* which gives us better name for the node (Figure 3.6).

Moreover, some resources may use their other known names in RDF input data, e.g., *Neva* is used instead of *Neva\_River* in RDF data. Therefore, query results from DBpedia may not be satisfactory. The solution we propose is to look for all objects of predicate `dbpedia-owl:wikiPageRedirects` in DBpedia for each resource.

As it is shown in Figure 3.7, the DBpedia page for *Neva River* shows that `dbpedia:Neva` is one of the `dbpedia-owl:wikiPageRedirects` values of `http://dbpedia.org/page/Neva_River`. Therefore, by sending `http://en.wikipedia.org/wiki/Neva` in the query we can ob-

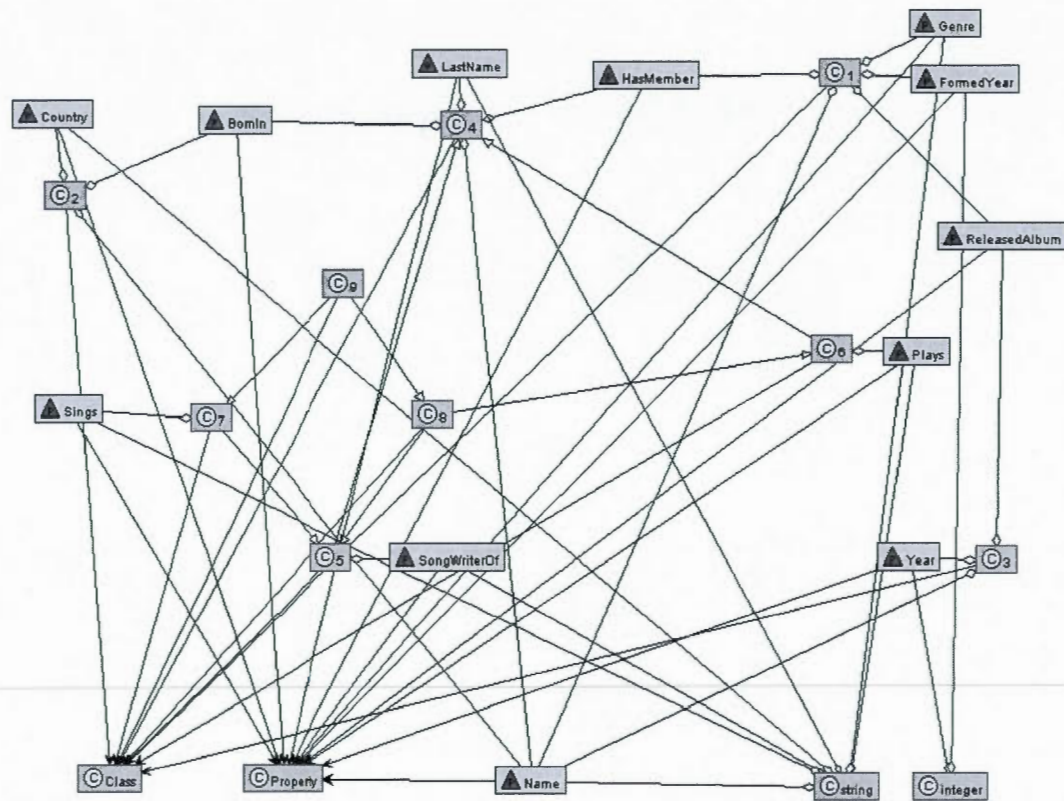


Figure 3.5: RDF Gravity representation of music dataset's RDFS graph

| Lake Onega                 | Neva River                 |
|----------------------------|----------------------------|
| ■ dbpedia-owl:Place        | ■ dbpedia-owl:Place        |
| ■ dbpedia-owl:BodyOfWater  | ■ dbpedia-owl:River        |
| ■ dbpedia-owl:Lake         | ■ dbpedia-owl:BodyOfWater  |
| ■ dbpedia-owl:NaturalPlace | ■ dbpedia-owl:NaturalPlace |

Figure 3.6: Objects of `rdf:type` predicate with `dbpedia-owl` prefix of *Lake Onega* and *Neva River* in DBpedia



tain the results for [http://en.wikipedia.org/wiki/Neva\\_River](http://en.wikipedia.org/wiki/Neva_River).

```
is dbpedia-owl:wikiPageRedirects of dbpedia:Neva
■ dbpedia:Neva_river
■ dbpedia:River_Neva
■ dbpedia:Большая_Невка
■ dbpedia:The_Neva
```

Figure 3.7: Objects of `dbpedia-owl:Wikipagesdirect` predicate for *Neva River* in DBpedia

Further, if all resources of one node have `dbpedia-owl:Person` as an object of their `rdf:type` predicate in DBpedia, considering `dc:description` predicate leads to more appropriate name for the node since objects of `dc:description` predicate contains more specific data. Therefore, we shall look for the objects corresponded to `dc:description` predicate in order to choose a suitable name for the node only by applying a few simple Natural Language Processing (NLP) methods.

For example, assume *Rihanna* and *John Bottomley* as resources belong to a node. Both of them have `dbpedia-owl:Person` as object of their `rdf:type` predicate in DBpedia, i.e., both are of type `Person`. Therefore, the objects of their `dc:descriptions` predicate should be extracted and considered instead of `rdf:type` predicate. The object of `dc:descriptions` predicate for *Rihanna* is “*Singer, songwriter*” and for *John Bottomley* is “*Canadian singer and songwriter*”.

Notice that “*and*” and “*,*” separate phrases inside the objects of `dc:description` predicate. Moreover, in the phrases composed of two or more words, usually the last word is noun and other words are adjectives, i.e., only the last word of every phrase should be considered for choosing a name for that node, e.g., the last word of phrase “*Argentine singer*” would be “*singer*”.

Therefore, with a simple NLP technique, “*Singer, songwriter*” which belongs to *Rihanna* can be converted to a list with two elements “*Singer*” and “*Songwriter*” by considering of capitalizing the first letter of each word. Moreover, for *John Bottomley*, “*Canadian singer and songwriter*” is converted to a list with two elements “*Singer*” and “*Songwriter*”.

After making list of plausible names for each resource of a class using `dc:description`,

we shall intersect the results for all resources of the node. The intersection between both above-mentioned lists is “*Singer*” and “*Songwriter*”. Hence, the class name would be “*Singer Songwriter*”.

After choosing names for nodes, there may happen to be some nodes with similar names. To reduce the occurrences of nodes with similar names, a technique is applied. For the nodes with the similar name as their parents, the reduced labeling attributes of those nodes should be added to the name of the nodes. The technique is illustrated with more detail with an example in chapter 4.

## 3.2 Implementation

In this section, first the Java platforms and APIs including Jena, RDF API, SPARQL, Galicia and RDF Gravity used for implementation are introduced. To illustrate the process of implementation of the algorithm, each of the steps mentioned in the previous section will be discussed based on their representative usage.

### 3.2.1 Java Frameworks and APIs

#### 3.2.1.1 Jena

Jena<sup>1</sup> is an open source semantic web framework which has been built by the members of the Semantic Web research group at HP labs. After 2009, Jena was released into the open source community to be developed and supported openly.

Jena is a java library which provides the abilities to work with semantic web applications. It provides an API for RDF, RDFS and OWL ontologies which makes it easier for programmers to use them in their applications rather than doing alternative implementations [McB01]. Moreover, Jena has the ability to read, process and write RDF data into/from XML [Arn10], N-triples and Turtle formats. The framework was first built

---

<sup>1</sup><http://jena.sourceforge.net/>

only for RDF type languages and then it added features for using in OWL languages too. Jena also provides SPARQL query engine.

The main components of Jena frameworks are shown in Figure 3.8.

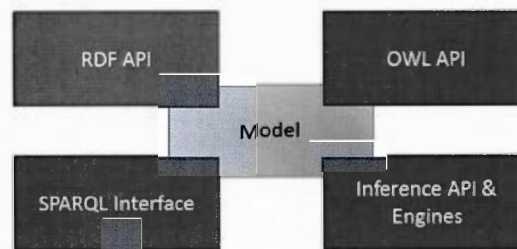


Figure 3.8: Jena framework [Lin10]

The central model interface contains 4 main components including: RDF, OWL API, SPARQL search engine and the Inference API & Engines which is useful for external and built-in reasoners.

#### 3.2.1.1.1 RDF API

RDF API is a Java API which allows the creation and manipulation of RDF graphs. As it is said before, Jena was first developed for data with RDF type languages. All information provided by RDF data is stored in a data structure called model.

RDF uses the standard Jena *model*. The model includes RDF statements and some functions for querying, adding and removing those statements.

Compared to RDF, RDFS has ontological interface; therefore, it uses Jena's *OntModel* which has the ability to create and manipulate models from ontologies in RDFS or OWL. Based on its type, a model can provide methods for querying, manipulating, creating RDF, RDFS and OWL data such as *listProperties*, *listStatements*, *createResource*, *createLiteral*, *createStatement*, *createProperty*, *createClass*, *createOntProperty* and so on.

To have java access to RDF/RDFS models, RDF API has libraries which allow any java program to create and manipulate RDF/RDFS based models easily [McB01].

## 3.2.1.1.2 SPARQL

Querying the accessible data stored in a model is possible by the ARQ search engine for Jena. ARQ provides SPARQL [PS06], an RDF Query language, on RDF type models. SPARQL is a standard query language to be used in RDF type models. SPARQL queries information in a similar way that SQL does but it uses RDF model instead of relational data model, i.e., it uses triple patterns in WHERE clause. A triple pattern is an expression of RDF statement composed of three components: subject, predicate and object. Each component can be a variable, e.g., `?person`, a name added to the default namespace, e.g., `custom:Name` and a value of an attribute belong to an entity or a full URI. Moreover, the third component can be a numeric literal, e.g., `3.14`, a plain literal, e.g., `"Duke"`, a plain literal with a language-tag, e.g., `"Duke"@en`, or a typed literal, e.g., `"123"8sd:int` (`xsd:int`<sup>1</sup> stands for integer datatype in XML Schema).

Figure 3.9 presents an example which return the names of persons who has a pet called "Duke". Three triple patterns have been chained to create the WHERE clause of the query. Namespaces are shortcuts for full URIs and it enhances both readability and writability. Among all four commands used in SPARQL to perform the query (SELECT, CONSTRUCT,

|   |                        |
|---|------------------------|
| <b>PREFIX</b> <code>custom: &lt;http://mySite/MySchema/&gt;</code>  | <b>Namespaces</b>      |
| <b>SELECT</b> <code>?personName</code>  | <b>Output columns</b>  |
| <b>WHERE</b> {  |                        |
| <div style="border: 1px solid black; padding: 5px; display: inline-block;"> <code>?person custom:Name ?personName.</code><br/> <code>?pet custom:Owner ?person;</code><br/> <code>custom:Name 'Duke'.</code> </div> | <b>Triple Patterns</b> |
| }   |                        |

Figure 3.9: SPARQL Example [Szl09]

ASK and DESCRIBE), SELECT is the most common used and it extracts columns of table based on WHERE clause.

<sup>1</sup><http://www.w3.org/TR/xmlschema-2/>

### 3.2.1.2 Galicia

Galicia [VG03] is an open source platform that supports tasks related to FCA such as creating, visualizing, and storing lattices. Being written in java, Galicia<sup>2</sup> can be run on different platforms without changes in its functionality. Moreover, Galicia can be used with both a command line interface (CLI) and a graphic user interface (GUI).

Galicia supports different input types including binary data, lattice, multi-valued context and relational context families. Galicia's name -Galois Lattice-Based Incremental closed Itemset Approach- comes from its incremental data mining algorithms, which are used for mining association rules in transaction databases. It is not our interest in this thesis. Binary relationships between objects and attributes can be stored as Relational Context Family (.rcf) format in Galicia. One can generate an XML format of an RCF context and import into Galicia to create a lattice from it.

Figure 3.10 shows a loaded data with binary relationship within context family editor of Galicia as well as the output lattice of data.

### 3.2.1.3 RDF Gravity

RDF Gravity is an open source tool for visualizing directed graphs in RDF/OWL format [GW06, DK07]. RDF Gravity has pre-built functionality that allows the user to filter out and choose to visualize the desired part of a graph. However, large data are not easily readable by RDF Gravity.

RDF Gravity's main features include: graph visualization, global and local filters (enabling specific views on a graph), full text search/ RDQL queries and visualizing multiple RDF files.

RDF Gravity is implemented on top of the JUNG Graph API and the Jena framework. A screen shot of RDF Gravity tool is shown in Figure 3.11.

In Table 3.2, the notations used to display RDF graphs in RDF Gravity are shown with

<sup>2</sup><http://www.iro.umontreal.ca/~galicia/>

<sup>1</sup><http://www.softpedia.com/>



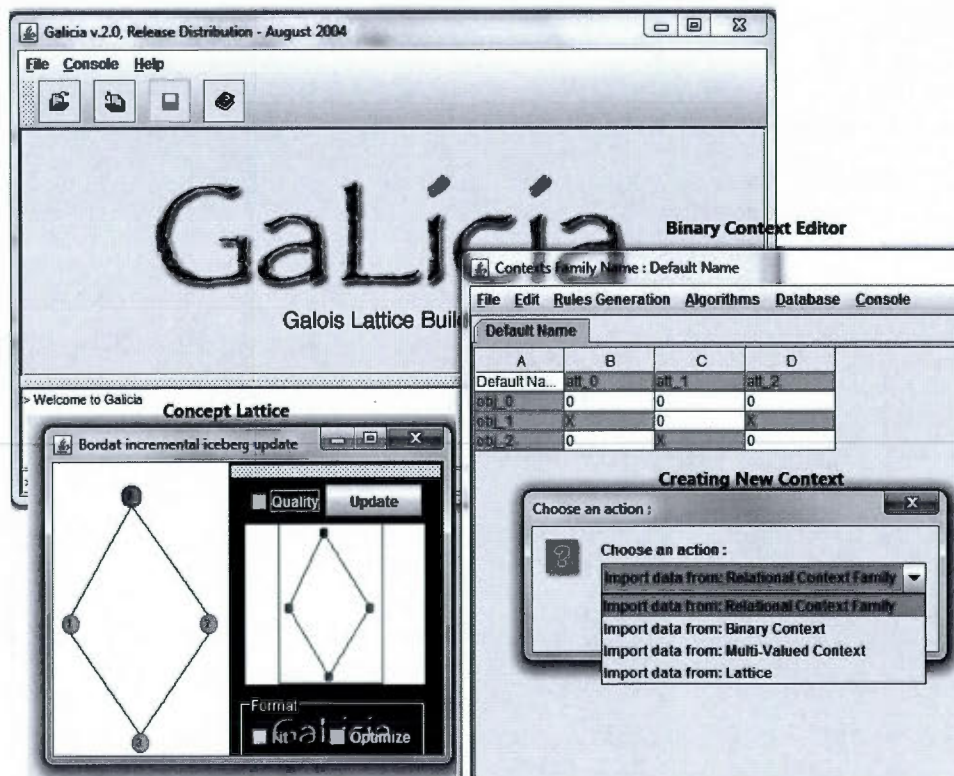
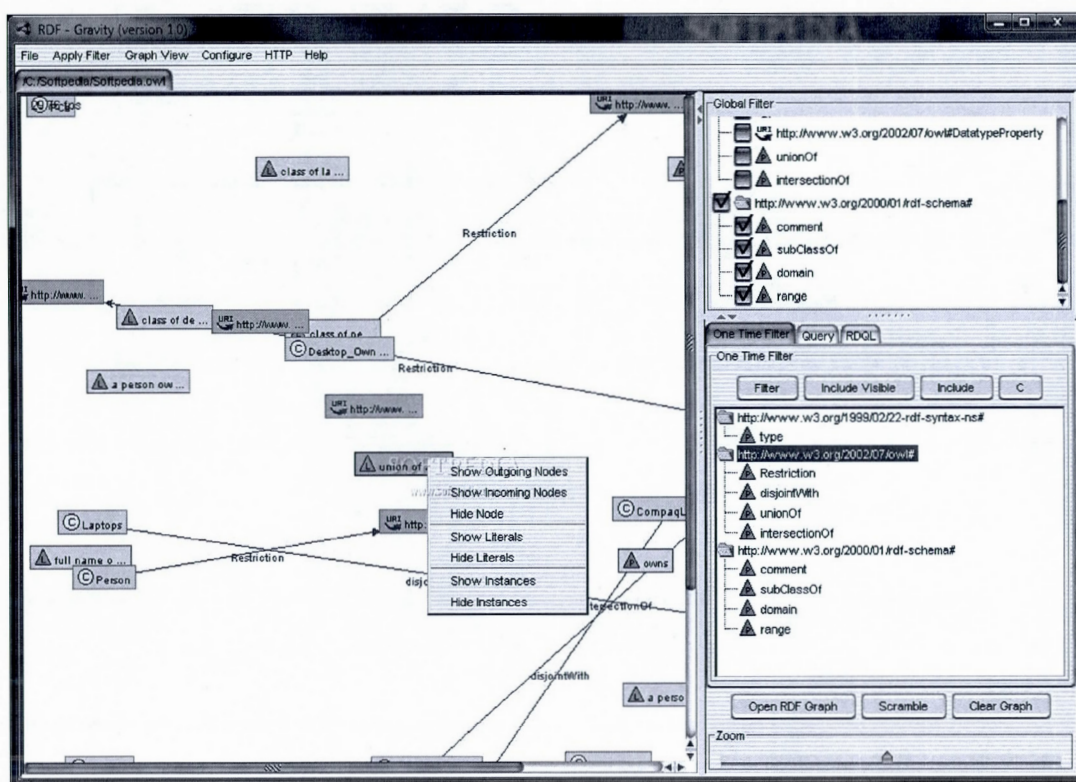


Figure 3.10: Galicia v.2 beta view

Figure 3.11: RDF Gravity View<sup>1</sup>







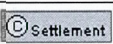
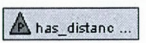
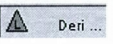


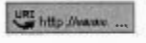
|  |   |
|--|---|
|  domain     | Violet edges refer to <code>rdfs:domain</code> predicate  |
|  range      | Green edges refer to <code>rdfs:Range</code> predicate  |
|  subClassOf | Blue edges refer to <code>rdfs:subClassOf</code> predicate  |
|  type       | Black edges refer to <code>rdf:type</code> predicate.   |
|             | Refers to concepts or classes.<br>Here, "Settlement" is a concept.  |
|             | Refers to Properties.<br>Here, "has_distance" is a property.  |
|             | Refers to a literal value (string, integer) etc.<br>Here, "Deri" is a Literal.  |
|             | Refers to Anonymous nodes   |
|             | Refers to instances. Any anonymous node which has <code>rdf:type</code> predicate associated to a class is also shown as an instance. |
|             | Refers to URI strings which cannot be identified as any of the above items  |

Table 3.2: RDF Gravity notations

brief descriptions.

### 3.2.2 Generating RDFS from RDF data

#### 3.2.2.1 Step One: Converting .rdf to .rcf.xml

The first step converts the RDF input data into the XML format of a RCF file (.rcf.xml). We chose Galicia as FCA tool, since Galicia can easily read and process XML data and transform .rcf.xml to lattice. First, relations between resources and properties in RDF data are determined and generated as <resource, property> pairs. Later, these relations will be described in a binary relationship table by Relational Context Family (.rcf) format.

Reading and extracting properties and resources from RDF data is performed by using Jena's RDF API (introduced in section 3.2.1.1).

A list of pairs of resources and their corresponding properties is used to create `.rcf.xml` file readable by Galicia.

DOMXML parser is a programming API that helps in creating and writing into/from XML files. DOM considers XML documents as tree structures. To do this, first a XML document is created using `DocumentBuilder` class; then, all the XML content are defined. The XML file requires three main child elements called `OBJS`, `ATTS`, `RELS` respectively used for objects, attributes and relations between them. Finally, the `Transformer` class is used to write the entire XML content to a file with `rcf.xml` format.

After creating RCF file in XML format, Galicia is able to construct a lattice from the relation between objects and attributes. We may select Lattice option from the export menu of Galicia while saving the lattice. Galicia saves the lattice in XML format, i.e., `lat.xml`.

### 3.2.2.2 Step Two: Converting `lat.xml` to RDFS

After obtaining XML format of LAT file from Galicia, we read the lattice file in order to create the RDFS file. The Lattice format has four specific tags: `PARENT`, `NOD`, `ATT` and `OBJ`. DOMXML provides functions for parsing and extracting data from XML files. Before creating an RDFS file (file with `.rdf` format but including schema level instead of instance level) from lattice, each node is named properly according to the step three. Usually the *naming class* step comes before the generation of RDFS file, but for the facility of understanding our methodology here, the creation of RDFS file is explained before explaining the naming of the classes of RDFS file.

RDFS file is similar to RDF file but the content contains schema level. The RDFS file uses Jena's *OntModel* for manipulating models from ontologies. It is produced based on three rules including Node rule, `rdfs:subclass` rule and Predicate rule fully described in the previous section.

### 3.2.2.3 Step Three: Naming Classes Using DBpedia

To obtain names for RDFS classes, DBpedia is queried using the SPARQL endpoint. Information in DBpedia could be retrieved online via SPARQL or offline from downloaded the DBpedia dataset. In section 3.1.3, it is mentioned which information of DBpedia needs to be retrieved for naming RDFS classes properly. In our case, the information has been retrieved from online DBpedia.

DBpedia allows users to retrieve data from DBpedia by providing a public SPARQL endpoint at <http://dbpedia.org/>.

Moreover, SNORQL query explorer allows users to have a preview of their results by providing a simpler interface to the DBpedia SPARQL endpoint. Figure 3.12 shows the results of an example in SNORQL.



Figure 3.12: SNORQL

### 3.3 Summary

In this chapter, the approach and implementation of our methodology are covered. The main goal of our study is to extract schema from RDF data. The approach is done by classifying RDF individuals according to the properties they share. Therefore, RDF individuals that share common properties are considered to belong to the same class. Then the classes are named according to their individuals' names. For each individual, the type of individual is obtained by searching its name in DBpedia. Finally, the class name is selected according to one of the common types shared by its individuals in DBpedia. In the next chapter, the experimental results will be presented.



## CHAPTER IV

### EXPERIMENTS AND RESULTS

The results for each of the steps detailed in the previous chapter are presented in this chapter. The effectiveness of our algorithm in terms of Precision, Recall and F-Measure is also analyzed and discussed.

#### 4.1 Dataset

Both validity and diversity of the dataset are important factors in order to achieve the best performance from our tool. Besides, the dataset must be large enough to show the efficiency of the tool. Therefore, finding an interesting and suitable dataset appears to be a challenging issue. The dataset used for the experiment contains a high variety of information about Russia and is large enough to prove the efficiency of our tool. The RDF dataset used in our experiment is a set of 1613 triples about Russia. The information includes data about Russia's cultural (theaters, museum, galleries, etc.) and natural (rivers, lakes, parks, etc.) sites. It also includes data about famous people in Russia, entertainment and other features.

The Russia dataset contains both instance and schema level of RDF model. Since our goal is to detect RDF schema level only by knowing the RDF instance level, we only considered the instance level and ignore the information at the schema level. The schema level of dataset is used to evaluate our results.

## 4.2 Results

### 4.2.1 Binary Relation Table

The data extracted from the RDF file are used to create a binary relation table (also called a formal context) for the FCA tool. The number of objects and attributes of our dataset are respectively 92 and 47. By exploring the XML version of Russia's RCF file, we also found that 256 relations exist between those objects and attributes. In the next section, a concept lattice is constructed from the binary table using Galicia.

### 4.2.2 Concept Lattice

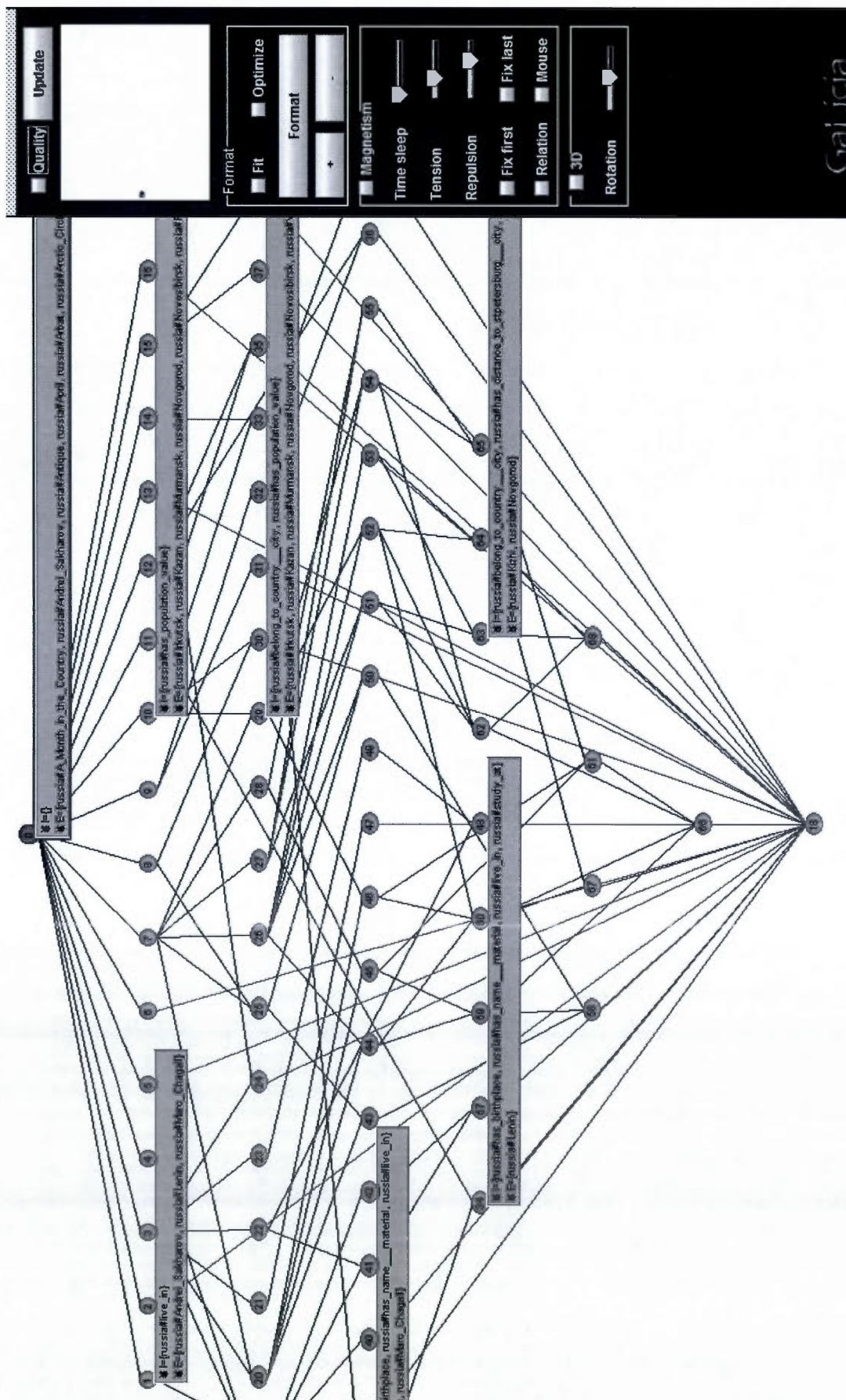
The lattice of the dataset generated by Galicia is shown in Figure 4.1. Each node's content can appear by clicking right on the node in the Galicia software. The lack of space has prevented us from showing full information of all nodes in the image. The concept lattice in the dataset includes 69 nodes where the uppermost node contains all objects and the lower most contains no objects. Therefore, in reference to the previous chapter, all nodes except the upper most and the lower most are used to build the RDFS graph.

Even though the concepts are created from resources using FCA, it should be considered that based on the open world assumption (OWA), creating a class even for a single resource may be necessary since the data is potentially incomplete [CBHS05].

### 4.2.3 RDFS Graph

Extracting schema from the dataset and translating it to RDFS are done by following the steps described in the previous chapter. Figure 4.2 shows the RDFS graph extracted from the dataset. We used DBpedia to name the nodes according to the common information their objects share in DBpedia. Therefore, there is a reduction in the number of





nodes compared to the concept lattice due to the similar names applied to some nodes. In order to tune this reduction, the aforementioned technique (refer to page 63 in the text) has been applied for naming similar nodes differently as much as possible.

The technique is used for nodes with the similar name as their parents. The solution is to add reduced labeling attributes of the node to its current name. For example, node 7 and node 23 both obtained name “*PopulatedPlace*”. Since node 23 has node 7 as its parent, we shall add one of node 7’s attributes to its name which node 23 doesn’t have, in other words, adding the reduced labeling attribute of node 7 (*distance\_unit\_\_region*) to its name. Finally, node 7 is named as “*PopulatedPlace with distance\_unit\_\_region*”.

We used the RDF Gravity version 1.0 to visualize the RDFS graph of our results from the dataset. For more details on the notations of RDF Gravity refer to section 3.2.1.3 of chapter 3.

---

### 4.3 Discussion of the Experiments

The original hypothesis of this methodology is to convert RDF to RDFS data by taking advantage of FCA for clustering RDF individuals according to the common properties they share. The resulting RDFS graph should provide a proper classification of individuals by considering the properties they have in the RDF data. Further, the generated RDFS graph should have classes with proper names.

The experiment was performed on a Windows 7 Home Premium operating system running on Intel Duo Core 2 2.40GHz PC with 3GB of RAM. Details on the dataset used in our experiment are already given in section 4.1. In the following, quantitative results are provided in terms of Precision, Recall and F-Measure.

As mentioned before, Russia dataset is also provided with schema level. To evaluate our tool against the dataset, we only consider the relevant classes in the dataset.

In this experiment, the relevant classes stands for the classes which have instances and at least one of their instances is used to construct the concept lattice, i.e., instances with

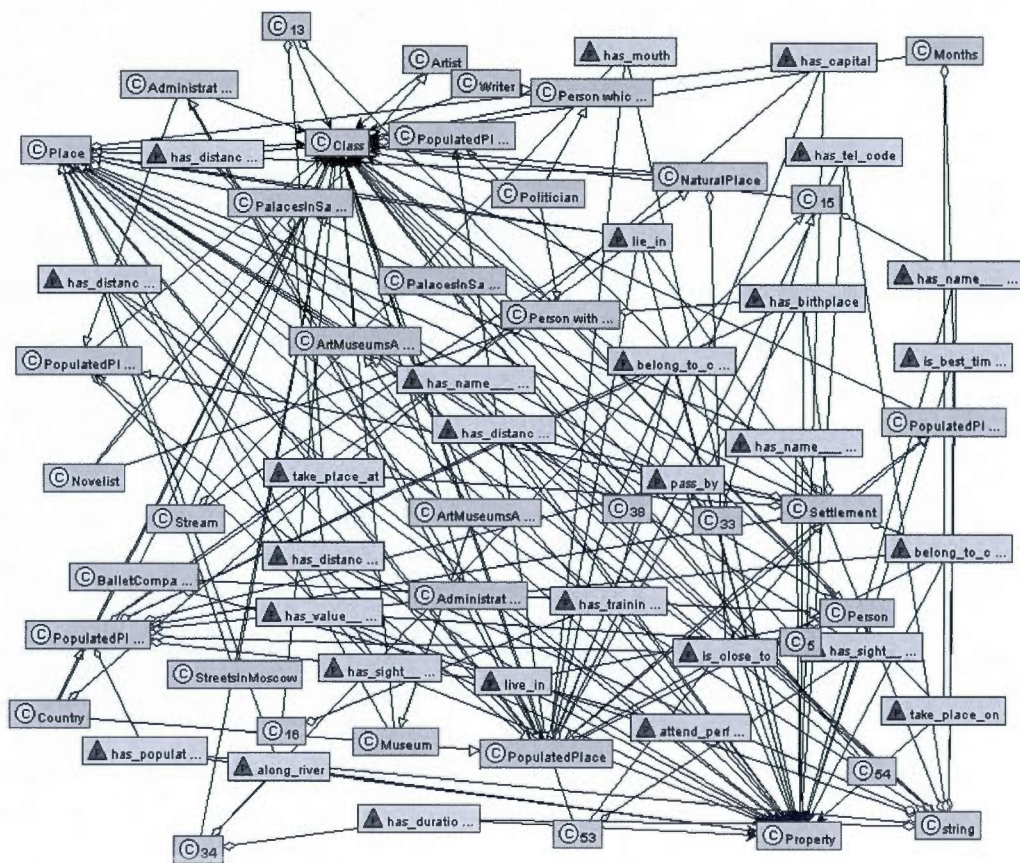


Figure 4.2: Full RDFS graph of Russia dataset by RDF Gravity



properties.

Recall is a measure which calculates the number of the relevant classes extracted from the dataset and it is calculated as follows:

$$Recall = \frac{relevant\ classes \cap retri\text{v}ed\ classes}{relevant\ classes}$$

The numerator is the intersection between the relevant and retrieved classes. The number of relevant classes in the dataset is 35 and the number of classes which are extracted to construct the resulted RDFS graph is 36. Among all of the extracted classes only 27 are relevant and used to construct the final RDFS graph since some of them combined due to the similar names. The criteria used to evaluate the retrieved classes is based on the classes names; therefore, it is important to have classes with different names. To resolve this issue, we should consider the occurrence of classes with same name only once. Therefore, the numerator of the fraction is 27 and the denominator is 35.

Precision shows the number of extracted classes from the dataset which are relevant and it is calculated as follows:

$$Precision = \frac{relevant\ classes \cap retri\text{v}ed\ classes}{retri\text{v}ed\ classes}$$

The numerator of the fraction is 27 and the denominator is 36.

F-Measure is a measure of accuracy which takes both Recall and Precision metrics into account ( $\beta$  is one which shows the equal weights for Recall and Precision). It is calculated as follows:

$$F\text{-Measure} = \frac{(\beta^2 + 1) \times Precision \times Recall}{(\beta^2 \times Precision) + Recall}$$

As mentioned before, by applying the technique (refer to page 63 of the text) the number of relevant classes appearing in the resulted RDFS graph is changed from 17 to 27.

The below shows the amount of Precision, Recall and F-Measure before and after applying the mentioned technique. Table shows a significant improvement in the amount of Precision, Recall and F-Measure after applying the technique.

|                               | Precision | Recall | F-Measure |
|-------------------------------|-----------|--------|-----------|
| After applying the technique  | 0.75      | 0.771  | 0.763     |
| Before applying the technique | 0.472     | 0.485  | 0.478     |

Table 4.1: Measurement table

In order to demonstrate the results more clearly, some filtration are applied to Figure 4.2 by using RDF Gravity's filter feature (represented in Figure 4.3.). All the generated classes with their complete names in the RDFS graph are shown in the figure. The names have been applied according to the common information extracted from DBpedia for the objects of each node. Some of the names are added by applying the technique described in section 4.2.3.

The classes generated from the RDF dataset to construct the RDFS graph are:

*Person, Months, Place, PopulatedPlace, PopulatedPlace with population\_value, Person which study\_at, Artist, Settlement, NaturalPlace, ArtMuseumsAndGalleriesInRussia, PalacesInSaintPetersburg, PopulatedPlace with sight\_\_city, PopulatedPlace with distance\_unit\_\_region, PopulatedPlace with distance\_to\_moscow\_\_city, BalletCompaniesInRussia, Person with birthplace, Country, Writer, Stream, PalacesInSaintPetersburg with position\_to\_stpetersburg \_\_palace, StreetsInMoscow, Museum, AdministrativeRegion, Novelist, Politician, ArtMuseumsAndGalleriesInRussia which closed\_on, AdministrativeRegion with distance\_to\_moscow \_\_city*

We believe that the resulting RDFS graph entails enough classes in which RDF individuals from the RDF dataset can be identified by them. Non-named classes are issued by the names of individuals in the RDF dataset since their names don't lead to any useful information to be retrieved from DBpedia. For example, node 13 which doesn't get any name includes two objects *Russian\_Winter\_Folk\_Festival* and *the\_Festival\_of\_the\_North*. Since no information is retrieved from DBpedia for those objects, their nodes don't get any name. The other example is node 16 which includes the object with name *ice\_skating* which are identified as an *sport* type but nothing is retrieved from DBpedia by searching *ice\_skating*.

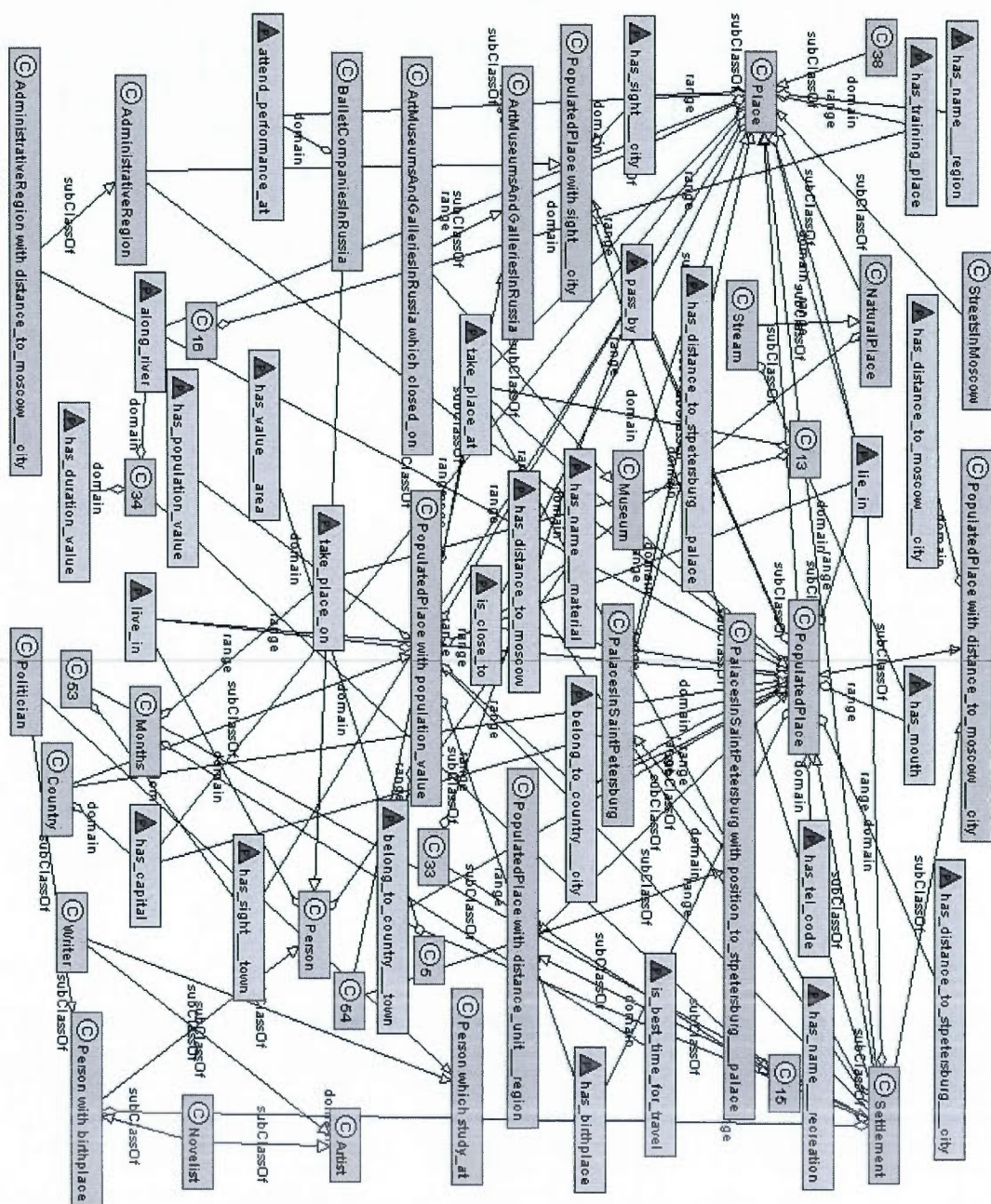


Figure 4.3: Parts of RDFS graph of Russia dataset

#### 4.4 Summary

In this chapter, we used an adequate dataset to prove the efficiency of our methodology by using measurement metrics. FCA helps us detect similar RDF individuals in the dataset according to the common properties they share. The concept lattice created by Galicia includes concepts where each concept contains similar individuals. After obtaining the schema level of the RDF dataset, each concept is named according to the names of its individuals using DBpedia. This chapter verifies the productivity of our approach on detecting schema from the RDF dataset and transforming it to RDFS model.





## CONCLUSION

This thesis has presented a methodology to obtain useful information from Web of Data by creating conceptual structure from RDF datasets. The proposed approach decreases the heterogeneousness of RDF datasets.

To fulfill this need, we used the advantage of FCA to build formal context and determine a binary relation between resources and attributes of RDF statements. The FCA tool used in our implementation is Galicia.

Afterward, by the usage of Galicia the formal context is converted into a concept lattice containing concepts and the hierarchical relation between those concepts. Each concept contains a set of resources called extension as well as a set of attributes shared by those resources called intension.

Gaining the conceptual structure from RDF data is a part of process for converting data from RDF to RDFS model. The obtained concept lattice converts into an RDFS graph. The concepts in concept lattice are considered as classes in RDFS model with hierarchical relation between them. Properties in the RDF data should also be taken into account for completing the RDFS graph. Necessity data properties also should be added properly.

Further, to name the classes of the RDFS graph, we used DBpedia to retrieve the common information that objects of each class share in DBpedia.

Since we use DBpedia to retrieve names for RDFS classes, for datasets containing complicated names for objects our algorithm may not always lead to precise names for the classes.

Although the presented results have demonstrated the effectiveness of our approach, it could be further developed. One way is to refine the names for the classes. It can be done by using WordNET in addition to DBpedia while applying more NLP techniques.



## BIBLIOGRAPHY

- [ABKLC08] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. and Z. Ives, DBpedia: a nucleus for a web of open data , In Proceedings of the 6th International Semantic Web Conference (ISWC), Lecture Notes in Computer Science, Vol. 4825, Springer, pp. 722-35, 2008.
- [AM11] M. d'Aquin, E. Motta, Extracting relevant questions to an RDF dataset using formal concept analysis, In Proceedings of the 6th International Conference on Knowledge Capture (K-CAP), ACM, pp. 121-128, 2011.
- [Arn10] Arnaud le Hors et al , Document Object Model (DOM) Level 2 Core Specification, <http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/>, 2000.
- [BVM04] D. Brickley, R. V. Guha and B. McBride, RDF Vocabulary Description Language 1.0: RDF Schema, <http://www.w3.org/TR/rdf-schema/>, 2004.
- [CBHS05] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler, Named Graphs, Provenance and Trust, In Proceedings of the 14th International World Wide Web Conference (WWW ), ACM Press, 2005.
- [DDFLH90] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, Indexing by Latent Semantic Analysis, Journal of the American Society for Information Science, 41(6):pp. 391-407, 1990.
- [DFD02] A. Delteil, C. Faron, R. Dieng, Building Concept Lattices by Learning Concepts from RDF Graphs Annotating Web Documents. In: Priss U, Corbett D, Angelova G (eds) LNAI 2393, Springer, pp 191-204, 2002.
- [DK07] L. Deligiannidis, K. Kochut, A. Sheth, RDF data exploration and visualization, In Proceedings of the ACM first workshop on CyberInfrastructure 2007, pp. 39-46, New York, 2007.
- [ES07] J. Euzenat, P. Shvaiko, Ontology Matching, Springer, Heidelberg, 2007.

- [Fer10] S. Ferre, Conceptual Navigation in RDF Graphs with SPARQL-Like Queries, In: Kwuida, L., Sertkaya, B. (eds.), ICFCA 2010. LNCS, vol. 5986, pp. 193-208, Springer, Heidelberg, 2010.
- [GW99] B. Gamter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer Verlag, Berlin, 1999.
- [GW06] S. Goyal, R. Westenthaler, RDF Gravity (RDF Graph Visualization Tool), Salzburg Research, Austria, <http://semweb.salzburgresearch.at/apps/rdf-gravity/index.html>, 2006.
- [HKLM09] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R.J. Miller, and M. Wang, A framework for semantic link discovery over relational data, In Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM), D.W.-L. Cheung, I.-Y. Song, W.W. Chu, X. Hu, and J.J. Lin, eds, ACM, pp. 1027-1036, 2009.
- [HXML09] O. Hassanzadeh, R. Xin, R. J. Miller, L. Lim, A. Kementsietsidis, M. Wang, Linkage Query Writer, In Proceedings of the 35th International Conference on Very Large Data Bases (VLDB 2009)-Demonstrations Track, 2009.
- [JGM07] A. Jaffri, H. Glaser, I. Millard, URI Identity Management for Semantic Web Data Integration and Linkage, In Proceedings of the Workshop on Scalable Semantic Web Systems, Springer, Vilamoura, Portugal, 2007.
- [JGM08] A. Jaffri, H. Glaser, and I. Millard. Managing uri synonymity to enable consistent reference on the semantic web. In IRSW2008 - Identity and Reference on the Semantic Web 2008 at ESWC, 2008.
- [KL11] M. Kirchberg, E. Leonardi, Y. Shyang Tan, R.K.L. Ko, S. Link, B.S. Lee, Beyond RDF Links: Exploring the Semantic Web with the Help of Formal Concepts, Semantic Web Challenge, 2011.

- [KL12] M. Kichberg, E. Leonardi, S. Link, R. Ko, F. Lee, Formal Concept Discovery in Semantic Web Data, Springer, 2012.
- [Lev65] V. Levenshtein, Binary codes capable of correcting spurious insertions and deletions of ones, Probl. Inf. Transmission 1, pp. 8-17, 1965.
- [Li13] Y. Li, A Federated Query Answering System for Semantic Web Data, Ph.D Thesis, Lehigh University, Pennsylvania, 2013.
- [Lin10] F. Lindörfer, Semantic Web Frameworks, CS341 Distributed Information Systems, HS 2010 Universität Basel, 2010.
- [LS99] O. Lassila, R. R. Swick, Resource Description Framework (RDF) Model and Syntax Specification, <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>, 1999.
- [MA13] M.W. Chekol, A. Napoli, An FCA Framework for Knowledge Discovery in SPARQL Query Answers, The Semantic Web (iswc2013), Sydney, 2013.
- [McB01] B. McBride, Jena: Implementing the RDF Model and Syntax Specification, In Steffen Staab et al (eds.): Proceedings of the Second International Workshop on the Semantic Web - SemWeb'2001, 2001.
- [MCLYP09] J. Mi, H. Chen, B. Lu, T. Yu, and G. Pan, Deriving similarity graphs from open linked data on semantic web, In Proceedings of the 10th IEEE International Conference on Information Reuse and Integration, pp. 157-162, 2009.
- [MLASH12] M. Morsey, J. Lehmann, S. Auer, C. Stadler, S. Hellmann, Dbpedia and the live extraction of structured data from wikipedia, Program: Electronic Library and Information Systems, pp 46-27, 2012.
- [MM04] F. Manola, E. Miller, RDF Primer, <http://www.w3.org/TR/rdf-primer/>, 2004.
- [MV10] T. Markotschi, J. Völker, GuessWhat?!-Human Intelligence for Mining Linked Data, In Proceedings of the Workshop on Knowledge Injection into and Extraction from Linked Data (KIELD) at the International Conference on Knowledge Engineering and Knowledge Management (EKAW), 2010.



- [NUM07] A. Nikolov, V. Uren, E. Motta, Knofuss: a comprehensive architecture for knowledge fusion, In Proceedings of the 4th International Conference on Knowledge Capture, K-CAP 2007, pp. 185-186, ACM, New York, 2007.
- [PHH04] P. Patel-Schneider, P. Hayes, and I. Horrocks, OWL Web Ontology Language semantics and abstract syntax Recommendation, <http://www.w3.org/TR/owl-semantics/>, 2004.
- [PS06] E. Prud'hommeaux, and A. Seaborne, SPARQL Query Language for RDF, <http://www.w3.org/TR/rdf-sparql-query/>, 2006.
- [RDF04] RDF/XML Syntax Specification (Revised), World Wide Web Consortium, <http://www.w3.org/TR/rdf-syntax-grammar/>, 2004.
- [ROH05] L. Rutledge, J. Ossenbruggen, L. Hardman, Making RDF Presentable: Integrated Global and Local Semantic Web Browsing. In: Proc WWW2005, Chiba, pp 199-206, 2005.
- [Row09] M. Rowe, Interlinking Distributed Social Graphs, Linked Data on the Web, Workshop, LDOW, 2009.
- [JN07] H. Jia, J. Newman, H. Tianfield, A new formal concept analysis based learning approach to ontology building, in Proceedings of the 2nd International Conference on Metadata and Semantics Research (MTSR 2007), pp. 433-444, 2007.
- [RSS08] Y. Raimond, C. Sutton, and M. Sandler, Automatic interlinking of music datasets on the semantic web, In Proceedings of the Linking Data On the Web workshop at WWW'2008, 2008.
- [SB10] T. Schandl, A. Blumauer, Poolparty: Skos thesaurus management utilizing linked data, In Lora Aroyo, Grigoris Antoniou, Eero Hyvri, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, ESWC (2), volume 6089 of Lecture Notes in Computer Science, pages 421-425, Springer, 2010.



- [SBO09] H. Stoermer, P. Bouquet, A Novel Approach for Entity Linkage, In Proceedings of IRI 2009, the 10th IEEE International Conference on Information Reuse and Integration, vol. 10, pp. 151-156, USA, 2009.
- [SCRGDS09] M. Salvadores, G. Correndo, B. Rodriguez-Castro, N. Gibbins, J. Darlington, N. R. Shadbolt, LinksB2N: Automatic data integration for the semantic web, In OTM Conferenc , pp. 1121-1138, 2009.
- [SLZ09] F. Scharffe, Y. Liu, and C. Zhou. RDF-AI: an architecture for RDF datasets matching, fusion and interlink, In Workshop on Identity and Reference in Knowledge Representation, IJCAI 2009, 2009.
- [SSK05] G. Stoilos, G. Stamou, and S. Kollias, A string metric for ontology alignment, In Proc. 4th International Semantic Web Conference (ISWC), volume 3729 of Lecture notes in computer science, pp. 624-637, Galway (IE), 2005.
- [Szl09] G. Szlechtman, On the Semantic Web: SPARQL queries, <http://blogs.southworks.net/gabrielsz/category/semanticweb/>, January 9th, 2009.
- [VBGK09] J. Volz, C. Bizer, M. Gaedke, G. Kobilarov, Silk - A link discovery framework for the web of data, In Proceedings of the 2nd Linked Data on theWebWorkshop, April 2009.
- [VG03] P. Valtchev, D. Gosser, C. Roume, and M. Hacene, Galicia: an open platform for lattices, In Using Conceptual Structures: Contributions to ICCS 2003, B. Ganter and A. de Moor, Eds. Shaker Verlag, pp. 241-254, 2003.
- [WB04] B. Wormuth, P. Becker, Introduction to formal concept analysis, in: 2nd International Conference of Formal Concept Analysis, Sydney, 2004.
- [WH06] J. Wang, K. He, Towards representing FCA-based ontologies in Semantic Web Rule Language, In Proceedings of the Sixth IEEE International Conference on Computer and Information Technology (CIT'06), Korea, 2006.

- [Zha07] Y. Zhao, Using Formal Concept Analysis for Semantic Web Applications, Springer-Verlag Berlin Heidelberg, SCI (42):157-176, 2007.
-