Insert here your thesis' task.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Framework for Extraction of Wikipedia Articles Content

Oleksandr Husiev

Department of theoretical computer science
Supervisor: Ing. Milan Dojčinovski, Ph. D.

September 17, 2020

Acknowledgements

I would like to thank my supervisor Milan Dojčinovski, family and friends for their prolonged support during writing this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on September 17, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Husiev, Oleksandr. *Framework for Extraction of Wikipedia Articles Content.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Tato diplomová práce se zabývá extrakcí obsahu Wikipedie pro DBpedia - crowd-sourced projekt. Hlavním cílem této práce bylo vyvinout rámec pro extrakci obsahu, struktury a anotací článků z Wikipedie. Výsledkem je framework, který zpracovává velké skládky XML na Wikipedii v několika populárních jazycích s možností dynamicky přidávat nové jazyky a vytváří čistý textový výstup, odkazy a strukturu stránky ve formátu N-Triples.

Klíčová slova NIF, RDF, propojená data, web škrábání.

Abstract

This thesis describes the development process of the extraction of Wikipedia articles content for a DBpedia, a crowd-sourced community effort. The main goal of this thesis was to develop a framework for extraction of Wikipedia articles content, structure, and annotations. The result is a framework that processes large Wikipedia XML dumps in several popular languages, with the possibility to dynamically add new languages, and produces clean text output, links, and page structure in N-Triples format.

Keywords NIF, RDF, linked data, web scraping, knowledge graph.

Contents

List of Figures

Introduction

Motivation

Knowledge bases are growing up in importance as a Web and enterprise search engine. At the moment, knowledge bases cover only specific niches and are not useful outside of their primary purpose. As part of a broader DBpedia initiative, this thesis has an objective to structure information, store it in a machine-readable form, and provide better ways for information to be collected, organized, searched, and utilized.

A DBpedia is a knowledge base, which information is organized as an open knowledge graph. DBpedia data is served as Linked Data, which opens a new way to access the Web for applications: via browser, automated crawlers, or complex SQL-like queries. For example, current technologies do not allow to combine information about cities, criminal rates, climate, and open job postings into one search. The goal of DBpedia is to enable such queries to happen.

The creation of the Framework for Extraction of Wikipedia Articles Content is important as it will allow the DBpedia initiative to receive formatted article data from the Wikipedia on a regular basis.

Objectives

The main sight of the thesis is to extract structured content from Wikipedia articles. This content can be divided into several main parts: context, structure, and links. Context is the text itself. The structure is how the article is organized and split into sections, subsections and paragraphs, and links are either links to other Wikipedia articles or external websites. Additionally, it is essential to take care of article publication dates, clean up the non-standard articles and sections, and cover other Wikipedia languages.

1. Accept and process input data in the form of Wikipedia XML dumps

2. Extract context.
3. Extract page structure.
4. Extract links.
5. Provide outputs for context, links and page structure in the form of N-Triples.
6. Implement language extensibility.
7. Provide a user interface.

Challenges

The main problem, however, is the current size of Wikipedia. Only a full English Wikipedia dump containing only text and XML structures takes around 16 GB of space. Therefore, a thesis should research the design and implementation of not only functional but an efficient parser with both horizontal and vertical scalability in mind.

An additional challenge lies in the structure of a general wiki page. Not all pages and their components are structured in the same way. For example, some of the pages might have a line break in the middle of a citation link, that can be ambiguously interpreted as a new paragraph start. This can cause unexpected errors and exceptions during the parsing, and the goal of the project is also to minimize and gracefully handle such exceptions.

Chapter 1

Background and related works

1.1 The Concept of Semantic Web

The Semantic Web is essentially a Web of Data, an extension to the Web that links the related data. The collection of Semantic Web technologies (such as Resource Description Framework (RDF), Web Ontology Language (OWL), Simple Knowledge Organization System (SKOS), SPARQL Protocol and RDF Query Language) (SPARQL), etc.) provides an environment where application can query that data, draw inferences using vocabularies, etc. The goal of the Semantic Web is to transform current Internet data and eventually make it machine-readable. The term "Semantic Web" was initially coined by Tim Berners-Lee[1] for a web of data where not simply text, but also meaning, or logical connections can be processed by machines.

The Semantic Web is an extension of the current World Wide Web, defined by standards set by the World Wide Web Consortium (W3C).

Although Semantic Web is a term that has often been criticized as confusing, opaque, and academic, it does nonetheless capture two of the most critical aspects of these technologies:

- **Semantic:** The meaning of the data is not only explicitly represented and richly expressive, but it also "travels" along with the data itself;

- **Web:** Individual pieces of data are linked together into a network of information, just as documents are linked together on the World Wide Web.

Less familiar synonyms to the Semantic Web are the following: Linked Data Web, the Web of Data, Web 3.0, the Enterprise Information Web, or the Giant Global Graph.

1.2 What is Linked Data?

In order to make Semantic Web, or Web of Data, a reality, it is necessary to support a vast amount of data that are manageable and reachable by a variety of Semantic Web tools. Not only access to data but also relationships among data should be provided. This collection of interconnected datasets can also be referred to as Linked Data.

The term was introduced in 2006 by Tim Berners-Lee's four rules for publishing data on the Web[2], stating the following expectations for HyperText Markup Language (HTML) or RDF standards:

1. Use Uniform Resource Identifier (URI) as an identifier.
2. Use Hypertext Transfer Protocol (HTTP) URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other URIs. so that they can discover more things.

While these four rules are the basis of Linked Data and related developments, their exact implementation can be done differently and evolves over time. Specifically, the way URI is now serialized either in RDF/Extensible Markup Language (XML) or via N3(also known as Turtle, or N-triples).

1.2.1 Web Ontology Language Overview

There is a long history of ontological development in philosophy and computer science. Since the 1990s, several research efforts have explored how the idea of knowledge representation (KR) from artificial intelligence (AI) could be made useful on the World Wide Web. These included languages based on HTML (called SHOE), based on XML (called XOL, later OIL), and various frame-based languages and knowledge acquisition approaches.

The OWL is a family of knowledge representation languages for authoring ontologies[3]. Ontologies are a formal way to describe taxonomies and classification networks, essentially defining the structure of knowledge for various domains: the nouns representing classes of objects and the verbs representing relations between the objects. Ontologies resemble class hierarchies in object-oriented programming, but there are several critical differences. Class hierarchies represent structures used in source code that evolve fairly slowly (typically monthly revisions), whereas ontologies are meant to represent information on the Internet and are expected to be evolving almost constantly. Similarly, ontologies are typically far more flexible as they are meant to represent information on the Internet coming from all sorts of heterogeneous data sources. On the other hand, class hierarchies are meant to be fairly static and

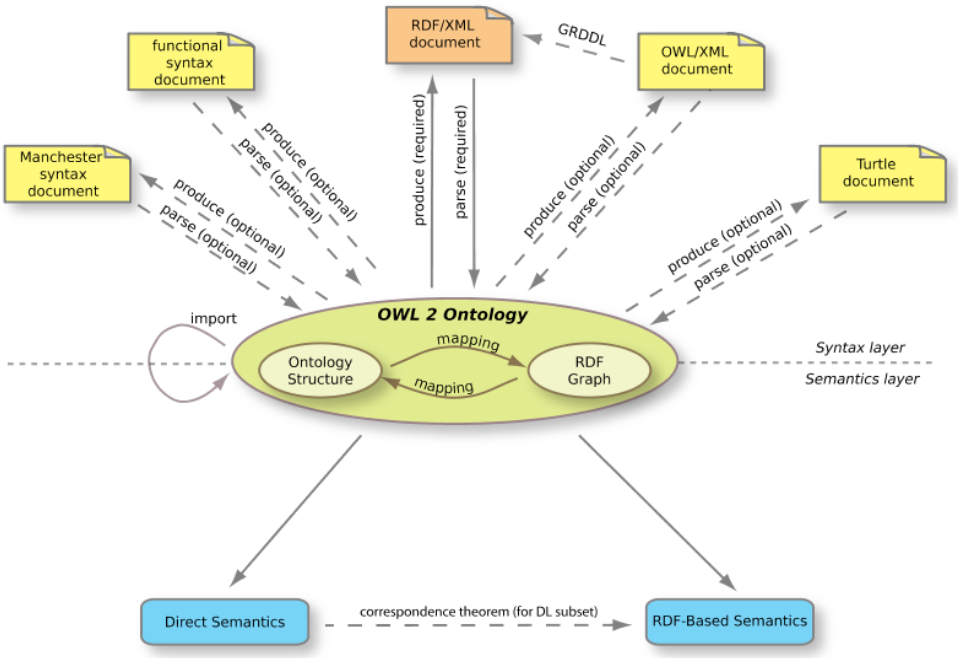

Figure 1.1: The Structure of OWL 2

rely on far less diverse and more structured sources of data such as corporate databases. The OWL family contains many species, serializations, syntaxes, and specifications with similar names. OWL and OWL 2 are used to refer to the 2004 and 2009 specifications, respectively. Full OWL 2 Ontology Structure is shown on 1.1. Most users of OWL 2 will need only one syntax and one semantics; for them, this diagram would be much simpler, with only their one syntax at the top, their one semantics at the bottom, and rarely a need to see what's inside the ellipse in the center.

The OWL languages are characterized by formal semantics. They are built upon the W3C XML standard for objects called the RDF. OWL and RDF have attracted significant academic, medical, and commercial interest.

1.2.2 RDF Description

The RDF is a family of W3C specifications initially designed as a metadata data model. The RDF data model is similar to classical conceptual modeling approaches, such as entity-relationship or class diagrams. It is based on the idea of making statements about the resource in expressions of the form subject–predicate–object, also known as triples. The subject denotes the resource, and the predicate denotes traits or aspects of the resource, and expresses a relationship between the subject and the object.

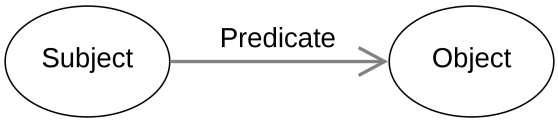

Figure 1.2: Basic RDF Graph

In the statement1.1, the subject is <The Nightwatch>, the object is <Rembrandt van Rijn>, and a predicate <was created by>defines a relation between the subject and the object.

Listing 1.1: Example of an RDF statement

```
<The Nightwatch> <was created by> <Rembrandt van Rijn> .
```

The subject of an RDFstatement is either a uniform resource identifier (URI) or a blank node, both of which denote resources. Resources indicated by blank nodes are called anonymous resources. They are not directly identifiable from the RDF statement. The predicate is a URI, which also shows a resource representing a relationship. The object is a URI, blank node, or a Unicode string literal.

In Semantic Web applications and relatively popular applications of RDF like RDF Site Summary (RSS) and Friend Of A Friend (FOAF), resources tend to be represented by URIs that intentionally denote, and can be used to access, actual data on the World Wide Web. But RDF, in general, is not limited to the description of Internet-based resources.

An RDF database, also called triplestore contains triples of interrelated statements that can be visualized with a network graph. A traditional relational database might split attributes about artworks and features about artists into separate tables. In an RDF/graph database, all these data points belong to the same interconnected graph, allowing users maximum flexibility in deciding how they wish to query it.

RDF is an abstract model with several serialization formats (i.e., file formats), so the particular encoding for resources or triples varies from format to format. Complete list of RDF serialization formats includes:

- **Turtle** - a compact human-friendly format.
- **N3** - format that is similar to Turtle, but allows for additional features like inference, or transformation rules.
- **N-Triples** - a very simple, easy-to-parse, line-based format that is not as compact as Turtle.
- **N-Quads** - a superset of N-Triples for serializing multiple RDF graphs.

- **RDF/XML** - an XML-based syntax that was the first standard format for serializing RDF.
- **RDF/JSON** - an alternative syntax for expressing RDF triples using a simple JSON notation.
- **JSON-LD** - a JSON-based serialization, allows data to be serialized in a way that is similar to traditional JSON.

For example, it is required to write an RDF file, say <http://example.org/smith>, local identifiers, say #albert, #brian and #carol. This RDF file will look differently in XML(1.2), Turtle format(1.3). It can be observed that Turtle format is more concise than the XML one.

Listing 1.2: Example of RDF serialization in XML

```
<rdf:Description about="#albert"
<fam:child rdf:Resource="#brian">
<fam:child rdf:Resource="#carol">
</rdf:Description>
```

Listing 1.3: Example of RDF serialization in N3/Turtle

```
<#albert> fam:child <#brian>, <#carol>.
```

The World Wide Web (WWW) architecture now gives a global identifier "http://example.org/smith#albert" to Albert. Anyone can now use this global identifier to refer to and provide more information about Albert.

In addition to the ways of describing a link, it is important to know when to make a link. One important pattern is a set of data which you can explore as you go link by link by fetching data. Whenever one looks up the URI for a node in the RDF graph, the server returns information about the arcs out of that node, and the arcs in. In other words, it returns any RDF statements in which the term appears as either subject or object.

Formally, call a graph G *browsable* if, for the URI of any node in G, if I look up that URI I will be returned information which describes the node, where describing a node means:

1. Returning all statements where the node is a subject or object; and
2. Describing all blank nodes attached to the node by one arc.

There are also the next limitations on such browseable data, mainly regarding data consistency across separate documents. By these definitions, statements which relate things in two different documents must be repeated. This clearly goes against the knowledge principle Don't Repeat Yourself (DRY), or in this case, not to store data in other places, as the problems with keeping the data consistent will arise eventually. A set of completely browsable data

with links in both directions has to be completely consistent, and that takes coordination, especially if different authors or programs are involved.

One of the solutions to this repetition problem is to have links of a certain property in a separate document. A person's homepage doesn't list all their publications but instead puts a link to it a separate document listing them.

In conclusion, linked data is essential for linking the Semantic Web. It is quite easy to implement linked data in both new and already existing applications or websites. Various common-sense considerations determine when to make a link and when not to.

1.2.3 SPARQL Query Language for RDF

As for querying the linked data, sometimes the data volume makes serving it in lots of files hard for efficient remote queries over the dataset. In this scenario, it seems reasonable to provide a query service. To make the data be effectively linked, someone who only has the URI of something must find their way. For that purpose, an RDF query language specification was developed. An RDF query language is a computer language, specifically a query language for databases, able to retrieve and manipulate data stored in RDF format.

Properties relevant to RDF query language design include support for the RDF format principles[4]:

- Support for RDF data, which is a collection of triples that form the RDF graph.
- Support for RDF semantics and inference that allows for entailment, the reasoning about the meaning of RDF graphs.
- Support for schema data types, such as XML schema.

Also every RDF query language should have desirable language features:

- **Expressiveness:** the power of query expression that may be constructed
- **Closure:** data operations on an RDF graph should result in another RDF graph
- **Orthogonality:** data operations are independent of the context in which they are used
- **Safety:** every expression returns a finite set of results.

RDF query languages can be grouped into language families, each family comprising a set of closely related languages.

The SPARQL family of languages includes several languages, including SquishQL, RDQL, SPARQL, and TriQL. These languages treat RDF data

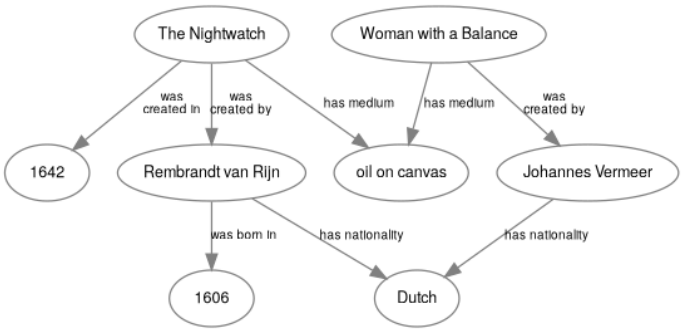

Figure 1.3: A network visualization of the triplestore

stores as triplestores that do not necessarily have ontology or schema information associated. Members of the SPARQL family are considered relational query languages because they have relational(or pattern-based) operations.

Aside from SPARQL, there are also other variations of query languages. The RQL family of languages includes RQL, SeRQL, and eRQL. These languages support the querying of both data and schema. RQL, an acronym for RDF Query Language, is known for using types defined in RDF schemas (RDFS) to query the schema class hierarchy and to support data querying by type. RQL is considered more expressive than the SPARQL family of languages but has been criticized for too many features and unusual syntactic constructs. There is a family of RDF query languages inspired by XML query technology. XQuery for RDF uses the XML query language XQuery to query RDF data by serializing RDF into an XML format and then using XQuery on the result.

To best show the capabilities of RDF, it is easiest to study the example of a SPARQL query. Continuing with the previous Rembrandt's painting, we should use the slightly extended triplestore, as shown in 1.4. A traditional relational database might split attributes about artworks and features about artists into separate tables. In an RDF/graph database, all these data points belong to the same interconnected graph, visualized in 1.3.

Listing 1.4: Example triplestore for SPARQL query

```
<The Nightwatch> <was created by> <Rembrandt van Rijn> .
<The Nightwatch> <was created in> <1642> .
<The Nightwatch> <has medium> <oil on canvas> .
<Rembrandt van Rijn> <was born in> <1606> .
<Rembrandt van Rijn> <has nationality> <Dutch> .
<Johannes Vermeer> <has nationality> <Dutch> .
<The Nightwatch> <was created by> <Johannes Vermeer> .
<Woman with a Balance> <has medium> <oil on canvas> .
```

Listing 1.5: Example triplestore for SPARQL query

```
SELECT ?painting
WHERE {
       ?painting <has medium> <oil on canvas> .
}
```

`?painting` in this query stands in for the node (or nodes) that the database will return. On receiving this query, the database will search for all values of ?painting that properly complete the RDF statement <has medium>oil on <canvas>. When the query runs against the full database, it looks for the subjects, predicates, and objects that match this statement, and finds paintings: The Nightwatch and Woman with a Balance.

1.3 NLP Interchange Format

It is important to first explain what Natural Language Processing (NLP) is. NLP is a field of science that combines lingustics, computer science and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data.

The NLP Interchange Format (NIF) is an RDF/OWL-based format that aims to achieve between NLP tools, language resources and annotations. NIF consists of specifications, ontologies and software (overview), which are combined under the common version identifier "NIF 2.0", but are also versioned individually[5]. The initial specification of NIF was released in November 2011.

NIF is being developed as a result and to facilitate the needs of Linked Data and related tools. NIF addresses the interoperability problem on three layers: the structural, conceptual and access layer. NIF is based on a Linked Data enabled URI scheme for identifying elements in (hyper-)texts that are described by the NIF Core Ontology (structural layer) and a selection of ontologies for describing common NLP terms and concepts (conceptual layer). NIF-aware applications will produce output adhering to the NIF Core Ontology as REST services (access layer). NIF enables the creation of heterogeneous, distributed and loosely coupled NLP applications, which use the Web as an integration platform. Another benefit is that a NIF wrapper has to be only created once for a particular tool, but enables the tool to interoperate with a potentially large number of other tools without additional adaptations. Ultimately, we envision an ecosystem of NLP tools and services to emerge using NIF for exchanging and integrating rich annotations.

NIF consists of several core components that are described below.

URI Schemes The idea behind NIF is to allow NLP tools to exchange annotations about text in RDF. Hence, the main prerequisite is that text becomes referenceable by URIs, so that they can be used as resources in RDF statements. In NIF, there is a distinction between the document d, the text t contained in the document and possible substrings s_t of this text. We call an algorithm to systematically create identifiers for t and s_t a *URI Scheme*. The canonical URI scheme of NIF is based on RFC 5147[6], which standardizes fragment ids for the text/plain media type. According to RFC 5147, the following URI can address the first occurrence of the substring "Semantic Web" in the text (26610 characters) of the document `http://www.w3.org/DesignIssues/LinkedData.html` with the separator #: `http://www.w3.org/DesignIssues/LinkedData.html#char=717,729`.

NIF Core Ontology The NIF Core Ontology[7] provides classes and properties to describe the relations between substrings, text, documents and their URI schemes. The main class in the ontology is nif:String, which is the class of all words over the alphabet of Unicode characters (sometimes called Σ^*). We built NIF upon the Unicode Normalization Form C, as this follows the recommendation of the RDF standard for rdf:Literal. Indices are to be counted in code units. Each URI scheme is a subclass of nif:String and puts further restrictions over the syntax of the URIs. For example, instances of type nif:RFC5147String have to adhere to the NIF URI scheme based on RFC 5147. Users of NIF can create their own URI schemes by subclassing nif:String and providing documentation on the Web in the rdfs:comment field.

Another important to the Framework for Extraction of Wikipedia Articles Content subclass of nif:String is the nif:Context OWL class. This class is assigned to the whole string of the text (i.e. all characters). The purpose of an individual of this class is special, because the string of this individual is used to calculate the indices for all substrings. Therefore, all substrings have to have a relation nif:referenceContext pointing to an instance of nif:Context.

Workflows and Modularity of NIF Workflows: NIF web services are loosely coupled and can receive either text or RDF. To allow seamless NLP integration, clients should create work flows where the text is normalized (Unicode) at the beginning and tokenization is provided.

Modularity: The NIF ontology is split in three parts: The terminological model is lightweight in terms of expressivity and contains the core classes and properties. Overall, it has 125 axioms, 28 classes, 16 data properties and 28 object properties. The inference model contains further axioms, which are typically used to infer additional knowledge, such as transitive property axioms. The validation model contains axioms, which are usually relevant for consistency checking or constraint validation. Depending on the use case, the inference and validation model can optionally be loaded.

1.3.1 Existing Use Cases for NIF

Internationalization Tag Set The *Internationalization Tag Set (ITS)* Version 2.0 is a W3C working draft, which is in the final phase of becoming a W3C recommendation. Among other things, ITS standardizes HTML and XML attributes which can be leveraged by the localization industry (especially language service providers) to annotate HTML and XML nodes with processing information for their data value chain.

An example of three attributes in an HTML document is given here1.6:

Listing 1.6: Example of Internationalization Tag Set HTML Code

```
<html>
       <body>
       <h2 translate="yes">
              Welcome to <span its-ta-ident-ref="http://dbpedia
                  .org/resource/Dublin" its-within- text="yes"
                  translate ="no"> Dublin </span> in <b
                  translate ="no" its-within-text="yes"> Ireland
                   </b>!
       </h2>
       </body>
</html>
```

NIF successfully creates a bridge between ITS and RDF and a round-trip conversion was recently implemented as a proof-of-concept. Therefore, NIF can be expected to receive a wide adoption by machine translation and industrial language service providers. Additionally, the ITS Ontology provides well modeled and accepted properties, which can in turn be used to provide best practices for NLP annotations.

Ontologies of Linguistic Annotation The Ontologies of Linguistic Annotation (OLiA) provide stable identifiers for morpho-syntactical annotation tag sets, so that NLP applications can use these identifiers as an interface for interoperability. OLiA provides Annotation Models (AMs) for fine-grained identifiers of NLP tag sets. The individuals of these annotation models are then linked via rdf:type to coarse-grained classes from a Reference Model (RM), which provides the interface for applications. NIF provides two properties: nif:oliaLink links a nif:String to an OLiA Annotation Model. Although a reasoner could automatically deduce the abstract type of each OLiA individual from the RM, it was a requirement that the coarse-grained types should be linked redundantly to the strings as well in case reasoning services are not available or would cause high overhead. Therefore, an OWL annotation property nif:oliaCategory was created as illustrated in the following example[**?**].

Listing 1.7: Example of Internationalization Tag Set HTML Code

```
<char=342,345> a nif:String, nif:RFC5147String;
nif:oliaLink penn:NNP;
nif:oliaCategory olia:Noun, olia:ProperNoun .
# deducable by a reasoner :
penn:NNP a olia:Noun, olia:ProperNoun .
```

1.4 Linked Open Data and DBpedia

A typical example of a large Linked Dataset is DBpedia. DBpedia and related tools are supported by the Leipzig University research group. As stated in the related article, is a community effort to extract structured information from Wikipedia and to make this information available on the Web. DBpedia allows internet users to ask sophisticated queries against datasets derived from Wikipedia and to link other datasets on the Web to Wikipedia data. This section will describe the extraction of the DBpedia datasets, and how the resulting information is published on the Web for human and machine consumption.

The most effective way of spurring synergistic research along these directions is to provide a rich corpus of diverse data. This would enable researchers to develop, compare, and evaluate different extraction, reasoning, and uncertainty management techniques, and to deploy operational systems on the Web. The DBpedia project has derived such a data corpus from the Wikipedia encyclopedia.

Wikipedia editions are available in over 250 languages, with the English one accounting for more than 1.95 million articles. Like many other web applications, Wikipedia has the problem that its search capabilities are limited to full-text search, which only allows very limited access to this valuable knowledge base. As has been highly publicized, Wikipedia also exhibits many of the challenging properties of collaboratively edited data: it has contradictory data, inconsistent taxonomical conventions, errors, and even spam.

The DBpedia project focuses on the task of converting Wikipedia content into structured knowledge, such that Semantic Web techniques can be employed against it — asking sophisticated queries against Wikipedia, linking it to other datasets on the Web, or creating new applications or mashups.

The DBpedia project focuses on the task of converting Wikipedia content into structured knowledge, such that Semantic Web techniques can be employed against it — asking sophisticated queries against Wikipedia, linking it to other datasets on the Web, or creating new applications or mashups. DBpedia project makes the following contributions:

- Develop an information extraction framework, which converts Wikipedia content to RDF. The basic components form a foundation upon which

further research into information extraction, clustering, uncertainty management, and query processing may be conducted.

- Provide Wikipedia content as a large, multi-domain RDF dataset, which can be used in a variety of Semantic Web applications. The DBpedia dataset consists of 103 million RDF triples.
- Iinterlink the DBpedia dataset with other open datasets. This results in a large Web of data containing altogether around 2 billion RDF triples.
- Develop a series of interfaces and access modules, such that the dataset can be accessed via Web services and linked to other sites.

1.4.1 Extracting Structured Information from Wikipedia

Wikipedia articles contain different types of structured information, such as infobox templates, categorisation information, images, geo-coordinates, links to external Web pages and links across different language editions of Wikipedia. To process this, Wikipedia uses Mediawik software. Due to the nature of this Wiki system, basically all editing, linking, annotating with meta-data is done inside article texts by adding special syntactic constructs. Hence, structured information can be obtained by parsing article texts for these syntactic constructs. Example of Wikipedia XML and then cleaned text that will be shown to the user can be seen in listings 1.8 and 1.9 respectively.

Listing 1.8: Raw Wikipedia XML

```
{{basic forms of government}}
'''Anarchism''' is an [[Anti-authoritarianism|anti-authoritarian
   ]] [[Political philosophy|political]] and [[Social
   philosophy|social philosophy]]{{sfnm|1a1=McLaughlin|1y
   =2007|1p=59|2a1=Flint|2y=2009|2p=27}} that rejects [[
   Hierarchy|hierarchies]] deemed unjust and advocates their
   replacement with [[Workers' self-management|self-managed]],
   [[Self-governance|self-governed]] societies based on
   voluntary, [[cooperative]] institutions. These institutions
   are often described as [[Stateless society|stateless
   societies]],{{sfnm|1a1=Sheehan|1y=2003|1p=85|2a1=Craig|2y
   =2005|2p=14}} ...
```

Listing 1.9: Cleaned Wikipedia text

```
Anarchism is an anti-authoritarian political and social
   philosophy that rejects hierarchies deemed unjust and
   advocates their replacement with self-managed, self-governed
    societies based on voluntary, cooperative institutions.
```

```
These institutions are often described as stateless
societies, ...
```

The XML extraction algorithm detects such Mediawiki templates and recognizes their structure using pattern matching techniques. It selects significant templates, which are then parsed and transformed to RDF triples. The algorithm uses post-processing techniques to increase the quality of the extraction. MediaWiki links are recognized and transformed to suitable URIs, common units are detected and transformed to data types. Furthermore, the algorithm can detect lists of objects, which are transformed to RDF lists.

1.4.2 DBpedia Dataset

As stated on the DBpedia's official website[8], the English version of the DBpedia dataset currently provides information about more than 4.58 million things, out of which 4.22 million are classified in a consistent ontology, including at least 1,445,000 persons, 735,000 places(including 478.000 populated places), 411,000 creative works (including 123,000 music albums, 87,000 films and 19,000 video games), 241,000 organizations (including 58,000 companies and 49,000 educational institutions), 251,000 species and 6,000 diseases.

DBpedia concepts are described by short and long abstracts in 125 languages. All these versions together describe 38.3 million things, out of which 23.8 million are localized descriptions of things that also exist in the English version of DBpedia. The full DBpedia data set features 38 million labels and abstracts in 125 different languages, 25.2 million links to images and 29.8 million links to external web pages; 80.9 million links to Wikipedia categories, and 41.2 million links to YAGO categories. DBpedia is connected with other Linked Datasets by around 50 million RDF links.

1.4.3 Triplestore

A triplestore is a software program capable of storing and indexing RDF data, in order to enable querying this data efficiently. Most triplestores support the SPARQL query language for querying RDF data. Virtuoso, Sesame, and BigOWLIM are typical examples of triplestores. DBpedia is using Virtuoso as the underlying triplestore.

1.4.4 DBpedia Dataset Web Endpoints

DBpedia website provides three access mechanisms to the DBpedia dataset: Linked Data, the SPARQL protocol, and downloadable RDF dumps. Royalty-free access to these interfaces is granted under the terms of the GNU Free Documentation License.

Linked Data. DBpedia resource identifiers, are set up to return RDF descriptions when accessed by Semantic Web agents, and a simple HTML

view of the same information to traditional web browsers. HTTP content negotiation is used to deliver the appropriate format.

SPARQL Endpoint. Client applications can send queries over the SPARQL protocol to this endpoint at http://dbpedia.org/sparql. This interface is appropriate when the client application developer knows in advance exactly what information is needed. In addition to standard SPARQL, the endpoint supports several extensions of the query language that have proved useful for developing user interfaces: full text search over selected RDF predicates, and aggregate functions, notably COUNT. To protect the service from overload, limits on query cost and result size are in place. For example, a query that asks for the store's entire contents is rejected as too costly, and SELECT results are truncated at 1000 rows.

RDF Dumps. N-Triple serializations of the datasets are available for download at the DBpedia website and can be used by sites that are interested in larger parts of the dataset.

1.5 Related works

1.5.1 DBpedia Information Extraction Framework

Prior to the current project, a few projects have already been made in order to facilitate DBpedia's need for extracting information from Wikipedia and related resources, namely **DBpedia Information Extraction Framework**[9].

DBpedia Information Extraction Framework focuses on the main disadvantage of DBpedia: heavy-weight release process. Producing a DBpedia dataset release through the traditional dump-based extraction requires manual effort and – since dumps of the Wikipedia database are created on a monthly basis – DBpedia has never reflected the current state of Wikipedia. Hence, this prokect extended the DBpedia extraction framework to support a *live extraction*, which works on a continuous stream of updates from Wikipedia and processes that stream on the fly. More importantly, the extraction framework focuses on other parts of the Wikipeida articles. The framework has 19 extractors that process the following Wikipedia content, most important of which are list below:

- *Labels.* All Wikipedia articles have a title, which is used as an rdfs:label for the corresponding DBpedia resource.

- *Abstracts.* Those include a short abstract (first paragraph, represented by using rdfs:comment) and a long abstract (text before a table of contents, using the property dbpedia:abstract) from each article.

- *Interlanguage links.*

- *Images.*
- *Redirects.*
- *Disambiguation.*
- *External links.*
- *Page links.*
- *Person data.* It extracts personal information such as surname, and birth date. This information is represented in predicates such as foaf:surname, and dbpedia:birthDate.
- *Infobox*
- *Category label* Wikipedia articles are arranged in categories, and this extractor extracts the labels for those categories.

Chapter 2

Analysis and Implementation

2.1 Requirements

Before starting to design the project, it is beneficial to specify the requirements. There several ways to layout the requirements, mainly writing down formal requirements or use cases. Because the project is oriented on delivering results to a smaller group of developers, it will be better to use formal requirements, as opposed to user-oriented use cases.

A Functional Requirement (FR) is a description of the service that the software, specifically the Extraction Framework, must offer to the user. It describes a software system or its component. Functional requirements should include the following things:

1. Details of operations conducted in the system;
2. Description of system inputs and outputs or other reports;
3. Information about the workflows performed by the system.

Functional requirements have next advantages:

- Helps to check whether the application is providing all the functionalities that were mentioned;
- A functional requirement document helps you to define the functionality of a system or one of its subsystems;
- Functional requirements along with requirement analysis help identify missing requirements. They help clearly define the expected system service and behavior;
- Errors caught in the Functional requirement gathering stage are the cheapest to fix;

- Support user goals, tasks, or activities.

The list of functional requirements for the Framework for Extraction of Wikipedia Articles Content:

1. **Accept input data:** The framework should accept the official Wikipedia dumps in the XML format, provided by the Wikipedia. The dumps can contain an amount of information up to 20 GB of text data. The framework should be able to parse dumps in English and at least 4 other popular Wikipedia languages.

2. **Provide outputs:** The framework should print all the outputs in the NIF triples, concatenating processed data from all articles in a single XML input file and writing the data to .nt output file.

3. **Extract context:** The framework should extract clean text from the Wikipedia page, removing or processing all the XML and Wikipedia-specific markup, including the core text but excluding infoboxes, files, images, and footers.

4. **Extract page structure:** The framework should extract a page tree, where every page section is a node, preserving the relation to the page context. The page tree should include The page tree should be printed in an output file separately from the page context.

5. **Extract links:** In addition to context, the framework should supplement the context by extracting internal Wikipedia links from the page. These links should refer to their respective sections of the text, and include only other Wikipedia articles, excluding possible external links to other web pages. The links should also be printed in another output file, separately from the page structure and context.

6. **Implement extensibility:** The framework should be easily extensible by other developers to include new languages.

7. **Provide an interface:** The framework is required to have an intuitive interface for the user to easily leverage the framework in other works.

8. **Evaluate the results:** The framework should contain the metrics that will provide user with a feedback about every execution.

While functional requirements are the most important part of the project, there can be other, less specific requirements that go along with the functional requirements, commonly known as non-functional requirements. For this project, the non-functional requirements included the research of Wikipedia XML Dump Structure, NIF data format and the ways to facilitate the goals of the DBpedia project.

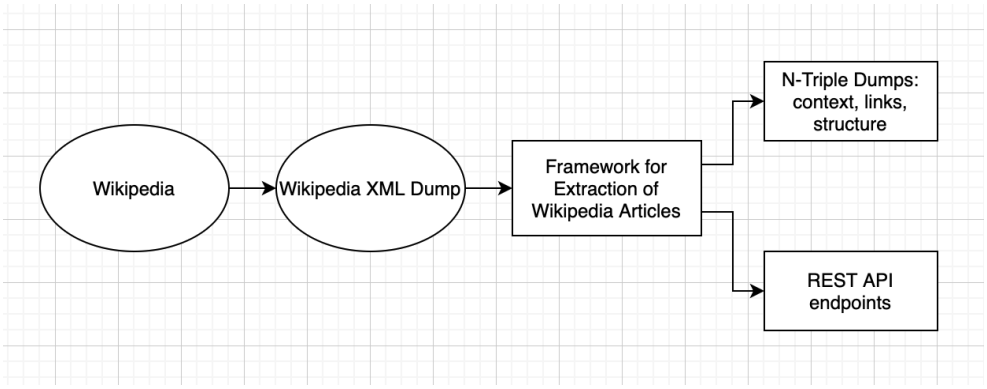

Figure 2.1: General Extraction Framework data workflow

2.2 Design

2.3 General Workflow

The general data workflow of the Framework for Extraction of Wikipedia Articles is depicted in Figure 2.1.

- Wikipedia: the main Wikipedia website is the primary sources of information.
- Wikipedia XML dump: Wikipedia runs a daily database archivation process and releases all the archived data in the form of XML dumps. These dumps can then be downloaded, unpacked and fed to the framework.
- Framework for Extraction of Wikipedia Articles: the framework processes the XML dump to get the context, structure and links and provide several options for the output.
- N-Triples Dumps: one of the outputs is to write the processed information to text files as N-Triples
- REST API endpoints: the other option is to provide the REST API endpoints for more convenient view of the output.

2.4 Usability considerations

One of the goals of the application is to make it easy to use, both by researchers and machines. In order to achieve that, application provides several interfaces. For the machines, it might be easier to connect to the application

via Representational state transfer (REST) Application Programming interface (API). Humans prefer other interfaces, such as Graphic User Interface (GUI), or Command Line Interface (CLI).

2.4.1 REST API

REST is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

An API is a computing interface which defines interactions between multiple software intermediaries. It defines the kinds of calls or requests that can be made, how to make them, the data formats that should be used, the conventions to follow, etc.

In a RESTful Web service, requests made to a resource's URI will elicit a response with a payload formatted in HTML, XML, JSON, or some other format. The response can confirm that some alteration has been made to the resource state, and the response can provide hypertext links to other related resources. When HTTP is used, as is most common, the operations (HTTP methods) available are GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS and TRACE. Therefore, HTTP-based RESTful APIs are defined with the following aspects:

- A base URI, such as http://api.example.com/collection/;
- Standard HTTP methods (e.g., GET, POST, PUT, PATCH and DELETE);
- A media type that defines state transition data elements (e.g., Atom, microformats, application/vnd.collection+json, xml, etc.). The current representation tells the client how to compose requests for transitions to all the next available application states. This could be as simple as a URI or as complex as a Java applet.

The framework gives an option to use REST API endpoints over the CLI interface. To understand the reasoning of why to include an API in the application, it is best to start with its pros and cons in the context of this project.

Using REST API Pros

- REST is a defined way of communication between machines
- REST API will allow users to retrieve information in chunks rather than having a complete output at once

- REST API can be easily tested by using applications tools Postman or curl to produce automated and granulated integration tests

Cons

- REST API is not cut out to transfer large amounts of data, and is usually limited by the machine's RAM, making it unsuitable for serving the amounts produced by this project.

- REST API is not a user-friendly out-of-the-box solution, as navigating it will require either some special tools or an additional development of the client interface.

Considering these points, it can be concluded that while REST API is useful for some particular tasks, it is better to use it as a secondary interface.

2.4.1.1 REST API Endpoints

In API terminology, communication endpoint, or simply endpoint is a unique URL address that users can access to exchange information with the server. Or in other words, APIs work using requests and responses. When an API requests information from a web application or web server, it will receive a response. The place that APIs send requests and where the resource lives, is called an endpoint.

Designing the endpoints is an intricate process on its own. While there is no single standard on how to design and name the endpoints, there are several recommendations followed by the programming community[10]:

- **Use Nouns in URI.** While this rule is not hard, the API is oriented towards resources, and nouns that define resources are generally preferred over verbs or adjectives.

- **Plurals over Singulars.** The ideology behind using plurals is that usually we operate on one resource from a collection of resources.

- **Let the HTTP Verb Define Action.** Continuing on the first point, HTTP already has verbs(such as GET, POST, PUT, DELETE) in place to define the action of a request.

- **Do not misuse idempotent methods.** Safe, or idempotent, methods in HTTP are the methods which will return the same response irrespective of how many times they are called by the client. GET, HEAD, OPTIONS and TRACE methods are defined as safe. It is important to use HTTP methods according to the action which needs to be performed.

- **Depict Resource Hierarchy Through URI.** If a resource contains sub-resources, make sure to depict this in the API to make it more explicit. For example, if a user has posts and we want to retrieve a specific post by user, API can be defined as GET /users/123/posts/1 which will retrieve Post with id 1 by user with id 123.
- **Version Your APIs** Versioning APIs always helps to ensure backward compatibility of a service while adding new features or updating existing functionality for new clients.

The framework provides the next REST endpoints:

- **POST /articles** - Submission endpoint allows you to submit the XML dump or its part to the server. After that is done, the server will asynchronously parse the provided XML, adding the articles to the database as it goes through the submitted articles.
- **GET /articles/{title}/context** Get the context N-Triples of an article with a given title.
- **GET /articles/{title}/structure** Similarly, get the page structure of an article.
- **GET /articles/{title}/links** Get the links associated with an article with a given title.
- **GET /articles/count** Get the total count of articles in a server's database.

2.4.2 Command Line Interface

Parsing of large xml files imposes limitations on the technologies that can be used. Particularly, the size of English part of the Wikipedia xml dump has a size of 16 GB. This means that the file cannot be normally loaded into Random Access Memory (RAM), as a single modern computer will usually have from 4 to 16 GB of RAM, with Java heap utilizing a quarter of that capability by default.

Furthermore, modern internet communication is better built around frequent exchange with small packets, and imposes a limit of maximum amount of requests that can be sent in a second. For example, it will not be possible to use Wikipedia's API for this task, as the Wikipedia's server might ban all further requests. For that reason, all the processing should be done offline and not rely on the internet connection at all.

Considering the limitations described above, it was decided to use CLI as the main way to use the application.

2.4.3 CLI Design Principles

Developers can get a lot more done by using a well-designed CLI. Usability and discovery are paramount in a CLI application. There are next important points to consider when designing a good CLI:

1. **Provide a Help Screen** Getting started with a CLI is unlike using other software for the first time. There is not always a welcome screen, no confirmation email with a link to documentation. Only through the command itself can developers explore what's possible. That experience begins in the help screen, accessible via a command in your CLI application, usually via runing command with a *help* parameter.

2. **Consider following already created CLI** For example, there a few general parameters that are included in every CLI:

   -h or –help Display the help screen

   -v or –verbose: Show less succinct output of the command, usually for debugging purposes. This one may be contentious, as some CLIs use -v for version.

   -V or –version: It's important to know which version of the CLI you're using, but not as often as you want verbose output.

3. **Allow Developers to Customize Their CLI Experience.** This one usually achieved via providing profiles. In this project's case, it was simplified by using an existing CLI library.

To simplify further development process, it was decided to use an existing picocli library for simple CLI implementation.

2.4.4 Command Line Input Options

The library used to create a CLI provides a good mechanisms to generate help text, from which the list of possible arguments, both mandatory and optional, can be extracted:

- **<xmlFile>** - The relative path to the XML Wiki dump.
- **-c, –clean** - The optional argument to clear output files of content before writing a new information. Useful option for testing the framework.
- **-h, –help** - Show this help message and exit.
- **-l, –language=<language>** - Provide the language of the XML dump that is being parsed. Default language is English.
- **-o, –output=<outputPath>** - The NIF files output folder.
- **-V, –version** - Print framework version information.

2.5 Project Architecture

Since the related works mentioned in 1.5.1 are based on Java and other Java Virtual Machine (JVM) based technologies such as Scala, it is best to build the project on those technologies in order to leverage the existing knowledge and reuse already developed libraries where possible. It is, however, worth noting that a large amount of dependencies used will introduce more complexity into the system. When the dependency' behavior that is not controlled by the framework is changed, the whole application workflow may be affected until the fix is implemented. Therefore, using only necessary dependencies is important.

Following the Java application standards, the codebase was split into packages, with **org.dbpedia** as the main package prefix, common for all DBpedia-related projects. A package in Java is used to group related classes, and is similar to a folder or a directory. We use packages to avoid name conflicts, and to write a better maintainable code.

- **application** - package for main Java classes. The application is developed in such a way that it has several main methods, and depending on the configuration, only one of those will be used.
- **cli** - package responsible for generating CLI.
- **configuration** - package that contains necessary Spring configuration, further described in 2.6.2.1.
- **exception** - package responsible for custom exception handling. While Java has its own Exception handling system, for a better handling it is recommended to extend the existing interfaces and catch custom exceptions instead.
- **extractor** - main package that contains all the code responsible for the extraction, parsing and structuring the information.
- **splitter** - support package that helps to split the xml dump into separate pages.

After designing the package structure, it is important to identify the general class diagram outlay. This outlay is presented in Figure 2.2. The classes and their purpose are broken down below:

- **ExtractionApplicationCLI** - main executable class that contains Java's main method.
- **XmlInput** - a class that processes the CLI input parameters, by using the picocly library[11].

- **LanguageIdentifierBean** - a singleton class that defines the language of an XML dump that is processed, set to English by default. A singleton classes can only have one instance and usually contains project-wide settings. Singleton mechanic is handled by the Spring framework, described in section 2.6.2.1.
- **XmlDumpService** - a service class that is a wrapper for all XML-processing operations.
- **XmlDumpParser** - a specific class that processes XML dump and breaks it up into pages.
- **WikipediaPageParser** - a parser class that focuses on processing a single page.
- **OutputFolderWriter** - a writer class that facilitates the output of a current project.

2.6 Implementation

2.6.1 Tools and libraries

The chosen language for framework development is Java, as it is used in DBpedia Extraction Framework, described in section 1.5.1. It is, however, important to notice that plain Java is not sufficient to achieve the goal of implementing modern Web Framework. For that reason, many other supplementary tools, libraries and other technologies were developed. This current ecosystem will be described below.

The project was developed using an IntelliJ IDEA[12], a modern Integrated Development Environment (IDE) that is maintained by a Czech company JetBrains. While initially inferior to its counterparts, such as Eclipse and NetBeans, this IDE has over time grew into an industry standard, currently becoming by far the most popular IDE used by developers[13], as can be seen on Figure 2.3. The company is now also developing a JVM-based language called Kotlin, that will be able to overcome some of Java's own shortcomings, such as a lack of support for functional programming paradigm. For the current project, however, it was decided to eliminate unnecessary dependencies and use Java as a main development language.

2.6.2 Spring Framework

Spring framework is an open source Java platform. It was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.

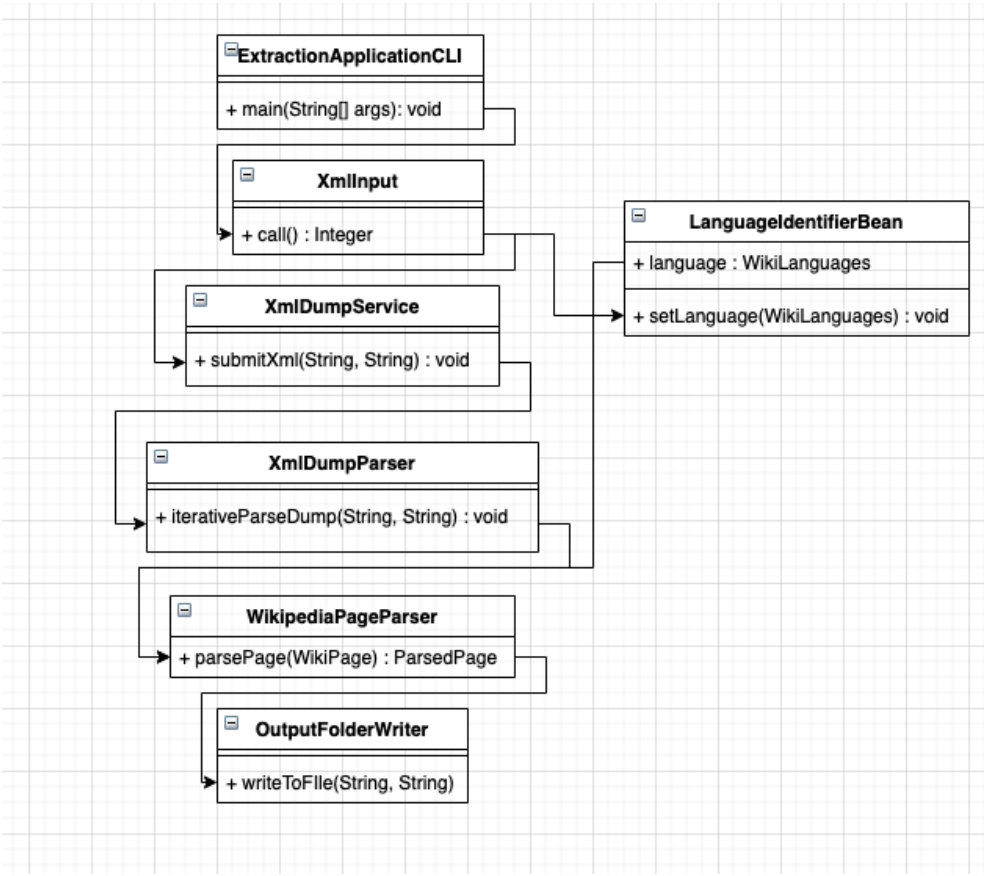

Figure 2.2: Framework Main Class Diagram

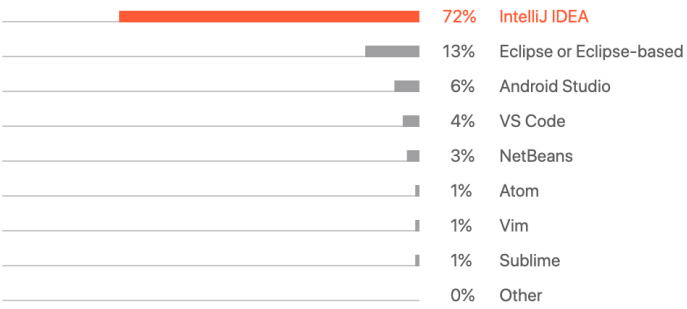

Figure 2.3: Usage of IDEs for Java development

The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java Enterprise Edition platform. Spring framework targets to make Java 2 Platform Enterprise Edition (J2EE) development easier to use and promotes good programming practices by enabling a Plain Old Java Object (POJO)-based programming model.

The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.

A key element of Spring is infrastructural support at the application level: Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

Following is the list of few of the great benefits of using Spring Framework:

- Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust servlet container such as Tomcat or some commercial product.
- Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about the ones you need and ignore the rest.
- Spring makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, and other view technologies.
- Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBeanstyle POJOs, it becomes easier to use dependency injection for injecting test data.
- Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks.
- Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.
- Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.

- Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

There is a several versions of Spring Framework, namely the original Spring and a newer version called Spring Boot. The difference between those two is that the original Spring leverages the use of XML configurations, while Spring Boot instead uses Java Configuration classes.

An Inversion of Control(IoC) container is a common characteristic of frameworks that implement IoC.

In the Spring framework, the IoC container is represented by the interface *ApplicationContext*. The Spring container is responsible for instantiating, configuring and assembling objects known as beans, as well as managing their lifecycle.

One of the important aspects of the Spring Framework is a **Bean**. Beans are the objects that form the backbone of your application and that are managed by the Spring Inversion of Control container. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that supplied to the container.

Beans have so-called Spring Bean Scopes, that allow developers to have more granular control of the bean lifecycles. There are 5 maing Bean scopes in Spring:

- singleton – only one instance of the spring bean will be created for the spring container. This is the default spring bean scope. While using this scope, make sure bean doesn't have shared instance variables otherwise it might lead to data inconsistency issues.

- prototype – A new instance will be created every time the bean is requested from the spring container.

- request – This is same as prototype scope, however it's meant to be used for web applications. A new instance of the bean will be created for each HTTP request.

- session – A new bean will be created for each HTTP session by the container.

- global-session – This is used to create global session beans for Portlet applications.

For this project, mostly prototype and singleton Bean scopes were used, with singleton being a default one.

2.6.2.1 Spring Dependency Injection

Inversion of Control **Inversion of Control** is a principle in software engineering by which the control of objects or portions of a program is transferred to a container or framework. It's most often used in the context of object-oriented programming.

By contrast with traditional programming, in which our custom code makes calls to a library, IoC enables a framework to take control of the flow of a program and make calls to our custom code. To enable this, frameworks use abstractions with additional behavior built in. If we want to add our own behavior, we need to extend the classes of the framework or plugin our own classes.

The advantages of this architecture are:

- Decoupling the execution of a task from its implementation.
- Making it easier to switch between different implementations.
- Greater modularity of a program.
- Greater ease in testing a program by isolating a component or mocking its dependencies and allowing components to communicate through contracts.

Inversion of Control can be achieved through various mechanisms such as: Strategy design pattern, Service Locator pattern, Factory pattern, and Dependency Injection (DI).

Dependency Injection Dependency injection is a pattern through which to implement IoC, where the control being inverted is the setting of object's dependencies.

The act of connecting objects with other objects, or "injecting" objects into other objects, is done by an assembler rather than by the objects themselves.

You can see an example on how to create an object dependency in traditional programming in Listing 2.1

Listing 2.1: Example class without a Dependency Injection

```
public class Store {
	private Item item;

	public Store() {
		item = new ItemImpl1();
	}
}
```

In the example above, we need to instantiate an implementation of the Item interface within the Store class itself. By using Dependency Injection pattern, this example can be rewritten without specifying the implementation of Item that we want, as can be seen in Listing 2.2.

Listing 2.2: Example class with a Dependency Injection

```
public class Store {
       private Item item;
       public Store(Item item) {
              this.item = item;
       }
}
```

2.6.3 Java Jackson XML Library

The Jackson project is a collection of data processing tools for the Java language and the JVM platform. It supports a wide range of data formats such as Comma-separated values (CSV), Java Properties, XML, and Yet ANother Markup Language (YAML) through extension components that support the specific language.

The Jackson XML component is meant for reading and writing XML data by emulating how Jakarta XML Binding (JAXB) works, although not conclusively.

In this project, Jackson library will be used to serialize Java objects into XML and deserialize them back into Java objects, in order to eventually produce text output.

XmlMapper Class XmlMapper is the main class from Jackson 2.x that helps the developers in serialization. This mapper is available in jackson-dataformat-xml jar, that can be easily added to the project using Apache Maven - project's dependency management[14].

An example of deserialization can be seen in Listing 2.3. Here, a Mediawiki class is a POJO class, or in other words a simple mapping of an XML scheme to a Java class hierarchy, where every subcomponent of a schema is mapped to a Java subclass.

Listing 2.3: Example of an XML Deserialization

```
  private XmlMapper xmlMapper = new XmlMapper();

  private Mediawiki deserializeXml(String dump) throws
      IOException {
              return xmlMapper.readValue(dump, Mediawiki.class)
                  ;
```

```
        }
```

2.6.4 Dynamic Language Support

The XML-structure of the Wikipedia article does not differ much from language to language. There are only a few points to be aware of: Footer headings and categories. In English, those would be "See also", "References", "Further reading", "External Links", and "Related pages", and Categories simply have a heading "Category". Those are the parts that have to be removed from the articles.

One of the project's requirements was to implement an easy extensibility mechanism to add new languages, mention in the Requirements Section 2.1. This was achieved by adding an abstract class `LanguageFooterRemover`. This class has some general functions to parse parts of the text that might be unique to different languages. Before the framework starts, it processes the configuration file language_list.xml that is stored in the configuration folder. The examples of this file's contents can be seen in Listing 2.4. Such template can easily be reused to extend the number of supported languages if needed.

Listing 2.4: Example of an language configuration file

```
<languageContainer>
        <language>
                <langName>ENGLISH</langName>
                <categoryName>Category</categoryName>
                <footer>See also</footer>
                <footer>References</footer>
                <footer>Further reading</footer>
                <footer>External Links</footer>
                <footer>Related pages</footer>
        </language>
        <language>
        <langName>POLISH</langName>
                <categoryName>Kategoria</categoryName>
                <footer>Przypisy</footer>
                <footer>Uwagi</footer>
        </language>
</languageContainer>
```

Chapter 3

Testing and Results

Before we dive into the testing, it might be interesting to describe a general approach to testing that was taken while implementing this framework.

Software testing is a procedure of implementing software or the application to identify the defects or bugs. For testing an application or software, we need to follow some principles to make our product defects free, and that also helps the test engineers to test the software with their effort and time. It is important to also keep in mind the essential principles of software testing.

- **Testing shows the presence of defects.** The primary purpose of doing testing is to identify the numbers of unknown bugs with the help of various methods and testing techniques because the entire test should be traceable to the customer requirement, which means that to find any defects that might cause the failure to meet the requirements. By doing testing on any application, we can decrease the number of bugs, which does not mean that the application is defect-free because sometimes the software seems to be bug-free while performing multiple types of testing on it.

- **Exhaustive Testing is not possible.** Instead of performing the exhaustive testing as it takes boundless determinations and most of the hard work is unsuccessful, it is better to focus on tests according to the importance of the modules because the timelines will not permit to perform a full testing scenario.

- **Early Testing.** This means that all the testing activities should start in the early stages of the software development life cycle's requirement analysis stage to identify the defects because if we find the bugs at an early stage, it will be fixed in the initial stage itself, which may cost us very less as compared to those which are identified in the future phase of the testing process.

- **Defect Clustering.** The defect clustering defined that throughout the testing process, we can detect the numbers of bugs which are correlated to a small number of modules. We have various reasons for this, such as the modules could be complicated; the coding part may be complex, and so on. These types of software or the application will follow the Pareto Principle, which states that we can identify that approx. Eighty percent of the complication is present in 20 percent of the modules. With the help of this, we can find the uncertain modules, but this method has its difficulties if the same tests are performing regularly, hence the same test will not able to identify the new defects.

- **Pesticide Paradox.** This principle defined that if we are executing the same set of test cases again and again over a particular time, then these kinds of the test will not be able to find the new bugs in the software or the application. To get over these pesticide paradoxes, it is very significant to review all the test cases frequently.

- **Testing is context-dependent.** Testing is a context-dependent principle states that we have multiple fields such as e-commerce websites, commercial websites, and so on are available in the market. There is a definite way to test the commercial site as well as the e-commerce websites because every application has its own needs, features, and functionality. To check this type of application, we will take the help of various kinds of testing, different technique, approaches, and multiple methods. Therefore, the testing depends on the context of the application.

- **Absence of errors fallacy.** Once the application is completely tested and there are no bugs identified before the release, so we can say that the application is 99 percent bug-free. But there is the chance when the application is tested beside the incorrect requirements, identified the flaws, and fixed them on a given period would not help as testing is done on the wrong specification, which does not apply to the client's requirements. The absence of error fallacy means identifying and fixing the bugs would not help if the application is impractical and not able to accomplish the client's requirements and needs.

The main machine that was used for testing, unless specified otherwise, had next specifications:

- **Processor:** 2,7 Hz Dual-Core Intel Core i5;
- **Memory:** 8 GB 1867 MHz DDR3;
- **Graphics Card:** Intel Iris Graphics 6100 1536 MB;
- **Operating System:** macOS Catalina, Version 10.15.6.

Project benchmarks **Benchmark Testing** measures a repeatable set of quantifiable results that serves as a point of reference against which products/services can be compared. The purpose of benchmark testing results is to compare the present and future software releases with their respective benchmarks.

A benchmark must be repeatable. For instance, with every iteration of load a test, if the response times varies too much, system performance be benchmarked. Response time needs to be stable amongst different load conditions.

A benchmark must be quantifiable. For example, the user experience cannot be quantified in numbers, but time a user spends on a webpage due to good UI can be quantified.

While testing this project, the results were constantly compared to an ideal output, that has been provided by the DBpedia project[15].

Three different output components were compared separately. On the listings below, the ideal result can be seen, for the context output in Listing 3.1, for links

Listing 3.1: Exemplary result for NIF Context

```
<http://dbpedia.org/resource/Anarchism?dbpv=2020-02&nif=context>
    <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-
   core#isString> "Anarchism is a radical political ... \n*
   Textbooks from Wikibooks \n* Data from Wikidata \n* Anarchy
   Archives. Anarchy Archives is an online research center on
   the history and theory of anarchism" .
```

Listing 3.2: Exemplary result for NIF Links

```
<http://dbpedia.org/resource/Austroasiatic_languages?dbpv
   =2016-04&nif=phrase_94_109> <http://www.w3.org/1999/02/22-
   rdf-syntax-ns#type> <http://persistence.uni-leipzig.org/
   nlp2rdf/ontologies/nif-core#Phrase> .
<http://dbpedia.org/resource/Austroasiatic_languages?dbpv
   =2016-04&nif=phrase_94_109> <http://persistence.uni-leipzig.
   org/nlp2rdf/ontologies/nif-core#referenceContext> <http://
   dbpedia.org/resource/Austroasiatic_languages?dbpv=2016-04&
   nif=context> .
<http://dbpedia.org/resource/Austroasiatic_languages?dbpv
   =2016-04&nif=phrase_94_109> <http://persistence.uni-leipzig.
   org/nlp2rdf/ontologies/nif-core#beginIndex> "94"^^<http://
   www.w3.org/2001/XMLSchema#nonNegativeInteger> .
<http://dbpedia.org/resource/Austroasiatic_languages?dbpv
   =2016-04&nif=phrase_94_109> <http://persistence.uni-leipzig.
   org/nlp2rdf/ontologies/nif-core#endIndex> "109"^^<http://www
   .w3.org/2001/XMLSchema#nonNegativeInteger> .
```

```
<http://dbpedia.org/resource/Austroasiatic_languages?dbpv
    =2016-04&nif=phrase_94_109> <http://persistence.uni-leipzig.
    org/nlp2rdf/ontologies/nif-core#superString>
```

Listing 3.3: Exemplary result for NIF Page Structure

```
<http://dbpedia.org/resource/Ada?dbpv=2016-04&nif=section_0_17>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://
    persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#
    Section> .
<http://dbpedia.org/resource/Ada?dbpv=2016-04&nif=section_0_17>
    <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-
    core#beginIndex> "0"^^<http://www.w3.org/2001/XMLSchema#
    nonNegativeInteger> .
<http://dbpedia.org/resource/Ada?dbpv=2016-04&nif=section_0_17>
    <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-
    core#endIndex> "17"^^<http://www.w3.org/2001/XMLSchema#
    nonNegativeInteger> .
<http://dbpedia.org/resource/Ada?dbpv=2016-04&nif=section_0_17>
    <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-
    core#referenceContext> <http://dbpedia.org/resource/Ada?dbpv
    =2016-04&nif=context> .
...
```

The output format should be in N-triples[16], better described in Section 1.2.2.

3.1 Smoke Testing

Smoke test is a test or a test suite that covers the main functionality of a component or system to determine whether it works properly before planned testing begins. Smoke testing, also known as "Build Verification Testing", is a type of software testing that comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work. The result of this testing is used to decide if a build is stable enough to proceed with further testing. It can also be used to decide whether to announce a production release or to revert. The term 'smoke testing', it is said, came to software testing from a similar type of hardware testing, in which the device passed the test if it did not catch fire (or smoked) the first time it was turned on. Smoke testing covers most of the major functions of the software but none of them in depth. The result of this test is used to decide whether to proceed with further testing. If the smoke test passes, go ahead with further testing. If it fails, halt further tests and ask for a new build with the required fixes. If an application is badly broken, detailed testing might be a waste of time and

effort. Smoke test helps in exposing integration and major problems early in the cycle. It can be conducted on both newly created software and enhanced software. Smoke test is performed manually or with the help of automation tools/scripts. If builds are prepared frequently, it is best to automate smoke testing.

To list the information, those are the advantages of an early and continuous smoke testing:

- It exposes integration issues.
- It uncovers problems early.
- It provides some level of confidence that changes to the software have not adversely affected major areas (the areas covered by smoke testing)

This kind of testing was performed during every stage of the implementation after each new part of functionality has been added to the framework. It includes CLI testing, API testing, functionality testing, and output verification. During development and testing the framework was running locally on a Mac-based machine.

A big amount of bugs and errors was revealed and subsequently fixed during these tests. For example, there were many errors related to the parsing of Wikipedia articles, and lots of inconsistencies happening during the page structure translation into the page.

Furthermore, scaling the framework has caused a lot of problems. The size of an English Wikipedia dump is about 16 GB of data, and parsing it takes a lot of time. During those tests, it was discovered that such amount of data can not be handled by the server, and therefore an adequate API for production purposes can not be easily provided.

For the smoke testing, the next tests were conducted:

- Single-article XML dump in English language.
- Two-page XML dump in English language.
- Other languages support testing, conducted for a single page from a German segment of Wikipedia.

3.2 Unit Test coverage

Unit testing is a software testing method by which individual units of source code - sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures - are tested to determine whether they are fit for use. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A

unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits.

Unit testing finds problems early in the development cycle. This includes both bugs in the programmer's implementation and flaws or missing parts of the specification for the unit. The process of writing a thorough set of tests forces the author to think through inputs, outputs, and error conditions, and thus more crisply define the unit's desired behavior. The cost of finding a bug before coding begins or when the code is first written is considerably lower than the cost of detecting, identifying, and correcting the bug later. Bugs in released code may also cause costly problems for the end-users of the software. Code can be impossible or difficult to unit test if poorly written, thus unit testing can force developers to structure functions and objects in better ways.

Unit testing allows the programmer to refactor code or upgrade system libraries at a later date, and make sure the module still works correctly (e.g., in regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified. Unit tests detect changes which may break a design contract.

Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

3.2.1 JUnit Framework

For the framework implementation, a JUnit library was used. **JUnit** is a Regression Testing Framework used by developers to implement unit testing in Java, and accelerate programming speed and increase the quality of code. JUnit Framework can be easily integrated with Maven.

JUnit test framework provides the following important features:

- **Fixtures** - is a fixed state of a set of objects used as a baseline for running tests. The purpose of a test fixture is to ensure that there is a well-known and fixed environment in which tests are run so that results are repeatable. It includes setUp() method, which runs before every test invocation, and tearDown() method, which runs after every test method.
- **Test suites.**A test suite bundles a few unit test cases and runs them together. In JUnit, both @RunWith and @Suite annotation are used to run the suite test.
- **Test runners.** Test runner is used for executing the test cases.
- **JUnit classes** JUnit classes are important classes, used in writing and testing JUnits. Examples of those classes are:

  Assert - contains a set of assert methods.

TestCase - contains a test case that defines the fixture to run multiple tests.

TestResult - contains methods to collect the results of executing a test case.

For a given project, unit tests were used to test the execution of a WIkipediaPageParser class, that is used to parse separate pages, as well as its supplementary classes, such as a DumpSplitService. You can see the examples of a unit test used in the project in the Listing 3.4:

Listing 3.4: JUnit Paragraph Parsing Unit Test Class

```
@Log4j
public class WikipediaPageParserTest {

        private static WikipediaPageParser pageParser;
        private static WikiPage wikiPage;
        private static XmlTransformer contextLanguageTransformer
            ;

        @BeforeAll
        public static void beforeAll() throws IOException {
                pageParser = new WikipediaPageParser(new
                    ContextLanguageTransformer());

                URL textUrl = Resources.getResource("page_test.
                    txt");
                wikiPage = new WikiPage("Anarchism",
                Resources.toString(textUrl, StandardCharsets.
                    UTF_8));
        }

        @Test
        public void parseParagraphsTest() throws IOException,
            ParsingException {
                Subdivision root = pageParser.buildPageStructure(
                    wikiPage);
                // check that the paragraphs are parsed
                assertTrue(root.getParagraphs().size() > 1);
                // check that the page has a meaningful structure
                assertTrue(root.getChildren().size() > 1);
        }
        ...
}
```

3.3 End-to-End Testing

End-to-end testing is a Software testing methodology to test an application flow from start to end. The purpose of End-to-end testing is to simulate the real user scenario and validate the system under test and its components for integration and data integrity. What it means for this project is that we will need to do a test from downloading an XML dump from Wikipedia to receiving the processed results. Here is the general outlay of a testing process:

1. Download the latest Wikipedia dump[17]. They are released at least monthly and usually twice a month. Different languages are listed after the metawiki dumps.

2. Unzip the file, in Unix-like operating systems usually done in `bzip2 -d wikidatawiki-*-pages-meta-history1.xml-p1p224.bz2`

Optional Because parsing of the whole XML dump is time-consuming, I have used a way to reduce the amount of articles that is processed. To do it, it is possible to execute this command `head -n 100000 enwiki-20191101-pages-articles-multistream1.xml > short_test_100k.xml`, and then properly close off the XML by removing the last article and close the XML brackets.

3. Build the framework and pass the input parameters to a file. I have created a script `parse_xml_dump.sh` that will build the framework if necessary and run the executable with the XML dump path as the parameter.

3.3.1 English language parsing

Initially the testing was done on a single English-language article. This testing uncovered many problems with the parsing model, such as the need to update the recursive function to build the page structure. Most importantly, the first testing helped to understand the vast number of Wikipedia XML components. Of those, I had to drop the parts that were already covered by the previous frameworks mentioned in Section 1.5.1, such as infoboxes, images, files, categories and others, and instead focus only on the text. I also dropped the citations mentioned in the article's footer, and instead focused on the text itself.

3.3.2 Testing other languages

Additionally to English language, I have added other popular Wikipedia languages. According to the latest Wikipedia statistics, those are the main languages of Wikipedia[18]:

1. **English:** 2,567,509 articles, 22.5% of the total number of articles;
2. **German:** 808,044 articles, 7.1%;
3. **French:** 709,312 articles, 6.2%;
4. **Polish:** 539,688 articles, 4.7%;
5. **Japanese:** 523,629 articles, 4.6%.

3.3.3 Output validation

There are several utilities that can be used to validate the output, most notable Apache Jena and rapper[19]. For the output validation, I have picked rapper, as it is more lightweight and has the ability to count or parse the provided N-triples. For example, the command `rapper --input ntriples --output rdfxml --show-graphs nif\_links.nt` will parse the links and transform it into RDF/XML format. If this transformation will not throw any exceptions, this will mean that the output is a valid set of N-triples.

3.3.4 Scale Testing

For the scale testing, I have implemented logging and a simple parsing success metric, as some of the articles have a syntax that may deviate from the standard or the framework's programmed expectations. For example, some might contain links that are broken with line separators, or differently encoded XML components.

To further measure the time that the application will take to parse the code, I have added the execution time metric.

I have run several tests over the English Wiki dump with the next results:

- Total pages parsed: 258. Success rate: 84.88%. Seconds passed: 31.
- Total pages parsed: 2087. Success rate: 88.55%. Seconds passed: 109.
- Total pages parsed: 6738. Success rate: 87.90%. Seconds passed: 237.

Chapter 4

Conclusions

The result of this thesis is a Java Framework that allows users to parse and retrieve the Wikipedia XML dump and achieves most of the original objectives, in some places with a room for improvement:

1. **Accept and process input data in the form of Wikipedia XML dumps.** The Wikipedia XML Dump parsing was achieved, and the process to do so best described in Section 3.3. The statistics show that the parsing success rate averages on 88% over the large amounts of articles, meaning that around 12% of articles will contain some kind of component that will not be parsable by the framework and will be skipped. These systemic errors can be avoided by further investigation of Wikipedia's XML Format.

2. **Extract context.** Context is extracted and stored in the form of N-Triples. Some of the contexts might still contain traces of the original XML code. This can be later fixed by improving the XML removal code.

3. **Extract page structure.** Page structure is extracted and recursively built in the form of N-Triples.

4. **Extract links.** Links are extracted, URLs that link them to the page structure are created.

5. **Provide outputs for context, links and page structure in the form of N-Triples.** Output is printed.

6. **Implement language extensibility.** Language extensibility mechanism is implemented, new languages can be added in the form of an XML that is parsed into POJO when the application is starting, better described in Section 2.6.4.

7. **Provide a user interface.** User interface is provided in two different forms and is described in Section 2.4.

Chapter 5

Acronyms

API Application Programming interface.

CLI Command Line Interface.

CSV Comma-separated values.

DRY Don't Repeat Yourself.

FOAF Friend Of A Friend.

FR Functional Requirement.

GUI Graphic User Interface.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

IDE Integrated Development Environment.

J2EE Java 2 Platform Enterprise Edition.

JAXB Jakarta XML Binding.

JVM Java Virtual Machine.

NIF NLP Interchange Format.

NLP Natural Language Processing.

OWL Web Ontology Language.

POJO Plain Old Java Object.

RAM Random Access Memory.

RDF Resource Description Framework.

REST Representational state transfer.

RSS RDF Site Summary.

SKOS Simple Knowledge Organization System.

SPARQL SPARQL Protocol and RDF Query Language).

URI Uniform Resource Identifier.

W3C World Wide Web Consortium.

WWW World Wide Web.

XML Extensible Markup Language.

YAML Yet ANother Markup Language.

Bibliography

[1] Berners-Lee, T. Q&A with Tim Berners-Lee [online]. April 2007. Available from: https://www.bloomberg.com/news/articles/2007-04-09/q-and-a-with-tim-berners-leebusinessweek-business-news-stock-market-and-financial-advice

[2] Berners-Lee, T. Linked Data [online]. July 2006. Available from: https://www.w3.org/DesignIssues/LinkedData.html

[3] Sean Bechofer, F. v. H. OWL Web Ontology Language Reference. In *W3C Recommendation*, February 2004.

[4] Peter Haase, A. E., Jeen Broekstra. A Comparison of RDF Query Languages. In *AWSC: Asian Semantic Web Conference*, October 2004, pp. 502–517.

[5] Sebastian Hellmann, S. A., Jens Lehmann; Brümmer, M. Integrating NLP using Linked Data. In *12th International Semantic Web Conference*, Sydney, Australia, October 2013.

[6] E. Wilde, M. D. URI Fragment Identifiers for the text/plain Media Type.

[7] NIF 2.0 Core Ontology [online]. 2020. Available from: https://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core/nif-core.html

[8] DBpedia Facts & Figures [online]. September 2020. Available from: https://dbpediawww.informatik.uni-leipzig.de/about/facts-figures

[9] Mohamed Morsey, J. L. DBpedia and the Live Extraction of Structured Data from Wikipedia.

[10] Kapadnis, J. REST: Good Practices for API Design[online]. March 2018. Available from: https://medium.com/hashmapinc/rest-good-practices-for-api-design-881439796dc9

[11] Picocli - command line interface library [online]. 2020. Available from: https://picocli.info/

[12] IntelliJ IDEA - Capable and Ergonomic IDE [online]. 2020. Available from: https://www.jetbrains.com/idea/

[13] JetBrains Java development ecosystem research [online]. 2020. Available from: https://www.jetbrains.com/lp/devecosystem-2020

[14] Apache Maven Project [online]. 2020. Available from: https://maven.apache.org/pom.html

[15] DBpedia Project Link [online]. 2020. Available from: https://wiki.dbpedia.org/

[16] N-Triples - a line-based syntax for an RDF graph [online]. 2020. Available from: https://www.w3.org/TR/n-triples/

[17] Wikimedia Downloads [online]. 2020. Available from: https://www.w3.org/TR/n-triples/

[18] Top Ten Wikipedias [online]. 2020. Available from: https://meta.wikimedia.org/wiki/Top_Ten_Wikipedias

[19] Raptor RDF Syntax Library - Raptor RDF parser utility [online]. 2014. Available from: http://librdf.org/raptor/rapper.html