TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK

MASTER THESIS

Software Implementation for Taxonomy Browsing and Ontology Evaluation for the case of Wikidata

Author:
Serghei STRATAN

Supervisor:
Dr. Markus KRÖTZSCH

INTERNATIONAL MASTER IN COMPUTATIONAL LOGIC

March 2016

Declaration

I hereby declare, that except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole, or in part to obtain a degree or any other qualification neither in this nor in any other university. This thesis is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text. No other resources apart from the references and auxiliary means indicated in the bibliography were used in the development of the presented work.

(Place, Date)	Serghei Stratan
	Matriculation number : 4024771

Abstract

Ontologies represent one of the basic elements in the Semantic Web. They contain background knowledge represented by relevant terms and formal relations between them, so that machines can read it automatically. Ontologies on the Semantic Web come from a vast variety of different sources, spanning institutions, and persons aiming for different goals and quality criteria.

Measuring some of the aspects of ontologies, enables us to answer the question: *"How to assess the quality of an ontology on the Web?"*. The ontology evaluation task is essential for a wide adoption of ontologies, in the Semantic Web and in other semantically enabled technologies.

The Wikidata project, being a part of a big Wikimedia family, is widely known for its data that is provided freely to the public. The data is structured and stored in a web ontology. This ontology has turned immediately into a resource of big value, with a large range of potential applications across all areas of science, technology, and culture. Unfortunately, at the moment there is no possibility to assess the quality of the Wikidata ontology.

We have chosen Wikidata as a basic scenario for implementation of an automated system which would allow to browse the class hierarchy contained in the Wikidata ontology. The class hierarchy is created by the class entities and two properties (*'subclass of'* and *'instance of'*) which denote relations between classes. Besides the Wikidata taxonomy browser, additionally we implemented a component for ontology evaluation, which shows how the quality of the Wikidata ontology is assessed for several evaluation criteria.

Contents

Chapter 1

Introduction

In the last years, *Semantic Web Technologies* gave possibility to develop tools which are easily accessible and usable by users, providing access to different web resources and their shared knowledge. This infrastructure is based mainly on formal domain models (*ontologies*) that are linked to each other on the Web. These ontologies provide the applications with shared terminologies and understandings. The World Wide Web Consortium (W3C) standardized the *Web Ontology Language* (OWL) as the standard format in which ontologies are represented online [McGuinness and van Harmelen, 2004]. It is designed to enable *domain experts* to express their *domain knowledge* in a precise and sharable manner via the Web in order to provide machine accessible knowledge content.

Ontologies are the main component in the emerging Semantic Web [Cruz et al., 2002]. They are used to share the knowledge by expressing relevant concepts and the relation between them. Providing formal semantics to contained terms enables machines to process the knowledge automatically and to understand it. Ontologies allow not only to understand the meaning of the data that is being exchanged by the machines, but also allow sharing and formalization of different conceptualizations by humans.

The importance of ontologies in the Semantic Web makes the ontology evaluation task very important [Vrandečić, 2010]. Ontologies can contain a lot of mistakes and omissions that can lead to difficulties in processing the data. Good ontologies make possible to reuse the knowledge in a simpler way for a large number of applications.

This thesis mainly discusses the designing techniques applicable for a taxonomy browser application and evaluation of Web ontologies, i.e. ontologies specified in the standard *Web Ontology Language* (OWL) and published on the Web, so that they can be used and extended in ways not expected by the creators of the ontology.

1.1 Motivation

An example of a large Web ontology is the Wikidata ontology. Today, the Wikidata project represents a central storage for the *structured data* of its Wikimedia[1] sister projects [wik, Dec, 2015]. The knowledge from a very large range of areas is contained in one of the biggest ontologies available on the web which is accesible freely by the public. The data is entered and maintained by thousands of the Wikidata editors, and of course, nobody can guarantee the good quality of this ontology.

To help editors from the Wikidata community to understand the Wikidata ontology and to represent graphically the contained class hierarchy, we decided to build a system which would allow them to visualize the class hierarchy as a graph and to browse the taxonomy.

On the other side, we have noticed the absence of some methods and automated tools which would allow users and editors to verify the quality of the Wikidata ontology. Therefore, this motivated us to design and develop an additional component to the main application which would give possibility to evaluate the ontology for several criteria.

1.2 Outline

In short, readers who know Semantic Web and web ontologies in particular, and who are also familiar with the Wikidata project, can safely skip Chapter 2. We start with our application design and implementation in Chapter 3, together with the subsequent chapters represent the core work of this thesis.

Chapter 2 gives necessary preliminaries and terminology for understanding the rest of the thesis. We start with describing what are taxonomies (Section 2.1.1) and ontologies (Section 2.1.2). We continue with Section 2.1.3 where we define the *Web Ontology Language* (OWL) and its semantics. In the following section, we introduce the Wikidata project and its ontology (Section 2.2). In the last Section 2.3, we talk about ontology evaluation, criteria, and methods that we use in our application.

Chapter 3 introduces the developed application for taxonomy browsing and ontology evaluation for the case of Wikidata which constitute the main practical contribution of this thesis. We briefly outline the basic structure and components of the application in Section 3.1. Then we specify the general user stories in Section 3.2, that we have used in the software development process. In the same section, we describe all functionalities and features that have been implemented in the final application. In Section 3.3 we describe the errors and redundancies from the Wikidata ontology,

[1] https://wikimediafoundation.org/wiki/Home

which are created by the incorrect use of the entity relation properties, and generalize them into distinctive patterns. These patterns will be used in the automation of ontology evaluation process.

Chapter 4 presents the architecture of the developed system (Section 4.1), together with a short description of its components (Section 4.2). The main algorithms for the ontology evaluation are shown in Section 4.3, together with a short explanation of their implementation.

Chapter 5 shows an initial evaluation of the system for runtime performance (Section 5.1). Then we talk about the problems that we encountered during the developing process (Section 5.2), and what we have learned from the Wikidata ontology evaluation (Section 5.3). Finally we present some of the users' feedback that we received after the official launch of the application to the Wikidata community and its usability (Section 5.4).

Chapter 6 provides a retrospective summary, leading to some general conclusions and assessment of our work. We conclude with an overview on the related work, in Section 6.1, and lastly present an outlook in Section 6.2 on potential future work and application improvements.

Chapter 2

Preliminaries

In this chapter is introduced the preliminary materials necessary for the reader to understand the theoretical background of the whole thesis. We start with defining what is a taxonomy (Section 2.1.1), then continue with defining ontologies and explaining their basic elements (Section 2.1.2). We continue with Section 2.1.3 where we describe the *Web Ontology Language* (OWL) and its semantics. In the following section, we introduce the Wikidata project, explain how the data is structured in its web ontology, and which entity's statements will be used in the following (Section 2.2). In the last Section 2.3, we talk about the ontology evaluation, and what evaluation criteria and methods are applicable in our case.

2.1 The Semantic Web

The goal of the *Semantic Web* initiative is to provide a description of the content and the ability to create and to access different web resources that can be read by machines.

The Semantic Web, which is also known as **Web of Data**, is actually an extension of the World Wide Web. It has similar goals with the Web in general: to make the knowledge widely accessible and to increase the utility of this knowledge by enabling advanced applications for searching, browsing, and evaluation [Berners-Lee et al., 2001].

Ontologies plays a central role in the process of sharing the knowledge between different systems and within the systems by the various components. Ontologies are defined as *"explicit specification of a conceptualization*[1]*"* by [Gruber, 1993]. Since they define the formal semantics of the terms and the relations between them, ontologies ensure the meaning of the shared data to be consistent.

[1] A *conceptualization* is way of thinking about part of the world.

2.1.1 Taxonomy

To make the knowledge readable automatically by machines, first it needs to be structured. Taxonomies provide a specific representation relationship between different semantic entities. These information entities are presented in the taxonomies in the form of hierarchies, which are generated based on the presumed connections of the real-world entities.

According to [Daconta et al., 2003], a *taxonomy* is defined as a *classification*[2]. A taxonomy organizes the information entities hierarchically based on a special relation between them. The special relation is represented by *'subclass of'* relationship which denotes the subsumption relation and is similar to OWL's `SubClassOf` construct that is explained later in Section 2.1.3. The information entities are defined as classes of objects, where a *class* is a collection of objects which have the same characteristics.

In the most cases, a taxonomy is represented graphically as a graph. In Figure 2.1 we can see an excerpt of hierarchical taxonomy contained in the Wikidata ontology, represented as a directed graph. The example includes the class *Star* and its subclasses of *Giant star* and *Supernova*, and its superclass of *Astronomical Object*. We can see that the root class for this example is the *Entity* class.

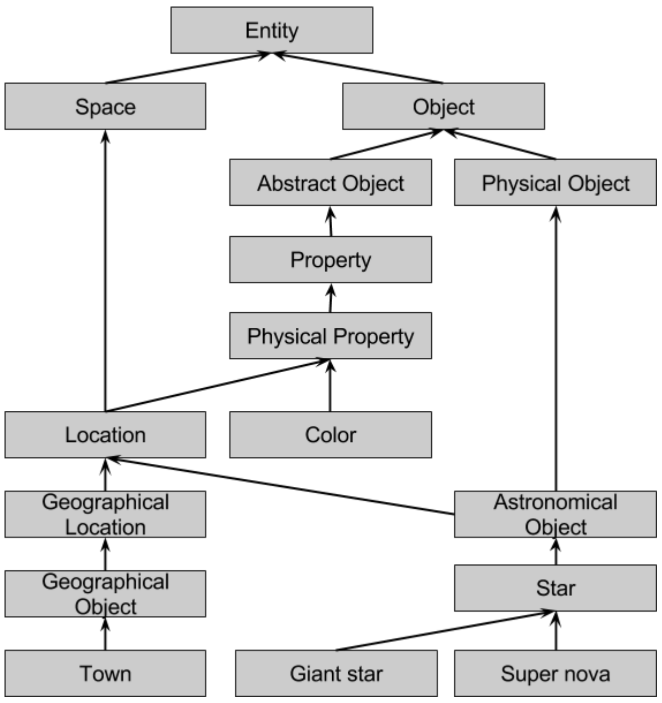


FIGURE 2.1: Taxonomy example from Wikidata (*November, 2015*).

The edges in the graph represent the *'subclass of'* relations between classes. Following the direction of the edges in the graph, we can infer the relevance of the information entities in the

[2] the act of establishing groupings based on ways that they are alike

semantical representation. The class entities at the top of the taxonomy are more general than the class entities situated on a lower level in the hierarchy. This representation helps to classify the information entities semantically according to their real-world representation.

The most usual use of a taxonomy is the possibility to browse and navigate the information entities contained in it. Taxonomies do not allow to add some additional attributes to the information entities [Breitman et al., 2007]. Hence, it is necessary to use the ontologies if this is required.

2.1.2 Ontologies

An *ontology*, as it is defined by [Gruber, 1993], is a formal specification of a shared conceptualisation of a domain of interest. It means that this formal specification should be interpretable by machines and shared between them based on some consensus.

An ontology is a representation of the knowledge from a particular area of interest that is corresponding to a human representation of that domain. The knowledge is represented by *concepts* and relationships between these concepts. The concepts describe the *meaning* of the knowledge from any area of interest.

The modern ontologies have similar structure based on the language that is used to express them. The most of ontologies describe individuals, classes, and properties:

- *Individuals*: instances or objects (basic or “ground level” objects);
- *Classes*: sets, collections, concepts, types of objects, or kinds of things;
- *Properties*: mainly represent the relationships between objects (and classes).

Within this thesis we regard only *Web ontologies* that are encoded in a standardized Semantic Web ontology language.

Different expressive representation languages are used to model complex ontologies which capture the knowledge from a specific domain. Based on this, we can do logical reasoning on them to get the knowledge contained within modeled ontologies.

OWL (*Web Ontology Language*) is a W3C recommended standard for the modeling of ontologies. For a detailed description of OWL, see Section 2.1.3.

An ontology is represented as a set of *axioms* that are stated in the web ontology language OWL. In Sections 2.1.3.2 and 2.1.3.3, we describe the available types of *entities*, followed by the types of *axioms* which consist the basic elements of an ontology.

2.1.3 OWL

The Web Ontology Language **OWL** extends RDF[3] and RDFS[4]. The most important task of it is to bring the expressive and reasoning power of *Description Logic*[5] to the Semantic Web. Since 2004, OWL is a W3C recommended standard for the modelling of ontologies [McGuinness and van Harmelen, 2004]. OWL was designed to simplify ontology development and to share the ontology via the Web.

Since 2009, the working group of World Wide Web Consortium proposed a major extension of the original version of the *Web Ontology Language* OWL. The next revision of OWL was called OWL 2 [Group, 2012].

OWL 2 comes with three different *profiles*. The OWL 2 Profiles represent sublanguages (or syntactic subsets) of OWL 2 that offer some expressive power for the efficiency of reasoning. The following description of OWL 2 Profiles is taken from [Motik et al., 2012a]:

- **OWL 2 EL**
  - used for ontologies with very large number of properties and/or classes,
  - computational complexity: *Polynomial* to the size of ontology,
  - is a part of $\mathcal{EL}$ family of Description Logics, that supports only Existential quantifiers.
- **OWL 2 QL**
  - used for large volumes of instance data,
  - orientated for query answering,
  - implemented using conventional relational database systems,
  - the expression power is limited,
  - implemented by rewriting queries into a standard relational Query Language.
- **OWL 2 RL**
  - less expressible,
  - orientated for rule-based reasoning,
  - computational complexity: *Polynomial* to the size of ontology,
  - implemented using a standard Rule Language.

Besides the OWL 2 Profiles, the web ontology language has two semantic specifications that define the meaning of OWL 2 ontologies: **OWL 2 Direct Semantics** and **OWL 2 RDF-Based Semantics**. To help reasoners and other automated tools to answer class consistency, subsumption, and instance retrieval queries, these two semantics are used. The *OWL 2 Direct Semantics*

[3] http://www.w3.org/RDF
[4] http://www.w3.org/TR/rdf-schema
[5] a family of formal knowledge representation languages

is shortly explained later in Section 2.1.3.1. For a detailed description of both semantics, please refer to the [Group, 2012].

In comparison to the first version of OWL language, OWL 2 adds the following new functionalities [Group, 2012]:

- keys;
- property chains;
- richer datatypes, data ranges;
- qualified cardinality restrictions;
- asymmetric, reflexive, and disjoint properties;
- enhanced annotation capabilities.

OWL collects the information into ontologies and then store them as documents. Each of these OWL documents contain ontology headers (generally not more than one), class axioms, property axioms, and facts about individuals [Smith et al., 2004].

Throughout this thesis we are using the OWL 2 Functional Syntax [Group, 2012] for defining axioms and thus ontology itself. We chose this syntax because it is easy to read and very concise.

The basic elements of OWL are classes, properties, and individuals (declared as instances of classes). Classes are defined in OWL 2 Functional Syntax as following:

```
Declaration(Class(Pizza))
```

From the example above, we can see that a new class is declared with the name *Pizza* which can be used later to reference to that class.

In OWL exist two classes which are predefined, called *owl:Thing* and *owl:Nothing*. The class *owl:Thing* is considered the most general class in the ontology, and every individual is an instance of that class. The class *owl:Nothing* does not have instances at all.

Individuals can be defined as instances of classes. This is called *class assertion*.

```
ClassAssertion(Pizza pizzaMargherita)
```

OWL contains two different types of properties: *object properties* and *data properties*. The object properties connect pairs of individuals. The data properties connect individuals with data values (literals), i.e. with elements of datatypes. Properties are declared similarly as classes.

```
Declaration(ObjectProperty(hasIngredients))
Declaration(DataProperty(size))
```

The first declaration is an object property declaration and expresses which ingredients a given pizza has. The second one is a data property declaration which assignes a size to a pizza.

From the representation below we can see, an individual called *pizzaMargherita* is declared as instance to the previously declared class *Pizza*. Also, it assignes two types of ingredients to it: *cheese* and *tomato*, and a value *"large"^^xsd:string* to the size property.

```
ClassAssertion(Pizza pizzaMargherita)
ObjectPropertyAssertion(hasIngredients pizzaMargherita cheese)
ObjectPropertyAssertion(hasIngredients pizzaMargherita tomato)
DataPropertyAssertion(size pizzaMargherita "large"^^xsd:string)
```

If there are multiple declared classes in the ontology, then they can be put in a relation to each other. For more details about the *class inclusion* and other types of relationships, please see Section 2.1.3.3.

Every individual in OWL is a member of the class `owl:Thing`. Thus each defined class is implicitly a subclass of `owl:Thing`, and the predefined class `owl:Nothing` is a subclass of every other class [Smith et al., 2004].

2.1.3.1 Semantics

This section specifies the *direct model-theoretic semantics* of OWL 2 ontologies as it is presented in [Group, 2012]. The given semantics is close related to the semantics of Description Logics and especially it "extends the semantics of the description logic $\mathcal{SROIQ}$" [Motik et al., 2012b].

In order to define a formal semantics, we consider *interpretation*[6] $\mathcal{I}$ that consists of a non-empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation) and a mapping function $\cdot^{\mathcal{I}}$ [Baader et al., 2003]. An axiom or an ontology is *satisfied* in this given interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, if the appropriate conditions hold to each axiom type [Motik et al., 2012b].

Table 2.1 describes a very small part of OWL 2 axiom types and their direct set semantics which are relevant for the case of the Wikidata ontology. Table 2.1 is compiled from [Motik et al., 2012b]. To see the complete list of axiom types for OWL 2, please see the original source.

2.1.3.2 Entities

As it is defined in [Group, 2012], **entities** are the essential element of structural representation of OWL 2 ontologies. They use to define the named terms of an ontology which then form the vocabulary[7] of it.

[6] "an assignment of meaning to the symbols of a formal language"
[7] a set of URI references

TABLE 2.1: Semantics of OWL 2 axioms, relevant to Wikidata

Functional syntax	**Set semantics**
`ClassAssertion(`C a`)`	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
`PropertyAssertion(`R a b`)`	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
`SubClassOf(`C D`)`	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
`EquivalentClasses(`C D`)`*	$C^{\mathcal{I}} = D^{\mathcal{I}}$
`TransitiveProperty(`R`)`	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}} \wedge (b^{\mathcal{I}}, c^{\mathcal{I}}) \in R \rightarrow (a^{\mathcal{I}}, c^{\mathcal{I}}) \in R^{\mathcal{I}}$

where $a, b, c \in N_{\mathcal{I}}$ are individual names, $C, D \in \mathbf{C}$ are concepts, $R \in \mathbf{R}$ is a role. The axiom type noted with * may hold more than the given parameters.

In OWL 2, **ontology entities** are defined as *individuals*, *classes*, *properties* (data properties and object properties), or *ontologies* etc. For a complete list of entity types, please refer to the original source [Group, 2012].

Individuals represent the actual objects from the domain. There are two types of individuals which can be given by their *name* or defined as *anonymous individuals*. An individual can be any entity with an identity (otherwise it would not be possible to identify that entity with an identifier). An *anonymous individual* does not have a global name and is thus local to the ontology where it is contained. This means that the individual can not be identified directly from outside of the given ontology.

A **class** can be understood as sets of individuals. A class is defined by a *class expression*. It can be a *class name* or a *complex class description*. A *class name* is just the name, i.e. an IRI of a class. Class names do not have any formal information about the defined class.

The *property expressions* define the **properties**. Usually, the most simple *property expressions* are just *property names*. *Properties* can be of two types: *object properties* or *data properties*. **Object properties** connect two individuals with each other. **Data properties** connect an individual with a *data value*. Section 2.1.3 provides an example how the property expressions are declared and used in the ontology.

Besides entities, OWL 2 ontologies can also contain literals of string or integer types. A *data value* is represented by a *literal* which is the syntactic representation of a concrete value. A *datatype map* gives a mapping between the *literal* and the *data value*. For example, the typed literal `"7"^^xsd:int` is mapped to the number 7 and the literal `"7"^^xsd:string` is mapped to the string ‘7’.

2.1.3.3 Axioms

In the following, we will describe one of the important components of any ontology. The main definitions and explanations are based on the official recommendations of W3C group which can be found in more details on [McGuinness and van Harmelen, 2004, Smith et al., 2004, Group, 2012].

The basic element of the knowledge within an ontology is an **axiom**. OWL contains different type of axioms, such as *facts* (or *assertions*), *terminological axioms*, and *annotations*. There are more types of axioms defined in OWL 2, but we will not refer to them in this thesis since they are not relevant for the Wikidata project. The complete list is available on [Group, 2012].

A **fact** is defined as a *class assertion*, an *object property assertion*, or a *data property assertion*.

An **instantiation** or **class assertion** is declared in the following way:

$$\texttt{ClassAssertion(}C\ a\texttt{)}$$

where C is a *class expression* and a being an *individual name*. This declaration can also say that a has the type C. If to explain it semantically, then the individual named with a is in the extension of the defined set C.

An **object property assertion** has the form:

$$\texttt{PropertyAssertion(}R\ a\ b\texttt{)}$$

where a and b are *individual names* and R is an *object property assertion*. This axiom allows to state that individual a is connected by an object property expression R to an individual b, e.g. saying that “Paris is the capital of France”. In this example, *Paris* and *France* are individual names, and *capital* is the name of the property that holds between them. The relation is defined as the instantiation of this property. Semantical definition of this declaration means that the tuple (a, b) is in the extension of the set R.

A **data property assertion** is declared almost the same as an object property assertion:

$$\texttt{PropertyAssertion(}R\ a\ v\texttt{)}$$

Data property assertion axiom allows to state that an individual a is connected by a data property expression R to a literal v.

A **terminological axiom** is defined as a *class axiom* or a *property axiom*. OWL 2 has axioms that allow relationships to be established between different class expressions. **Class axiom** can either be a *subsumption*, *class equivalence*, *disjoint*, or *disjoint union*.

In OWL 2 a **subsumption** relation is declared in the following way:

$$\texttt{SubClassOf(}C\ D\texttt{)}$$

where C (the subclass) and D (the superclass) are *class expressions*. This subsumption means that every individual of the class C is also an individual of the class D. The `SubClassOf` axiom allows to construct a hierarchy of classes. For example,

```
SubClassOf(Bread Food)
SubClassOf(FlatBread Bread)
SubClassOf(Pizza FlatBread)
```

From this example we can infer that the class *`Pizza`* is a subclass of *`Food`*.

In the same way in OWL 2 are defined `EquivalentClasses` and `DisjointClasses` axioms. For a complete description, please refer to [Hitzler et al., 2012].

In OWL applications is often necessary some possibilities to associate additional details with ontologies, entities, and axioms. In this way are defined **annotations** which connect an element by an *annotation property* with an *annotation value*. These elements can be defined as *entities*, *ontologies*, or *axioms*. An annotation adds further information about the elements. The most broadly used annotation is `rdf:label`. In this way a human-readable label is connected with an element.

2.2 Wikidata

In this section we will describe the most relevant things about the Wikidata project, some details about how the contained data is structured in the Wikidata ontology, and which are the main components of this ontology. The following information is reproduced from different official project documentation sources [Introduction, Dec, 2015], [Items, Dec, 2015], [Data, Dec, 2015] and is presented here in a short form.

"*Wikidata* is a free linked database that can be read and edited by both humans and machines" [wik, Dec, 2015]. Wikidata is storing the structured data of its Wikimedia sister projects, such as *Wikipedia*, *Wikivoyage*, *Wikisource* etc., and plays an important role. It has become a good source of verified knowledge for many other sites and services besides the Wikimedia projects. Its content is available under a free license, which gives possibility to download it freely in different data formats.

Until today, almost every Wikipedia page in any available language is linked with a similar page from Wikidata. The same as Wikipedia, Wikidata incorporates a very large range of knowledge that is applicable in different areas. All the data is permanently edited and supplied by users from Wikimedia community, making it new and original.

According to [Krötzsch and Vrandečić, 2014] the Wikidata project can be characterized:

- **It's free**. All the data in Wikidata is published under the *Creative Commons Public Domain Dedication 1.0*[8]. This means that everyone can use the provided data for own purpose. The data can be copied, modified, distributed, and used without special permission.
- **It's collaborative**. All the data in Wikidata is added and maintained by thousands of Wikidata editors. They make decisions how to structure the data and manage it.
- **It's multilingual**. All the data in Wikidata is translatable. All the new entered data to the system is available for translation into other languages.
- **A secondary database**. All the data from Wikidata is linked to its source. This reflects the diversity of the knowledge and verifies the source.
- **Collects structured data**. If to compare the Wikidata project with other its syster projects such as Wikipedia, which store information in format of encyclopedic articles, it collects the data in a structured form. This gives possibility to reuse the stored data by other projects, and to easily process and "understand" it.
- **Large Support**. Wikidata is supporting a big range of different services where its data is used.

2.2.1 The Wikidata Repository

All the Wikidata information is stored on a repository. Data on the Wikidata repository is structured in items. Each item is defined with a *label*, a *description* and possibly with one or more *aliases*. Each item is possible to link with a corresponding page from other Wikimedia project through *sitelinks*. Items have *statements* which are used to describe in details all of their characteristics. Each statement consists of a *property* and a *value* [Introduction, Dec, 2015].

In Wikidata can be found any kind of information. For example, Wikidata allows to add a new food dish and to link it with the ingredients that might contain; to add a person and to link it to his or her country of birth or death; to add a new astronomical star with its location in a star constellation; to add a vulcano and its geographical coordinates etc.

Wikidata is a knowledge base that is editable by anyone and also it is useable for free. As any of its Wikimedia sister projects, Wikidata is built on a wiki package, called MediaWiki[9] [Krötzsch and Vrandečić, 2014]. It allows any user to manage every page from Wikidata without a deep knowledge in computer science, such that editing, deleting, and adding information is easy and is made in collaboration with others. Wikidata uses Wikibase[10] software which allows collaborative editing of structured data.

[8] https://creativecommons.org/publicdomain/zero/1.0/
[9] https://www.mediawiki.org
[10] http://wikiba.se

Concepts, objects, topics, or any other type of information, are represented as items in the Wikidata repository [Erxleben et al., 2014]. For example, the *2012 Summer Olympics*, *love*, *Oxigen*, and *ship* are all items in Wikidata.

Since Wikidata's items are quite important, there are few important characteristics about them [Items, Dec, 2015]:

- **All items are unique**: Each item is a unique identifiable concept or object, or an instance of them. For example, in Wikidata is possible to have items for general concepts as *star (Q523)*, and an instance of those general concepts, such as *Sun (Q525)*.
- **All items are notable**: This mean that any item from Wikidata have a corresponding page to any of the Wikimedia sites such as Wikipedia, Wikivoyage, Wikisource, Wikiquote, or Wikimedia Commons. Nevertheless, could exists some exceptions to this rule.
- **Linked items**: In Wikidata, items can be linked to each other. Every item is identified by a unique ID (starting with a *Q* prefix) and has its own page in the Wikidata main namespace. For example, for the items listed above, *2012 Summer Olympics (Q8577)*[11], *love (Q316)*, *Oxigen (Q629)* and *ship (Q11446)* are the corresponding item pages. On these pages all the data for these items can be added, edited, and maintained.

All the structured data is organized according to a **data model** [Data, Dec, 2015]. This data model allows machines to read and to understand the data. Also, it allows to translate a human natural language representation of things into a machine-readable structure. For example, in English we can say:

"Challenger Deep is the deepest point in the world."

In Wikidata, this sentence can be translated using statements, which give possibility to link different properties of items with their values. In this example we have item *Earth (Q2)*:

Earth (Q2) (item) → deepest point (P1589) (property) → Challenger Deep (Q459173) (value)

Additionally, Wikidata would also hold a statement about the item for Challenger Deep (indicating it is a landform[12]):

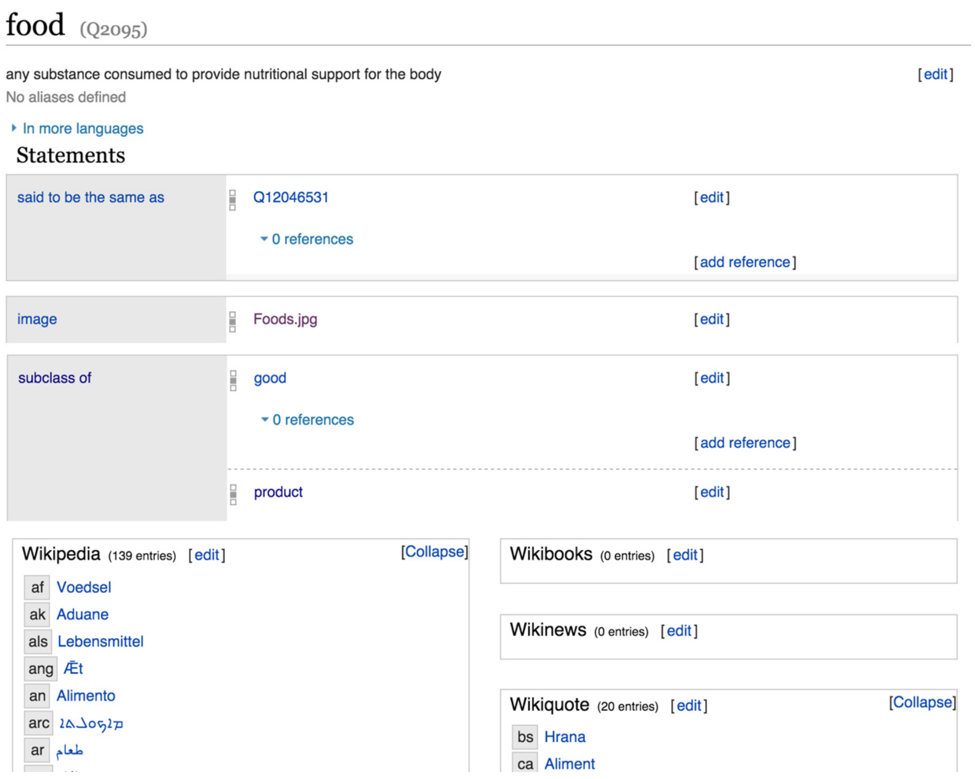
Challenger Deep (Q459173) (item) → instance of (P31) (property) → landform (Q271669) (value)

As we can see, other items from Wikidata can be used as the values for some statements. This means that all items can be linked to each other through different statements. Since the data models are machine-readable, these connections of data allow to discover and process new relationships and connections between items automatically by machines.

[11] https://www.wikidata.org/wiki/Q8577
[12] a geomorphological unit in the earth sciences

2.2.2 The Wikidata Statements

As is it described by [Erxleben et al., 2014], to every item from Wikidata corresponds a page. In this way the data is structured on the Wikidata repository. Every piece of information from a knowledge, such as a topic, a concept, or an object, has a separate page and is called an *entity*. Wikidata has two different types of entities: *items* and *properties*. Items are represented by *individuals* and *classes*. The Wikidata properties are similar to some of *RDF properties*, such as `SubClassOf` relation from OWL (Section 2.1.3).



food (Q2095)

any substance consumed to provide nutritional support for the body [edit]

No aliases defined

‣ In more languages

Statements

said to be the same as	Q12046531	[edit]
	▾ 0 references	[add reference]
image	Foods.jpg	[edit]
subclass of	good	[edit]
	▾ 0 references	[add reference]
	product	[edit]

Wikipedia (139 entries) [edit] [Collapse]

af	Voedsel
ak	Aduane
als	Lebensmittel
ang	Ǣt
an	Alimento
arc	ܡܐܟܘܠܬܐ
ar	طعام

Wikibooks (0 entries) [edit]

Wikinews (0 entries) [edit]

Wikiquote (20 entries) [edit] [Collapse]

bs	Hrana
ca	Aliment

Figure 2.2: Excerpt of a typical Wikidata item page with terms, statements, and site links (*November, 2015*).

Any piece of information from Wikidata has a corresponding page where the data can be viewed or edited by users. For example, an excerpt for the item page for *food (Q2095)*[13] in English from Wikidata, is shown in Figure 2.2. The unique identifier of the Wikidata items is used as a title for the item page, e.g. the title of the page *food (Q2095)* is *"Q2095"* rather than its label *"food"*. It is useful for the case of internationalization, where labels can be translated into any language.

The following sections are contained in every item page from Wikidata [Erxleben et al., 2014]:

- the *label* (e.g. "food");
- a *short description* (e.g. "any substance consumed to provide nutritional support for the body");

[13] https://www.wikidata.org/wiki/Q2095

- *aliases*;
- a list of *statements* (the item properties);
- the list of *site links* (the links that connect the Wikidata item to other Wikimedia projects).

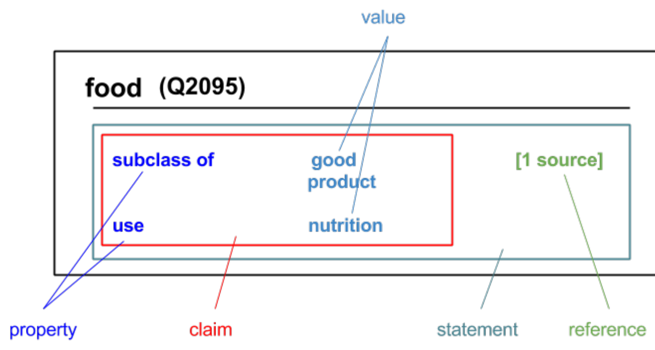


FIGURE 2.3: Elements of a statement in Wikidata.

The *label*, *description*, and *aliases* are generally known as *terms* and can be translated into any available language on Wikidata. These terms are used to distinguish the items and to search them.

To every *property* from Wikidata, belongs a separate page, similar to items. Each page is identifiable by a unique ID which starts with *"P"* and continues with a numerical value. Similar to *RDF properties*, the Wikidata properties have data types that determine the accepted value. Table 2.2 shows the list of all available datatypes in Wikidata. The original source can be found in [Erxleben et al., 2014].

As we said, every item in Wikidata has a corresponding page. Every page might have several **statements** which represent the way how the information, that we know about, is recorded in Wikidata. Statements link an item property with its value. Each of the statements can have a reference or some optional qualifiers. Also, they are used to link different items from Wikidata to each other, making a linked data structure. The basic elements of statements from Wikidata are shown in Figure 2.3 [Introduction, Dec, 2015].

The principal properties of an item from Wikidata, which are relevant to our work, are:

- *ID*;
- the *label*;
- the value for *subclass of (P279)* relation property;
- the value for *instance of (P31)* relation property;
- *description*.

TABLE 2.2: The Wikidata datatypes and their current member fields and field types.

Datatype	**Member fields**
Item	item id (IRI)
String	string
URL	URL (IRI)
Commons Media file	article title (string)
Time	point in time (dateTime), timezone offset (int), preferred calendar (IRI), precision (byte), before tolerance (int), after tolerance (int)
Geographic coordinates	latitude (decimal), longitude (decimal), globe (IRI), precision (decimal)
Quantity	value (decimal), lower bound (decimal), upper bound (decimal)

Subclass of (P279) relation property represents subsumption relation between two involved classes. The equivalent RDFS representation for it is: `rdfs:subClassOf`. This relation property is similar to subsumption relation from OWL 2 which is described in more details in Section 2.1.3. The same as `SubClassOf` relation, it is a *transitive* property.

Instance of (P31) relation property represents the item as a specific example and a member of some class. The equivalent RDF representation is: `rdf:type`.

2.3 Ontology Evaluation

After presenting ontologies and how they are structured, is arising a question about how to assess the quality of a web ontology. In this section we will discuss several evaluation criteria and methods that have been proposed by [Vrandečić, 2010] in his work, to guarantee the good quality of an ontology on the web.

According to [Vrandečić, 2010], ontology evaluations can be done on several different levels, as it is described below:

1. Ontologies can be evaluated by themselves;
2. Ontologies can be evaluated with some context;
3. Ontologies can be evaluated within an application. This approach was introduced by [Brank et al., 2005] in his work and it is called application based ontology evaluation;
4. Ontologies can be evaluated in the context of an application and a task. This method was described first by [Porzel and Malaka, August 2004] and is called task based ontology evaluation approach.

Each of the evaluation levels listed above, evolves from the previous ones. It means that every ontology should be evaluated first by itself and with some context, and later can be evaluated already within an application. The most of the possible errors can be found in the evaluation during the first two levels.

The task of ontology evaluation which consists in checking if the ontology has been modeled correctly, is called *ontology verification* task [Vrandečić, 2010]. During the verification process is tried to find out if the initial specifications are met correctly in the modeled ontology. Errors such as cycles contained in the class hierarchy, redundant axioms, and other types of errors are easily detected by ontology verification task. Once the ontology has been verified, conclusions can be drawn to confirm or not if the ontology has been modeled correctly according to certain specified quality criteria.

2.3.1 Criteria

Ontology verification is made based on some evaluation criteria. These criteria tell us: if an ontology is good or not; if the intended specifications have been implemented correctly; if the encoded knowledge can be inferred correctly from the created ontology etc.

Evaluation of an ontology for some criteria is possible within a specially designed system. This implementation and ontology verification depends on the initially defined requirements. Before the implementation, first need to define several evaluation criteria that an ontology has to satisfy to assess its quality.

In the following, we will list a number of criteria which comply with those presented by [Hitzler et al., 2010] and [Vrandečić, 2010]. We will try to aggregate them to form a coherent and concise set, and to discuss their applicability and relevance for the Wikidata ontology.

Consistency, *coherency*, *completeness* are considered logical criteria. These criteria have characteristics that can be checked on a logical level [Hitzler et al., 2010].

According to [Vrandečić, 2010], the **consistency** criteria says if an ontology is interpretable at all, and if there are no contained contradictions. The *logical consistency* which is a part of it, tells if the contained comments and descriptions are not contradicting to each other and if they are linked correctly with the set of axioms from the ontology. Thus, the **consistency** criteria is one of the important conditions for an ontology to be useful.

Now, we will explore an example where a logical inconsistency can appear. Let's assume that an ontology contains the class entity `Java` which is defined as *"Java is an island in the archipelago of Indonesia."* and contains the following axiom

`ClassAssertion(`*`Island Java`*`)`

The inconsistency appears when the following axiom is added to the ontology:

`ClassAssertion(`*`Programming_language Java`*`)`

These problems are often the result of an incorrect ontology modeling or a badly management.

Besides the presented logical criteria, ontologies can be automatically evaluated for other criteria that indicate the possible presence of problems. These criteria refer to the structural representation of the ontology. For example, the detection of taxonomy *cycles* can be done directly from the ontology.

```
SubClassOf(Informatics Faculty)
SubClassOf(Faculty University)
SubClassOf(University Building)
SubClassOf(Building Informatics)
```

From the example above, we can infer that in this ontology is present a *cycle*. If to follow the subsumption statements we will arrive to the first axiom, which means that "this taxonomy collapses semantically" [Hitzler et al., 2010]. We can infer that the presented classes `Informatics`, `Faculty`, `University`, and `Building` are all equivalent. Therefore, it's unlikely to have so many equivalent classes in the ontology. It can be a sign that ontology has a problem.

Each of the class entities in the ontology can have several characteristics, such as *identity*, *unity*, *dependence*, and *rigidity*. The last characteristic shows if a class is considered rigid in the class hierarchy or not. As it is described in [Hitzler et al., 2010], the rigidity means that every instance of that class cannot change its state without losing its existence.

Accuracy criterion tells whether the ontology is modeled accurately with respect to the initial specifications. Also it states if the contained axioms in the ontology comply with the represented knowledge about the domain [Hitzler et al., 2010]. Of course, it is considered one of the important requirements and therefore evaluation criterion for an ontology.

[Vrandečić, 2010] says that a good accuracy of a modeled ontology comes from "correct definitions and descriptions of its classes, properties, and individuals". Correctness is coming from the correspondence with the defined standards and conceptualizations. Defined axioms should encode the possible interpretations of an ontology, so that the resulting models are compatible with the intended conceptualizations.

Unfortunately, this criterion is impossible to check fully automatically. This task relies on humans and how they represent the real-world knowledge. But there are some possibilities to check if exists some problems and omissions where a domain is modeled incorrectly to the initial intention [Hitzler et al., 2010].

The biggest goal for us is to achieve good results for all analyzed criteria, as long as they are not contradicting each other. For example, the **conciseness** criterion that tells if there are some irrelevant elements about the domain of interest, and **completeness** criterion that describes if this domain is "appropriately covered" [Vrandečić, 2010] – sometimes can have a contradictory results. Usually, the person who is in charge for ontology evaluation, needs to decide which criteria is better for the verified ontology and which methods to apply to analyze it.

Besides the criteria listed above, exist other criteria which can be checked to improve the quality of the modeled ontology. For example: *adaptability*, *clarity*, *computational efficiency* etc. All these criteria are explained in more details in [Vrandečić, 2010].

2.3.2 Methods

To assess the quality of an ontology, some **evaluation methods** are applied. These methods describe the way how to verify an ontology for several criteria or specify the expected results. [Vrandečić, 2010] defines the evaluation criteria, which are checked in this case, as premisses for some evaluation methods, where the methods give the results how the ontology satisfies or not these quality criteria.

In the following we will describe few of the evaluation methods that are applicable to the scenario of the Wikidata ontology evaluation. These evaluation methods are compiled from [Vrandečić, 2010].

Maximum depth of the taxonomy

[Vrandečić, 2010] defines the *maximum depth* of a class hierarchy as *"...the largest existing path following the inheritance relationships leading through the taxonomy"*.

However, he does not agree fully with this definition. There are some situations when the maximum depth of a class hierarchy in the taxonomy can be infinite, which is almost impossible and not very useful result. This is happening because of *cycles* in the taxonomy.

Consider the following ontology example proposed by Vrandečić [2010], shown in Figure 2.4:

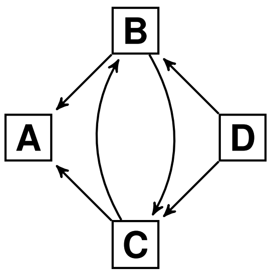


FIGURE 2.4: Example for a cycle hierarchy path.

```
SubClassOf(B A)    SubClassOf(B C)
SubClassOf(C A)    SubClassOf(C B)
SubClassOf(D C)    SubClassOf(D B)
```

The longest *path* in this taxonomy is **4** (either *DBCBA* or *DCBCA*), even though we would expect the maximal depth of the class hierarchy to be **3** (since *B* and *C* are on the same hierarchy level which means that these classes are equivalent).

Semantic similarity measure

Another characteristics of a modeled ontology is semantics. It is important to represent the semantics in most accurate way to run correctly automated reasoning.

Semantic similarity measure method was firstly introduced by [Alani and Brewster, 2006]. To understand how this method works, [Vrandečić, 2010] came with an example which is shown below.

In Figure 2.5 we can see two ontologies which have similar semantics. The hierarchy from the right side has several *"subclass of"* relations which are redundant. If to remove them, the semantics of both hierarchies remains unchanged. Even though, the *semantic similarity measure* method should give the same result for both ontologies, which is not the case.

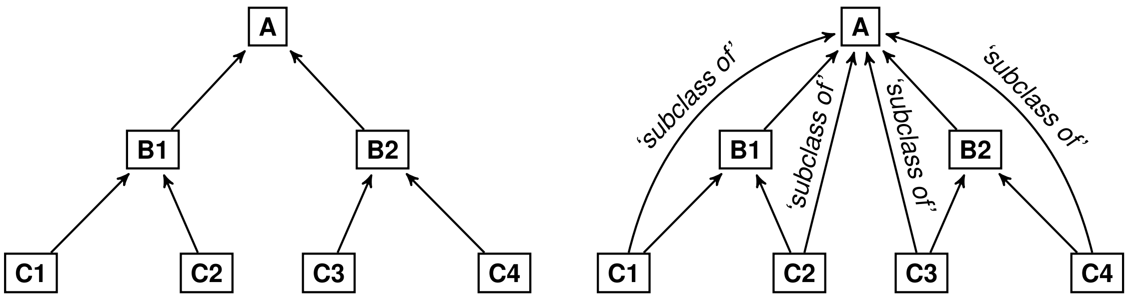


FIGURE 2.5: Two class hierarchies with identical semantics.

Actually, any of the classes *C1, C2, C3,* or *C4* can easily be connected using a subsumption relation to the class *A* from the top. This will not change the general semantics of the hierarchy. As we know, the subsumption relation is a transitive property which makes the new added links redundant, because each of the classes is connected to the top in two ways.

The presence of some redundancies in the ontology, not always leads to complications. Even though a redundant subsumption axiom may be represented in the ontology without breaking the quality measurements of the ontology. The inclusion of such an axiom often has a reason. In human-engineered ontologies it may express a specific relation (even though that does not mean anything in the formal semantics). These explicit connections may indicate some semantic closeness that the formal semantics do not properly capture.

Chapter 3

Application Design

The main contribution to this thesis is implementation of an automated system for taxonomy browsing and ontology evaluation. The Wikidata project and its ontology were chosen as a basic scenario for our application.

In Section 3.1 we describe the most important components and functionalities of the implemented system. In the same section, we describe the main approaches and strategies of designing this application. To organize the developing process, we elaborated user stories that are presented in Section 3.2.

Besides the taxonomy browser, we give some methods and tools for the ontology evaluation task. To evaluate the ontology of Wikidata, we check it for several criteria. The errors and redundancies that are found in the ontology are generalized in several distinctive cases and schematically are shown as patterns in Section 3.3, together with some representative examples from the Wikidata ontology.

3.1 Components and Functionality

Before we have started the implementation, we did a small research in the area to collect information regarding the main strategies and techniques about how to develop a system for taxonomy browsing and visualization of a class hierarchy. Since we have also decided to implement additional methods and tools which would allow to assess the quality of ontology, we need to adjust the developing process to this specific scenario and to find good solutions how to design the system.

At the beginning we decided to design the application with two separate components (frontend and backend). The first component will extract data from the Wikidata repository and the second component will allow users to navigate through the Wikidata taxonomy and to verify it for several quality criteria. For more details about the architecture of the developed application, see Section 4.1.

The first step to make the application functional, is to supply it with the data that will be processed later. The Backend component is designed to extract the data from the weekly data dumps from an external repository that is accesible freely. The downloaded data from the external repository contains millions of items. To reduce the amount of processed data, we extract only the *class entities*[1] that are contained in the class hierarchy.

To make the application more interactive and dynamic, we decided to build the Frontend component using the JavaScript programming language. In this way, users can interact with the system, without reloading the web page to get some results. The Frontend component allows users to browse the Wikidata taxonomy. Navigation on the taxonomy, up to the root class or down to the leaves, gives a general overview for the modeled ontology.

Besides the taxonomy browsing, the system gives some methods to verify the Wikidata ontology for several evaluation criteria and outputs the results in different sections, according to the analyzed criteria. Users will be able to select each of the found issues and to navigate to the area of the taxonomy where it is contained, for a complete graphical representation of the problem.

We know that the Wikidata ontology is maintained by different editors. Sometimes they discuss about a specific situation where an error or an omission can occur in the ontology. To help them to visualize the segment of the taxonomy where such problems can occur, we implemented a possibility to share the current state of the class hierarchy with some other users or to save it for a later visualization.

Within the application, the user will be able to view additional details about a class entity, such as: label, ID, description, how many direct up or down neighbors are still hidden or displayed in the graph etc. The user can also find the direct link to the Wikidata.org website where the selected class entity is located, to see the full list of statements.

For the case when too many nodes are displayed in the graph, the user will have the possibility to adjust space between the nodes. Two options for spacing: *small* and *large*, will allow user to change it for a better visualization of the graph.

The system is designed to browse the taxonomy and to visualize the class hierarchy that is contained in the Wikidata ontology. This class hierarchy is constructed using the class entities and two of their relation properties: *'subclass of'* and *'instance of'*. In order to display only the class subsumptions in the class hierarchy, it will be possible to exclude the *'instance of'* relation, which is represented by another type of edge in the graph, by disabling it from the general menu.

[1] "A class entity is a type of item which refers to a group of instances." [Wikidata.org]

3.2 User Stories

A user story is a description in one or more sentences of a task or a job that a user or a component of an application needs to do. Usually, the user story is written in a simpler language, without some technical terms or notions. User stories are important in the *agile development methodology*[2] that we use for our implementation.

To summarize all the functional tasks that are proposed to be implemented in the application, we formulate some general user stories. We can divide these stories into three different groups: a first group regarding the Wikidata *taxonomy browsing*, a second group about *ontology evaluation*, and the last third group about *data extraction* from the Wikidata repository.

The complete list of user stories is presented in Table 3.1, Table 3.2, and Table 3.3.

3.3 Relation Properties Patterns and Examples

In this section we show what kind of general errors and redundancies can be found in the Wikidata ontology. All the errors and redundancies have been generalized in some representative patterns. These patterns will help us in the ontology evaluation task to detect any possible problems. The errors and redundancies are generated by two types of relation properties (*'subclass of'* and *'instance of'*), which belong to the class entities from the taxonomy.

The taxonomy from Wikidata is represented as an acyclic graph. The class entities are displayed as nodes of the graph, and *'subclass of'* and *'instance of'* relationships as edges.

In the following patterns representation, we display nodes of the graph as black circles ●. Edges are shown with two different types of lines: solid lines for *'subclass of'* relationships, and dashed-lines for *'instance of'* connections. We consider the direction of the relations from the bottom to the top. All the situations that we consider as errors are depicted in Figure 3.1.

In Figure 3.2 we present three different examples of errors that can be found in the Wikidata ontology. These examples correspond to the generalized patterns shown in Figure 3.1.

To generalize the redundancy cases from the Wikidata ontology created by the relation properties, we have two types of patterns, shown in Figure 3.3. These patterns are exemplified with two concrete situations from the Wikidata ontology (Figure 3.4).

To understand why all these cases are considered errors or redundancies in the taxonomy, see Section 4.3.3 and 4.3.4 for more explanations.

[2] "a software development method ... which promotes adaptive planning, evolutionary development, ... continuous improvement, and encourages rapid and flexible response to change." [Wikipedia.org]

TABLE 3.1: User stories regarding the Wikidata **taxonomy browsing**.

#	*As a user, I want to …*
UN1	… browse the Wikidata taxonomy.
UN2	… have the possibility to expand step-by-step all the upwards and downwards neighbors of each node in the graph.
UN3	… choose which neighbors to expand of those nodes that have more direct up or down neighbors.
UN4	… generate a link of the current graph state, to be shared with other users or saved for a later visualization.
UN5	… have a menu for each node, opened by a right-click, with additional possibilities.
UN6	… have the possibility to select nodes in the graph and to apply the following menu options to the selected nodes: *Select all*, *Hide selected*, *Invert selection*, *Deselect all*, *Expand all path*, *Expand only up neighbors*, *Expand only down neighbors*, and *Undo* the last changes.
UN7	… have the possibility to change the spacing between nodes in the graph, for any suitable visualization.
UN8	… have the possibility to show or hide in the graph the *'instance of'* relation property.
UN9	… have a general map navigation of the displayed graph.
UN10	… have a tooltip which will display all the known details about the class entity, available by hovering on the node.
UN11	… have the possibility to zoom-in and zoom-out on the graph.
UN12	… find a class from the Wikidata ontology by its label or ID, and to visualize it in the class hierarchy.

TABLE 3.2: User stories for the Wikidata **ontology evaluation**.

#	*As a user, I want to …*
OE1	… see the Wikidata ontology evaluation results in a list for all analyzed criteria.
OE2	… have a legend with all the colors and their explanations, used in the graph navigation.
OE3	… know from which date is the latest data dump from the Wikidata repository, used in the application.
OE4	… have a Help page which explains shortly all details how to use and understand the application.
OE5	… select one of the detected issues and to view the area of the graph where that issue is contained.

TABLE 3.3: User stories for the Wikidata **data extraction**.

#	*The backend component should be able to …*
AJ1	… download weekly data dumps from the Wikidata repository.
AJ2	… extract from the external repository all the class entities and their properties (ID, label, description).
AJ3	… collect all the class entities from the Wikidata ontology which have only *'subclass of'* and/or *'instance of'* relation properties.

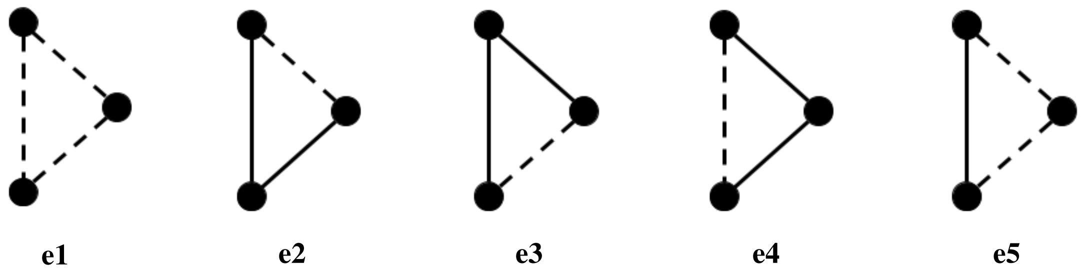


FIGURE 3.1: Error patterns for *'subclass of'* and *'instance of'* relation properties from Wikidata.

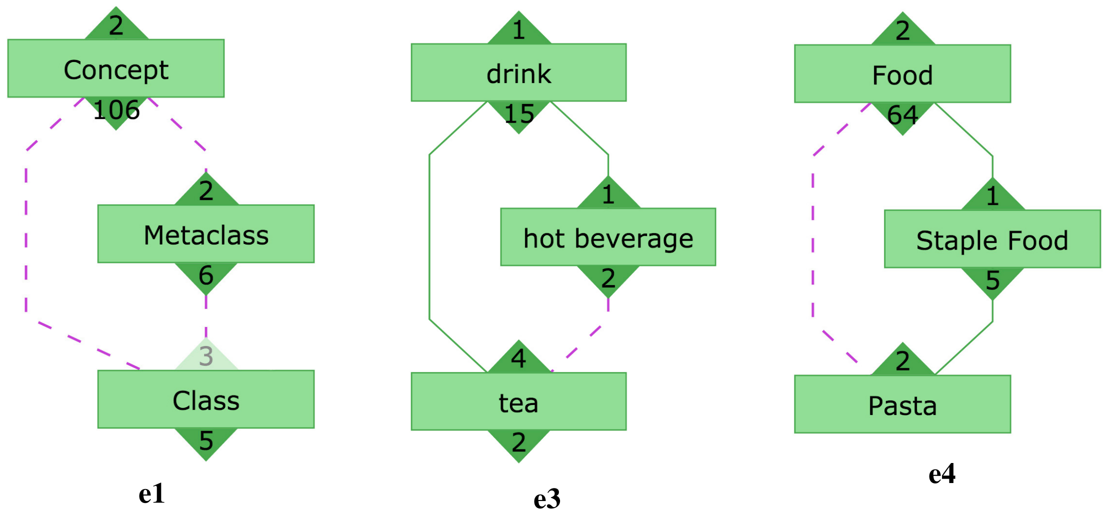


FIGURE 3.2: Error examples for *'subclass of'* and *'instance of'* relation properties from Wikidata (*November, 2015*).

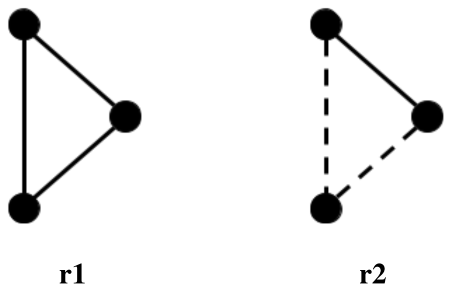


FIGURE 3.3: Redundancy patterns for *'subclass of'* and *'instance of'* relation properties from Wikidata.

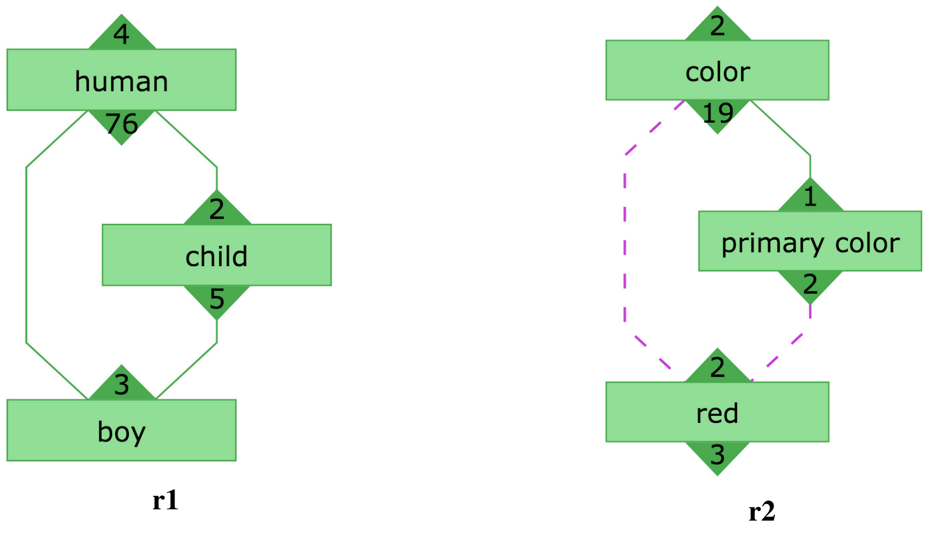


FIGURE 3.4: Redundancy examples for *'subclass of'* and *'instance of'* relation properties from Wikidata (*November, 2015*).

In the following, we describe Algorithm 1, which detects all the redundancies and errors from the Wikidata ontology. It is an iterative algorithm that starts verification in one node of the graph and goes upwards on its path, checking all the parent nodes until the root node is reached. The algorithm counts the number of parsed edges by their type (*'subclass of'* or *'instance of'*), to categorize the found problem as a redundancy or an error based on the checked pattern. In the end, the algorithm returns the pair of the first node and the last node.

Algorithm 1 The algorithm for detection the redundancies and errors of relation properties.

```
 1: for each node in nodes do
 2:     var cSubclasses = 0;
 3:     var cInstances = 0;
 4:     // check 'subclass of' parents
 5:     if node.parent_nodes.length > 0 then
 6:         for each node.parent_nodes (key, value) do
 7:             cSubclasses++;
 8:             PARENTS_ITERATION (value, node, node.id, node.rank);
 9:             cSubclasses = 0;
10:         end for
11:     end if
12:     // check 'instance of' parents
13:     if node.parent_instance.length > 0 then
14:         for each node.parent_instance (key, value) do
15:             cInstances++;
16:             PARENTS_ITERATION (value, node, node.id, node.rank);
17:             cInstances = 0;
18:         end for
19:     end if
20: end for
```

```
21: function PARENTS_ITERATION(node, initNode, nodeID, nodeRank)
22:     var found = FALSE; var source = “Error”;
23:     // check ‘subclass of’ parents
24:     if node.parent_nodes.length > 0 then
25:         for each node.parent_nodes (key, value) do
26:             if initNode.parents_nodes_subclassOf has key then
27:                 cSubclasses += cSubclasses + 2;
28:                 found = TRUE;
29:             else if initNode.parents_nodes_instanceOf has key then
30:                 cSubclasses += cSubclasses + 1;
31:                 cInstances += cInstances + 1;
32:                 found = TRUE;
33:             end if
34:
35:             if found then
36:                 if (cSubclasses==1 and cInstances!=0) or cInstances==0 then
37:                     source = “Redundancy”; found = FALSE;
38:                     if nodeID != key then
39:                         return (value, initNode);
40:                     end if
41:                 end if
42:             else
43:                 for each value.parents_nodes_subclassOf (k,v) do
44:                     if initNode.parents_nodes_subclassOf has k then
45:                         cSubclasses += cSubclasses + 3;
46:                         found = TRUE;
47:                     else if initNode.parents_nodes_instanceOf has k then
48:                         cSubclasses += count_subclasses + 2;
49:                         cInstances += cInstances + 2;
50:                         found = TRUE;
51:                     end if
52:                     if found then
53:                         if (cSubclasses==1 and cInstances!=0) or cInstances==0 then
54:                             source = “Redundancy”;
55:                         end if
56:                         found = FALSE;
57:                         if nodeID != k then
58:                             return (v, initNode);
59:                         end if
60:                     else
61:                         if v.rank < nodeRank then
62:                             // check parents of next upper level
63:                             return PARENTS_ITERATION (v, initNode, nodeID, v.rank);
64:                         end if
65:                     end if
66:                 end for
67:                 . . . // repeat [43-66] for ‘instance of’ parents
68:             end if
69:         end for
70:     end if
71:     . . . // repeat [24-70] for ‘instance of’ parents
72: end function
```

Chapter 4

Implementation

There are already a number of different applications and tools for taxonomy browsing, but none of them is built to give possibility to browse the taxonomy from Wikidata. The main goal of this thesis is to implement a system that allows users to navigate through the Wikidata taxonomy and to visualize the class hierarchy. Also, the same system gives some methods to evaluate automatically the Wikidata ontology for several criteria.

In this chapter, we discuss about the implementation of this system. Figure 4.1 provides a general overview of the developed system's core components and architecture. Section 4.1 explains in details the components of the system, what kind of libraries, tools, and methods were used in the developing process, and describes some technical details of the application. Section 4.2 describes the main algorithms and implemented methods for the Wikidata ontology evaluation task.

4.1 Architecture and Technical Details

Application consists of two separate components: frontend and backend. The Backend component is built using the Java programming language. The main task of this component is to extract data from the Wikidata external repository, to parse the downloaded ontology file, and to collect only the data that we need. The Wikidata ontology has reached millions of items. Obviously, we do not need in our application all the information contained in the ontology. For this reason, we developed a backend component to help us in collecting the data that we need. The extracted data is converted into JSON[1] format and then it is saved into a file which later will be used as data source for the Frontend component of the application.

[1] "an open standard format that uses human-readable text to transmit data objects consisting of attribute-value pairs" [Wikipedia.org]

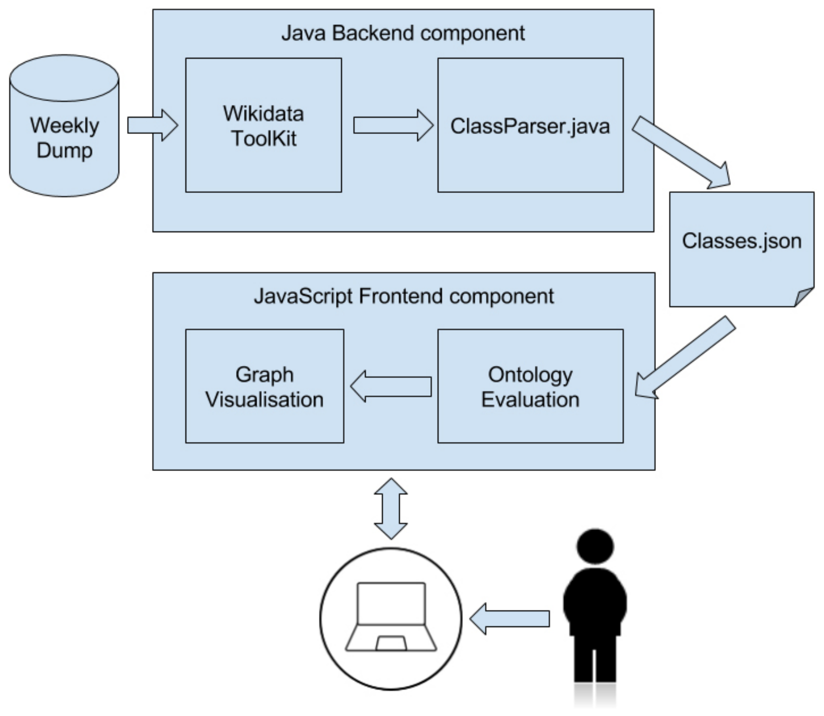


FIGURE 4.1: Architecture of the developed system

For data extraction from the Wikidata external repository, we use Wikidata Toolkit[2] (version 0.4.0) – a collection of libraries which gives access to the data stored on the Wikidata repository and to other available services. In our application we use the weekly data dumps from Wikidata which are provided freely to the public. Weekly dumps are stored in a GZIP compressed JSON format file. The Wikidata Toolkit helps to parse the dump file and to analyze each of the entities contained in the ontology. A separate controller, called *"ClassParser.java"*, was built to help in collecting only the class entities which have *'subclass of'* and *'instance of'* properties. The *'subclass of'* property is present only for items which represent *classes* in the Wikidata ontology. Since we want to construct only the class hierarchy from the ontology in our application, other items are skipped.

Because the Wikidata ontology counts millions of items and most of the items has multiple statements, for a memory saving reason, we store only the values for: ID, the label and description in English, and the value for *'subclass of'* and *'instance of'* statements of each found item.

The process is automated and runs weekly, once a new data dump is provided by the Wikidata. The final generated JSON file (*classes.json*) is saved and used as source data for the second component of the application.

The Frontend component is built mainly using the JavaScript programming language, and HTML5 and CSS3 as markup languages. This component is accessible by any user from a web browser.

[2] https://github.com/Wikidata/Wikidata-Toolkit/releases

As we already explained in Section 3.3, the taxonomy from the Wikidata ontology is displayed as an acyclic directed graph. To build an acyclic graph from the provided data in JSON format, we use additional JavaScript libraries, such as:

- *Dagre.js*[3] (version 0.0.6) – a JavaScript library that makes easy to lay out directed graphs on the client-side;
- *D3.js*[4] (version 3.0) – a JavaScript library for manipulating documents based on data;
- *JQuery*[5] (version 1.9.0) – for general implementation;
- *JQuery UI*[6] (version 1.11) – to adjust user interface.

First, *Dagre.js* librabry renders a graph structure in the virtual memory of the web browser for the given data in JSON format. It produces an acyclic graph out of the provided data and uses the idea of *a minimum spanning tree for ranking* which is explained in more details in [Gansner et al., 1993]. Then, the *D3.js* library uses the rendered acyclic graph to build a visual representation out of it, helping to bring data to life using HTML, SVG, and CSS.

SVG (or *Scalable Vector Graphics*)[7] is an XML-based vector image format for two-dimensional graphics with support of interactivity and animation. Most of the modern web browsers have at least some degree of SVG rendering support which gives possibility to use it in our application. It is relatively fast in generating data graphs with several hundreds of nodes.

To build a graphical representation of the Wikidata taxonomy, the *D3.js* library is using the following SVG elements:

- **path** – used to display graph edges as lines with different properties;
- **g** – shows the graph node, a container used to group objects;
- **rect** – gives to a graph node the shape of rectangle or triangle, with styling options;
- **text** – displays textual details of the node: the label, number of up and down neighbors.

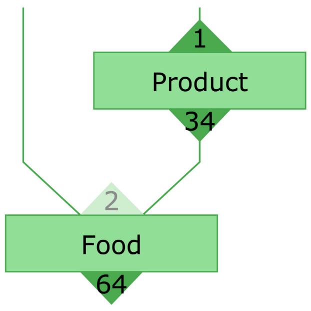
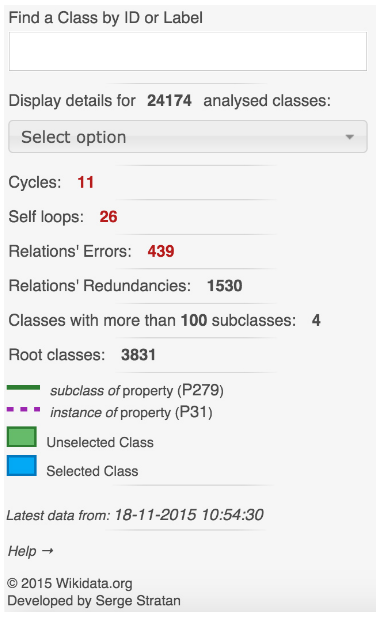


FIGURE 4.2: Graphical representation of the nodes and edges, using SVG elements.

[3] https://github.com/cpettitt/dagre/wiki
[4] http://d3js.org
[5] https://jquery.com
[6] http://jqueryui.com
[7] http://www.w3.org/Graphics/SVG

4.2 Components

When the Frontend component of the application is accessed from a web browser, it generates the results for the ontology evaluation task and displays them in a component placed on the right side of the window. Figure 4.3 shows how the evaluation results are grouped based on the analyzed criteria and how they are displayed in that component.


Find a Class by ID or Label

Display details for **24174** analysed classes:

Select option

Cycles: **11**

Self loops: **26**

Relations' Errors: **439**

Relations' Redundancies: **1530**

Classes with more than **100** subclasses: **4**

Root classes: **3831**

— *subclass of* property (P279)
- - - *instance of* property (P31)
🟩 Unselected Class
🟦 Selected Class

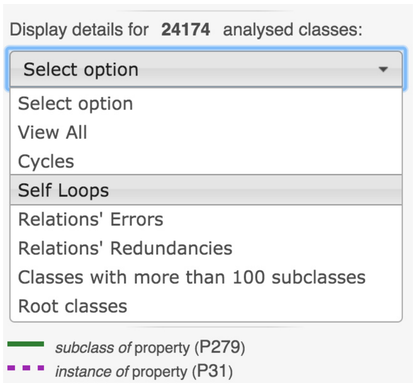
Latest data from: 18-11-2015 10:54:30

Help →

© 2015 Wikidata.org
Developed by Serge Stratan

FIGURE 4.3: The evaluation component from "Wikidata Taxonomy Browser" application (*November, 2015*).

Here we can see how some of the user stories described in Section 3.2 (Table 3.2: *OE1-OE5*) have already been implemented. Figure 4.4 shows how a user can select one of the evaluation criteria to see a complete list of found issues. After that, the user can click on one of them to jump to the area in the Wikidata taxonomy where it is contained (Figure 4.5).

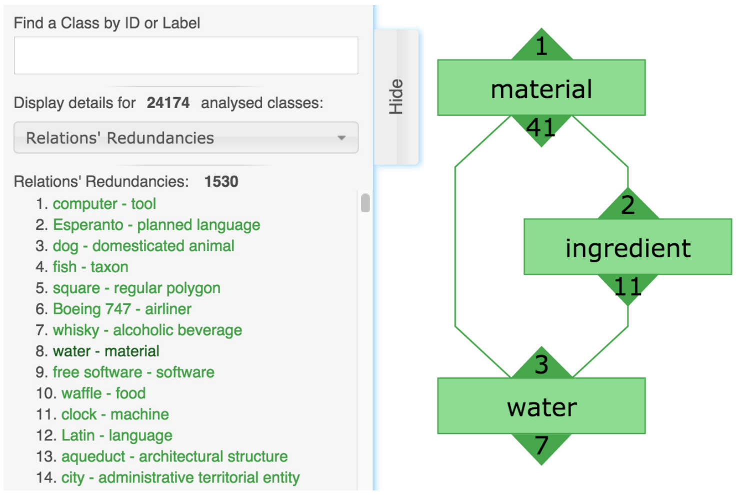

Display details for **24174** analysed classes:

Select option

- Select option
- View All
- Cycles
- Self Loops
- Relations' Errors
- Relations' Redundancies
- Classes with more than 100 subclasses
- Root classes

— *subclass of* property (P279)
- - - *instance of* property (P31)

FIGURE 4.4: Evaluation criteria selection in "Wikidata Taxonomy Browser" application (*November, 2015*).


Find a Class by ID or Label

Display details for 24174 analysed classes:

Relations' Redundancies

Relations' Redundancies: 1530

1. computer - tool
2. Esperanto - planned language
3. dog - domesticated animal
4. fish - taxon
5. square - regular polygon
6. Boeing 747 - airliner
7. whisky - alcoholic beverage
8. water - material
9. free software - software
10. waffle - food
11. clock - machine
12. Latin - language
13. aqueduct - architectural structure
14. city - administrative territorial entity

FIGURE 4.5: An example of how a redundancy is found and displayed graphically in "Wikidata Taxonomy Browser" application (*November, 2015*).

The next component of the application, shown in Figure 4.6, allows to navigate through the class hierarchy, either by selecting nodes in the graph, and then applying some of the main menu options for showing or hiding neighbors, or by expanding step by step all children or parents of a node in the graph. These functionalities are described by the user stories which are listed in Table 3.1: *UN1-UN12*.

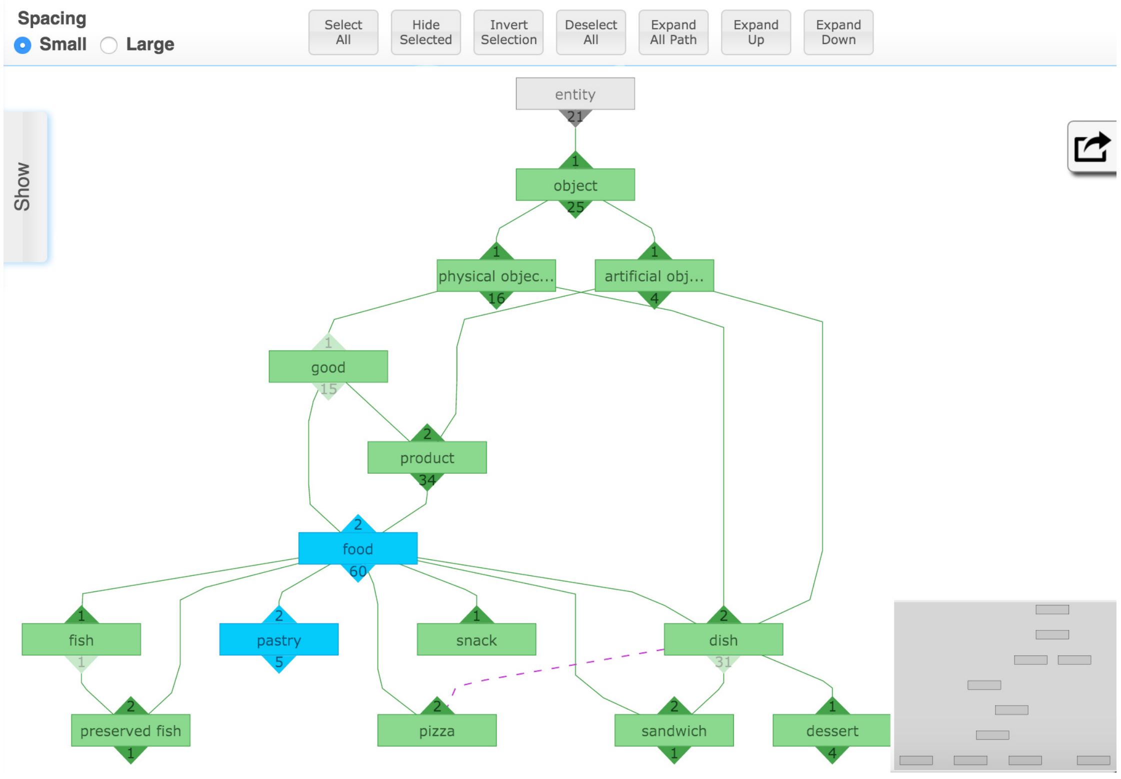


FIGURE 4.6: Navigation through the class hierarchy in "Wikidata Taxonomy Browser" application (*November, 2015*).

4.3 Algorithms

In this section we will present algorithms which have been implemented in the application, toward evaluation of the Wikidata ontology. Some of the used methods have already been introduced in Section 2.3.2.

In total we verify the Wikidata ontology for six evaluation criteria:

- cycles detection;
- self-loops detection;
- checking for errors of *'subclass of'* and *'instance of'* relation properties;
- checking for redundancies of *'subclass of'* and *'instance of'* relation properties;
- identifing classes which have more than 100 direct subclasses;
- root classes detection.

4.3.1 Cycles Detection

A correct graphical representation of the Wikidata taxonomy is an acyclic graph. However, because of some wrong modeling actions which have been done by the Wikidata editors, can lead to a *cycle* in the taxonomy and break the acyclicity[8] of the graph, e.g. when a class is declared as subclass of another class which is situated on a lower rank in the class hierarchy. The existence of a cycle in directed graph, can be determined by whether the *Depth-First Search Algorithm (DFS)*[9] finds an edge that points to an ancestor of the current node. It means that the graph contains a *back edge*. This approach is described in more details by [Gansner et al., 1993] in his work. Such situations should be detected, considered as errors, and fixed.

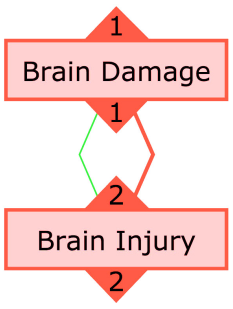


FIGURE 4.7: A cycle example from the Wikidata taxonomy (*November, 2015*).

In Figure 4.7 is presented one of the cycle examples detected in the Wikidata taxonomy. Classes *brain damage (Q720026)* and *brain injury (Q3280669)* are part of a cycle in the taxonomy, since the path which starts in one of the nodes, leads in the end to the same node. The edge with a

[8] the condition of being acyclic
[9] "an algorithm for traversing or searching tree or graph data structures" [Wikipedia.org]

bigger width is considered a *back edge* that has a wrong direction in the class hierarchy. Also the different colouring of the nodes and edges shows the presence of the cycle in the taxonomy.

4.3.2 Self-loops Detection

Second criterion that we analyze is the presence of self-loops in the taxonomy. Sometimes the Wikidata editors declare intentionally or by mistake the *'subclass of'* or *'instance of'* relation properties to the class entity itself. Occasionally this can lead to errors. A self-loop is an edge that connects a node to itself. We should avoid such situations and to fix them if exist.

The pseudo-code for self-loop edges detection algorithm is presented below:

Algorithm 2 Self-loop edges detection algorithm

```
1: for each node in nodes do
2:     if node.parent_nodes.id==node.id or node.parents_nodes_instanceOf.id==node.id then
3:         return node
4:     end if
5: end for
```

Figure 4.8 shows an example of a self-loop from the Wikidata taxonomy for the class *music school (Q184644)*. Because the developed application does not support visualization of self-loop edges, we cannot see it graphically, but we can notice it from the different colouring (pink) of the node.

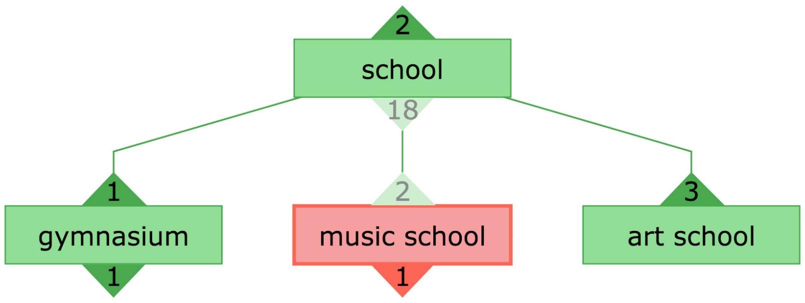


Figure 4.8: A self-loop example for class *music school (Q184644)* from the Wikidata taxonomy (*November, 2015*).

4.3.3 Errors Detection for Relation Properties

As it was described in Section 2.2.2, a *subclass of (P279)* property from Wikidata is a *transitive* property – a binary relation over a set of two classes, and *instance of (P31)* property represents the item as an instance of some class entity. These properties can create errors in the graph, if they are not used correctly.

Let's analyze a general example, which explains how an error is created when the *'subclass of'* and *'instance of'* relation properties are not used correctly according to their characteristics.

From Figure 4.9 we can infer that the class *A* is in a *'subclass of'* relation with the class *C*, based on the transitivity property of the *'subclass of'* relation. If whenever the class *A* is related to another class *B*, and the class *B* is in turn related to the class *C* with the same *'subclass of'* relation type. On other side, if the class *A* is in an *'instance of'* relation with the class *C*, then this situation is considered an error, because an entity cannot be a *'subclass of'* and an *'instance of'* another class simultaneously.

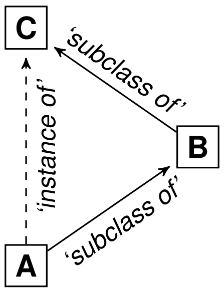


FIGURE 4.9: General representation of an error with *'subclass of'* and *'instance of'* relation properties in Wikidata.

An error example for relation properties from the Wikidata taxonomy is shown in Figure 4.10. Here we can see that the class *noun (Q1084)* is an instance of the class *part of speech (Q82042)*. However it is also a subclass of the same class which can be inferred from the transitivity property of the *'subclass of'* relation and its connection via the class *name (Q82799)*.

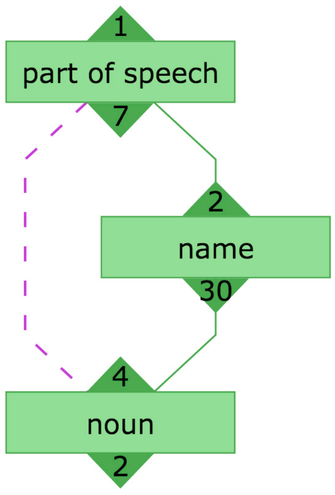


FIGURE 4.10: Example of an error from the Wikidata taxonomy (*November, 2015*).

Errors in the taxonomy are detected based on several patterns. For a complete list of the used patterns, additional examples, and the pseudo-code for the implemented algorithm, see Section 3.3.

4.3.4 Redundancies Detection for Relation Properties

Similar to errors created by incorrect use of the relation properties, we detect redundancies which are not considered strong errors, but rather some duplications.

For example, Figure 4.11 shows one of the redundancy examples. We can see that the class *boy (Q3010)* is linked via a *'subclass of'* relation to the class *child (Q7569)*, which in its turn is linked with the same relation type to the class *human (Q5)*. The redundancy is created by the direct *'subclass of'* relation between the classes *boy (Q3010)* and *human (Q5)*, which can be removed, because it is resulting from the transitivity property of the *'subclass of'* relation.

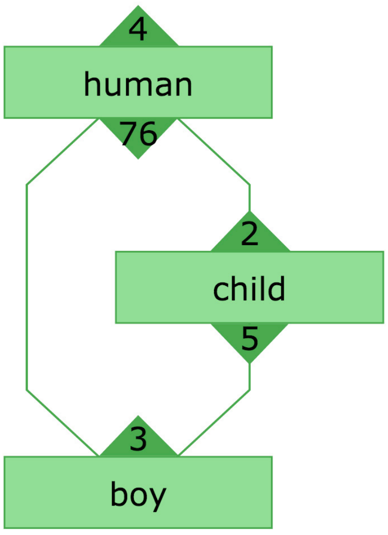


FIGURE 4.11: Example of a redundancy from the Wikidata taxonomy (*November, 2015*).

The complete list of redundancy patterns for relation properties can be found in Section 3.3, with some additional examples.

4.3.5 Identifing classes which have more than 100 direct subclasses

Classes which have more than 100 direct children linked with *'subclass of'* relation, also need to be analyzed. These classes are not treated as being a part of errors, but it is recommended to have less direct up and down neighbors of a class for a better visualization of the taxonomy and a better semantic representation of the entities.

As an example from the Wikidata taxonomy can be used the class *disease (Q12136)* (Figure 4.12) which has hundreds of direct subclasses representing different type of known diseases. It is impossible to visualize all the direct neighbors of this class. It is recommended to link these direct subclasses to other entities which can express more directly the real semantics.

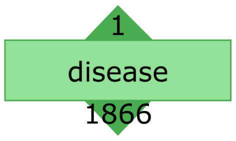


FIGURE 4.12: Class *disease (Q12136)* and the amount of its direct neighbors (*November, 2015*).

The pseudo-code for algorithm that distinguishes classes which have more than 100 direct subclasses, is presented below:

Algorithm 3 Algorithm for detection classes with more than 100 direct subclasses.

```
1: for each node in nodes do
2:     if node.parent_nodes.length ≥ 100 then
3:         return node
4:     end if
5: end for
```

4.3.6 Root Classes

Another problem for the Wikidata taxonomy is the presence of a big number of root classes without any direct up neighbors (parents). As a recommendation, is better to have fewer root classes in the taxonomy as possible. Here are some examples of root classes from the Wikidata ontology: *pottery (Q11642)*, *fashion (Q12684)*, *symmetry (Q12485)*, *fee (Q11765)* etc. (Figure 4.13).

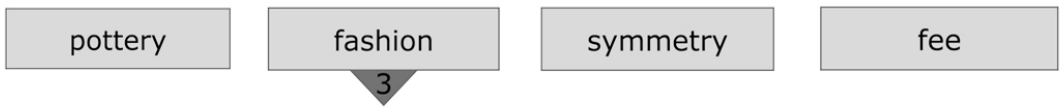


FIGURE 4.13: Some examples of root classes from the Wikidata ontology (*November, 2015*).

The pseudo-code for detection of root classes in the taxonomy is listed below:

Algorithm 4 Algorithm for root classes detection

```
1: for each node in nodes do
2:     if node.parent_nodes == $Empty$ then
3:         return node
4:     end if
5: end for
```

Chapter 5

Evaluation

To demonstrate capabilities and usefulness of the “Wikidata Taxonomy Browser” application, we tested it for different technical criteria. The complete performance testing results are presented in Section 5.1. In Section 5.2 we show what kind of problems we confronted during the developing process and what approaches we applied to fix them. Also, we present what we have learned from the Wikidata ontology evaluation and what problems have been found (Section 5.3). Finally, we discuss about the feedback received from the users after the official launch of the application to the Wikidata community (Section 5.4).

5.1 Performance

To evaluate the performance and efficiency of the “Wikidata Taxonomy Browser” application, regarding the taxonomy browsing and evaluation tasks, we did several tests to check it, and then compared the results.

The performance testing of the application has been done separately for each of the application components (frontend and backend). For more details about the architecture of the developed system, see Section 4.1. The most important factor is the task completion time measurement which shows how fast the system is in computing the results.

5.1.1 The Frontend Component Performance Testing

To check the performance of the Frontend component, we did the following testing tasks:

- **T1**: to search a class by *label* or *ID* in the Wikidata taxonomy;
- **T2**: to run a full evaluation of the Wikidata ontology for all analyzed criteria;
- **T3**: to display the complete path of a chosen class in the class hierarchy.

TABLE 5.1: Runtime comparison for the Frontend component testing (*November, 2015*). The time is given in seconds.

System	T1.a	T1.b	T1.c	T2	T3.a	T3.b	T3.c
Mac OS (El Capitan v. 10.11.1), Chrome (v. 46)	0.566	0.574	0.522	2.741	0.590	0.173	0.278
Mac OS (El Capitan v. 10.11.1), Firefox (v. 42)	0.643	0.605	0.596	2.399	0.891	0.223	0.412
Mac OS (El Capitan v. 10.11.1), Safari (v. 9.0.1)	0.458	0.448	0.441	4.267	0.562	0.162	0.243
Windows 8, Chrome (v. 46)	3.345	3.795	3.473	7.783	3.555	1.448	1.377
Windows 8, Firefox (v. 42)	4.651	3.982	4.429	7.015	5.967	2.128	2.944

For the initial setup of the experiment, the tasks **T1**, **T2**, and **T3** have been tested on Windows 8 and Mac OS (El Capitan v. 10.11.1) machine platforms. For tests were used the latest version of the web browsers which support JavaScript, such as Chrome (v. 46), Firefox (v. 42), and Safari (v. 9.0.1). On the moment of testing, the application was using the Wikidata ontology dump from 16th of November, 2015.

The tasks **T1** and **T3** have been repeated three times with different searched class entities which are situated on different *levels*[1] in the class hierarchy:

- **T1.a / T3.a**: class *food (Q2095)*, level 5;
- **T1.b / T3.b**: class *cake (Q13276)*, level 8;
- **T1.c / T3.c**: class *Gouda (Q593675)* , level 17.

The complete results of testing and the runtime comparison are displayed in Table 5.1. The total amount of class entities from the Wikidata taxonomy used in the application on the moment of testing, was 24,030 classes (*November, 2015*). Analyzing the evaluation results, we can conclude that our system behaves differently for particular machine platforms and web browsers. It is obvious that the runtime results depend on the technical configuration of the used machine. Figure 5.1 shows the difference between using application on Windows and Mac OS platforms.

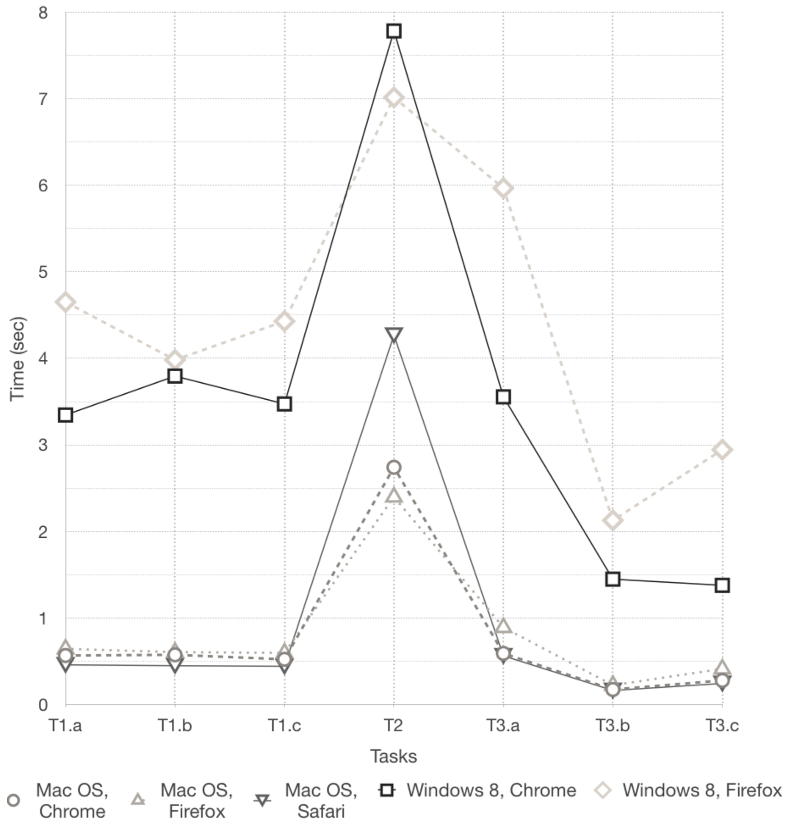
The values from Table 5.1 represent the time in which the system parses the whole taxonomy to find the searched entity, to execute the task, and to output the results. We are happy to see that the Task **T2** is executed in less than 30 seconds, because a typical web browser returns an average timeout response already after this limit. Otherwise, we could run in a problem. See Section 5.2 for other problems that we have encountered during the developing process, and solutions that we applied.

We run the test with three different class entities that are situated on different levels in the hierarchy. We expected to see a difference in time for running the tests. But the difference is very small and not important. We can conclude that the location of the class entities in the

[1] the maximal distance from the root class *Entity (Q35120)* to the class itself

hierarchy does not have a big impact to the runtime of the system. Probably the difference in time will be more obvious for bigger taxonomies which include thousands of entities.

From the charts represented in Figure 5.1, we can conclude that the Chrome browser is slower when is loading the first data (executing the task **T2** for ontology evaluation), but then outputs the results faster for the tasks **T1** and **T3**. Where the Firefox browser is faster in executing the task **T2** first and then the rest of the tasks. Based on this results, we can conclude and recommend to use the system with Chrome browser, which can give a better usability of it.



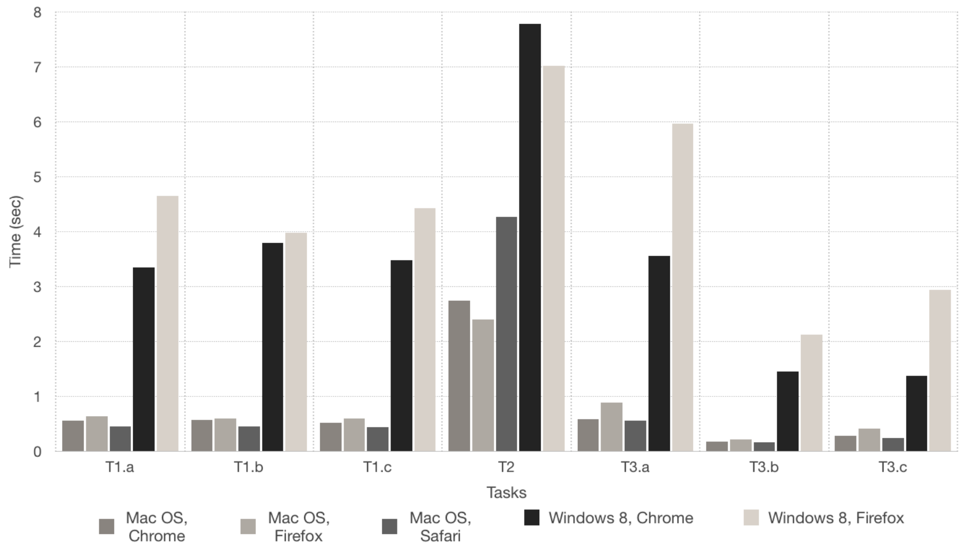


FIGURE 5.1: Runtime comparison charts for the performance testing of the Frontend component.

TABLE 5.2: Runtime comparison for the performance testing of the Backend component (*November, 2015*).

System	T4.a	T4.b	T4.c
MacBook Pro • *Mavericks v10.9.5;* • *Processor 2,4 Ghz Intel Core 2 Duo;* • *6 GB RAM*	31min 21sec	32min 11sec	33min
MacBook Pro • *El Capitan v10.11.1;* • *Processor 2,2 GHz Intel Core i7;* • *16 GB RAM*	12min 55sec	13min	13min 09sec

5.1.2 The Backend Component Performance Testing

To test the Backend component, we run a single task **T4**: "data extraction from the external Wikidata repository". The system has been tested only on Mac OS (Mavericks and EL Capitan) machine platforms with different configurations, on which have been installed and runs the Java SE Development Kit (version 8u66)[2]. The tests have been executed several times with different versions of data dumps. Note, that every week new class entities are added to the Wikidata ontology, such that the quantity of analyzed data is growing continuously. This should influence the runtime performance of the application. For more processed data, we will have a longer time for computations.

The task **T4** has been repeated three times, using several versions of the data dumps from the Wikidata repository which include different amount of entities in the ontology:

- **T4.a**: weekly dump from *26th of October, 2015* (with 18,699,771 entities in total);
- **T4.b**: weekly dump from *9th of November, 2015* (with 18,896,048 entities in total);
- **T4.c**: weekly dump from *16th of November, 2015* (with 18,959,744 entities in total).

Table 5.2 displays runtime comparison of the task **T4** for the Backend component. The time shown in results, is taken by the system to parse the whole ontology file, to collect the entities and statements that we need, to convert the data into JSON format, and to save it into a file.

The results tell us that the runtime for the system is increasing with the amount of growing processed entities. As we can see, the system is running faster on machines with a better configuration. A machine with bigger RAM memory and a faster processor, increases the speed of computations. However, the difference in time for processing different data dumps for machines with a better configuration, is very small. We can estimate that the system can be useful even when the amount of processed data will be doubled.

[2] http://www.oracle.com/technetwork/java/javase/overview/index.html

5.2 Technical Problems

Throughout the first phase of designing and developing application, we had encountered several problems that were necessary to fix to continue the implementation. The first problem was a **very slow response time** for processing big amount of data from the Wikidata ontology. We could not admit slow response time for the system on the taxonomy visualization and ontology evaluation tasks.

For example, the first runtime for the ontology evaluation task was around 4 mins 30 sec to output the first results. It is very slow, since a typical web browser returns an average timeout response already after 30 seconds. At this point, the main goal for us was to improve the used algorithms and libraries, to reduce the response time to the minimum.

Several changes have been done to the JavaScript libraries that we used in our application (see Section 4.1 for a complete list of the used libraries). Especially, *Dagre.js* and *D3.js* libraries were adapted to our scenario. Many of the additional features of these libraries have been deleted to improve the performance. In this way, the total time for the graph construction was reduced to the minimum. We achieved the highest response time for the ontology evaluation task, to be smaller than the default timeout response of a web browser.

Another technical problem that we confronted, was **running out of virtual memory** for data extraction task. Every dump file contains millions of entities from the Wikidata ontology. Collecting the class entities which represent only a small part of the whole ontology, did not help us to escape from this problem.

To solve this problem we decided to store for each of the class entities only the properties and statements that we need in our application, such as: *ID*, *label*, *description*, the values for *'subclass of'* and *'instance of'* relation properties. Another decision was to transform the *ID* of each class entity, which is stored as a string value, into an integer type, by removing the *"Q"* char from it (To remark that integers occupy less memory). Finally we could manage to run a full data export without exceeding the memory limit of the machine that is used in the processing. Usually the memory limit depends on the machine's configuration and other installed tools that are used.

5.3 What we've learned about the Wikidata Ontology

After several evaluations of the Wikidata ontology we have learned about the main problems that occur often in the ontology. Many of the class entities have been added to the ontology without a correct configuration of their statements (i.e. taxonomy relations, such as *'subclass of'* and *'instance of'*).

For example, from the total amount of analyzed class entities (24,174) on 18th of November 2015, we have encountered:

- 11 cycles which are considered strong errors in the ontology modeling;
- 26 self-loop edges;
- about 439 errors created by *Subclass of (P279)* and *Instance of (P31)* relation properties;
- about 1530 redundancies in which *Subclass of (P279)* and *Instance of (P31)* relation properties are involved;
- 4 classes which have more than 100 direct subclass children;
- and 3,831 root classes.

The most of errors and redundancies in the taxonomy appear because the *Subclass of (P279)* and *Instance of (P31)* taxonomy relations are configured incorrectly. A general example explaining how an error is created in the taxonomy, is depicted in Figure 5.2. From the representation below we can see that the class *A* is declared *instance of* the class *B*. In its turn, the class *B* is declared as *subclass of* the class *A*. Because *subclass of* property is a transitive relation, we can infer that all the instances of the class *B* are also instances of the class *A*. In this case we can logically infer that the class *A* is an *instance of* itself, which can not always be the case. In this way is created a cycle error in the taxonomy.

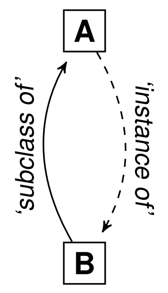


FIGURE 5.2: A general representation of a cycle error from the Wikidata taxonomy.

Another issue that we've noticed, was the presence in the taxonomy of too many root classes. These entities have not been placed correctly in the class hierarchy. Some of them are not linked to any other entity in the taxonomy. A good modeled ontology should not have many root classes.

5.4 Feedback

After the "Wikidata Taxonomy Browser" application was official launched in *October 2015*, we received some feedback from the community of Wikidata. In this way we can conclude: how useful the system is in the completion of a specific task and how users are satisfied using the application in general.

We would like to outline few things about the “Wikidata Taxonomy Browser” application which differentiate it from any other system:

- There is no other tool which would allow to visualize and to browse the taxonomy of Wikidata, beyond the “Wikidata Taxonomy Browser” application that is especially designed to do it.
- The developed application gives possibility to verify the Wikidata ontology for several evaluation criteria, in order to asses its quality.

Here are some of the feedback that we received from users of the Wikidata community:

- *“Great tool ! The error detection is precious !”*;
- *“This is fantastic. :)”*;
- *“Nice work! Thanks for sharing”*.

The feedback from the regular users and the Wikidata community was extremely positive and confirmed our main design objectives.

Also we would like to remark that the application is used frequently in the users discussions. Usually users share the link of our application with some class entity in the hierarchy to visualize its neighbours and the whole path, in a particular discussion on the Wikidata platform. This makes us happy to see that the system is really useful.

Chapter 6

Conclusions

The main goals of this thesis were to implement a system for taxonomy browsing and to provide some tools and methods for ontology evaluation. As a basic scenario for this system, has been chosen the Wikidata ontology, since there are no other systems which would accomplish these tasks for it.

The biggest challenge for us was to find the appropriate solutions to combine in a single application so many different functionalities and requirements. In Chapter 3 we described general designing procedures about developing the application.

The first step toward our goals, was to build an infrastructure which would allow users to navigate through the Wikidata taxonomy. Finding the best JavaScript libraries with the possibility to construct a visual and structural representation of the class hierarchy, was a very important task. Based on the chosen libraries, listed in Section 4.1, we implemented functionalities described by the user stories elaborated in Section 3.2.

In order to provide some methods for ontology evaluation, we decided to divide the application into two different components, to separate the basic tasks: *data extraction from the Wikidata repository*, the possibility to *navigate through the taxonomy*, and to run the *ontology evaluation*. For the Wikidata ontology verification, we elaborated several evaluation criteria which are applicable for this case. Based on them, we distinguish certain patterns that indicate the presence of some errors or redundancies in the ontology (Section 3.3). This application should be very useful to detect any possible errors that appear in the ontology, both for simple users and for the Wikidata editors.

When the application was ready, we did a performance testing to see how useful it is. The runtime results presented in Section 5.1, show how fast is the application in processing the data. Despite the technical problems that we confronted at the beginning of its implementation, such as *very slow response time* and *running out of virtual memory*, we improved our techniques to avoid them and to reach our initial goals (Section 5.2). Finally, we could achieve very good results, from minimising the maximal time for data processing from 4min 30sec to less than 30sec.

The possibility to navigate through the taxonomy of Wikidata, shows how the entities are placed in the hierarchy. And from the ontology evaluation we have learned that the Wikidata ontology contains a lot of issues that have not been discovered before and which need to be fixed to assess the good quality of it.

6.1 Related Work

Our work is closely related to the work of Denny Vrandečić about the ontology evaluation [Vrandečić, 2010]. He presents a theoretical framework and describes several methods which are used to evaluate the web ontologies. Within this thesis we implemented some of his algorithms and methods (see Section 2.3) which are applicable to the case of evaluation the Wikidata ontology. Besides that, we designed our own algorithms (see Algorithms 1, 2, 3, and 4) and evaluation criteria (see Section 3.3) for the cases that we did not find in his work.

Unfortunately, the Vrandečić work about the ontology evaluation is one of the most recent works that could be found. No other work is describing in more details the evaluation process and the procedures that can be adapted and applied to the case of Wikidata, to achieve the goals that we have.

As we know, there are numerous quality evaluation methods which have been suggested in other literature. But only few of them have been properly designed, defined, implemented, and experimentally verified. "The relation between evaluation methods and ontology quality criteria is only badly understood and superficially investigated, if at all", says [Vrandečić, 2010].

6.2 Future Work

The work in this thesis can be seen as the first step toward a complex system which can give the possibility to browse a bigger taxonomy and to improve the quality verification of the Wikidata ontology. This system can be used not only for the Wikidata ontology, but also can be adjusted to any other web ontology.

In the following, we will outline briefly the potential future work, some of them merely represent ideas which might be worth to consider, whereas others are concrete improvements or simply open issues.

6.2.1 Open Issues and Optimization

First, the application which was implemented as a *beta*-version, has many issues that need to be fixed or improved. The technical problems such as *memory limit* or *timeout response*, do not allow us to use completely all the knowledge contained in the Wikidata ontology. It is still required to do some improvements to the algorithms that we are using, to be able to use and process a bigger quantity of data.

A possible improvement of the user interface would be required, in order to increase the usability of the application in general. Possibly that some of the application features which are not used at all by the users or are not understandable how to apply, need to be changed.

6.2.2 Extensions

Besides the taxonomy browsing or the ontology evaluation tasks that have been implemented in the application, would be good to add additional features to make the system more attractive and useful to the users. One of the future tasks is to integrate the Wikidata API services. This would allow the accredited users to make changes to the Wikidata repository directly from our application. For example, to rearrange *"on-the-fly"* the class entities in the class hierarchy or to fix immediately any found errors or redundancies in the ontology, would be possible if to implement such techniques.

On other side, the Wikidata API integration would be a difficult task to implement. The API integration will become a security treat to the Wikidata system. Also, the API it's not always a very convenient way to use when multiple changes need to be done to the repository. So, this implementation is open for discussions and future proposals.

Another future task that needs to be accomplished, is to add additional evaluation criteria to the existing ones. More evaluation criteria will allow to discover from different perspectives any possible problems in the ontology.

The implemented system can be adjusted to other scenarios where it can be used. We can use the system to navigate through other class hierarchies, instead of the one which is created by the class entities and their *'subclass of'* and *'instance of'* relations. For example, we can use the power of the Wikidata ontology which includes a very large knowledge from different areas, to extract the entities and relations which refer to the biological classification. In this way, we can browse in our application the biological taxonomy which classifies all the groups of known organisms. And also, we can evaluate it for several criteria.

Bibliography

Deborah L. McGuinness and Frank van Harmelen. *OWL Web Ontology Language Overview*. W3C Recommendation, 2004. URL https://www.w3.org/TR/owl-features/.

Isabel Cruz, Stefan Decker, Jérôme Euzenat, and Deborah McGuinness, editors. *The emerging semantic web*. Frontiers in artificial intelligence and applications. IOS press, Amsterdam (NL), 2002. ISBN 1-58603-255-0.

Denny Vrandečić. *Ontology Evaluation*. PhD thesis, Karlsruhe Institute of Technology, 2010.

Wikidata.org. https://www.wikidata.org, Dec, 2015.

Tim Berners-Lee, Jim Hendler, and Ora Lassila. The semantic web. *Scientific American*, 2001.

Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

Michael C. Daconta, Leo J. Obrst, and Kevin T. Smith. *The Semantic Web*. Wiley Publishing, Inc., Indianapolis, Indiana, USA, 2003.

Karin K. Breitman, Marco Antonio Casanova, and Walter Truszkowski, editors. *Semantic Web: Concepts, Technologies and Applications*. Springer, 2007.

W3C OWL Working Group. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. W3C Recommendation, 2012. URL https://www.w3.org/TR/owl2-overview/.

Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. *OWL 2 Web Ontology Language Profiles (Second Edition)*. W3C Recommendation, 2012a. URL https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/.

Michael K. Smith, Chris Welty, and Deborah L. McGuinness. *OWL Web Ontology Language Guide*. W3C Recommendation, 2004. URL https://www.w3.org/TR/owl-guide/.

Boris Motik, Peter F. Patel-Schneider, Bernardo Cuenca Grau, Ian Horrocks, Uli Sattler, and Bijan Parsia. *OWL 2 Web Ontology Language Direct Semantics (Second Edition)*. W3C Recommendation, 2012b. URL http://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. *OWL 2 Web Ontology Language Primer (Second Edition)*. W3C Recommendation, 2012. URL https://www.w3.org/TR/2012/REC-owl2-primer-20121211/.

Wikidata Introduction. *Wikidata Introduction*. Wikidata.org, Dec, 2015. URL https://www.wikidata.org/wiki/Wikidata:Introduction.

Wikidata Help. Items. *Wikidata Help. Items*. Wikidata.org, Dec, 2015. URL https://www.wikidata.org/wiki/Help:Items.

Wikidata Help. About Data. *Wikidata Help. About Data*. Wikidata.org, Dec, 2015. URL https://www.wikidata.org/wiki/Help:About_data.

Markus Krötzsch and Denny Vrandečić. Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez, and Denny Vrandečić. *Introducing Wikidata to the Linked Data Web*. Technische Universität Dresden, Germany, 2014.

Janez Brank, Marko Grobelnik, and Dunja Mladenić. A survey of ontology evaluation techniques. *Proceedings of 8th International Multi-Conference of the Information Society*, pages 166–169, 2005.

Robert Porzel and Rainer Malaka. A task-based approach for ontology evaluation. *Proceedings of ECAI 2004 Workshop on Ontology Learning and Population*, August 2004.

Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. Springer, 2010.

Harith Alani and Christopher Brewster. Metrics for ranking ontologies. *Proceedings of the 4th International Workshop on Evaluation of Ontologies for the Web (EON2006) at the 15th International World Wide Web Conference (WWW 2006)*, pages 24–30, 2006.

Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem-Phong Vo. A technique for drawing directed graphs. *Software Engineering, IEEE Transactions on*, 19:214–230, 1993.