Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Flexible RDF data extraction from Wiktionary

Leveraging the power of community build linguistic wikis

Masterarbeit
im Studiengang Master Informatik

eingereicht von: Jonas Brekle (Matrikel: 1498088)

eingereicht am: 4. September 2012

betreuender Professor: Prof. Dr. Ing. habil. Klaus-Peter Fähnrich

Betreuer: Dipl. Inf. Sebastian Hellmann

We present a declarative approach implemented in a comprehensive open-source framework (based on *DBpedia*) to extract lexical-semantic resources (an ontology about language use) from *Wiktionary*. The data currently includes language, part of speech, senses, definitions, synonyms, taxonomies (hyponyms, hyperonyms, synonyms, antonyms) and translations for each lexical word. Main focus is on flexibility to the loose schema and configurability towards differing language-editions of *Wiktionary*. This is achieved by a declarative mediator/wrapper approach. The goal is, to allow the addition of languages just by configuration without the need of programming, thus enabling the swift and resource-conserving adaptation of wrappers by domain experts. The extracted data is as fine granular as the source data in *Wiktionary* and additionally follows the *lemon* model. It enables use cases like disambiguation or machine translation. By offering a linked data service, we hope to extend DBpedia's central role in the LOD infrastructure to the world of Open Linguistics.

I thank Sebastian Hellmann for the courageous supervision and a professional research environment; furthermore Theresa for her support. Without them, this work would not have been possible.

Contents

1. Introduction

In the following I will present a short overview on the problem and how we solve it.

1.1. Motivation

The topic of this thesis will be the flexible extraction of semantically rich data from *Wiktionary*. The resulting data set is a lexical resource for computer linguistics. Nevertheless, the focus is not on linguistics, but on data integration: information extraction from wikis can be conducted in two ways—in perspective of either text mining, where the Wiki is seen as a corpus or alternatively interpreting the Wiki as a collection of semi-structured documents. The latter is our perspective and how DBpedia was designed, as it enables the definition of *extractors*, that interpret wiki pages. Also opposed to text mining, DBpedia allows to tailor the extracted data step by step closer to the intended semantic of the wikitext. Additionally weak signals (facts that don't have much support, that are only stated once) can be taken account of. DBpedia creates an ontology from Wikipedia, that is roughly said, a database of world knowledge. Opposed to Wikipedia, the DBpedia knowledge base can be queried like a database, combining information from multiple articles. To conduct an analogous transformation on *Wiktionary*, we analysed the major differences and found that *Wiktionary* is on one hand richer in structured information, but on the other hand this structure varies widely. So we propose a declarative framework, built on top of DBpedia, to convert *Wiktionary* into a linguistic ontology about languages, about the use of words, about their properties and relations. We will show the unique properties such a knowledge base has and what possible applications are. The declarative extraction rules can be maintained by a community of domain experts, that don't necessarily need programming skills. As will be shown, this is crucial for the approach to succeed on a long term. DBpedia has proven such an approach to be working and scaling. The goal of DBpedia is to provide a tool for unsupervised but highly configurable ontology construction. By using and extending DBpedia *Wiktionary* can be automatically transformed into a machine readable dictionary — with substantial quantity and quality.

1.2. Problem

The exploitation of community-built lexical resources has been discussed repeatedly. *Wiktionary* is one of the biggest collaboratively created lexical-semantic and linguistic resources available, written in 171 languages (of which approximately 147 can be considered active[1]), containing information about hundreds of spoken and even ancient languages. For example, the English *Wiktionary* contains nearly 3 million words[2]. For a lexical word a *Wiktionary* page provides a hierarchical disambiguation to its language, part of speech, sometimes etymologies and most prominently senses. Within this tree numerous kinds of linguistic properties are given, including synonyms, hyponyms,

[1] `http://s23.org/wikistats/wiktionaries_html.php`

[2] See `http://en.wiktionary.org/wiki/semantic` for a simple example page.

hyperonyms, example sentences, links to Wikipedia and many more. [23] gave a comprehensive overview on why this dataset is so promising and how the extracted data can be automatically enriched and consolidated. Aside from building an upper-level ontology, one can use the data to improve NLP solutions, using it as comprehensive background knowledge. The noise should be lower when compared to other automatic generated text corpora (e.g. by web crawling) as all information in *Wiktionary* is entered and curated by humans. Opposed to expert-built resources, the openness attracts a huge number of editors and thus enables a faster adaptation to changes within the language.

The fast changing nature together with the fragmentation of the project into *Wiktionary* language editions (*WLE*) with independent layout rules. We identified this as a serious problem in the automated transformation into a structured knowledge base: Although the value of *Wiktionary* is known and usage scenarios are obvious, only some rudimentary tools exist to extract data from it. Either they focus on a specific subset of the data or they only cover one or two WLE. The development of a flexible and powerful tool is challenging to be accommodated in a mature software architecture and has been neglected in the past. Existing tools can be seen as adapters to single WLE — they are hard to maintain and there are too many languages, that constantly change. Each change in the *Wiktionary* layout requires a programmer to refactor complex code. The last years showed, that only a fraction of the available data is extracted and there is no comprehensive RDF dataset available yet. The key question is: Can the lessons learned by the successful DBpedia project be applied to *Wiktionary*, although it is fundamentally different from Wikipedia? The critical difference is that only word forms are formatted in infobox-like structures (e.g. tables). Most information is formatted covering the complete page with custom headings and often lists. Even the infoboxes itself are not easily extractable by default DBpedia mechanisms, because in contrast to DBpedia's *one entity per page* paradigm, *Wiktionary* pages contain information about *several* entities forming a complex graph, i.e. the pages describe the lexical word, which occurs in several languages with different senses per part of speech and most properties are defined *in context* of such child entities.

1.3. Solution

Opposed to the currently employed classic and straight-forward approach (implementing software adapters for scraping), we propose a declarative mediator/wrapper pattern. The aim is to enable non-programmers (the community of adopters and domain experts) to tailor and maintain the WLE wrappers themselves. We created a simple XML dialect to encode the "entry layout explained" (ELE) guidelines and declare triple patterns, that define how the resulting RDF should be built. This configuration is interpreted and run against *Wiktionary* dumps. The resulting dataset is open in every aspect and hosted as linked data[3]. Furthermore, the presented approach can be extended easily to interpret (or *triplify*) other MediaWiki installations or even general document

[3] `http://wiktionary.dbpedia.org/`

collections, if they follow a global layout.

1.4. Structure

In section 2 we will introduce the domain of information extraction from wikis and RDF and related concepts, that form the basis of this thesis. Related work will be examined in section 3]. An overview over competing approaches and their properties is given. In section 4 and 5 we give an overview on requirements of the developed software, that arise in context of the DBpedia project. In addition resulting specifications and software architecture will be presented. In the following section 6 we will present some implementation details, that turned out to be essential for the success of the approach. Finally in section 7 the created dataset will be evaluated and compared with existing datasets.

2. Background

In the following, a short overview on underlying technologies is given.

2.1. Semantic Web

The world wide web is one of the most important inventions of our time. It enables the global access to documents and real time communication between individuals. This could be achieved by an interoperable infrastructure of independent networks, that can route any communication between two points. From our current perspective on the last three decades, this even seems technologically simple and maintainable at a reasonable cost. Furthermore the resulting benefits to our economy and society are beyond all expectations. A hole new industry was created and most industries are substantially influenced in their processes. The costs of communications dropped and both public and private communications switched to the new medium mostly. The world wide web is enabled by a set of related technologies, which can be summarized to the following core concepts:

- IP addressing, TCP transmission and routing
- client/server communication protocols like HTTP
- interlinked documents containing data (e.g. XML based) or services (e.g. interactive content) valuable for humans

The first mentioned technology provides for the global reachability of web servers; the second for accessing documents and data in them and the last for structuring information on them, so it can be processed for presentation or other purposes. While these technologies are well established and scale up to a huge amount of data, the users—humans—can barely cope with this amount of data offered. If one considers the WWW an information system, it only offers basic retrieval methods (a full text index via search engines) and most critical, it is fragmented into heterogeneous subsystems. Those, however, can offer very good retrieval methods. But there are several fundamental issues about the way data is handled today: Global interoperability is not supported on content level. Data currently is controlled by applications and each application keeps it to itself[4]. From a perspective of data integration, semantics are often vague or undefined. It may be unclear which entity a document refers to. Documents are only interlinked by untyped relations. These are strong limitations: If the dataset is too huge for a human to read and machines do not have deeper insight into it, the dataset as a whole can be seen as inaccessible. Of course the way the WWW works today seems to be well suited. Programmable web servers e.g. with PHP made the web interactive, enabling social interaction or collaborative editing. But future development is still blocked by the human-centric nature of the web. If all the knowledge humanity acquired and wrote down in e.g. Wikipedia would be also available to information

[4] http://www.w3.org/2001/sw/

systems in a structured way, even more knowledge could be inferred automatically and e.g. artificial intelligence, expert systems or search would be boosted dramatically. According to [3], the key to solving this issue lies in the establishment of *Linked Data*, which will be explained in the next section. The use of machine readable data and annotation of text with such data is crucial for information technology to enter the next level. So to conclude: The problem is the amount of knowledge and the lack of formalization e.g. machine readability. Even data that is already structured often lacks a defined semantic or the semantic is not formalized. The web as we know it is a web of documents, the target is the web of data. Instead of just linking documents to each other, *entities of the universe of discourse* should be linked. Consider this example: A company stores costumer information about you in its relational database; facebook keeps record of your activities in their distributed database and you yourself have a blog on your own web server. Why shouldn't all this personal information be linked[5]? This should be achieved by a global identifier for *you*, that is reused. Why do we have to supply contact information over and over again, although its database 101 that redundancy is bad. The answer is incompatibility on many levels. The ideal would be that all data is interoperable by design. The same approach of using unique identifiers can be applied to *all* entities (not just social profiles as in the example). Shortly after the invention of concepts for the WWW, Tim Berners-Lee et al. came up with the idea of the Semantic Web: an WWW where intelligent agents can act on behalf of users, to find information or communicate. They have a shared syntax and vocabulary, use ontologies as background knowledge and thus get deeper to the intended semantic of things. The Semantic Web (SW) is a set of complementary technologies, which are depicted in figure 1. It is a layered model to represent, query and manage information.
The effort was initiated in 2001 by Tim Berners-Lee, who defined it as

> "an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." [3]

There are several ways to augment information for better machine readability: One is the annotation of old fashioned documents with well defined shared vocabularies. An example for annotations is RDFa[6]. It allows to add structured information to arbitrary HTML documents. The second way is the creation of stand alone ontologies: "An ontology is a specification of a conceptualization" [8].

A conceptualization refers to some model of the world, created by a subject, for some purpose, that is shared by a group of individuals.

> "A body of formally represented knowledge is based on a conceptualization: the objects, concepts, and other entities that are assumed to exist in

[5] The reader may object privacy issues; but the focus of this thesis is on data integration. These two topics have to be considered separately: Just because data is interoperable, it is not accessible. Even more: If you avoid redundancy, you gain control over your data. How this control can be achieved is topic to current research but already very promising. Cf. WebID: `http://www.w3.org/wiki/WebID`

[6] `http://www.w3.org/TR/xhtml-rdfa-primer/`

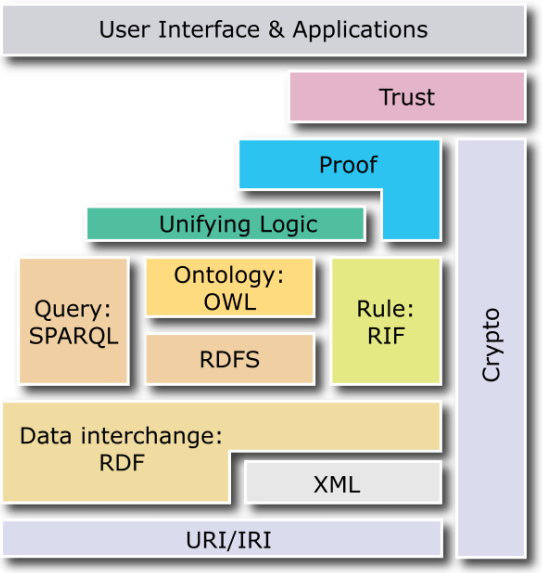

Figure 1: Semantic Web technologies

> some area of interest and the relationships that hold among them [7]. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly." [8]

A specification refers to a formal notion of the conceptualized knowledge.

> "When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects, and the describable relationships among them, are reflected in the representational vocabulary with which a knowledge-based program represents knowledge. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. In such an ontology, definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. [...] Formally, an ontology is the statement of a logical theory. [...] Ontologies are often equated with taxonomic hierarchies of classes, but class definitions, and the subsumption relation, but ontologies need not be limited to these forms. [...] In short, a commitment to a common ontology is a guarantee of consistency." [8]

The usage of shared, formal ontologies (which implies the agreement on a vocabulary and hence a common *language*) and the development of open standards provides

contracts for agents that act on behalf of a information interest of humans. The reuse of existing web standards (like URIs or the HTTP protocol) shall promote a fast adoption. The target is to make knowledge globally interoperable between different schemata and both humans and machines alike. Therefore knowledge becomes usable for software systems which allows for pioneering usage scenarios (cf. [14]).

2.2. RDF

The central technology to enable these ideas is RDF[7]. RDF itself is a universal data model, that dictates how information may be modelled on a structural level. The syntax is ruled by standards like RDF/XML[8] or N3[9], that describe how RDF is serialized to text. RDF is an acronym for Resource Description Framework. The basic idea is the identification of the *things and relations* within the universe of discourse (not just websites), with a URI[10]. About those resources, statements can be made in the form of *subject predicate object*—analogously to a natural language sentence. These statements are also called triples. For example the triple

```
<http://example.com/Alice> rdf:type foaf:Person
```

may encode the fact that "Alice is a person" (if the vocabulary has been defined accordingly). Notice: subject, predicate and object are URIs, but the predicate uses the prefix `rdf`, which is an abbreviation for an actual URI. The definition of this prefix would be:

```
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

The used class `Person` has been taken from the foaf vocabulary[11], which offers terms for people in a social environment. The triple can be interpreted as a directed graph: subject and object are nodes; the predicate is a typed edge.

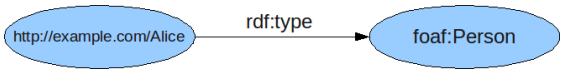

Figure 2: The triple as a graph

Multiple triples can be grouped in a graph, which is in turn identified by a URI. Such a graph can be used as an ontology. Thus ontologies in RDF format are actually graph databases which are inherently easily accessible for algorithms as opposed to natural language which always introduces ambiguity and noise. It is surprisingly easy to model even complex data or schemata in graphs. Additionally the formalized knowledge becomes easily accessible. Implicit information can be inferred (e.g. transitive

[7] http://www.w3.org/RDF/
[8] http://www.w3.org/TR/rdf-syntax-grammar/
[9] www.w3.org/DesignIssues/Notation3
[10] http://tools.ietf.org/html/rfc3986
[11] http://xmlns.com/foaf/0.1/

relations or inherited properties) or contractions can be detected[12].
In contrast to tree based models like XML, a relation does not have to be allocated to one of the participating entities. This eliminates a frequent modelling dilemma. Furthermore, such a flat net resembles the decentral nature of the web: The information is distributed and when merging two knowledge bases—at least at this level—there is no problem with schema normalization (cf. [14]). Of course there can be different schemas, but according to the classification of heterogeneity in [28] the problem can be reduced to its core—semantic heterogeneity, but for example schematic heterogeneity as in relational systems or XML can be solved by design. Object matching can be avoided where possible or wanted, reusing vocabularies and URIs in the ontology creation process.[13]
To complete the introduction of RDF, it is necessary to present the notion of data values. For example the number 5 or the string "abc" are not assigned URIs. It would make no sense as they are no independent entities within the universe of discourse. They are the value of properties and so they are assigned the special node type Literal[14]. They can not be subject for further statements. There are three types of literals:

- *plain literals*,
- *plain literals* with optional *language tag* and
- *typed literals*, that are augmented with a data type, that is given by a URI.

The NTriples serialization of a literal could be:

```
1 ex:Alice foaf:birthday "22.09.1986"^^xsd:date
```

In the example a typed literal is used to represent a date. The XSD vocabulary[15] for basic data types is used.

RDF can be stored either in text files or within special databases called *triple stores*, where common database techniques like indexing, query languages or backups are used. Some widely known triple stores are Virtuoso[16], Sesame [17] or Jena[18]. Storage backends vary widely from relational over graphs to in-memory.

2.3. Linked Data

Linked Data is the simplest and yet most important mechanism to retrieve RDF data: As described, entities are identified by URI's. When choosing the URL's one should pick a namespace under her authority, so that she can provide *some* data about the entity under the URL. As defined by W3C[19], the requirements for Linked Data are as follows:

1. Use URIs as names for things

[12] Access Control, Logic and Proof: http://www.w3.org/2000/01/sw/#access
[13] In practice the problem still persists due to administrative autonomy. Existing vocabularies are not reused due to lack of information, strategic decisions or suspected semantic mismatches. Schema and object matching in the semantic web is subject to ongoing research.
[14] http://www.w3.org/TR/rdf-concepts/#section-Graph-Literal
[15] http://www.w3.org/2001/XMLSchema#
[16] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/
[17] http://www.openrdf.org/
[18] http://jena.apache.org/
[19] http://www.w3.org/DesignIssues/LinkedData.html

2. Use HTTP URIs so that people can look up those names
3. When someone looks up a URI, provide useful information, using RDF
4. Include links to other URIs, so that they can discover more things

What is the rational behind doing so?

> "The first *Linked Data* principle advocates using URI references to identify, not just Web documents and digital content, but also real world objects and abstract concepts. These may include tangible things such as people, places and cars, or those that are more abstract, such as the relationship type of knowing somebody, the set of all green cars in the world, or the colour green itself. This principle can be seen as extending the scope of the Web from online resources to encompass any object or concept in the world. The HTTP protocol is the Web's universal access mechanism. In the classic Web, HTTP URIs are used to combine globally unique identification with a simple, well-understood retrieval mechanism. Thus, the second Linked Data principle advocates the use of HTTP URIs to identify objects and abstract concepts, enabling these URIs to be dereferenced (i.e., looked up) over the HTTP protocol into a description of the identified object or concept." [12]

The lookup of URI can even be enhanced with a mechanism called *Content Negotiation*: Based on the HTTP *accept* header that is set by the application requesting, different types of formatting can be used in the response. If for example a human browses RDF data using a web browser, a HTML version of the RDF data can be generated easily. If an application requests RDF data it would set the accept header to `application/rdf+xml` and get XML, which is easy to read by machines, but not humans. The mechanism is also transparent, it happens server side at request time. Modern RDF stores like *Virtuoso* have built in support for *Linked Data* with *Content Negotiation*.

These four basic principles together make a fundamental difference regarding the architecture of Linked Data seen as a database. Instead of custom communication protocols, well established web standards are used. This makes it interoperable at a technical level and easy to adopt. And it is backward compatible to very simple solutions: If one wants to publish *Linked Data*, no triple store is required at all, one could as well use a file server, with the documents available materialized. Also *Linked Data* implicitly is a *basic distributed database*: Because the query language is limited to a simple GET (for now), one can easily distribute data on different physical servers, while having constant access costs. Also mirroring and load balancing is easy by deploying any default web proxy, because Linked Data integrates transparently with established web technologies. But besides these technical benefits, the most important is, that linked data changes the way people administrate data. The integration aspect is being catered for by design and more important: The design also promotes data producers to regard integration issues at production time. If tool support grows fast enough, initial adoption costs and fears could be reduced. This would result in an overall decrease of integration efforts,

which in turn would prosper knowledge intensive applications. As presented in the introduction, solving knowledge intensive problems will be one of the key challenges of the next decades.

To embed the technology centric RDF standard into general publishing decision (for example by governments), Tim Berners-Lee suggested a simple rating system regarding openness and interoperability:

Table 1: Five star rating of Linked Data

★	Available on the web (whatever format) (optionally with an open licence, to be *Open Data*)
★★	Available as machine-readable structured data (e.g. excel instead of an image scan of a table)
★★★	two stars plus: Use of a non-proprietary format (e.g. CSV instead of excel)
★★★★	All the above plus: Use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff
★★★★★	All the above plus: Link your data to other people's data to provide context

It gives data producers a roadmap to high quality data and consumers objective hint about published datasets.

2.4. SPARQL

How can one query RDF data in a more sophisticated way than just the retrieval of single resources? One may want to search data that matches certain constraints or access a full text index over the data. SPARQL[20] is a recursive acronym *SPARQL Protocol and RDF Query Language* which is a query language for RDF graphs and a protocol for its use over a web service. It enables the matching of graph patterns (containing variables) in RDF graphs, thus it allows for the search and extraction of certain subgraphs. It developed from several other languages like SquishQL[21], RDQL[22], RQL[23] or SeRQL[24] (cf. [11]), and is an official W3C recommendation since 2008.
A very short overview about the features of the language shall be given in the following:

There are four types of SPARQL Queries:

- SELECT to extract parts of a subgraph that match a certain criteria
- ASK to check, if a subgraph exists in the graph

[20] `http://www.w3.org/TR/rdf-sparql-query/`
[21] cf. [9]
[22] `http://www.w3.org/Submission/RDQL/`
[23] `http://139.91.183.30:9090/RDF/RQL/`
[24] cf. [4]

- CONSTRUCT to creates a graph, which might be constructed by a search pattern
- DESCRIBE to gather information about matching resources. What is returned is not standardised, but mostly certain properties will be used to generate the description

A SPARQL SELECT query consists of the following parts:

- Prolog
  to declare prefixes and the base URI. Relative URIs, within the query are interpreted relative to the base URI. Prefixes are abbreviations for frequent or long URIs
- Projection
  similar to SQL: Variables (columns from SQL), which should be visible in the result or "*" for all used variables
- Result modifiers
  - DISTINCT for deduplication
  - REDUCED *can* remove duplicates, if it benefits the execution time.
  - ORDER BY sorts by an expression (mostly a variable)
  - LIMIT and OFFSET restrict a certain interval of results
- GraphPattern
  declares a subgraph to be searched for, consists of triples or optional graph patterns and more. But instead of explicitly declaring a graph, variables can be used to represent resources or literals, therefore the name pattern. Possible bindings for the variables are the result of the query.
- Filter
  can be part of a GraphPattern and restricts variable bindings by evaluation certain expressions upon them. When the expression is evaluated to false, the result is omitted from the overall result.

An example query:

```
1 PREFIX ex: <http://example.org/ns#>
2 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3 SELECT ?subj ?name
4 WHERE {
5    ?subj rdf:type foaf:Person .
6    ?subj foaf:age ?age
7    OPTIONAL { ?subj foaf:name ?name }
8    FILTER( ?age > 25 )
9 }
```

The query is evaluated against this dataset:

```
1 ex:Alice a foaf:Person
2 ex:Alice foaf:age "24"
3 ex:Bob a foaf:Person
4 ex:Bob foaf:age "26"
```

The query finds all persons who are older than 25. In detail the query does the following: First, two prefixes are declared, one for the FOAF vocabulary and one for our custom namespace; the prefixes are then used to abbreviate URIs within the GraphPattern. The WHERE part declares a GraphPattern, containing the variable `?subj` that is bound with all resources that are a `Person` from the FOAF vocabulary. In the seventh line the corresponding name is optionally bound to a variable. Optional implies that even if this triple can not be satisfied, the remaining pattern matches. In the example the name is not given, so the result will contain no value for the `?age` variable. In the eighth line the age is restricted be greater than 25. The triple store will try to match the query against the data and returns an answer in an XML based result format[25]. In our example there will be one result, containing a bound and an unbound variable:

subj	**age**
ex:Bob	

2.5. Scenarios

Now that we introduced many basics of the semantic web, one may ask: *"What is it all good for? Why cant we solve this with traditional approaches?"*

Consider this example: Alice works as a journalist. She often has to investigate in specific fields of history or science—fields she is no expert in—to support her articles. One day she has the information need for *"Olympia winners before 1972 from countries with less than 10 million inhabitants"*. How long will it take her? A few hours maybe. Then her boss comes and asks her to change that article to *"Olympia winners whose height is 10% above their countries average, who were born on Mondays"*. This may seem unrealistic, but apparently the information is available somewhere in the internet—most probably even in Wikipedia alone. And one could find it—but it would take ages. Except if this information would be available in RDF. Then it would take only seconds. This example should show that the computational access to information exposes a vast amount of new knowledge that was hidden inside existing data. Very complex knowledge intensive searches can be solved easily if the data is well structured.

Two problems come into play when trying to solve this scenario, which is an example for the research area of *question answering*:

1. understanding the question
2. finding the answer

In context of the semantic web, the first step could be solved by formalizing the question into a SPARQL query (by some black box of natural language processing magic), step two would require the information to be available in RDF.

The solution of the first one could be even assisted by the outcome of this thesis, as it provides for a large language resource, that can disambiguate query terms. The second

[25] `http://www.w3.org/TR/rdf-sparql-XMLres/`

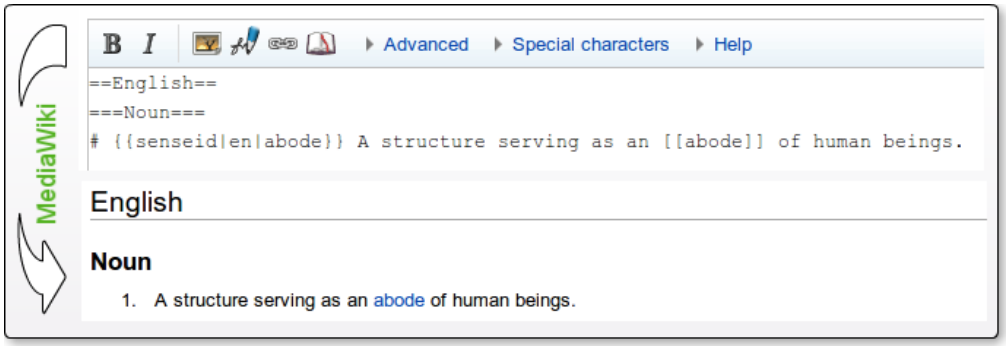

Figure 3: An excerpt of the *Wiktionary* page *house* with the rendered HTML.

one can be tackled by a related (and yet larger and more important) project, which is presented in the section after the next.

2.6. Wikitext

Pages in *Wiktionary* are formatted using the *wikitext* markup language[26], which is designed to be lightweight (easy to learn, quick to write—by humans). Upon request of a page, the MediaWiki engine renders this to an HTML page and sends it to the user's browser. An excerpt of the *Wiktionary* page *house* and the resulting rendered page are shown in Figure 3.

The markup == is used to denote headings, # denotes a numbered list (* for bullets), `[[link label]]` denotes links and `{{}}` calls a template. Templates are user-defined rendering functions that provide shortcuts aiming to simplify manual editing and ensuring consistency among similarly structured content elements. In MediaWiki, they are defined on special pages in the `Template:` namespace. Templates can contain any wikitext expansion, HTML rendering instructions and placeholders for arguments. In the example page in Figure 3, the `senseid` template[27] is used, which does nothing being visible on the rendered page, but adds an id attribute to the HTML `li`-tag (which is created by using #).

2.7. DBpedia

To describe the Wikipedia and DBpedia project, we found it is highly sufficient to simply quote from each of their self-portrayals:

> "Wikipedia is a free, collaboratively edited, and multilingual Internet encyclopedia supported by the non-profit Wikimedia Foundation. Its 22 million articles (over 4 million in English alone) have been written collaboratively by volunteers around the world. Almost all of its articles can be edited

[26] `http://www.mediawiki.org/wiki/Markup_spec`
[27] `http://en.wiktionary.org/wiki/Template:senseid`

> by anyone with access to the site, and it has about 100,000 regularly active contributors. As of August 2012, there are editions of Wikipedia in 285 languages. It has become the largest and most popular general reference work on the Internet, ranking sixth globally among all websites on Alexa and having an estimated 365 million readers worldwide. It is estimated that Wikipedia receives 2.7 billion monthly pageviews from the United States alone."[28]

DBpedia in turn wants to exploit this rich but unstructured dataset:

> "The DBpedia project is a community effort to extract structured information from Wikipedia and to make this information accessible on the Web. The resulting DBpedia knowledge base currently describes over 2.6 million entities. For each of these entities, DBpedia defines a globally unique identifier that can be dereferenced over the Web into a rich RDF description of the entity, including human-readable definitions in 30 languages, relationships to other resources, classifications in four concept hierarchies, various facts as well as data-level links to other Web data sources describing the entity. Over the last year, an increasing number of data publishers have begun to set data-level links to DBpedia resources, making DBpedia a central interlinking hub for the emerging Web of data. Currently, the Web of interlinked data sources around DBpedia provides approximately 4.7 billion pieces of information and covers domains such as geographic information, people, companies, films, music, genes, drugs, books, and scientific publications." [18]

In addition "DBpedia allows you to ask sophisticated queries against datasets derived from Wikipedia and to link other datasets on the Web to Wikipedia data"[1], because all extracted data becomes part of the semantic web and is accessible via powerful query languages or tools. DBpedia is one of the most successful semantic web projects and has become a central linking hub of Linked Data. The architecture of DBpedia is build around so called *Extractors*. The most important extractor is the infobox extractor; it tries to interpret tabular data that can be found on many Wikipedia pages. For example the page of Leipzig[29], contains a table on the right, that presents some essential facts and statistics. See figure 4 for example.

The underlying wikitext that creates the infobox on the right looks like this:

```
{{Infobox German location
|Art = Stadt
|image_flag = Flag of Leipzig.svg
|Wappen = Coat of arms of Leipzig.svg
|lat_deg = 51 |lat_min = 20 | lat_sec=0
|lon_deg = 12 |lon_min = 23 | lon_sec=0
|Lageplan = Lage der kreisfreien Stadt Leipzig in Deutschland.png
|Bundesland = Sachsen
|Regierungsbezirk = Leipzig
[...]
```

[28] http://en.wikipedia.org/wiki/Wikipedia

[29] http://en.wikipedia.org/wiki/Leipzig

Leipzig

From Wikipedia, the free encyclopedia

Coordinates: 51°20'0"N 12°23'0"E

For other uses, see Leipzig, Saskatchewan.

Leipzig (/ˈlaɪptsɪɡ/; German pronunciation: [ˈlaɪp͡tsɪç] (listen)) is one of the two largest cities (along with Dresden) in the federal state of Saxony, Germany. Leipzig is situated about 200 km south of Berlin at the confluence of the Weisse Elster, Pleisse and Parthe rivers at the southerly end of the North German Plain.

Leipzig has always been a trade city, situated during the time of the Holy Roman Empire at the intersection of the Via Regia and Via Imperii, two important trade routes. At one time, Leipzig was one of the major European centres of learning and culture in fields such as music and publishing.[2] After World War II, Leipzig became a major urban centre within the Communist German Democratic Republic but its cultural and economic importance declined.[2]

Leipzig later played a significant role in instigating the fall of communism in Eastern Europe, through events which took place in and around St. Nicholas Church. Since the reunification of Germany, Leipzig has undergone significant change with the restoration of some historical buildings, the demolition of others, and the development of a modern transport infrastructure. Leipzig has many institutions and opportunities for culture and recreation including a football stadium which has hosted some international matches, an opera house and one of the most modern zoos in Europe.[3]

In 2010, Leipzig was ranked 68th in the world as a livable city, by consulting firm Mercer in their quality of life survey. Also in 2010, Leipzig was included in the top 10 of cities to visit by the *New York Times*.

Leipzig	
Location of the town of Leipzig within Saxony [show]	
Coordinates	51°20'0"N 12°23'0"E
Administration	
Country	Germany
State	Saxony
Admin. region	Leipzig
District	Urban district
Lord Mayor	Burkhard Jung (SPD)
Basic statistics	
Area	297.60 km² (114.90 sq mi)
Population	531,809 (31 December

Figure 4: Wikipedia article about Leipzig with infobox

This information is already structured and by mapping the keys of the given properties to a RDF vocabulary it is simple to extract triples. The inner workings shall not be presented here. An important sub project of DBpedia is the *mappings wiki*[30], it provides a community maintained repository of those mappings from template argument keys to RDF properties.

3. Related Work

In the last five years, the importance of *Wiktionary* as a lexical-semantic resource has been examined by multiple studies. Meyer et al. ([22, 21]) present an impressive overview on the importance and richness of *Wiktionary*. In [33] the authors present the *JWKTL* framework to access *Wiktionary* dumps via a Java API. In [23] this JWKTL framework is used to construct an upper ontology called *OntoWiktionary*. The framework is reused within the *UBY project* [10], an effort to integrate multiple lexical resources (besides *Wiktionary* also *WordNet*, *GermaNet*, *OmegaWiki*, *FrameNet*, *VerbNet* and *Wikipedia*). The resulting dataset is modelled according to the *LMF ISO standard*[15]. [24] and [30] discuss the use of *Wiktionary* to canonize annotations on cultural heritage texts, namely the Thompson Motif-index. Zesch et. al. also showed, that *Wiktionary* is suitable for calculating semantic relatedness and synonym detection; and it outperforms classical approaches [34, 32]. Furthermore, other NLP tasks, such as sentiment analysis, have been conducted with the help of *Wiktionary* [5].
Several questions arise, when evaluating the above approaches: Why aren't there more NLP tools reusing the free *Wiktionary* data? Why aren't there web mashups of the data[31]? Why didn't *Wiktionary* became the central linking hub of lexical-semantic resources, yet?

[30] `http://mappings.dbpedia.org/`

[31] For example in an online dictionary from `http://en.wikipedia.org/wiki/List_of_online_dictionaries`

name	active	available	RDF	#triples	ld	languages
JWKTL	✓	dumps	✗	-	✗	en, de
wikokit	✓	source + dumps	✓	n/a	✗	en, ru
texai	✗	dumps	✓	∼ 2.7 million	✗	en
lemon scraper	✓	dumps	✓	∼16k per lang	✗	6
blexisma	✗	source	✗	-	✗	en
WISIGOTH	✗	dumps	✗	-	✗	en, fr
lexvo.org	✓	dumps	✓	∼353k	✓	en

Table 2: Comparison of existing Wiktionary approaches (ld = linked data hosting). None of the above include any crowd-sourcing approaches for data extraction. The wikokit dump is not in RDF.

From our point of view, the answer lies in the fact, that although the above papers present various desirable properties and many use cases, they did not solve the underlying knowledge extraction and data integration task sufficiently in terms of coverage, precision and flexibility. Each of the approaches presented in table 2 relies on tools to extract machine-readable data in the first place. In our opinion these tools should be seen independent from their respective usage and it is not our intention to comment on the scientific projects built upon them in any way here. We will show the state of the art and which open questions they raise.

JWKTL is used as data backend of *OntoWiktionary* as well as UBY[32] and features a modular architecture, which allows the easy addition of new extractors (for example *wikokit* [17] is incorporated). The Java binaries and the data dumps in LMF are publicly available. Among other things, the dump also contains a mapping from concepts to lexicalizations as well as properties for part of speech, definitions, synonyms and subsumption relations. The available languages are English, German (both natively) and Russian (through *wikokit*). According to our judgement, *JWKTL* can be considered the most mature approach regarding software architecture and coverage and is the current state of the art. *Texai*[33] and *Blexisma*[34] are also Java based APIs, but they are not maintained any more and were most probably made obsolete by changes to the *Wiktionary* layout since 2009. There is no documentation available regarding scope or intended granularity. A very fine grained extraction was conducted using WISIGOTH [29], but unfortunately there are no sources available and the project is unmaintained since 2010. Two newer approaches are the *lexvo.org* service and the algorithm presented in [19]. The *lexvo.org* service offers a linked data representation of *Wiktionary* with a limited granularity, namely it does not disambiguate on sense level. The source code is not available and only the English *Wiktionary* is parsed. As part of the Monnet project[35], McCrae et

[32] `http://www.ukp.tu-darmstadt.de/data/lexical-resources/uby/`, `http://www.ukp.tu-darmstadt.de/data/lexical-resources/uby/`

[33] `http://sourceforge.net/projects/texai/`

[34] `http://blexisma.ligforge.imag.fr/index.html`

[35] See `http://www.monnet-project.eu/`. A list of the adopted languages and dump files can be found at `http://monnetproject.deri.ie/lemonsource/Special:PublicLexica`

al. [19] presented a simple scraper to transform *Wiktionary* to the *lemon* RDF model [20]. The algorithm (like many others) makes assumptions about the used page schema and omits details about solving common difficulties as shown in the next section. At the point of writing, the sources are not available, but they are expected to be published in the future. Although this approach appears to be the state of the art regarding RDF modelling and linking, the described algorithm will *not scale to the community-driven heterogeneity* as to be defined in section 4. All in all, there exist various tools that implement extraction approaches at various levels of granularity or output format. In the next section, we will show several challenges that, in our opinion, are insufficiently tackled by the presented approaches. Note that this claim is not meant to diminish the contribution of the other approaches as they were mostly created for solving a single research challenge instead of aiming to establish *Wiktionary* as a stable point of reference in computational linguistics using linked data.

4. Problem Description

In order to conceive a flexible, effective and efficient solution, in this section we survey the challenges associated with Wiki syntax, *Wiktionary* and large-scale extraction.

4.1. Processing Wiki Syntax

As introduced in section 2, a wiki is a set of pages, formatted in wikitext. For display in a browser, they can be rendered to HTML. Operating on the parsed HTML pages, rendered by the *MediaWiki engine*, does not provide any significant benefit, because the rendered HTML does not add any valuable information for extraction. Processing the database backup XML dumps[36] instead, is convenient as we could reuse the DBpedia extraction framework[37] in our implementation. The framework mainly provides input and output handling and also has built-in multi-threading by design. If the English *Wiktionary* community decides to change the layout of sense definitions at some point in the future, only a single change to the template definition is required. Templates are used heavily throughout *Wiktionary*, because they substantially increase maintainability and consistency. But they also pose a problem to extraction: On the unparsed page only the template name and it's arguments are available. Mostly this is sufficient, but if the template adds static information or conducts complex operations on the arguments (which is fortunately rare), the template result can only be obtained by a running MediaWiki installation hosting the pages. The resolution of template calls at extraction time slows down the process notably and adds additional uncertainty.

4.2. Wiktionary

Wiktionary has some unique and valuable properties:

- **Crowd-sourced**
  Wiktionary is community edited, instead of expert-built or automatically generated from text corpora. Depending on the activeness of it's community, it is up-to-date to recent changes in the language, changing perspectives or new research. The editors are mostly semi-professionals (or guided by one) and enforce a strict editing policy. Vandalism is reverted quickly and bots support editors by fixing simple mistakes and adding automatically generated content. The community is smaller than Wikipedia's but still quite vital: Between 50 and 80 very active editors with more than 100 edits per month for the English *Wiktionary* in 2012[38].
- **Multilingual**
  The data is split into different Wiktionary Language Editions (WLE, one for each language). This enables the independent administration by communities and leaves the possibility to have different perspectives, focus and localization. Simultaneously one WLE describes multiple languages; only the representation

[36] `http://dumps.wikimedia.org/backup-index.html`
[37] `http://wiki.dbpedia.org/Documentation`
[38] `http://stats.wikimedia.org/wiktionary/EN/TablesWikipediaEN.htm`

language is restricted. For example, the German *Wiktionary* contains German descriptions of German words *as well as* German descriptions for English, Spanish or Chinese words. Particularly the linking across languages shapes the unique value of *Wiktionary* as a rich multi-lingual linguistic resource. Especially the WLE for not widely spread languages are valuable, as corpora might be rare and experts are hard to find.

- **Feature rich**
  As stated before, *Wiktionary* contains for each lexical word[39] a disambiguation regarding language, part of speech, etymology and senses. Numerous additional linguistic properties may be given for each part of speech usage or sense. Such properties include word forms, taxonomies (hyponyms, hyperonyms, synonyms, antonyms) and translations. Well maintained pages (e.g. frequently used words) often have more sophisticated properties such as derived terms, related terms and anagrams.
- **Open license**
  All the content is dual-licensed under both the *Creative Commons CC-BY-SA 3.0 Unported License*[40] as well as the *GNU Free Documentation License (GFDL)*.[41] All the data extracted by our approach derives these licences.
- **Big and growing**
  English contains 2.9M pages, French contains 2.1M, followed by Chinese with 1.2M and later German with 0.2 M pages. The overall size (12M pages) of *Wiktionary* is in the same order of magnitude as Wikipedia's size (20M pages)[42]. The number of edits per month in the English *Wiktionary* varies between 100k and 1M — with an average of 200k for 2012 so far. The number of pages grows: In the English *Wiktionary* with approx. 1k per day in 2012.[43]

The most important resource to understand how *Wiktionary* is organized are the *Entry Layout Explained* (ELE) help pages. As described above, a page is divided into sections that separate languages, part of speech etc. The table of content on the top of each page also gives an overview of the hierarchical structure. This hierarchy is already very valuable as it can be used to disambiguate a lexical word. The schema for this tree is restricted by the ELE guidelines[44]. The entities of the ER diagram illustrated in Figure 5 will be called *block* from now on. The schema can differ between WLEs and normally evolves over time.

[39] A lexical word is just a string of characters and has no disambiguated meaning yet
[40] `http://en.wiktionary.org/wiki/Wiktionary:Text_of_Creative_Commons_Attribution-ShareAlike_3.0_Unported_License`
[41] `http://en.wiktionary.org/wiki/Wiktionary:GNU_Free_Documentation_License`
[42] `http://meta.wikimedia.org/wiki/Template:Wikimedia_Growth`
[43] `http://stats.wikimedia.org/wiktionary/EN/TablesWikipediaEN.htm`
[44] For English see `http://en.wiktionary.org/wiki/Wiktionary:ELE`.

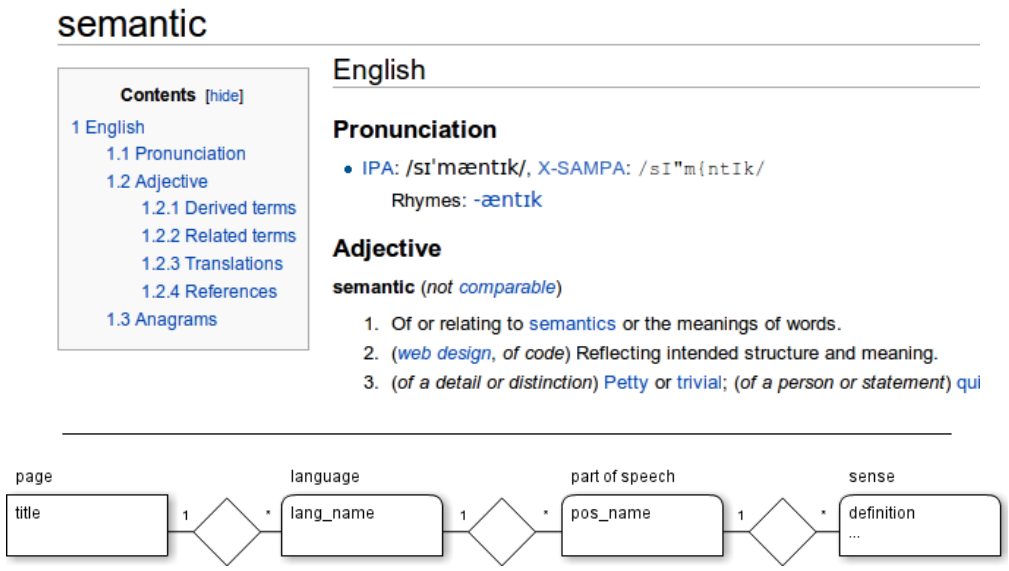

Figure 5: Example page `http://en.wiktionary.org/wiki/semantic` and underlying schema (only valid for the English *Wiktionary*, other WLE might look very different.)

4.3. Wiki-scale Data Extraction

The above listed properties that make *Wiktionary* so valuable, unfortunately pose a serious challenge to extraction and data integration efforts. Conducting an extraction for specific languages at a fixed point in time is indeed easy, but it eliminates some of the main features of the source. To fully synchronize a knowledge base with a community-driven source, one needs to make distinct design choices to fully capture all desired benefits. MediaWiki was designed to appeal to non-technical editors and abstains from intensive error checking as well as formally following a grammar — the community gives itself just layout guidelines. One will encounter fuzzy modelling and unexpected information. Editors often see no problem with such *noise* as long as the page's visual rendering is acceptable. Overall, the issues to face can be summed up as

1. the constant and frequent changes to data *and schema*,
2. the heterogeneity in WLE schemas and
3. the human-centric nature of a wiki.

From the perspective of requirements engineering, they result in a number of requirements, that do not easily fit together. Figure 6 illustrates the different requirements.

The requirement *Expressiveness* is the sum of all functional requirements. The other two are non-functional as they regard the long term sustainability of the development process. Approaches with hard coded algorithms mostly cover only the first requirement. If a modular architecture is used, it might be easy to cover the *Flexibility to new*

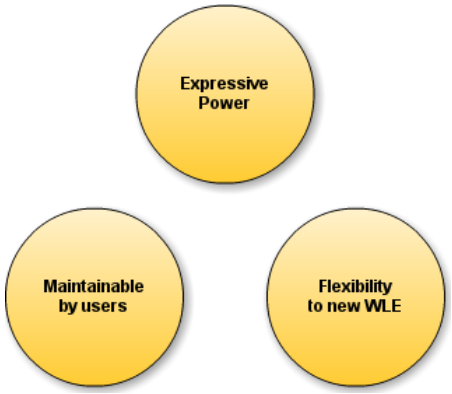

Figure 6: Competing requirements in wiki scale data extraction

WLE; an extractor for each WLE might be hard coded. Although it might be arguable whether the growing code base stays maintainable, with sufficient effort of software developers, such an approach theoretically could scale. However, development costs could be reduced drastically, if the extraction is maintained by users. The two non-functional requirements conflict with expressiveness as they are implemented with a declarative pattern in our case and we argue that it is essential to do so, because only a declarative approach can hide the complexity. But the more features the declarative language supports, the harder it becomes for non experts to use it. In section 6 it will be shown which trade off is chosen.

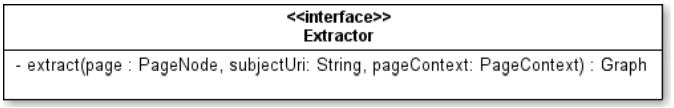

Figure 7: The extractor interface.

5. Specification

In the following we will describe the used architecture.

Existing extractors as presented in section 3 mostly suffer from their *inflexible* nature resulting from their narrow use cases at development time. Very often approaches were only implemented to accomplish a short term goal (e.g. prove a scientific claim) and only the needed data was extracted in an *ad-hoc* manner. Such evolutionary development generally makes it difficult to generalize the implementation to heterogeneous schemas of different WLE. Most importantly, however, they ignore the community nature of a *Wiktionary*. Fast changes of the data require ongoing maintenance, ideally by the wiki editors from the community itself or at least in tight collaboration with them. These circumstances pose serious requirements to software design choices and should not be neglected. All existing tools are rather monolithic, hard-coded black boxes. Implementing a new WLE or making a major change in the WLE's ELE guidelines will require a programmer to refactor most of its application logic. Even small changes like new properties or naming conventions will require software engineers to align settings. The amount of maintenance work necessary for the extraction correlates with change frequency in the source. Following this argumentation, a community-built resource can only be efficiently extracted by a community-configured extractor. This argument is supported by the successful crowd-sourcing of DBpedia's internationalization [16] and the non-existence of *open* alternatives with equal extensiveness.
Given these findings, we can now conclude four high-level design goals:

- declarative description of the page schema;
- declarative information/token extraction, using a terse syntax, maintainable by non-programmers;
- configurable mapping from language-specific tokens to a global vocabulary;
- fault tolerance (uninterpretable data is skipped).

The extractor is built on top of the the DBpedia framework, and thus it is required to conform to a simple interface, shown in figure 7.

Extractors are registered with the framework via a configuration file, and instantiated with a requested context. The context can be the DBpedia ontology or the selected language etc. Extractors are then subsequently invoked for every page of the MediaWiki XML dump. The framework passes the `page`—in parsed form of an AST, the `subjectURI`—the URI of the resource this page should be referring to (e.g. `http://dbpedia.org/resource/$PAGENAME`)— and the `pageContext`— a helper for URI generation. The interface defines the extractor to return a Graph in turn, which is basically a set of triples (or quads in this case). Internally the extractor will inspect the

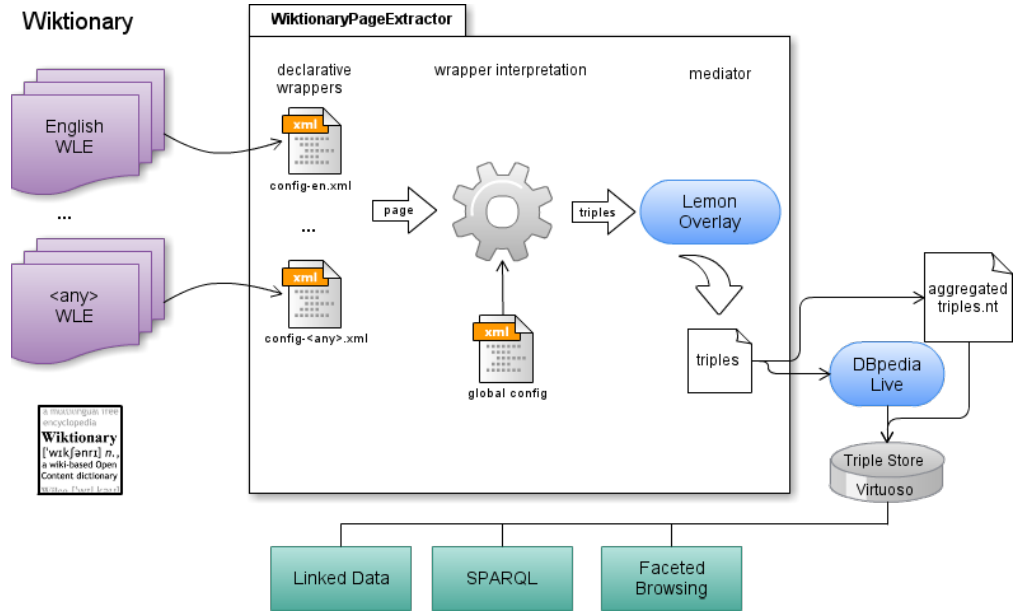

Figure 8: Architecture for extracting semantics from Wiktionary leveraging the DBpedia framework

AST and generate triples, when he finds information he interprets as relevant. This straightforward interface makes the DBpedia framework so modular. Input and output handling (parsing and serialization) is left to the framework, and the resulting RDF data can be directly inserted into a triple store.

We solve the shown requirements with an additional extractor, which internally follows a rather sophisticated workflow, shown in figure 8.

The *Wiktionary* extractor is invoked by the DBpedia framework to handle a page. It uses a language-specific configuration file, that has to be tailored to match the WLE's ELE guidelines to interpret the page, to extract the desired information. At first, the resulting triples still adhere to a language-specific schema, that directly reflects the configured layout of the WLE. A generic lossless transformation and annotation using the *lemon* vocabulary is then applied to enforce a global schema and reduce semantic heterogeneity. Afterwards the triples are returned to the DBpedia framework, which takes care of the serialization and (optionally) the synchronization with a triple store via DBpedia Live[45] [25]. The process of interpreting the declarative wrapper is explained more detailed in Figure 9.

The actual algorithm is quite complex and will be explained in-depth in the next section. The basic idea is the separation in three phases:

- preprocessing to skip pages that do not represent a lexical word
- the actual extraction with the three steps of analysing the page structure, match-

[45] `http://live.dbpedia.org/live`

Figure 9: Overview of the extractor work flow.

ing templates and generating triple from the result

- post-processing to normalize schemata and polishing the output

The extractor itself is split into two components: a generic template matcher, that takes a (possibly partially consumed) wiki page and a template. It then tries to *bind* the variable parts of the template to actual content from the page. When a template is successfully matched, the matched part of the page is consumed—removed from the page. This component is called the `VarBinder`. The `VarBinder` is a stateless tool that has no knowledge of the overall page layout

6. Design and Implementation

In the following we will present a bunch of noteworthy implementation details. First of all, the implementation is done in *Scala*[46] , because the DBpedia extraction framework is written in Scala as well, but by the nature of Scala, extensions can also be written in Java or any other JVM language[47]. The source code is available via the official DBpedia Mercurial repository[48] in the `wiktionary` branch.

6.1. Extraction Templates

As mentioned in section 4.2, we define a *block* as the part of the hierarchical page that is responsible for a certain entity in the extracted RDF graph. For each *block*, there can be declarations on how to process the page on that level. This is done by so called *extraction templates* (called *ET*; not to be confused with the templates of *wikitext*). Each possible section in the *Wiktionary* page layout (i.e. each linguistic property) has an ET configured, explained in detail below. The idea is to provide a declarative and intuitive way to encode *what to extract*. For example consider the following page snippet:

```
1 ===Synonyms===
2 * [[building]]
3 * [[company]]
```

Since the goal is to emit a triple for each link per line, we can write the ET in the following style:

```
1 ===Synonyms===
2 (* [[\$target]]
3 )+
```

Lets analyse what features are available to build ET:

Template matching: To match a template against a page, the VarBinder is employed by the Extractor. Page and template are compared node by node, as you can see in figure 10:

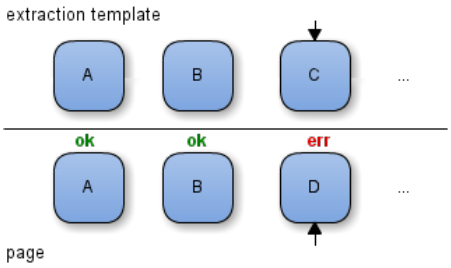

Figure 10: Matching an extraction template against a page

[46] http://www.scala-lang.org/
[47] http://en.wikipedia.org/wiki/List_of_JVM_languages
[48] http://dbpedia.hg.sourceforge.net/hgweb/dbpedia/extraction_framework/

Internally a `Stack`[49] is used; if the head nodes of both stacks are equal, they are consumed, if not an `Exception` is thrown to notify about a mismatch.

Variables: In the extraction template, there can be special nodes (e.g. variables). If a variable is encountered, all nodes from the page are being recorded and saved as a binding for that variable. All variables that are recorded within an extraction template are collected and returned as a result of the template being matched against the page.

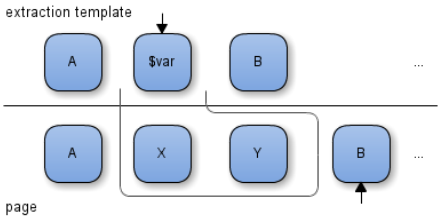

Figure 11: recording a variable

Some short notes about variables:

- The pattern to recognize them is `\$[a-zA-Z0-9]`.
- They stop recording the page when they encounter the token that follows the variable in the extraction template, as you can see in figure 11.
- If there is no node following them, they consume everything.
- If they record too many nodes (e.g. the whole page), they are assumed to be faulty and an exception is thrown.

Repetition: They implement the possibility to have subtemplates that can be repeated. Their syntax is intuitive by grouping a subtemplate with brackets and one of three modifiers:

- $\star$ for *0..n* matches,
- + for *1..n* matches and
- `?` for *0..1* matches.

To implement repetitions, the `VarBinder` extracts the nodes from within the brackets, treats them as a subtemplate and simply tries to match the subtemplate multiple times against the page. It stops if either the maximum number of repetitions is reached (only for the `?`-modifier) or the subtemplate does not match any more. Thus it expands the repetition as far as possible. *How do variables relate to subtemplate?* If a variable is used in a repetition and bound twice, it doubles the number of varbindings. Variables outside the subtemplate are then duplicated. Formalized: The flattened version of the (directed) *binding tree* is the *set of all paths* starting at the root.

Error tolerance: Due to the human-centric nature of a wiki, pages often contain unexpected information: An additional image, an editor note or a rare template. To compensate this, we decided to add the possibility to weaken the conditions for a template mismatch. When a node is encountered on the page, that is not expected from the template, the template is not immediately aborted, but instead the `VarBinder` keeps

[49] `http://www.scala-lang.org/api/current/scala/collection/mutable/Stack.html`

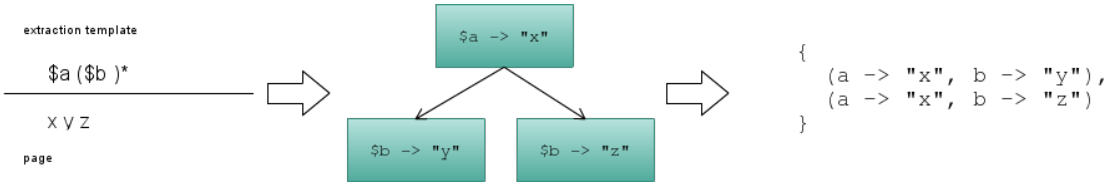

Figure 12: using variables in repetitions

record of the error and this unexpected node is skipped. To limit these skips, a window over the last s nodes is observed, to calculate an error threshold *maxError*. This allows the template to recover from local errors if it continues to match later. Additionally the edge case of templates with length 0 or 1 and 1 unexpected node, should be avoided to succeed by the *minCorrect* parameter that prevents templates from matching too easily.

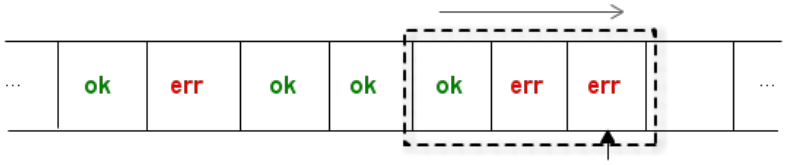

Figure 13: Error tolerance with a sliding window ($s = 3, minCorrect = 1, maxError = 1$)

The example in figure 13 shows how confined errors (the single one) are ignored but major errors (like the two consecutive ones) will prevent the template from matching. This implements a sliding window as only the last s nodes are considered and this window progresses with the page being consumed. These are the most important features of extraction templates.

Back to our example: The found *variable bindings* are `{($target -> "building"), ($target -> "company")}`. How do we transform these bindings into RDF triples? We simply invert the idea of extraction templates to *result templates* (called RT). We insert the value of variables into subject, predicate and object of a RDF triple:

```
1 <triple s="http://some.ns/$entityId" p="http://some.ns/hasSynonym" o="http://some.ns/
      $target" />
```

Notice the reuse of the `$target` variable: The data extracted from the page is inserted into a triple. The variable `$entityId` is a reserved global variable, that holds the page name i.e. the word. The created triples in N-Triples syntax are:

```
1 <http://some.ns/house-1> <http://some.ns/hasSynonym> <http://some.ns/building> .
2 <http://some.ns/house-1> <http://some.ns/hasSynonym> <http://some.ns/company> .
```

The used RT can be more complex, as explained below.

6.2. Algorithm

The algorithm of processing a page works as follows:
Input: The parsed page is obtained from the DBpedia Framework (essentially a lexer is

used to split the wikitext into tokens).

1. Filter irrelevant pages (user/admin pages, statistics, list of things, files, templates, etc.) by applying string comparisons on the page title. Return an empty set of triples in that case.

2. Build a *finite state automaton*[50] from the page layout encoded in the WLE specific XML configuration. This automaton contains a state for each *block*. This schema also contains so called *indicator templates* for each *block*, that—if they match at the current page token—indicate that their respective block starts. This way *indicator templates* trigger *state transitions* in the automaton, thus automaton recognizes the page layout and keeps track of the current block. In this regard the mechanism is similar to [19], but in contrast our approach is declarative — the automaton is constructed *on-the-fly* and not hard-coded. The current state represents the current position in the disambiguation tree.

3. The page is processed token by token:
   a) Check if *indicator templates* match. If yes, the corresponding block is entered. The *indicator templates* also emit triples like in the *extraction template* step below. These triples represent the block in RDF—for example the resource `http://wiktionary.dbpedia.org/resource/semantic-English` represents the English block of the page *semantic*.
   b) Check if any *extraction template* of the current block match. If yes, transform the variable bindings to triples.[51] Localization specific tokens are replaced as configured in the so called *language mapping* (explained in detail in section 6.3).

4. The triples are then *transformed*. In our implementation *transformation* means, that all triples are handed to a static function, which returns a set of triples again. One could easily load the triples into a triple store like JENA and apply arbitrary SPARQL Construct and Update transformations. This step basically allows post-processing, e.g. consolidation, enrichment or annotation. In our case, we apply the schema transformation (by the mediator) explained in detail in section 6.6.

5. The triples are sorted and de-duplicated to remove redundancy in the RDF dumps.

Output: Set of triples (handed back to the DBpedia Framework).

6.3. Language Mapping

The language mappings are a very simple way to translate and normalize tokens, that appear in a WLE. In the German WLE, for example, a noun is described with the Ger-

[50] Actually a finite state transducer, most similar to the Mealy-Model.

[51] In our implementation either declarative rules are given in the XML config or alternatively static methods are invoked on user-defined classes (implementing a special interface) for an imperative transformation. This can greatly simplify the writing of complex transformation.

man word *Substantiv*. Those tokens are translated to a shared vocabulary, before emitting them (as URIs for example). The configuration is also done within the language specific XML configuration:

```
<mapping from="Substantiv" to="Noun">
<mapping from="Deutsch" to="German">
...
```

The mapping consists currently of mappings for part of speech types and languages, but arbitrary usage is possible. Section 6.5 shows how this mapping is used.

6.4. Reference Matching

A *Wiktionary* specific requirement is the resolution of intra-page references: All Wiktionaries use *some* way to refer to parts of the traits of the word. For example on the page *house* senses are defined:

```
# A structure serving as an [[abode]] of human beings.
# {{politics}} A deliberative assembly forming a component of a legislature, or, more
    rarely, the room or building in which such an assembly normally meets.
# [[house music|House music]].
```

Later on relations to other words are noted — but *in context* of a sense. For example house *in context* of *abode* has the translation *Haus* in German. So the following notion is used:

```
====Translations====
{{trans-top|abode}}
...
* German: {{t+|de|Haus|n}}, {{t+|de|Häuser|p}}
...
```

The problem is to match the gloss that is given in the `trans-top` template argument against the available senses. The senses have been assigned URIs already; now those are needed to serve as the subject for the translation triples. There is no simple way to determine which sense URI belongs to which gloss. As described in [23] as *relation anchoring*, a string based measure is used[52]. There is a simple data structure that is initialized per page, to this matching mechanism. Which measure is used can be configured in the global configuration. Available measures are: Levenshtein and trigram set similarity with dice, jaccard or overlap coefficient. A sense can be registered with the `Matcher` by passing its definition sentence. An *id* is generated for that sense. Later on, glosses (short forms of the definition) that refer to a sense can be passed to look up which sense matches best and the corresponding *id* is returned. Opposed to existing approaches, we make no assumptions on how such references are noted. The English *Wiktionary* uses glosses; the German one uses explicit numbers (that don't need to be

[52] Opposed to the approach described in [23], we try to focus on explicit information. Determining the sense URI of a translation triple is already error prone, but so called *target anchoring* is not performed. *Target anchoring* refers to the disambiguation of the target word (or entity): this target of course has also a disambiguation tree, and it is possible to *bend* the link to point to a node deeper in that tree instead of just the root node. We consider this highly assumptious and it introduces noise. We leave that to postprocessing. Also it is not implementable easily within the DBpedia framework, because data extracted on other pages is not available to an extractor at runtime.

matched), the Russian and French uses a combination of both—sometimes senses are explicitly referred to by their numbers, sometimes with a gloss. So we came up with a customizable way to use the reference matcher. section 6.5 shows how this mechanism is used.

6.5. Formatting functions in Result Templates

The question arises, how this mapping is then used within the application. It is certainly not reasonable to replace *all* occurrences of those `from` tokens. This would lead to a number of false positive matches and screwed output. It is crucial to offer a possibility to configure in which context output should be handled like that. Therefore, we introduced formatting functions in result templates: When you define the triples that are generated, you can apply functions to e.g. variables. An example:

```
<triple s="http://some.ns/uri($entityId)" p="http://some.ns/hasLanguage" o="http://some
    .ns/map($target)" />
```

In this RT, two functions are used: `uri` and `map`. They are wrapped around variables and during rendering the RT to triples, they are evaluated. The following functions are available:

`uri(str)`	URL encode
`map(str)`	replace if mapping found
`assertMapped(str)`	don't emit triple if not in mapping vocabulary
`assertNumeric(str)`	don't emit triple if argument is not numeric
`getId(str)`	look up a gloss and get the id of the best matching sense
`getOrMakeId(str)`	as `getId` but generate id if below a similarity threshold
`makeId(str)`	save a sense and generate an id
`saveId(str, str)`	save a sense with a given id

The functions with `Id` in their name relate to the matching introduced in section 6.4. Continuing the example of senses and translations, one would configure the RT to save definition sentences into the `Matcher`, when generating triples about definitions and later looking up (matching) the gloss, when generating triples about the translation section:
The RT for the definitions may be:

```
<triple s="http://some.ns/$entityId-makeId($definition)" p="http://some.ns/
    hasDefinition" o="$definition" oType="literal" />
```

and for the translations:

```
<triple s="http://some.ns/$entityId-getId($gloss)" p="http://some.ns/hasTranslation" o=
    "http://some.ns/uri($target)" />
```

The idea is that, if the matching is correct, the subject URIs are equal (e.g. `http://some.ns/house-1`) in both triples: There are two triples about one resource, thus the information is successfully merged.

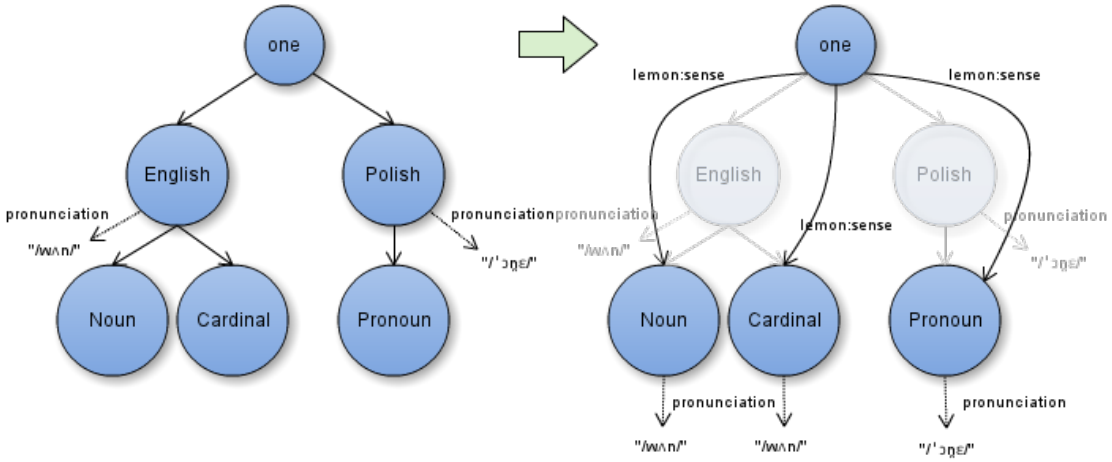

Figure 14: Schema normalization.

```
1 <http://some.ns/house-1> <http://some.ns/hasDefinition> "A structure serving as an
      abode of human beings." .
2 <http://some.ns/house-1> <http://some.ns/hasTranslation> <http://some.ns/Haus> .
```

6.6. Schema Mediation by Annotation with *lemon*

The last step of the data integration process is the schema normalization. The global schema of all WLE is not constructed in a centralized fashion—instead we found a way to both making the data globally navigable and keeping the heterogeneous schema without loosing information. *lemon* [20] is an RDF model for representing lexical information (with links to ontologies—possibly DBpedia). We use part of that model to encode the relation between *lexical entries* and *lexical senses*. *lemon* has great potential of becoming the *de facto* standard for representing dictionaries and lexica in RDF and is currently the topic of the OntoLex W3C Community group[53]. The rationale is to add *shortcuts* from *lexical entities* to *senses* and propagate properties that are along the intermediate nodes down to the senses. This can be accomplished with a generic tree transformation algorithm, regardless of the depth of the tree and used links. Applications assuming only a *lemon* model, can operate on the shortcuts and, if applied as an overlay and leaving the original tree intact, this still allows applications, to operate on the actual tree layout, too. The (simplified) procedure is presented in Figure 14[54]. The use of the *lemon* vocabulary and model as an additional schema layer can be seen as our mediator. This approach is both lightweight and effective as it takes advantage of *multi-schema modelling*.

[53] `http://www.w3.org/community/ontolex/`

[54] Note, that in the illustration it could seem like the information about part-of-speech would be missing in the *lemon* model. This is not the case: From the part-of-speech nodes, there is a link to corresponding language nodes, which is omitted for brevity. These links are also propagated down the tree.

6.7. Configuration

As presented in section 4, the most important requirement of the approach is configurability. The extractor itself is as generic as possible, it is not tailored to linguistics or even *Wiktionary*. It has a commitment to wiki syntax, but is also able to process plain text as it can be interpreted as wikitext without markup, thus the extractor may be suitable for most flat file formats. However the configuration makes up the heart of the extractor: it is a big XML file interpreted at runtime and describes how to interpret the page. We will go through the available options and show their relevance to the given requirements.

At first, the configuration is splitted into a generic part and a language specific part. The generic part is always loaded and does not need to be localized to a WLE. It contains options like the namespace, in which all URIs are created and an option that specifies which language configuration should be used. The language specific configuration is loaded, based on that option, at runtime. It has to be tailored to a WLE by the maintainer of the dataset.

Both configuration types are stored in the `config` folder. The naming convention is straight forward, `confix.xml` holds the generic config, the other files are language configurations:

```
config
├── config.xml
├── config-de.xml
├── config-en.xml
└── ...
```

The generic configuration has two parts—a properties list and a mapping. The properties are the mentioned namespace, the language, the `loglevel` (to configure debug verbosity) and options to configure the matcher. The mapping is — as explained in section 6.3 — a way to replace tokens found on the page to a global vocabulary. But opposed to language specific tokens, in the generic configuration, globally used tokens are configured. It is used to provide a mapping from ISO 639-1 and -2 codes to the *Wiktionary* vocabulary.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <config>
3     <properties>
4        <property name="logLevel" value="0"/>
5        <property name="language" value="ru"/>
6        <property name="ns" value="http://wiktionary.dbpedia.org/"/>
7        <property name="matchingStrategy" value="levenshtein"/>
8        <property name="matchingThreshold" value="0.5"/>
9     </properties>
10    <mappings>
11       <!-- ISO 639-1 -->
12       <mapping from="aa" to="Afar" />
13       <mapping from="ab" to="Abkhazian" />
14       <mapping from="ae" to="Avestan" />
15       ...
```

The language specific configuration is probably the most important part of this thesis. The created XML dialect directly reflects the expressiveness of the extraction approach. As explained in section 4, the expressiveness is limited by the complexity of the declar-

ative language. However, the declarative language should remain simple to keep it easily usable by non experts. The interpreter should be as generic as possible. In the following we will present which trade off was chosen and how the layout of a WLE is modelled in our XML dialect.
The configuration for English shall serve as an example here[55]. The XML is structured as

<config>
<ignore>
<mappings>
<postprocessing>
<saveVars>
<templateRepresentativeProperties>
<page>

The <ignore> section configures which pages shall be skipped and not used for extraction. This is used to skip help pages or user profiles, but it can also be used to skip pages like conjugation tables, as they are not handled yet. An example for this section could be

```
<ignore>
      <page startsWith="Help:" />
      <page endsWith=" (Conjugation)" />
      ...
```

There are two options to determine if a page should be skipped: Prefix or suffix matches in the page title.
The mappings section has been explained in 6.3, it is used to translate language specific terms for languages or part of speech types and can be invoked by formatting functions in result templates as explained in 6.5.
<postprocessing> configures whether and how the extracted triples of a page should be handled. It is possible to pass them to so called Postprocessors (that are JVM classes, visible in classpath, that need to implement a certain interface Postprocessor). This Postprocessor can be configured by arbitrary XML nodes. The interpretation of those is left to the class itself. An example for Postprocessors is the lemon overlay. It is invoked like this:

```
<postprocessing enabled="true" ppClass="org.dbpedia.extraction.mappings.wikitemplate.
    wiktionary.postprocessor.LemonOverlay">
  <config>
    <blockProperty uri="http://www.monnet-project.eu/lemon#sense"/>
    <inputTargetClass uri="http://wiktionary.dbpedia.org/terms/Sense"/>
    <followProperties>
      <property uri="http://wiktionary.dbpedia.org/terms/hasPoSUsage"/>
      <property uri="http://wiktionary.dbpedia.org/terms/hasLangUsage"/>
      <property uri="http://wiktionary.dbpedia.org/terms/hasSense"/>
    </followProperties>
    <collectProperties>
      <property uri="http://purl.org/dc/elements/1.1/language"/>
      <property uri="http://www.w3.org/2000/01/rdf-schema#label"/>
      <property uri="http://wiktionary.dbpedia.org/terms/hasMeaning"/>
      <property uri="http://wiktionary.dbpedia.org/terms/hasTranslation"/>
      <property uri="http://wiktionary.dbpedia.org/terms/hasExampleSentence"/>
```

[55] http://dbpedia.hg.sourceforge.net/hgweb/dbpedia/extraction_framework/file/tip/wiktionary/config/config-en.xml

```
    ...
  </collectProperties>
  <outputStartClass uri="http://www.monnet-project.eu/lemon#LexicalEntry"/>
  <outputAggregatedClass uri="http://www.monnet-project.eu/lemon#LexicalSense"/>
 </config>
</postprocessing>
```

`<saveVars>` allows to cache variables between template matches. Normally the only variables visible in result templates, are the ones bound in the extraction templates. We extended this with a cache, so certain variables can be kept. If a variable is set to be saved like

```
<saveVars>
 <var name="pos"/>
 <var name="language"/>
 <var name="definition"/>
</saveVars>
```

it's last value stays available to be used in further result templates. In other words: After being bound, the variable stays visible, it gets a global scope. This memory allows a kind of *context sensitivity* by means of a *look back*. The mechanism is currently not used. `templateRepresentativeProperties` works as a very simple *template resolution* mechanism. To resolve templates (render them to readable text), actually a running MediaWiki instance is necessary. But often this is superfluous: It might be sufficient to simply choose an argument of that template to represent it readable. For example the English template `term` is used to format links to other words e.g. `derived from {{term|hus}}, from {{proto|Germanic|husan}}, ...`; it is printed as the word itself, which is the first argument. Other informations can be ignored. So we came up with this simple but effective mechanism to declare which property is used to represent templates:

```
<templateRepresentativeProperties>
 <templateRepresentativeProperty tplName="term" pKey="1"/>
 <templateRepresentativeProperty tplName="proto" pKey="2"/>
 ...
</templateRepresentativeProperties>
```

Finally we come to the most important section, the `page` section. It describes the overall layout of a page within a WLE. As defined in section 4.2, a page is hierarchically divided into *blocks*. The need arises to configure

1. the hierarchy of the blocks,
2. how the start of a block is recognized and
3. which extraction templates are used in each block.

The first is achieved by nesting `<block>` nodes into each other:

```
<page>
 <block name="language">
  <block name="pos>
   ...
  </block>
 </block>
</page>
```

The second is realised by reusing templates. As introduced above, templates are matched against the page; additionally to extracting triples, they are used here, to react *when* they match. A block can have several *indicator templates* and while the extractor processes the page, it tries to match these. If they match, they trigger the start of a block. In the next step we will see how such a template is actually configured. The syntax for extraction templates and indicator templates is exactly the same[56]. The indicator templates are stored in each block:

```
<block name="language">
  <indicators>
    <indicator>
      <template ... (see below)
    </indicator>
  </indicators>
  ...
</page>
```

The third is done by the `templates` section within each block:

```
<block name="language">
  <indicators />
  <templates>
    <template name="example">
      <wikiTemplate>===Etymology===
$etymology
</wikiTemplate>
      <resultTemplates>
        <resultTemplate>
          <triples>
            <triple s="$block" p="http://wiktionary.dbpedia.org/terms/hasEtymology" o="
                $etymology" oType="literal"/>
          </triples>
        </resultTemplate>
      </resultTemplates>
      </template>
    </template>
  </templates>
</page>
```

The basics were explained in section 6.1, now we put them together: the `<template>` node is divided into two parts — the extraction template in `<wikiTemplate>` and the result templates.
The *extraction template* has been explained already. It is put in the `<wikiTemplate>` node. The only thing to keep in mind here is whitespace—whitespaces count. Also every indention or line break will be interpreted as wiki text and expected to match. Make sure you don't accidentally change the template, because it most likely won't match any more. Also set your text editor to show control characters like spaces, tabs or newlines (cf. the ¶ symbol).
Additionally to the *result template* basics introduced above, for a `<template>` there can be multiple RT and each consists of a set of triple templates. The rational is to respect missing bindings: If a variable is inside an optional repetition, it may not be present in the variable bindings. If those bindings are then converted to triples by a *result template*, missing variables will result in the current triple to fail. To avoid inconsistent triples (because some are missing), no triple shall be emitted. Thus RT are atomic—either all triples inside are emitted or none. This allows to model triples separately for varying

[56]The interpreting code is reused.

page content *within one template*. Another way to respect diverse layouts, is to declare a triple *optional*:

```
<resultTemplate>
  <triple s="http://some.ns/$entityId" p="http://some.ns/hasSense" o="http://some.ns/
      $entityId-$sense" />
  <triple s="http://some.ns/$entityId-$sense" p="http://some.ns/hasSource" o="$source"
      optional="true" />
</resultTemplate>
```

This disregards errors while rendering the RT to triples. A missing variable, will be ignored and the template finishes successfully.
A very important feature is the global `$block` variable together with the `oNewBlock` configuration option within *indicator templates*: Global variables have already been introduced by the `$entityId` variable and the possibility to save variables between templates. The `$entityId` variable is static, it keeps its value over time. Saved variables are local variables that become global. Now we introduce the `$block` variable, that holds the URI of the current block. For example, if the extractor currently processes the language block of a word, the value could be `http://some.ns/Haus-German`. This value can then be used to construct URIs that reuse this as a prefix:

```
<triple s="$block" p="http://some.ns/hasPosUsage" o="$block-$pos" oNewBlock="true" />
```

The object URI could be `http://some.ns/Haus-German-Noun` for example. But furthermore, if this RT is used within an *indicator template*, the need arises to indicate the start of a new block. When we introduced *indicator templates* above, we lied: Not the successful match of template triggers the state change, but only the successful rendering of its result template, including a triple with the `oNewBlock` option. The rational is, that to trigger that transition, the new block URI has to be known and there is no simple way to determine it from a set of unremarkable triples that are produced by the indicator template. Of course the `$block` variable can be used independently from the `oNewBlock` option in ordinary RT.

6.8. Utility Tools

To provide a complete deployment environment for the Wiktionary RDF dataset, it is also necessary to cater for tools to *load* the data into a database. We chose *Virtuoso* for this purpose and we created a set of tools that relate to data loading and cleaning. The set of available tools is presented in this table:

script	parameters	result	note
prepare	<lc>	cleaned nt-file	Applies `rapper` to the nt-file of the language code to clean common encoding problems, generates statistics and `bzip`'s the file.
virtuoso-load	<init> <lc>+	loaded language dumps in Virtuoso	Init (either *true* or *false*) specifies whether the database should be purged first. Then for each language code, the corresponding nt-file is loaded into Virtuoso, while setting up the graph layout accordingly.
publish-download	<lc>+	nt-files uploaded	Expects passwordless `ssh` access to the DBpedia download server, `gzip`s them, uses `scp` to upload the files and names them with the current date.

make_jarzip	-	executable jar file of the extractor	Creates a zip file that contains an executable jar of the DBpedia *Wiktionary* source code, containing all dependencies (to the framework and to configuration resources). Enables the easy distribution of the software for people without Mercurial knowledge.
statistics	<nt-file>	printed statistics	Generates statistics about an nt-file (e.g. triple count, property usage, sense counts)
translation-extract	-	translations CSV file	Retrieves all translation pairs from a SPARQL endpoint and serializes them in a fixed CSV format. The translation source word is disambiguated by language, part of speech and sense.
translation-loader.sql	-	relation of translations	execute from SQL console to load translations CSV file into a fixed relational table

The first four tools are used for deployment, the statistics tool is informative and the last two are descended from a specific use case that can also serve as a best practice for similar tasks. The task in this case was to export translations to a relational schema. We chose to do it via CSV as an intermediate step, as this format is both easy to serialize and easy to load from SQL. If the need arises to do something similar (for example with synonyms), this script is easy to adapt.

6.9. Graph Layout

Each WLE corresponds to a graph of its extracted triples. E.g. the English Wiktionary has been converted into the graph `http://en.wiktionary.dbpedia.org/`. A very interesting trait of this thesis is the merging of multiple Wiktionaries into on global resource: Each WLE describes various languages (in one reader language). For example, both the German and the English Wiktionary contain informations about Spanish words. Thus, without parsing the Spanish Wiktionary, you get an comprehensive Spanish language resource. The interesting part is, that *URIs fall together* — when the German and the English Wiktionary describe the same word, the resulting triples describe the same RDF resource, because the URIs are constructed canonically. When you merge both graphs, you get a multilingual linguistic knowledge base — at no additional cost. To avoid the decision if all graphs are only available merged or independent, for our deployment we chose to use the *Graph Group* feature of Virtuoso. This enables the configuration of special graphs that are substituted by the group of their member graphs when used in a query. This feature can be used to virtually import multiple graphs into one global graph:
Both the global and local graphs are visible simultaneously. The Linked Data view is based on the global graph. In SPARQL queries, it is up to the user, which graph to choose.

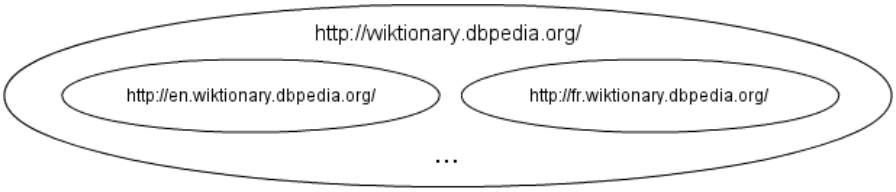

Figure 15: A graph for each WLE, virtually imported into one global graph

7. Evaluation

The extraction has been conducted as a proof-of-concept on four major WLE, the English, French, Russian and German *Wiktionary*. The datasets combined contain more than 80 million facts[57] about five million lexical words[58]. The data is available as N-Triples dumps[59], Linked Data[60], via the *Virtuoso Faceted Browser*[61] or a SPARQL endpoint[62]. The extraction has been conducted on August 15th 2012 and the used XML dumps were the latest as of that day[63].

7.1. Example Data

Just for reference, we present some sample data. It is extracted from the English Wiktionary entry for *goggles*[64]:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix wt: <http://wiktionary.dbpedia.org/terms/>.
@prefix wr: <http://wiktionary.dbpedia.org/resource/>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix doap: <http://usefulinc.com/ns/doap#>.
@prefix lemon: <http://www.monnet-project.eu/lemon#>

wr:goggles doap:creator <http://en.wiktionary.org/w/index.php?title=goggles&action=
    history> .
wr:goggles lemon:sense wr:goggles-English-Noun-1en .
wr:goggles rdfs:label "goggles"@en .
wr:goggles rdfs:seeAlso <http://ko.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://fi.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://sv.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://io.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://te.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://zh.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://de.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://my.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://pl.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://en.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://ta.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://kn.wiktionary.org/wiki/goggles> .
```

[57] SPARQL: `SELECT COUNT(*) WHERE ?s ?p ?o`
[58] SPARQL: `SELECT COUNT(?s) WHERE ?s a lemon:LexicalEntry`
[59] `http://downloads.dbpedia.org/wiktionary`
[60] for example `http://wiktionary.dbpedia.org/resource/dog`
[61] `http://wiktionary.dbpedia.org/fct`
[62] `http://wiktionary.dbpedia.org/sparql`
[63] In detail: en - 08/12, fr - 08/12, ru - 08/06, de - 08/11
[64] `http://en.wiktionary.org/wiki/googles`

```
wr:goggles rdfs:seeAlso <http://vi.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://fr.wiktionary.org/wiki/goggles> .
wr:goggles rdfs:seeAlso <http://et.wiktionary.org/wiki/goggles> .
wr:goggles rdf:type lemon:LexicalEntry .
wr:goggles rdf:type wt:LexicalEntity .
wr:goggles wt:hasLangUsage wr:goggles-English .
wr:goggles-English dc:language wt:English .
wr:goggles-English wt:hasPoSUsage wr:goggles-English-Noun .
wr:goggles-English wt:hasEtymology "Probably from goggle from the appearance it gives
    the wearer."@en .
wr:goggles-English wt:hasPronunciation "/\u02C8\u0261\u0251.\u0261\u0259lz/"@en .
wr:goggles-English-Noun wt:hasPoS wt:Noun .
wr:goggles-English-Noun wt:hasSense wr:goggles-English-Noun-1en .
wr:goggles-English-Noun-1en dc:language wt:English .
wr:goggles-English-Noun-1en wt:hasPoS wt:Noun .
wr:goggles-English-Noun-1en rdfs:label "goggles"@en .
wr:goggles-English-Noun-1en wt:hasMeaning "Protective eyewear set in a flexible frame
    to fit snugly against the face."@en .
wr:goggles-English-Noun-1en rdf:type lemon:LexicalSense .
wr:goggles-English-Noun-1en rdf:type wt:Sense .
wr:goggles-English-Noun-1en wt:hasEtymology "Probably from goggle from the appearance
    it gives the wearer."@en .
wr:goggles-English-Noun-1en wt:hasTranslation wr:Schutzbrille-German .
wr:goggles-English-Noun-1en wt:hasTranslation wr:occhiali_protettivi-Italian .
...
```

7.2. Possible Application

Now that we have this fine granular data, we can imagine many usage scenarios:

- reference, annotation (annotate corpora with unique identifiers)
- grammatical analysis (part of speech tagging, stemming, disambiguation)
- machine translation: The provided translations are defined on the senses, which enables context aware translation.

Most of these use cases require more data to be extracted then currently supported. But the data is already present within *Wiktionary*. Only the configuration has to be extended.
For example machine translation can be augmented with the provided data. To retrieve the translation pairs—with the given context—a simple SPARQL query is sufficient:

```
PREFIX wt:<http://wiktionary.dbpedia.org/terms/>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX dc:<http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?sword ?slang ?spos ?ssense ?tword ?tlang
FROM <http://wiktionary.dbpedia.org/>
WHERE {
   ?swordRes wt:hasTranslation ?twordRes .
   OPTIONAL {
      ?swordRes rdfs:label ?sword .
      ?swordRes dc:language ?slang .
      ?swordRes wt:hasPoS ?spos .
   }
   OPTIONAL { ?swordRes wt:hasMeaning ?ssense . }
   OPTIONAL {
      ?twordBaseRes wt:hasLangUsage ?twordRes .
      ?twordBaseRes rdfs:label ?tword .
   }
   OPTIONAL { ?twordRes dc:language ?tlang . }
}
```

This query retrieves all translation pairs with a disambiguation towards language, part of speech and sense definition of the source word. The data may look like:

"Hahn"	wt:German	wt:Noun	"das männliche Tier verschiedener Vogelarten"@de	"cock"	wt:English
"Hahn"	wt:German	wt:Noun	"eine mechanische Vorrichtung [...]"@de	"tap"	wt:English
...					

The example data shows, the disambiguated translation oh the German word *Hahn*, which has two meanings, which each a different translation. Analogously e.g. synonym pairs can be queried. This data is highly valuable, as it does not rely on noisy disambiguation algorithms, but is explicitly entered by human editors.

7.3. Quantity Measurement

We used some simple counting queries to measure the *dimensions* of the RDF data. This includes the number of its entries in the dictionary or, when seen as a graph, the number of edges and vertices or the number of distinct used predicates.

language	*#words*	*#triples*	*#resources*	*#predicates*	*#senses*
en	2,142,237	28,593,364	11,804,039	28	424,386
fr	4,657,817	35,032,121	20,462,349	22	592,351
ru	1,080,156	12,813,437	5,994,560	17	149,859
de	701,739	5,618,508	2,966,867	16	122,362

Table 4: Statistical quantity comparison of three *Wiktionary* extraction result datasets.

The statistics show, that the extraction produces a vast amount of data with broad coverage, thus resulting in the largest lexical linked data resource. (Note that even non-existing but referenced pages, result in the creation of a *word*. E.g. translations often reference pages that do not exist yet—red pages).

7.4. Quality Measurement

The measurement of data quality is a difficult topic, as there is no gold standard — no optimum to compare with. One could compare it with competing extractors, *but what, if you succeed them? What, if your scope is too different?* It is necessary to use absolute measures, either automatically calculated (which can be misleading) or by human rating (which requires substantial effort).
All presented index numbers are chosen at will, to give *some idea* about the coverage of the RDF data, but none of them is able to indicate the quality of the extraction — the completeness of the configuration — on a scale from zero to one. The numbers depend on the quality of the source data (which can not simply be assessed) and are not necessarily normed to one. It can be argued, that the last measure, triples per line, may the one most robust against source dependency. This tendency of the measure to vary with the quality of the source is desired to be low. It can be argued that this value should be close to one or even higher for a perfect extraction configuration, because each line

language	*t/w*	*#wws*	*s/wws*	*t/l*
en	13.35	591,073	1.39	2.70
fr	7.52	750,206	1.26	1.73
ru	11.86	211,195	1.40	2.25
de	8.01	176,122	1.43	1.06

Table 5: Statistical quality comparison of three *Wiktionary* extraction result datasets. *t/w*: *Triples per word.* The simplest measure of information density. *#wws*:*Words with senses.* The number of words, that have at least one sense extracted. An indicator for the ratio of pages for which valuable information could be extracted (but consider stub pages, that are actually empty). *s/wws*: *Senses per word with sense.* Gives an idea of the average senses per word while ignoring unmaintained pages. *t/l*:*Triples per line.* The number of triples divided by the number of line breaks in the page source (plus one). Averaged across all pages.

should contain *some* information (which results in a triple). Empty lines are disregarded and lines that produce multiple triples are rare. Therefore any value considerably lower than one, indicates that there are many uninterpreted lines. But again, these measures might be misleading as they are influenced by many unknown factors. One should not use them to compare two language editions of Wiktionary or a new configuration against one for a different language which is considered *good*. A safe way to use them is when comparing two versions of a configuration file with each other.
Both measurement types can be conducted with the `statistics` tool, which is part of the source code. It operates on the N-Triples dump of one language.
A reliable data quality assessment is only possible by human rating of randomly sampled entries. The procedure would include the random selection of a sufficiently large subset of extracted data (e.g. 200 words within one WLE), then these are assigned to a group of semi-experts to validate them against a check list of common errors (e.g. missing data, incorrect data, wrong data type, etc.).

7.5. Maintenance Experience

One of the claims of this thesis is the easy maintenance of the configuration and it is crucial that non-professionals can edit them. To evaluate this trait, we let a colleague with no special foreknowledge build the configuration for the Russian WLE. He took a few days to get familiar with the topic (and doing his normal work as well), and then was able to create a quite good configuration himself. To fully evaluate this claim we need to wait until the project is picked up in the wild and interview adopters. Unfortunately this is not possible within the temporal constraints of this thesis.

7.6. Limitations

There are limitations to the extractor: Some from the tailoring towards Wiktionary, some from the simple design of the `VarBinder`. Firstly, the extractor is meant to be as generic as possible, and should not contain direct assumptions about the page. But the schema of the page is modelled as the mentioned *nested blocks*. They directly reflect the layout of *Wiktionary*. Nesting describes the separation of one page into hierarchically organized sections—like the sections of this thesis. This concept is very basic and yet powerful, most texts fit into it. If the use case does not need nesting, it's fine, but if the layout is fundamentally different—maybe formats that are not directly text, think of binary formats or similar—the extractor may be unsuitable. Secondly the complexity of the configuration is limited to remain simple. A first example for limitations: A variable is ended by the node that follows it. Thus, this nodes can never be part of that variable. In RegEx there are more powerful options to end capturing groups. Therefore one may encounter situations where this limitation hinders the perfect extraction and requires dirty workarounds (in comparison to RegEx). A second example: If indicator templates or normal templates as well are too generic (They are not specific enough to only match when they are meant to match.) one may end with incorrect data. This is particularly annoying when the actual Wiki layout does not allow to be more specific. For example in the Vietnamese Wiktionary, language sections and part of speech sections are indicated (started) with almost the same pattern: `{{-viet-}}` vs. `{{-noun-}}` The corresponding indicator template would be: `{{-$lang-}}` vs. `{{-$pos-}}`. The templates would match both occurrences alike, because they are too generic. The template that is tried to be matched first (the outer one, regarding the nesting) would *steal* all the matches. A dirty workaround would be to explicitly write all possible variants without using variables, but it would be very verbose. One could also try to enhance the framework by extending the mapping mechanism to support some kind of classification (e.g. all mappings for part of speech tokens are classified *pos*) and modify e.g. the `assertMapped` RT formatting function, to support an additional argument that restricts the classification. For reference there is to say, that there are still RDF serialization bugs in DBpedia, that can cause triples to be missing, when they are cleansed by rapper.

8. Conclusion

Finally we want to present our vision of the future development of the project, of the underlying data source *Wiktionary* and of the LLOD cloud in general.

8.1. Vision

Making unstructured sources machine-readable creates feedback loops. The argument, that extracting structured data from an open data source and making it freely available in turn, encourages users of the extracted data to contribute to the unstructured source, seems reasonable. It is firstly not easily possible to continuously apply changes to the automatically extracted data when the source and the configuration changes arbitrarily. One could imagine a kind of a patch queue, but also how hard it is to maintain it. On the other hand, for humans it is much easier to curate a wiki. The co-evolution of ontologies is still an open research problem. Back to the claim: Users of the RDF version of Wiktionary could be NLP researchers or companies. Some of them have considerable human resources, that maintain internal databases of linguistic knowledge and some are willing to publish their data and integrate it with existing sources; but the clear incentive is to *get the data back again* — enriched. With a mature extraction infrastructure that can be deployed autonomously by adopters or centrally by us at the same time, it becomes reasonable for such third parties to contribute. This increase in participation, besides improving the source, also illustrates the advantages of machine readable data to common Wiktionarians, which are in turn more motivated to cooperate. Such considerations are fundamental; without community interaction and mutual benefits, extraction from community edited sources becomes dull scraping. There is an implicit requirement for a social effort to successfully transfer the desired properties of the source to the extracted data. Such a positive effect from DBpedia supported the current *Wikidata*[65] project.

8.2. Suggested changes to Wiktionary

Although it's hard to persuade the community of far-reaching changes, we want to conclude how *Wiktionary* can increase it's data quality and enable better extraction.

- **Homogenize Entry Layout across all WLE's.**
- **Use anchors to markup senses:** This implies creating URIs for senses. These can then be used to be more specific when referencing a *word* from another article. This would greatly benefit the evaluation of automatic anchoring approaches like in [23].
- **Word forms:** The notion of word forms (e.g. declensions or conjugations) is not consistent across articles. They are hard to extract and often not given.

[65] `http://meta.wikimedia.org/wiki/Wikidata`

8.3. Discussion

Our main contributions are

1. an extremely flexible extraction from *Wiktionary*, with simple adaption to new Wiktionaries and changes via a declarative configuration,
2. the provisioning of a linguistic knowledge base with unprecedented detail and coverage and
3. a mature, reusable infrastructure including a public Linked Data service and SPARQL endpoint (by reusing well established projects like DBpedia and Virtuoso).

In addition, all resources related to our *Wiktionary* extraction, such as source code, extraction results, pointers to applications etc. are available from our project page[66] under an open license.
As a result, we hope it will evolve into a *central resource and interlinking hub* on the currently emerging web of linguistic data, the *Linguistic Linked Open Data* cloud.

8.4. Next Steps

Wiktionary Live: Users constantly revise articles. Hence, data can quickly become outdated, and articles need to be re-extracted. DBpedia-Live enables such a continuous synchronization between DBpedia and Wikipedia. The WikiMedia foundation kindly provided us access to their update stream, the Wikipedia OAI-PMH[67] live feed. The approach is equally applicable to *Wiktionary*. The *Wiktionary* Live extraction will enable users to query *Wiktionary* like a database in real-time and receive up-to-date data in a machine-readable format for the first time ever. This will strengthen *Wiktionary* as a central resource and allow it to extend its coverage and quality even more.
Wiki based UI for the WLE configurations: To enable the crowd-sourcing of the extractor configuration, an intuitive web interface is desirable. Analogue to the mappings wiki[68] of DBpedia, a wiki could help to hide the technical details of the configuration even more. Therefore a JavaScript based WYSIWYG XML editor seems useful. There are various implementations, which can be easily adapted.
Linking: An alignment with existing linguistic resources like WordNet and general ontologies like YAGO or DBpedia is essential. That way *Wiktionary* will allow interoperability across a multilingual semantic web.
Quality Assessment: As described in the evaluation section, a data quality evaluation analogously to DBpedia and Zaveri's DBpedia evaluation[69] needs to be conducted.

[66] `http://wiktionary.dbpedia.org`
[67] Open Archives Initiative Protocol for Metadata Harvesting,
cf. `http://www.mediawiki.org/wiki/Extension:OAIRepository`
[68] `http://mappings.dbpedia.org/`
[69] `http://aksw.org/Projects/DBpediaDQ`

8.5. Open Research Questions

We found two important open research questions associated with lexicas and the Semantic Web:

8.5.1. Publishing Lexica as Linked Data

The need to publish lexical resources as linked data has been recognized recently [27]. Although principles for publishing RDF as Linked Data are already well established [2, 12], the choice of identifiers and first-class objects is crucial for any linking approach. A number of questions need to be clarified, such as which entities in the lexicon can be linked to others. Obvious candidates are entries, senses, synsets, lexical forms, languages, ontology instances and classes, but different levels of granularity have to be considered and a standard linking relation such as `owl:sameAs` will not be sufficient. Linking across data sources is at the heart of linked data. An open question is how lexical resources with differing schemata can be linked and how linguistic entities are to be linked with ontological ones. There is most certainly an impedance mismatch to bridge.
The success of DBpedia as "a crystallization point for the Web of Data" [18] is predicated on the stable identifiers provided by Wikipedia and those are an obvious prerequisite for *any* data authority. Our approach has the potential to drive this process by providing best practices, live showcases and data in the same way DBpedia has provided it for the LOD cloud. Especially, our work has to be seen in the context of the recently published Linguistic Linked Data Cloud [6] and the community effort around the Open Linguistics Working Group (OWLG)[70] and NIF [13]. Our Wiktionary conversion project provides valuable data dumps and linked data services to further fuel development in this area.

8.5.2. Algorithms and methods to bootstrap and maintain a Lexical Linked Data Web

State-of-the-art approaches for interlinking instances in RDF knowledge bases are mainly build upon similarity metrics [26, 31] to find duplicates in the data, linkable via `owl:sameAs`. Such approaches are not directly applicable to lexical data. Existing linking properties either carry strong formal implications (e.g. `owl:sameAs`) or do not carry sufficient domain-specific information for modelling semantic relations between lexical knowledge bases.

[70] `http://linguistics.okfn.org`

A. Literature

References

[1] Sören Auer, Chris Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A nucleus for a web of open data. In *Proceedings of the 6th International Semantic Web Conference (ISWC)*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2008.

[2] Sören Auer and Jens Lehmann. Making the web a data washing machine - creating knowledge out of interlinked data. *Semantic Web Journal*, 2010.

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *The Scientific American*, 2001.

[4] Jeen Broekstra and Arjohn Kampman. SeRQL. An RDF Query and Transformation Language. ISWC, 2004.

[5] P. Chesley, B. Vincent, L. Xu, and R. K. Srihari. Using verbs and adjectives to automatically classify blog sentiment. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, 2006.

[6] C. Chiarcos, S. Hellmann, S. Nordhoff, S. Moran, R. Littauer, J. Eckle-Kohler, I. Gurevych, S. Hartmann, M. Matuschek, and C. M. Meyer. The open linguistics working group. In *LREC*, 2012.

[7] Michael Genesereth and Nils Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, 1987.

[8] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, December 1995.

[9] R.V. Guha, Ora Lassila, Eric Miller, and Dan Brickley. Enabling Inference. W3C Query Language meeting, 1998.

[10] I. Gurevych, J. Eckle-Kohler, S. Hartmann, M. Matuschek, C. M. Meyer, and C. Wirth. Uby - a large-scale unified lexical-semantic resource based on lmf. In *EACL 2012*, 2012.

[11] P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A comparison of RDF query languages. ISWC, 2004.

[12] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan and Claypool, 2011.

[13] Sebastian Hellmann, Jens Lehmann, and Sören Auer. Linked-data aware uri schemes for referencing text fragments. In *EKAW 2012*, Lecture Notes in Computer Science (LNCS) 7603. Springer, 2012.

[14] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, and York Sure. *Semantic Web*. Springer, 2008.

[15] ISO 24613:2008. *Language resource management — Lexical markup framework*. ISO, Geneva, Switzerland.

[16] D. Kontokostas, C. Bratsas, S. Auer, S. Hellmann, I. Antoniou, and G. Metakides. Internationalization of Linked Data: The case of the Greek DBpedia edition. *Journal of Web Semantics*, 2012.

[17] A. A. Krizhanovsky. Transformation of wiktionary entry structure into tables and relations in a relational database schema. *CoRR*, 2010. http://arxiv.org/abs/1011.1368.

[18] Jens Lehmann, Chris Bizer, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009.

[19] J. McCrae, P. Cimiano, and E. Montiel-Ponsoda. Integrating WordNet and Wiktionary with lemon. In C. Chiarcos, S. Nordhoff, and S. Hellmann, editors, *Linked Data in Linguistics*. Springer, 2012.

[20] J. McCrae, D. Spohr, and P. Cimiano. Linking Lexical Resources and Ontologies on the Semantic Web with lemon. In *ESWC*, 2011.

[21] C. M. Meyer and I. Gurevych. How web communities analyze human language: Word senses in wiktionary. In *Second Web Science Conference*, 2010.

[22] C. M. Meyer and I. Gurevych. Worth its weight in gold or yet another resource – a comparative study of wiktionary, openthesaurus and germanet. In *CICLing*. 2010.

[23] C. M. Meyer and I. Gurevych. OntoWiktionary – Constructing an Ontology from the Collaborative Online Dictionary Wiktionary. In M.T. Pazienza and A. Stellato, editors, *Semi-Automatic Ontology Development: Processes and Resources*. IGI Global, 2011.

[24] K. Moerth, T. Declerck, P. Lendvai, and T. Váradi. Accessing multilingual data on the web for the semantic annotation of cultural heritage texts. In *2nd Workshop on the Multilingual Semantic Web, ISWC*, 2011.

[25] Mohamed Morsey, Jens Lehmann, Sören Auer, Claus Stadler, , and Sebastian Hellmann. Dbpedia and the live extraction of structured data from wikipedia. *Program: electronic library and information systems*, 46:27, 2012.

[26] Axel-Cyrille Ngonga Ngomo and Sören Auer. Limes - a time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of IJCAI*, 2011.

[27] A. G. Nuzzolese, A. Gangemi, and V. Presutti. Gathering lexical linked data and knowledge patterns from framenet. In *K-CAP*, 2011.

[28] Aris M. Ouksel and Amit P. Sheth. Semantic interoperability in global information systems: A brief introduction to the research area and the special section. *SIGMOD Record*, 28(1):5–12, 1999.

[29] Franck Sajous, Emmanuel Navarro, Bruno Gaume, Laurent Prévot, and Yannick Chudy. Semi-automatic Endogenous Enrichment of Collaboratively Constructed Lexical Resources: Piggybacking onto Wiktionary. In *Advances in Natural Language Processing*, volume 6233 of *Lecture Notes in Computer Science*, pages 332–344. 2010.

[30] K. Mörth G. Budin T. Declerck, P. Lendvai and T. Váradi. Towards linked language data for digital humanities. In C. Chiarcos, S. Nordhoff, and S. Hellmann, editors, *Linked Data in Linguistics*. Springer, 2012.

[31] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In *ISWC*, 2009.

[32] T. Weale, C. Brew, and E. Fosler-Lussier. Using the wiktionary graph structure for synonym detection. In *Proc. of the Workshop on The People's Web Meets NLP, ACL-IJCNLP*, 2009.

[33] T. Zesch, C. Müller, and Iryna Gurevych. Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary. In *LREC*, 2008.

[34] Torsten Zesch, C. Müller, and I. Gurevych. Using wiktionary for computing semantic relatedness. In *AAAI*, 2008.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet.
Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, den 31.08.2012