

MASTERARBEIT

EXPANDING THE NIF ECOSYSTEM
CORPUS CONVERSION, PARSING AND PROCESSING USING
THE NLP INTERCHANGE FORMAT 2.0

April 22, 2015

Martin Brümmer
`bruemmer@informatik.uni-leipzig.de`

Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik
Abteilung für betriebliche Informationssysteme

Betreuer:
Prof. Dr. Ing. habil. Klaus-Peter Fährnich
Dr. Ing. Sebastian Hellmann

Contents

1	Introduction	5
1.1	Motivation	5
1.2	Structure	5
1.3	Use Cases	6
2	Foundations	6
2.1	The Semantic Web	6
2.1.1	RDF	6
2.1.2	Linked Data	8
2.2	Natural Language Processing (NLP)	8
2.2.1	Text annotation	8
2.2.2	Sentence Boundary Detection	9
3	NIF 2.0	10
3.1	URI Schemes	11
3.2	NIF Core Ontology	12
3.3	OLiA	13
3.4	NERD	14
4	Conversion of corpora into NIF	15
4.1	Brown corpus	16
4.2	Tiger ConLL dependency corpus	19
4.3	Wikipedia abstract corpus	23
4.4	Wikilinks corpus	27
4.5	Other NIF corpora	29
5	Validation of NIF corpora	30
5.1	RDFUnit	30
5.2	Test Case Implementation for NIF	31
5.3	Validation of collected corpora	33
6	Implementation and scalability of NIF tools	36
6.1	OpenNLP NIF Wrapper	36
6.2	OpenNLP NIF parser	39
6.2.1	Creating NIF annotations via OpenNLP	42
6.2.2	Round-trip combining both resources	45
7	Interoperability and In-Use	45
7.1	The Unicode Text Segmentation use case	45
7.2	Obtaining abbreviation lists from LOD	46
7.3	Corpus-based evaluation of sentence boundary detection algorithms	50
7.4	Gerbil	55
8	Discussion and Future Work	55

Abstract This work presents a thorough examination and expansion of the NIF ecosystem. Both core use cases of NIF, as a format for NLP tool integration, as well as a corpus pivot format, are addressed. NLP tool integration is extended with a new wrapper for the OpenNLP framework including a NIF parser, as well as a CoNLL converter to convert the widely used CoNLL format to NIF. Tools are examined for scalability and data quality. On the corpus side, a large range of diverse existing corpora are converted into NIF. The complete contribution of corpora presented in this work adds up to 841,106,737 triples. Data quality is a special focus, validating the corpora to guarantee correctness using SPARQL test cases. Finally, an industry use case is presented in Unicode Text Segmentation, where abbreviation lists are extracted from Linked Data and their impact on sentence boundary detection is measured using the converted corpora and NLP tools.

Acknowledgements I would like to thank my supervisor Sebastian Hellmann for unconditionally supporting me in the course of this work. I would further like to thank my girlfriend Julia, for supporting me with unconditional love and coffee.

1 Introduction

1.1 Motivation

The Semantic Web as a movement to evolve the unstructured web of documents, largely limited to the consumption by humans, into a web of structured data, open to humans as well as machines, has made huge gains since its initial proposal¹ in 2001. Linked Open Data (LOD)² was established as a concept 5 years later and has since been realized in over 1000 datasets³ producing a cloud of interconnected open datasets, containing structured knowledge to be explored by machines and humans alike. Many of these datasets are sourced from already structured data, databases and collections of semi-structured documents, like XML files of various formats. To accomplish the vision of the Semantic Web and truly evolve the web of documents, the huge amount of knowledge contained in unstructured natural language text has to be made accessible. Knowledge in this sense does not mean mere information, it implies information that is structured according to a formal conceptualization of its domain and accessible to computer systems. Aware of these needs, one part of the Semantic Web research community⁴ started to apply techniques of Natural Language Processing (NLP) to facilitate the necessary knowledge extraction tasks. Besides the benefit Semantic Web researchers gain by using NLP, advantages are not a one way street.

The NLP Interchange format [10] presents one avenue to facilitate interoperability between NLP tools, corpora and the Linked Open Data cloud by formally specifying a way to address and annotate arbitrary strings, publish these annotations and infer new knowledge using an ontology. It presents a way to build rich NLP pipelines and can serve as glue between previously incompatible NLP output formats. Using the Linked Open Data cloud, NIF promotes the growth of knowledge in the web of data as well as the quality of NLP results by supporting access to existing linguistic data in the cloud. Although these benefits are already established by the adoption of NIF and are being realized in EU-funded projects like LIDER⁵, the ecosystem is in need of expansion, both in the quantity of tools and data available, as well as in the quality of existing components.

1.2 Structure

This work addresses these needs of growing and evaluating NIF components. First, a comprehensive introduction to NIF will be presented in Section 3, following the ISWC paper “Integrating NLP using Linked Data” [10], which I co-authored. Then the importance of NIF as a pivot format between corpora and NLP tools will be reinforced by presenting the conversion of important NLP corpora into NIF, contributing significantly to the LOD cloud. The conversion of four corpora will be shown, two of them based on existing structured data, two of them converted from semi-structured sources. They will be thoroughly validated in Section 5, following, extending and evaluating the procedure shown in [13], another paper I co-authored. The correctness specification of the NIF format will be addressed, as well as the feasibility and scalability of the validation procedure itself. The NLP tool part of the NIF ecosystem will be expanded in Section 6. A wrapper to integrate the well-known Apache OpenNLP⁶ framework into NIF will

¹<http://www.cs.umd.edu/~golbeck/LBSC690/SemanticWeb.html>

²<http://www.w3.org/DesignIssues/LinkedData.html>

³<http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/>

⁴Indicated by the success of workshops like *NLP & DBpedia* on the International Semantic Web Conference - http://iswc2014.semanticweb.org/workshops_and_tutorials#acceptedworkshop

⁵<http://lider-project.eu/>

⁶<https://opennlp.apache.org/>

be implemented, as well as a parser to use NIF annotations in OpenNLP. The software was be evaluated for scalability and tested for correctness demonstrating a roundtrip from text to NIF to text. Finally, an industry use case combining corpora, NLP tools and external linked open data will be demonstrated in 7. The basic NLP problem of sentence boundary detection and the positive impact of abbreviation lists extracted from linked open data will be examined, executed and evaluated, using the tools developed in this work, proving its positive contribution and pointing to further needs in the fields of NLP and the Semantic Web.

1.3 Use Cases

NIF can serve a number of use cases in the fields of NLP and the Semantic Web. As both, NLP tool output formats, as well as formats to represent NLP annotations are plentiful and heterogeneous, NIF can serve as a pivot format. The concept of the NLP pivot format can be attributed to [18], where the Linguistic Annotation Framework (LAF) is presented as a possible pivot format. Such a format allows for representation of arbitrary annotations, so that, if implemented in a number of tools, they share a common format for exchange. This procedure significantly reduces the development overhead for integration tasks spanning annotations from multiple tools and corpora. NLP tools would only need to support the pivot format, instead of supporting the wide array of possible formats that are commonly used. Similar to LAF, NIF presents a standard format to express annotations of arbitrary strings in a stand-off way, but does so openly⁷ and based on Semantic Web technologies.

Using NIF as a pivot format by converting corpora into NIF and writing wrappers so that NIF input can be consumed and put out by NLP tools opens up further use cases. Statistical models used by NLP tools for text annotation can be trained by using open NIF corpora as a gold standard. Evaluation of NLP tool performance is a similar and equally important use case also founded on open corpora. More generally, NIF wrappers facilitate use of NLP annotations produced by one tools in conjunction with other NLP tools, allowing them to interoperate and possibly increasing the quality by exploiting strengths of different tools for specific tasks.

NIF's linked data nature allows easily publishing the resulting annotation data in a machine-readable way. Because strings can be uniquely identified using the NIF URI schemes, third parties can add their own annotations, with NIF enabling a natural merge of the annotations. Altogether, NIF furthers extensible architectures consisting of diverse NLP tools based on a shared interoperable data model.

2 Foundations

2.1 The Semantic Web

2.1.1 RDF

RDF is a set of specifications developed by the World Wide Web Consortium (W3C) as a data model that can be used to formally describe resources. A resource can be anything that can be uniquely identified, ranging from digital documents to abstract concepts. Resources are identified by URIs, Uniform Resource Identifiers,⁸ distinct strings with uniform syntax. URIs can be classified into URLs (Uniform Resource Locator) and URNs (Uniform Resource Name). URLs additionally describe the primary method of access to the resource, most of them describing web documents, like <http://site.nlp2rdf.org/>, that can be visited to gain more information.

⁷LAF is an ISO standard and thus not open

⁸RFC3986 Uniform Resource Identifier (URI) <http://tools.ietf.org/html/rfc3986#section-1.1>

In the RDF data model, resources are described by statements in the form of “subject predicate object” which are called triples. These statements can be understood as metadata, “data about data” and in the context of RDF “data describing Web resources” [15]. The subject is the resource that is described by the statement, uniquely identified by its URI. The object is the content of the statement, the meta datum itself, like the name of the author of a web site or its publishing date. The predicate constitutes the semantic link between the subject and the object and describes the meaning of the link between subject and object. Using the example of the name of the author of a web site, the triple describing it could look like this:

```
1 <http://site.nlp2rdf.org> author "Sebastian Hellmann" .
```

The subject of the triple is `http://site.nlp2rdf.org`, its object is the string “Sebastian Hellmann” and the predicate is “author”. Assuming that there was no predicate, one could not ascribe a specific meaning to the datum “Sebastian Hellmann”, as one would not know the kind of relation between subject and object. Thus, the predicate is what assigns the meaning to the statement.

In the RDF data model, the difference between data and metadata is fluent. The author can be expressed as a simple *literal* that contains the author’s name, as seen in the example, but it can also be another resource, like a web page containing further information.

```
1 <http://site.nlp2rdf.org> author <http://aksw.org/SebastianHellmann> .
2 <http://aksw.org/SebastianHellmann> name "Sebastian Hellmann" .
```

This process of linking and creating resources explicitly describing their relations creates a labeled, directed graph. As explained, the semantic information is in the predicates. Although simple strings like “author” are already meaningful to humans, they still are ambiguous and could be misinterpreted. Is the object of the statement with the predicate “author” a mere string of characters or does it describe a person? How is the relation of being an author defined? Does it entail further information that could be valuable? Especially machines that could otherwise automatically interpret the data cannot automatically resolve this ambiguity, although being able to automatically answer these questions would be beneficial. Thus, in the RDF data model, the predicates also have URIs that can be looked up for further information and are then called properties.

```
1 <http://nlp2rdf.org> <http://purl.org/dc/terms/author> <http://aksw.org/
  SebastianHellmann.html> .
2 <http://aksw.org/SebastianHellmann.html> <http://purl.org/dc/terms/name> "
  Sebastian Hellmann" .
```

The additional benefit is, that sets of properties can be defined and documented by institutions or developers, like Dublin Core⁹ in the example above. They can then be reused by other users, increasing interoperability and reducing the work necessary for formal definition.

These sets of properties and associated classes of things needed to create and interpret RDF triples are called vocabularies or ontologies. A vocabulary or ontology is a set of entity classes and properties that models a conceptualization of a specific domain, allowing its formal representation with RDF as well as its automatic interpretation by machines. Classes divide a domain in a set of abstract objects while individual resources can instantiate a number of classes. Subclass relations allow for more complex hierarchies of inheritance. Properties specify the meaning of the link between different resources (their relation) and serve as predicates of RDF triples. A large number of these vocabularies already exist and can be reused.

RDF itself is only a data model, independent of the concrete serialization, that can be realized using different formats, like RDF/XML, N3, Turtle or JSON-LD. All serializations contain the same information but differ in readability, size and ease of parsing.

⁹<http://dublincore.org/>

2.1.2 Linked Data

Linked Data is defined by a set of *best practices* for publication and linking of structured data on the web[3]. It describes machine-readable data published on the world wide web with explicitly defined meaning that link further data. The result is a *Web of Data*, a machine-readable, semantic network of structured data in contrast to the unstructured HTML document web. In [2] Tim Berners-Lee defined the following rules of Linked Data:

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

Linked Data extends the demands of RDF as a data model by specifying HTTP as the access layer of choice and requiring the openness of the resource to be regarded of high quality.

2.2 Natural Language Processing (NLP)

Natural language processing is a field of computer science researching how natural human language can be processed, analyzed, understood and generated by computers. Major tasks in NLP involve basic analysis of text, like sentence boundary detection, tokenization and part of speech tagging but also more sophisticated techniques like named entity recognition, dependency parsing or morphological analysis. More complicated tasks often depend on basic analysis. Therefore NLP tools often work in pipelines, making the output of one tool the input of another.

Natural language corpora, collections of texts in different domains (like newspaper texts or literary prose), play an important role in NLP. The texts these corpora contain are usually annotated by humans marking certain features, like sentence and word borders or part of speech tags. Using these texts, NLP tools can be trained by letting them learn statistical features of these annotations that can later be used on unannotated text. Corpora can also be used to evaluate NLP tool quality, by checking if the output of the tool corresponds to the annotation in the corpus.

2.2.1 Text annotation

Text annotation in the field of NLP is the task of adding notes that describe certain features of a text - *annotations* - to the text itself. This can be done manually, for example by a linguist identifying and marking the part of speech of the words of a sentence, or automatically, by specific NLP tools. The representation and storage of these annotations is realized in a number of different ways. Major distinctions are:

- In-text annotation versus stand-off annotation, i.e. if the annotations are marked up in the text itself or in a different file.
- Identification of the annotated string.

For example, the annotated string can be identified by counting all characters of the text and denoting the starting character number and the ending character number of the string, a practice used in the NLP Interchange Format described in Section 3. A different way is to mark up identifiers in the text itself and refer to them, as used in GATE XML and shown in Listing 1.

```

1 <GateDocument>
2 <TextWithNodes>
3   <node id="0"/>This <node id="5"/>word<node id="9"/>is annotated.<node id="23"
   />
4 </TextWithNodes>
5 <AnnotationSet>
6   <Annotation Type="POS" StartNode="5" EndNode="9">
7     <Feature>
8       <Name>POS</Name>
9       <Value>NN</Value>
10    </Feature>
11  </Annotation>
12 </AnnotationSet>
13 </GateDocument>

```

Listing 1: Example text identification using XML elements in the text

Flat file based annotation formats like ConLL split up sentences in tokens, displaying one word and its annotations per line. A more in-depth description will follow in Section 4.2.

2.2.2 Sentence Boundary Detection

Sentence boundary detection is a basic NLP problem concerned with determining and annotating beginning and end of a sentence. The procedure is basic to NLP because many advanced NLP techniques like POS tagging or dependency parsing require single sentences as input. In general, there are three major approaches to sentence boundary detection:

Rule-based sentence boundary detection uses a number of heuristics that describe the surface form of a sentence. In the most simple of cases, an English sentence can be described as “*a string of characters, starting with an upper case letter and ending with sentence punctuation*”, for example expressed by a regular expression like `[A-Z].*?(.!!!?)`. However, there can be sentences that start with lower case letters and words like abbreviations or dates and numbers, can contain periods. In the Brown corpus that was converted in 4.1, 10.21% of periods occur at the end of abbreviations. However, the percentage of periods ending abbreviations can be much higher. In the Wall Street Journal part of the ACL/DCI corpus, 27.05% of periods end abbreviations¹⁰. To prevent large decreases in precision, abbreviation dictionaries can be used to alleviate this problem. Thus, lists of abbreviations are often used in addition and the number of rules employed is much more complex than in this very simple case. If abbreviation lists are used, the rule based detector itself is not changed but if the last word of the sentence is found in a precompiled abbreviation dictionary, the sentence is not considered to be finished and the next “sentence” is concatenated to it, thus making punctuation ending an abbreviation an exception to the rule. This approach will be explained in more detail in Section 7.1.

Supervised machine learning approaches employ training a software on a text annotated with sentence boundaries as a gold-standard before they can be used on general text. The sentence detection component included OpenNLP framework, for example, uses supervised Maximum Entropy learning as shown in [22], reporting a precision of 97% using the Brown corpus.

Lastly, unsupervised machine learning approaches can be trained on unannotated corpora. One of the most notable systems of this kind is *Punkt* [12], which identifies abbreviations by regarding them as collocations of the abbreviated word and the following period, using likelihood rations to detect them. They achieve a system that is not only domain independent but also largely language independent and features a small error rate, reporting 99.13% precision and 98.64% recall on the brown corpus. However, it has to be trained on large amounts of text to work properly.

¹⁰Percentages according to [12]

3 NIF 2.0

The NLP Interchange Format is an RDF/OWL based format created to facilitate interoperability between different NLP tools. Its current version 2.0 was first presented in [10], a paper I co-authored. A large number of NLP tools is currently available, enabling the application of diverse NLP algorithms for equally diverse tasks. The tools themselves accomplish similar tasks, however, they often are not compatible regarding their input and output formats. Most tools contain support for a number of standard formats, like ConLL or various flavours of XML¹¹. However, chaining them together freely, using the tokenization algorithm of one tool, comparing it to the tokenization of another tool, processing the outcome with another is an interesting use case without native support in the tools. They also have their particular strengths and weaknesses, working better or worse for particular use cases or languages. NIF serves as an interoperability layer, wrapping NLP tools by converting their output into a common format and merging the different annotations into a single file. For this merge to work, all that is necessary is assigning the same URI to the same string annotated by different frameworks. Because annotations in NIF are RDF triples, annotations from different tools will naturally merge using the assigned URI as identifier. Figure 3.1 shows an example of this merge process.

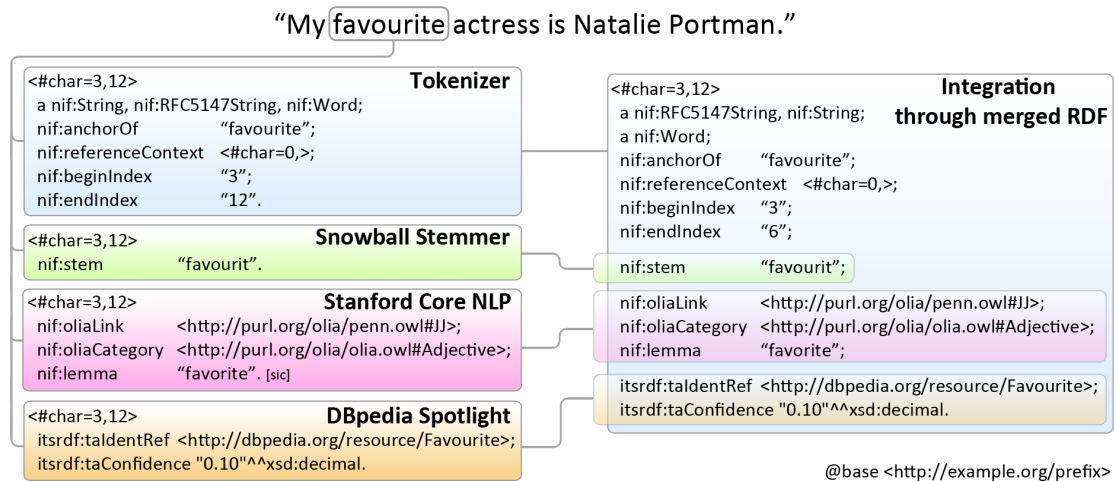


Figure 3.1: Different annotations by different tools on the same string naturally merge with NIF

Recently, NIF is also increasingly used as a corpus format ([10], [25]). This is done to further facilitate interoperability by integrating source data used by NLP tools for training and evaluation tasks into the NIF ecosystem, but also because NIF provides a way to annotate arbitrary strings and publish them via LOD. NIF achieves this by implementing a URI scheme for identifying and addressing elements in (hyper-)text and an ontology for describing common NLP terms. The following description largely follows [10].

¹¹for example GateXML, TEI XML or GrAF

3.1 URI Schemes

NIF was developed to allow NLP tools to exchange text annotations in RDF. This requires a way to make text strings referenceable by URIs to be able to use them as resources of RDF statements. An algorithm to create a URI for a specific string contained in a document is called a URI scheme. This scheme needs a number of parameters to create a URI, namely:

- the URI of the document itself, in the following addressed as *prefix*
- the character indices of beginning and end of the string to address
- a separator between the prefix and string position identifier

Character indices in NIF are counted offset based, as shown in figure 3.2, starting at zero before the first character and counting the gaps between the characters incrementally until after the last character of the referenced string.



Figure 3.2: Offset based string counting counts the gaps between characters

The canonical URI scheme of NIF is based on RFC 5147¹², which provides a standardization of fragment identifiers for the media type `text/plain`. Of course, most textual documents found on the web don't have this media type, but rather `text/html` or `application/xhtml+xml`. To be able to uniformly address strings in textual documents, NIF regards them as `text/plain` documents, thereby including markup characters. The typical example for the use of the standard NIF URI scheme is the first occurrence of the string "Semantic Web" in the document (26610 characters) `http://www.w3.org/DesignIssues/LinkedData.html`. It can be addressed as `http://www.w3.org/DesignIssues/LinkedData.html#char=1206,1218` according to RFC 5147, using the separator `#` and character offsets to identify the string. The whole document could in turn be addressed by `http://persistence.uni-leipzig.org/nlp2rdf/examples/doc/LinkedData.txt#char=0,26610`. This procedure also describes how to realize web annotation using the NIF URI scheme, by simply appending the character offsets of the string to be annotated to the web page URL. Because the document being annotated is not changed in the process of annotation, NIF functions as stand-off annotation format.

For purposes of web services or in applications involving database storage of the strings, the prefix can be a generated URN or a custom identifier, allowing shorter identifiers that are more uniform and therefore easier to handle. Because the focus of NLP web services lies in the processed text or annotations transferred, the generated RDF does not have to include resolvable resources that are mainly useful for republishing the annotations.

To publish the annotations themselves as Linked Data, additional attention has to be given to the URIs. Most of the time, the domains of the web pages annotated are not controlled by the annotating party themselves. Thus, using the URI scheme described will lead to resolvable URIs, but they won't contain the annotation but the original document. Using our example, a part of speech annotation for the string "Semantic Web" may look like this:

¹²URI Fragment Identifiers for the text/plain Media Type `http://tools.ietf.org/html/rfc5147`

```
1 <http://www.w3.org/DesignIssues/LinkedData.html#char=1206,1218> nif:oliaLink <
  http://purl.org/olia/penn.owl#NNP>.
```

The annotation states that the string has the part of speech “proper noun”. But to be able to access it as Linked Data, it needs a URI controlled by the annotating party. This can be achieved in different ways. The first would be a web service, that takes the original NIF URI as a parameter and returns the annotation resource, like in the case of the NIF Brown corpus introduced in 4.1. Resulting URIs are quite long, but the structure is easy to understand. The following example shows how a NIF web service URI for annotation of the example would look like.

```
1 http://example.org/service.php?t=url&f=text&i=http://www.w3.org/
  DesignIssues/LinkedData.html#char=1206,1218
```

The first part is the URI of the web service, shown in red. What follows is a query denoting the NIF input parameters, as explained in Section 6.1, here shown in blue. The `input` parameter (`-i`) is the URI of the web site that contains the string to annotate, marked in green. Finally, the fragment containing the strings position via start and end offsets (black) is added, completing the reference. Accessing this resource will then return the annotation of the string in question.

A different way would be using own prefix URIs and only referencing the original resources or relevant parts thereof via properties like `nif:sourceUrl` or `nif:wasConvertedFrom`.

3.2 NIF Core Ontology

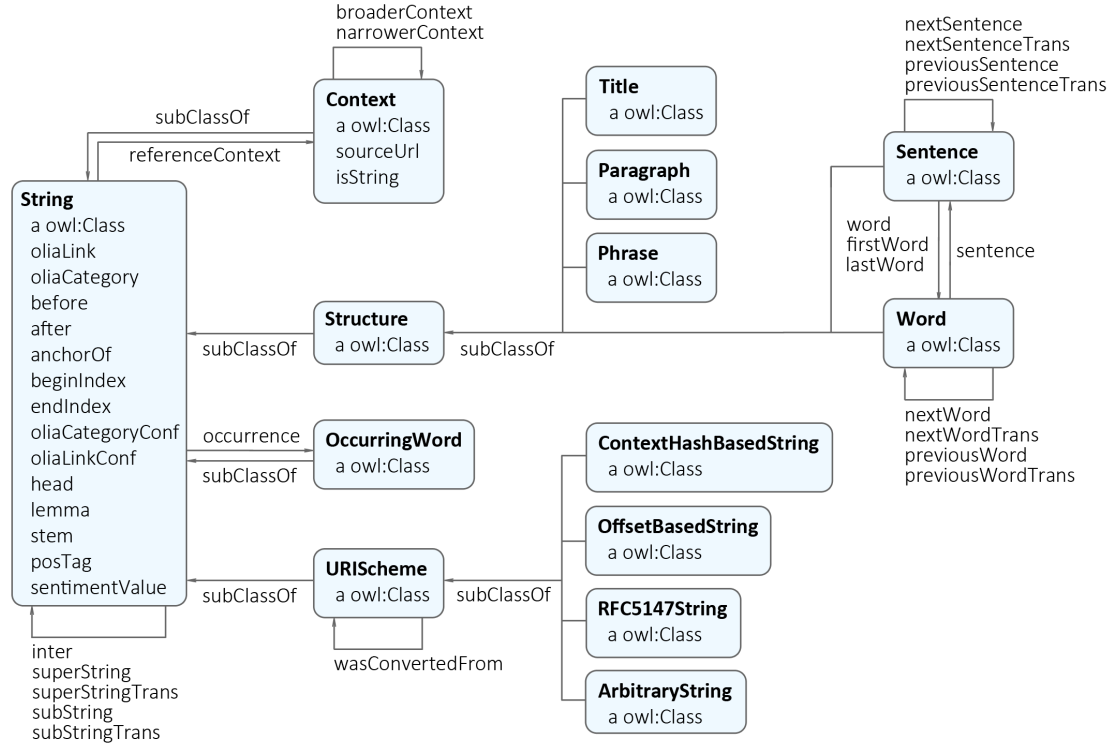
The NIF Core Ontology¹³ serves as the structural basis of every NIF-conformant resource. It provides classes and properties to describe arbitrary strings as well as documents, parts thereof and their mutual relations. It also includes complementary inference and validation models to extend the potential usefulness of the ontology while keeping it small enough to be easily understood. Figure 3.3 provides an overview over the ontology. The central class is `nif:String`, the class of all words consisting of Unicode characters. `Strings` offer properties to describe their content (the Unicode character string itself, via `nif:anchorOf`) and their position according to other strings with text indices (via `nif:beginIndex` and `nif:endIndex`) as well as various semantic information, like part of speech tags, sentiment values or word stems.

Subclasses of `nif:String` include the `nif:Context`, which contains the whole document’s text string and can be used to calculate the offsets of the individual substrings of the document. It also serves as a means of persisting the annotated text itself, should it either be changed in the original web resource or not be online as resource on the web any more. All substrings of the document link to the relevant `Context` element via `nif:referenceContext`. Such substrings can be categorized into classes such as `nif:Sentence` or `nif:Word`. A classification like that is important to be able to interface with existing NLP tools, that often do tokenization and sentence boundary detection as a basic step in the pipeline. `Words` and `Sentences` can be connected with additional properties to indicate that, for example, a `Word` is part of a `Sentence`, or is followed by another `Word`. These properties are necessary to keep track of the original ordering of the text for text traversal and similar tasks that rely on accessing one string at a time in their original order.

Because NIF is RDF-based, a NIF representation of a document is necessarily a graph. Thus, there is no “beginning”, as in a flat file or XML tree, where child elements of a root node can be parsed in order. An access point is presented by the `nif:Context` resource that contains the original document text, but if the individual string resources are to be consumed in order,

¹³<http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core/#>

with all their annotations and their original tokenization, only the string offsets can be used to order the strings. This especially applies for consecutive words in a sentence. Thus, properties like `nif:nextSentence`, `nif:firstWord` and `nif:nextWord` can be used for easy traversal of the individual strings of a document. This provides a less tedious way to parse the text in its original ordering than sorting character offset pairs.



Namespace nif: <<http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#>>

Figure 3.3: Class diagram of the NIF Core Ontology

3.3 OLiA

The Ontologies of Linguistic Annotation [5] are a number of modular ontologies, aiming to map strings commonly used in linguistic annotations to ontological entities. While NIF aims to provide syntactic interoperability between NLP tools and relevant corpora, OLiA provides a component of semantic interoperability by mapping the disparate annotation terms used by different tools and corpora to entities of a common reference model. It was originally developed in a project on “Sustainability of Linguistic Resources”¹⁴ which had the goal of creating accessible dissemination and storage of linguistic corpora of heterogeneous provenance, size, text type, linguistic theories, language and annotation.

Differences in annotation terminology range from minor differences in the choice of tag names to more fundamental variations. For each of these different annotation schemes, OLiA provides an

¹⁴<http://nachhalt.sfb632.uni-potsdam.de/owl/ontologies-background.html>

Annotation Model, as well as a *Linking Model* to the common *Reference Model*. The Annotation Model formalizes the annotations of the annotation scheme and provides OWL resources for the tags. These include the tag string as well as membership information to a descriptive `owl:class`, which is linked to a class of the Reference Model via `rdfs:subClassOf` using the Linking Model. Additionally, `rdfs:comments` exist for further details on use and meaning of the tags. For example, the OLiA mapping of the Penn Treebank[16] tag “NN” used to annotate common nouns can be found in listing 2. The first resource belongs to the annotation model, describing the tag via the `hasTag` property as well as the part of speech it encodes. The second resource belongs to the linking model, defining the specific part of speech as used in the corpus as a subclass of a matching part of speech in the OLiA reference model. Finally, the third resource belongs to the reference model.

```

1 #annotation model: penn.owl
2 <http://purl.org/olia/penn.owl#NN>
3   a <http://purl.org/olia/penn.owl#CommonNoun> ;
4   olia_system:hasTag "NN"^^xsd:string ;
5   rdfs:comment "Noun singular or mass"^^xsd:string .
6
7 #linking model: penn-link.owl
8 <http://purl.org/olia/penn.owl#CommonNoun>
9   rdfs:subClassOf <http://purl.org/olia/olia.owl#CommonNoun> .
10
11 #reference model: olia.owl
12 <http://purl.org/olia/olia.owl#CommonNoun>
13   rdfs:comment "A common noun is a noun that signifies a non-specific member of
14   a group."^^xsd:string ;
15   rdfs:subClassOf <http://purl.org/olia/olia.owl#Noun> .

```

Listing 2: Example mapping of the Penn tag “NN” to an entity of the reference model

In practice, aggregations of different corpora and tool outputs can thus be queried for links to this reference model. This allows, for example, to aggregate the output of an NLP tool that uses the Penn Treebank tagset¹⁵ and a corpus that uses the Stuttgart-Tübingen tagset (STTS)¹⁶ and query it for all words of the type “adjective”. Although they will be tagged with “JJ” according to the Penn tagset and with “ADJA” and “ADJD” according to the STTS tagset, they can be retrieved by querying words of `rdf:type olia:Adjective` using OLiA.

With regard to these properties, OLiA plays an important part in the NIF ecosystem to provide semantic interoperability between annotations of different tools that would otherwise not be interoperable.

3.4 NERD

Like morpho-syntactic annotations that are handled differently by different tools, *Named Entity* types identified by *Named Entity Recognition* (NER) algorithms may also be different and have to be mapped to a common class to make them interoperable. Named Entity types are pre-defined categories, such as persons, locations or organizations, used to classify text elements. What is considered as Named Entity type depends on the annotation tool. DBpedia Spotlight [17], for example, supports the manual selection of types based on the DBpedia ontology¹⁷, featuring 31 broader categories, such as person, place, organisation, event or work and 254 more granular types, such as politician, populated place, sports team, film festival or song. Other tools, such as the OpenNLP name finder tool with its pre-trained models, are limited to less granular categories,

¹⁵<http://www.comp.leeds.ac.uk/ccalas/tagsets/upenn.html>

¹⁶<http://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/tagsets/stts-table.html>

¹⁷http://downloads.dbpedia.org/3.9/dbpedia_3.9.owl

like date, location, money, organization, percentage, person and time for the English model¹⁸. Similar to the POS tag example given in the previous section, the types themselves can also be expressed with different strings.

The Named Entity and Disambiguation (NERD) ontology [24, 23] is a set of manual mappings between different taxonomies of named entity types, developed to solve this issue. It is composed of two parts, the NERD core, which is a list of 10 classes representing named entity types including a number of `owl:equivalentClass` references to equivalent types used by different NER tools and frameworks and the NERD inferred axioms, that provide granular `rdfs:subClassOf` relations between different fine grained entity types and core classes. Listing 3 shows an example mapping for the type “Person”, which also shows the differences between the frameworks as different letter case.

```

1  nerd:Person
2    a owl:Class ;
3    rdfs:subClassOf nerd:Thing ;
4    owl:equivalentClass
5      opencalais:Person , semitags:person ,
6      saplo:Person , zemanta:person ,
7      extractiv:PERSON , yahoo:person ,
8      alchemyapi:Person , wikimeta:PERS , dbpedia:Person .
10
11  nerd:Ambassador
12    rdfs:subClassOf
13      nerd:Person ,
14      extractiv:AMBASSADOR .

```

Listing 3: NERD core mappings for the named entity type “Person”

NERD thus provides the possibility of including interoperable Named Entity tags in NIF annotations by linking the relevant NERD class via the NIF property `itsrdf:taClassRef`.

4 Conversion of corpora into NIF

The usefulness of the NIF ecosystem depends on the availability of wrapped tools as well as corpora available in NIF. Before beginning this work, only few NIF corpora existed, which are presented in Section 4.5, all of them for named entity recognition and all of them relatively small, measuring from 2000 to 13000 triples. To expand the NIF ecosystem and evaluate its capability of acting as a corpus pivot format and to scalably validate and parse NIF datasets, more and larger NLP corpora were needed. Because a large number of corpora already are available, the task was to convert a few of them into NIF 2.0. The choice of which corpus to convert was defined in a number of criteria elicited from several discussions with Sebastian Hellmann and the wider NLP2RDF community:

- The license had to be open, as the finished corpus was to be re-published as Linked Data to enable other users of NIF to test their tools and re-use the data
- Annotation had to be of high quality to be able to serve as a gold standard. Corpora with automatic annotation without intellectual overview were not used because their precision is limited by the tools used to generate the annotations.
- Larger size corpora were preferred to evaluate the scalability of NIF. The heterogeneity of the data typically grows with larger size, which provides an additional challenge to conversion as well as the chance to show the adequacy of the format for real world big data.

¹⁸<http://opennlp.sourceforge.net/models-1.5/>

- The annotations of the corpora chosen should be varied, spanning a number of different NLP tasks, like POS tagging and Named Entity Recognition.

For sentence detection and POS tagging, the Brown corpus qualified for the purpose. Its main benefit lies in its open availability on one hand and its prominence as one of the most well-known corpora on the other hand. It will be described in Section 4.1. For named entity recognition evaluation, English Wikipedia abstracts (Section 4.3) were extracted. This dataset did not exist as a corpus before, but it is open, available and can serve as a source for a number of interesting experiments and thus qualified for conversion to NIF. The Wikilinks corpus (Section 4.4) was converted because of its massive size as well as the “messy” web data it contains. For named entity recognition training, corpora were already created by the community. They will be introduced in Section 4.5. Finally, to define how to represent dependency trees in NIF and provide a converter for the widely used CoNLL 2009 corpus format, the Tiger corpus was used and will be presented in Section 4.2. The complete contribution of corpora presented in this work adds up to 841,106,737 triples.

Generally, the development of the conversion algorithms was divided into two parts, a general data analysis and conversion implementation phase which will be presented separately for the corpora, followed by a data-test driven debugging and validation phase to find errors of the algorithms that produce incorrect data and ensure corpus correctness, presented in Section 5.

4.1 Brown corpus

The Brown Corpus [7] is a general corpus published in 1964 containing around 1 million words in 500 English texts from various sources, like newspapers texts on diverse topics, non-fiction and fiction books as well as government documents. It was manually annotated with word and sentence boundaries, as well as word class. Word class of annotations encompass:

- Part of speech, such as noun (common and proper), verb, adjective etc.
- Function words such as determiner, conjunctions etc.
- certain words of interest such as *not* and forms of *to be*.
- punctuation marks
- inflectional morphemes, such as noun plural, verb tense and adjective comparison
- special tags to mark foreign or cited words.

A more granular explanation to be found in the Brown corpus manual¹⁹. Annotations are expressed as *tags*, markers that consist of a number of upper case letters and stand for an individual category of words. For example, nouns that are either singular or don’t have a plural, so called “mass nouns”, are annotated with the tag “NN”.

Source corpus format The format of the Brown corpus used for conversion was XML according to TEI²⁰. Documents of the Brown corpus in TEI-XML can be parted into a `teiHeader` element containing metadata like title and source of the document and a `text` element containing the text itself. Figure 4.1 shows a sample of the original XML as well as NIF resources extracted from it.

¹⁹<http://clu.uni.no/icame/brown/bcm.html#bc5>

²⁰Text Encoding Initiative

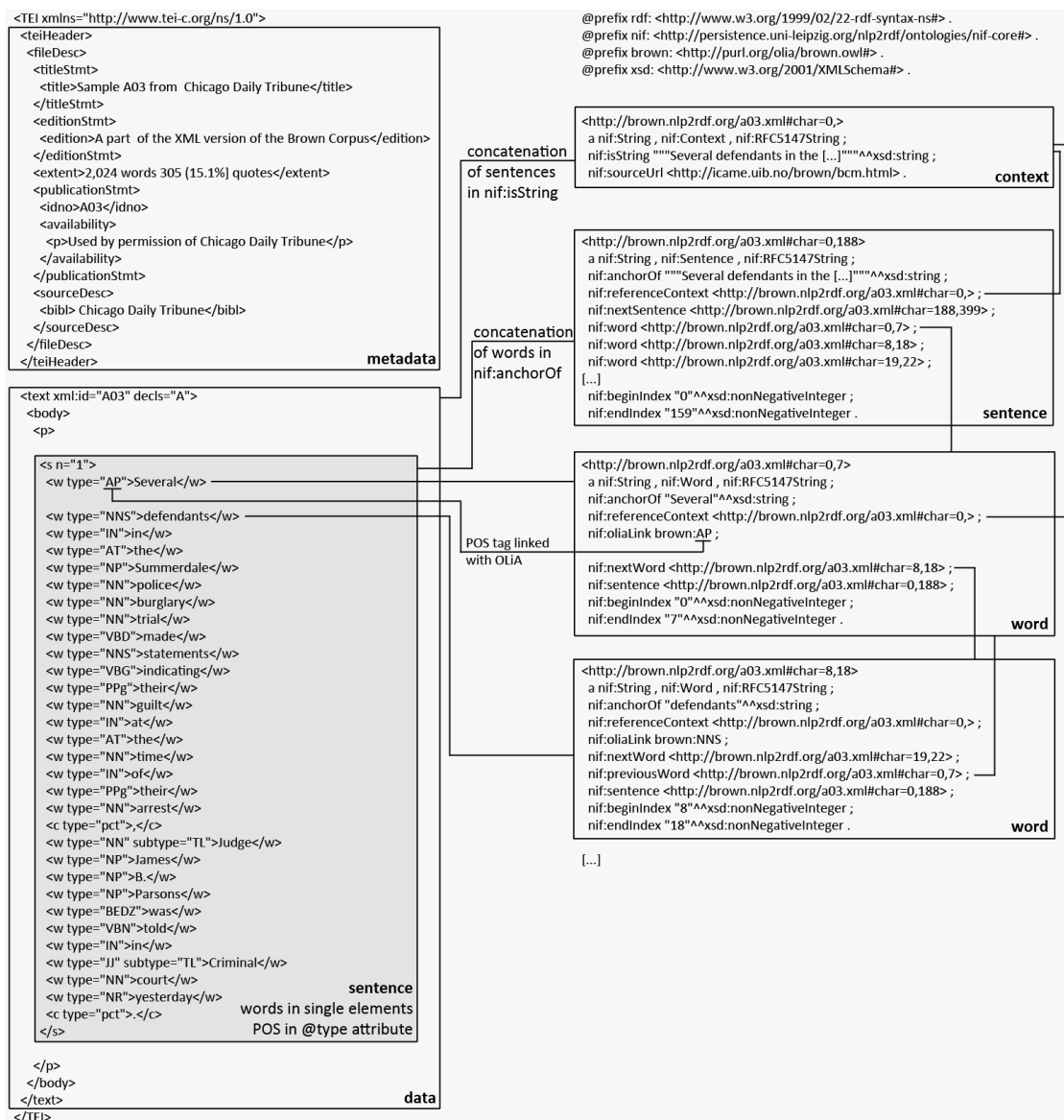


Figure 4.1: Original TEI corpus elements and according NIF resources, as well as links between NIF resources

The **text** contains a **body** element which in turn contains a number of **paragraphs** with **sentences** and **words** in individual elements. Word elements contain a text node that represents the words they annotate. Tags are found in **type** attributes of the word elements.

One of the problems of the format is, that, due to the non-standoff annotation, it is impossible to extract the original text of the document before it was annotated and added to the corpus. The extraction of the text of the nodes leads to an enclosure of every elements' text content with whitespaces. Therefore, punctuation is also separated from the rest of the text via a whitespace,

which does not reflect standard text. This can be alleviated with a set of rules specific to punctuation, so that whitespaces in front of punctuation marks are eliminated. This would imply directly changing the text of the corpus, which is problematic because of licensing on one hand - transferring into another format and changing the content of the data are very different, the last of which forbidden by the Brown license. On the other hand, it would be hard to document, as it would have to be explicitly stated in some kind of added documentation, although the semantically annotated data is supposed to stand on its own. Even though the conversion to NIF already implies changes to the text, because the source text was annotated in a non-standoff way, no special rules for punctuation marks were introduced for these reasons.

Target NIF format For each of the 500 original XML documents, one `nif:Context` resource containing the whole document text as well as the link to the source corpus was established. For every `sentence` element, a new `nif:Sentence` resource containing the sentence string in its `nif:anchorOf` predicate and linking to the context via `nif:referenceContext` was established. Similarly, for every `word` element, a new `nif:Word` resource was established. Each `nif:Word` resource links to its `nif:Sentence` as well as to the resource that represents the following word in the sentence. It also contains a `nif:oliaLink` for the POS-tag and the string of the word with `nif:anchorOf`. Finally, every sentence links to every word it contains. This results in two possible ways to iterate over all words of a sentence: 1. Getting all words linked via `nif:word` by a sentence element and 2. Getting only the first word of the sentence and then traversing via the `nif:nextWord` predicates.

The consistent use of `nif:anchorOf` properties in every sentence and word resource introduces some redundancy because the complete document string can be found in the `nif:Context`, the sentence strings are repeated in the `nif:Sentences` and the word strings again in the `nif:Words`. NIF suggests this procedure; every resource of the type `nif:String` that does not have the type `nif:Context` should have a `nif:anchorOf` property triple containing the string it represents. But through `nif:referenceContext` and the string offsets of the strings, the strings could easily be extracted as substrings of the reference context string. Having them redundantly available via `nif:anchorOf` is just convenient. The Brown corpus being a stable and established resource, `nif:anchorOf` could be dropped to lower the footprint of the final NIF corpus. As seen in table 1, this would lower the overall size compared to full NIF by 9,7%. The benefits of this relatively small reduction in size will in most use cases be outweighed by the usefulness of having the strings the resources annotate directly available for further processing. Not addressing the words at all and only annotating sentences is an interesting use case. Words and their part of speech may not be interesting to some use cases, like sentence boundary detection training and evaluation. The resulting corpus would be valid NIF and 94,8% smaller than the full NIF corpus, making it easier to parse, saving system memory and online bandwidth. This demonstrates that publishing different extractions and parts of corpora for different use cases might be beneficial for the users. However, this extraction be achieved by a simple SPARQL query executed by the users themselves. In contrast, adding the NIF properties for easier traversal of sentences and words, like `nif:word` to link sentences to words they contain, or `nif:nextSentence` to traverse sentences in order, nearly doubles the size of the corpus, making use of these properties ambivalent in terms of return of investment.

NIF conversion implementation For NIF conversion, a custom Java converter was written. Because the documents that comprise the corpus are relatively small in size, a DOM²¹ parser was used, because the XML documents that comprise the corpus are on average only 56.8 KB in size.

²¹Document Object Model

Corpus Version	File Size	Number of Triples
TEI-XML	28 MB	-
NIF 2.0 - only sentence resources, no anchorOf	26.8 MB	346,610
NIF 2.0 - only sentence resources, including anchorOf	28.7 MB	403,795
NIF 2.0 - sentence and word resources, no anchorOf	504.4 MB	8,473,806
NIF 2.0 - sentence and word resources, including anchorOf	553.4 MB	9,692,019
NIF 2.0 - complete, including nif:sentence, nif:word, nif:nextSentence and nif:nextWord	1.145 MB	14,335,131

Table 1: Size comparison between different NIF Turtle versions of the Brown corpus

Such small XML documents can be easily handled by a DOM parser without performance issues and simply do not merit the implementation of a SAX parser. The documents were iteratively read and parsed into DOM `Document` objects. All sentence elements were added to a `NodeList` that was iterated over. The text content of the sentence elements was extracted and concatenated in a document text string, as well as added to a `Sentence` object that also holds the strings start and end offsets. To determine the start offset of the first sentence, its position in the document string was used. End offsets were computed by adding the string length of sentences. Words were treated the same, adding them to a list of `Word` objects in the respective `Sentence` object. After parsing the XML into Java objects, they were iterated over and written to disk as NIF in the Turtle format by inserting the objects' data into Turtle string templates that could be written by a `BufferedWriter`. The final corpus features 14,335,131 triples, denoting 57,185 sentences containing 1,161,028 words.

4.2 Tiger ConLL dependency corpus

ConLL is a family of formats used for the shared tasks of the yearly Conference on Computational Natural Language learning²². The shared tasks are challenges for NLP researchers and change from year to year, typically following current trends and developments in NLP technology. Part of the task usually is a number of data files containing training and evaluation data used to benchmark the solutions presented by different algorithms. These files have a common format that, because of the prestigious conference, became a commonly used file format in NLP. Because of the shared nature of the tasks, there is also a lot of high quality, annotated data available in a number of languages that otherwise may be either strictly licensed or hard to obtain. This large acceptance and wide adoption are reasons to develop a component for conversion of ConLL data to NIF.

ConLL files are flat files containing a number of lines with tab separated values. Each line contains one word of a sentence in the first field and associated annotations in the other fields.

²²[urlhttp://ifarm.nl/signll/conll/](http://ifarm.nl/signll/conll/)

Sentences are ended by empty lines. Annotations themselves differ significantly based on the task presented. For example, the ConLL 2009 shared task was on dependency parsing. A short description²³ of the contained fields can be found in table 2.

As one of the most important German corpora featuring a research license, the Tiger Treebank [4] was also released in the ConLL format²⁴ and can serve as an example resource for transformation of ConLL files to NIF.

Conversion implementation Again, a converter was written for the ConLL format. Because there is a large number of ConLL files available and the converter can thus be reused instead of providing a static conversion like in the case of Brown and Wikipedia Abstracts, the converter for this corpus supports NIF API parameters according to the NIF Public API specification²⁵. Its needed parameters are either an input file or an input text, as well as the input type (file or direct) and input format (only text). An additional parameter `tagset` that is not included in the NIF API parameters was added. It allows specifying the name of the tagset used to annotate the corpus for OLiA linking of POS tags contained in the corpus. The tagset name has to be one of the tagsets included in OLiA and implemented in the NIF vocabulary module²⁶. After accepting the parameters, the converter creates a Jena OntModel as well as an initial `nif:Context` resource, that will contain the whole document content (all sentences concatenated) in its `nif:isString` property. It then consumes the input file or string line by line, adding all lines to an `ArrayList<String>` until a empty line comes up, that signals the end of the sentence, so that the List contains exactly one sentence. The elements of the `ArrayList` are then processed into a List of `ConLLWords` by splitting the strings via `split("\t")` and adding the resulting array elements to `ConLLWord` objects. `ConLLWords` contain the information from all fields of the original file in variables, so that they can be accessed easier. They also contain start and end offsets in regard to the document string. During this process, the sentence string is also being built.

Special attention is given to punctuation and parenthesis. Because ConLL encodes one token per line, the original string can not be recreated properly. If every token was treated the same, one had to assume that after each token, a whitespace would follow. However, this is not true for special characters like punctuation or parenthesis, because there is no whitespace in front of punctuation and closing parenthesis and no whitespace after opening parenthesis. This was addressed by checking if the words belong to these characters with two regular expressions, presented in the following listing.

```
1 ([\\.!?,;:)]|' '+)
2 ([(|'| '+)
```

The first expression checks for characters that should not have a leading whitespace, like punctuation, commas, colons or chains of single quotes. The second one checks if the character is an opening parenthesis or a chain of opening quotes, which should not have trailing whitespaces. In all other cases, tokens will be appended by whitespaces automatically.

After building the sentence string and the word objects, a Jena `Individual` was created for the sentence and each of the words, adding the `Individuals` to the Jena model and links to the word resources via `nif:word` to the sentence resource. The `Individuals` also get to link to the `nif:Context` resource. The word `Individuals` received additional properties according to the

²³Full format documentation here: <http://ufal.mff.cuni.cz/conll2009-st/task-description.html>

²⁴<http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/TIGERCorpus/download/tigercorpus-2.2.conll09.tar.gz>

²⁵<http://persistence.uni-leipzig.org/nlp2rdf/specification/api.html>

²⁶<https://github.com/NLP2RDF/software/tree/master/java-maven/vocabularymodule/OLiA/src/main/java/org/nlp2rdf/vm/olia/models>

Field	Content description	Comments
ID	unique identifier of the word	
FORM	word form	
LEMMA	Lemma / base form of the word	"-" for punctuation, equals FORM for proper nouns
PLEMMA	predicted LEMMA	based on independently trained tagger, always "_" in Tiger corpus
POS	Part of speech	Tagset depends on corpus, STTS for Tiger
PPOS	predicted POS	based on independently trained tagger, always "_" in Tiger corpus
FEAT	morphological features	in Tiger corpus case, number, gender divided by " "
PFEAT	predicted FEAT	based on independently trained tagger, always "_" in Tiger corpus
HEAD	head of the phrase	ID of the phrase head, 0 for root node
PHEAD	predicted HEAD	based on independently trained tagger, always "_" in Tiger corpus
DEPREL	type of dependency relation to HEAD	"-" for punctuation
PDEPREL	predicted DEPREL	based on independently trained tagger, always "_" in Tiger corpus
FILLPRED	Y for lines where PRED should be filled	always "_" in Tiger corpus
PRED	rolesets of semantic predicates in this sentence	always "_" in Tiger corpus
APREDS	columns of argument labels for semantic predicates in textual order	always "_" in Tiger corpus

Table 2: Description of the ConLL 2009 format and its realization in the Tiger corpus

available data in the ConLL file including `nif:lemma` and `nif:posTag`, as well as `nif:oliaLink` and `nif:oliaCategory` using the configured tagset. The dependency tree is also parsed and its information added to word resources via `nif:dependency` that points from the governing phrase to the dependent phrase. An additional property `nif:dependencyRelationType` was introduced, that contains the content of the ConLL DEPREL field. The properties retrieved by parsing the dependency tree were then added to their respective words. This steps concludes the parse of the sentence. The sentence string is added to `contextString` and the next set of lines is consumed.

After the whole ConLL document is parsed, the `contextString` is added to the `nif:Context` resource via `nif:isString` and its length is set as the end offset of the document string. The final model is then put out either via `System.out` or written to a file in the configured format. Listing 4 gives an example of the resulting NIF data.

```

1 <http://example.org/tiger_conll#char=29,32>
2   a      nif:String , nif:RFC5147String , nif:Word ,
3         nif:Phrase ;
4   nif:anchorOf      "ein" ;
5   nif:beginIndex    "29" ;
6   nif:dependencyRelationType "NK" ;
7   nif:endIndex      "32" ;
8   nif:lemma         "ein" ;
9   nif:oliaCategory  olia:Article , olia:PronounOrDeterminer ,
10  olia:Determiner ;
11  nif:oliaLink      <http://purl.org/olia/stts.owl#ART> ;
12  nif:posTag        "ART" ;
13  nif:referenceContext <http://example.org/tiger_conll#char=0,55>
14  ;
15  nif:sentence      <http://example.org/tiger_conll#char=0,55>
16  .
17
18 <http://example.org/tiger_conll#char=33,43>
19   a      nif:String , nif:Word , nif:RFC5147String ,
20         nif:Phrase ;
21   nif:anchorOf      "prächtiger" ;
22   nif:beginIndex    "33" ;
23   nif:dependencyRelationType "NK" ;
24   nif:endIndex      "43" ;
25   nif:lemma         "prächtigtig" ;
26   nif:oliaCategory  olia:Adjective , olia:AttributiveAdjective
27  ;
28   nif:oliaLink      <http://purl.org/olia/stts.owl#ADJA> ;
29   nif:posTag        "ADJA" ;
30   nif:referenceContext <http://example.org/tiger_conll#char=0,55>
31  ;
32  nif:sentence      <http://example.org/tiger_conll#char=0,55>
33  .
34
35 <http://example.org/tiger_conll#char=44,52>
36   a      nif:RFC5147String , nif:Word , nif:String ,
37         nif:Phrase ;
38   nif:anchorOf      "Diktator" ;
39   nif:beginIndex    "44" ;
40   nif:dependency    <http://example.org/tiger_conll#char=33,43>
41   , <http://example.org/tiger_conll#char=29,32> ;
42   nif:dependencyRelationType "PD" ;
43   nif:endIndex      "52" ;
44   nif:lemma         "Diktator" ;
45   nif:oliaCategory  olia:CommonNoun , olia:Noun ;
46   nif:oliaLink      <http://purl.org/olia/stts.owl#NN> ;
47   nif:posTag        "NN" ;
48   nif:referenceContext <http://example.org/tiger_conll#char=0,55>
49  ;
50  nif:sentence      <http://example.org/tiger_conll#char=0,55>
51  .

```

Listing 4: Example resource of the Tiger corpus

Tiger corpus example To evaluate the converter, the Tiger Treebank was converted into NIF. The 50,472 sentences and 888,238 words of the corpus were transformed into NIF in 14:03 minutes. Memory consumption was very high, needing over 5.7 GB memory to process the 60 MB ConLL file (figure 4.2). The final corpus features 17,428,613 triples.

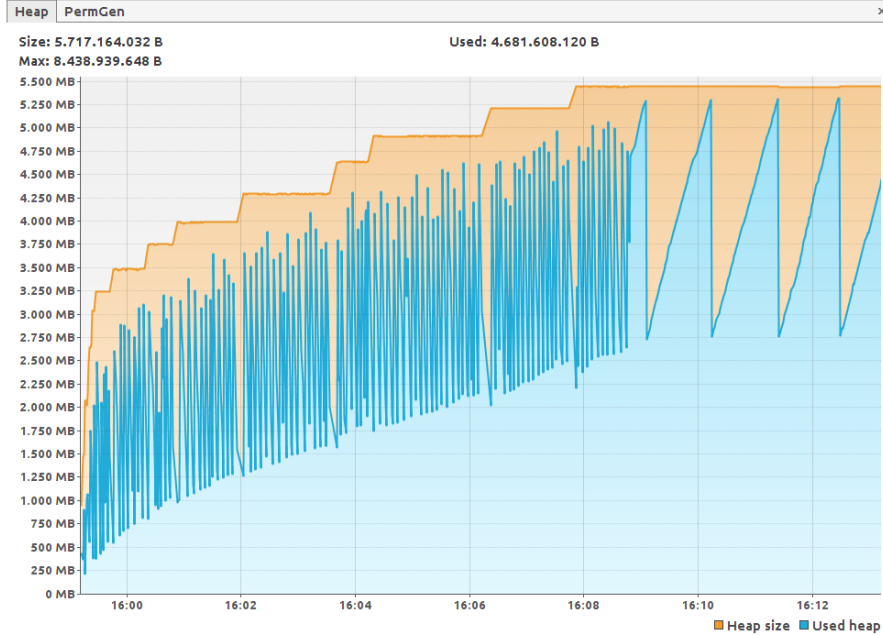


Figure 4.2: Memory consumption of the Tiger corpus conversion process

4.3 Wikipedia abstract corpus

Wikipedia is the most important and comprehensive source of open, encyclopedic knowledge. The English Wikipedia alone features over 4.280.000 entities described by basic data points, so called info boxes, as well as natural language texts. Besides the Wikipedia web site, Wikipedia data is available via a PHP API as well as complete XML dumps. However, web site and API access are officially limited to one request per second, prohibiting a naive web scraping approach to obtain the data. Even after downloading the XML dump files, consuming and processing them is a complicated task further hindering large-scale data extraction. To alleviate these issues, the DBpedia project [1] has been extracting, mapping, converting and publishing Wikipedia data since 2007, establishing the LOD cloud and becoming its center in the process. The main focus of the DBpedia extraction lies in mapping of info boxes, templates and other easily identified structured data found in Wikipedia articles to properties of an ontology, that is build in the process in a bottom up way. Article texts and the data they may contain are not especially focused, although they are the largest part of most articles in terms of time spent on writing, informational content and size. Only the text of the first introductory section, up until the first heading after the main article heading, of the articles is extracted and contained in the DBpedia, calling them *abstract*. Links inside the articles are only extracted as an unordered bag, showing only an unspecified relation between the linking and the linked articles, but not where in the

text the linked article was mentioned or which relation applies between the articles. As the links are set by the contributors to Wikipedia themselves, they represent entities intellectually disambiguated by URL. This property makes extracting the abstracts including the links and their exact position in the text an interesting opportunity to create a corpus usable for, among other cases, NER and NEL algorithm evaluation. In fact, Wikipedia abstracts are already widely used for training purposes.

Furthermore, the abstracts represent a special form of text that lends itself to be used for more sophisticated tasks, like open relation extraction. The articles have an encyclopaedic style [20], describing the topical entities by relating them to other entities often explicitly linked in the text. Following the official Wikipedia guidelines on article style and writing²⁷, the first sentence usually tries to put the article in a larger context. Therefore Wikipedia abstracts can be understood as “Is-A”-corpus, defining an entity in relation to a broader class of related entities or an overarching topic. This interesting property is exemplified by the fact that by clicking the first link in a Wikipedia article, one will eventually reach the article on “Philosophy” for 94,5% of all articles²⁸. The encyclopedic style also suggests that anaphers²⁹ appearing as subject of a sentence reference the topic of the article, easing coreference resolution. Due to the topical restriction and the Wikipedia guidelines on entity linking, it is also a reasonable assumption that words appearing in article texts linked on their first occurrence and reappearing unlinked later represent the same entity. Programmatically exploiting these properties strongly hinges on link extraction and support the high value of the corpus in spe.

Conversion implementation Because Wikipedia data is only obtainable as a very large XML dump with a proprietary structure, and other ways of extraction like web scraping are too slow, a processing pipeline had to be used to access the data. The main hindrance is, that raw Wikipedia articles are written in Wiki markup³⁰, a special syntax and keywords that are used to format Wikipedia articles and add further data. Besides common syntax, like for example representing a `==Heading==`, it also features calls to external LUA scripts and templates, making it a very tedious and wikipedia language-specific task to implement a tool to render Wikipedia articles just like Mediawiki³¹ does. Therefore a local Mediawiki instance as mirror of the Wikipedia is installed and configured as first part of the extraction pipeline, tasked with rendering the Wiki markup to plain text.

Figure 4.3 shows the data flow of the pipeline used. Central to the extraction was the DBpedia extraction framework³² an open source framework to load and convert Wikipedia data to LOD written in Scala. It provides tools and documentation for necessary tasks, such as the import of the Wikipedia dump into the local Mediawiki’s database as well as the extraction itself.

For the extraction of abstracts, the framework uses Mediawiki’s API. For every page to extract data from, an HTTP request is made to the locally hosted API, using the `TextExtracts`³³ extension with the parameters `exintro` and `explaintext`. This way, the plain text of the first section as rendered by Mediawiki, without any markup can be obtained. Because this makes it impossible to extract the links, a new extractor class was written for the corpus conversion,

²⁷Wikipedia Manual of Style, Lead section, Opening paragraph: http://en.wikipedia.org/wiki/Wikipedia:MOSBEGIN/#Opening_paragraph

²⁸Data compiled by Wikipedia user “Ilmari_Karonen” using English Wikipedia articles from 26 May 2011: http://en.wikipedia.org/wiki/User:Ilmari_Karonen/First_link

²⁹like pronouns or synonyms, hyponyms, hypernyms or holonyms

³⁰http://en.wikipedia.org/wiki/Help:Wiki_markup

³¹Wikipedia’s software, <http://www.mediawiki.org/>

³²<https://github.com/dbpedia/extraction-framework>

³³<http://www.mediawiki.org/wiki/Extension:TextExtracts>

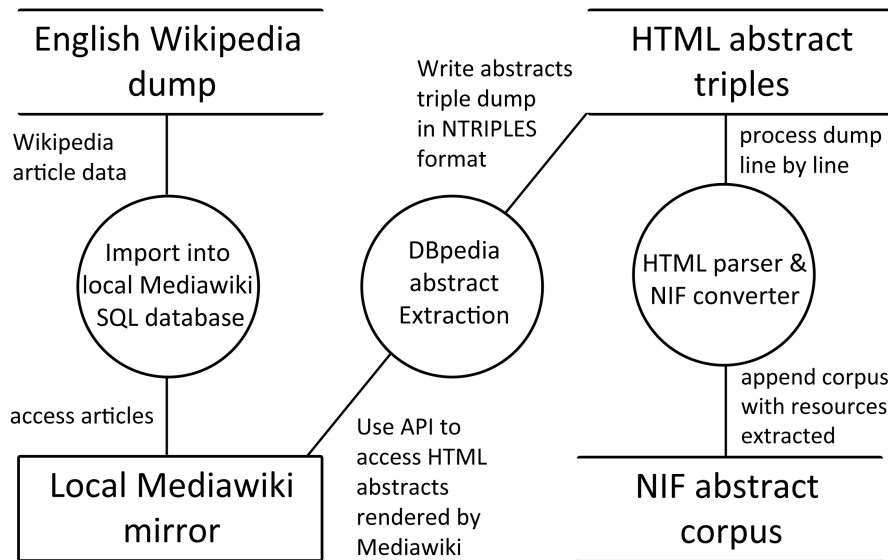


Figure 4.3: Data flow diagram showing the data conversion process

that uses a different API. This `AbstractLinkExtractor` uses the standard Mediawiki API with the parameters `action=parse` and `section=0` to obtain the complete HTML source of the first section of each article. Running this framework using this extractor and the local Mediawiki mirror produced a 30GB file in ntriples format, containing triples of the following form.

```
1 <$DBpediaUri> <http://dbpedia.org/ontology/abstract> "$htmlAbstract".
```

These triples present an intermediary data format that only serves for further extraction of HTML abstracts and the respective DBpedia URI. The `$htmlAbstract` literal containing the HTML text was then split into relevant `paragraph` elements, which were parsed with a SAX-style³⁴ parser.

Opposite to DOM-style parsers that have to parse the whole document before allowing operations on it, SAX-parsers operate by sequentially parsing the documents, reporting events such as encountering the start or end of an element as well as its element name. This is less memory intensive than DOM-parsers. Because the whole HTML string had to be parsed anyway to obtain the whole text, the loss of DOM functions like access to specific elements could be disregarded. No attention had to be paid to the validity of the HTML either, because Mediawiki HTML is already valid because of the Mediawiki's software correctness.

All child element of the relevant `paragraphs` were traversed by the parser. If a `#text` element was encountered, its content was appended to the abstract string. If an `a` element with attribute `href` was encountered, a `Link` object was created, containing the start and end offsets of the `#text` child element of the `a` element, as well as the link text itself and the linked URL. Other elements were skipped to exclude `tables` and `spans` containing various other information not representable as plain text. Special consideration was given to Mediawiki specific elements used for visual styling. For example `kbd` elements were not skipped, because they are used to style the text they contain as if it was a keyboard letter. Thus they contain important content easily

³⁴Simple API for XML

representable as text. After completely parsing the paragraphs, the resulting strings were concatenated to the abstract string, the `Link` objects created were aggregated. Using simple string processing, the resulting data was written to disk in NIF 2.0 serialized in the turtle format before consuming the next line. This streaming procedure allows for the small memory footprint needed when processing large amounts of textual data on consumer hardware, like in this case.

Target NIF format A `nif:Context` resource was established for each article, containing the article abstract in the `nif:isString` property, as well as the string offsets denoting its length and the URL of the source Wikipedia page. For each link, another resource was created, containing the link text in the `nif:anchorOf` property, its position in the string of the referenced `nif:Context` resource, as well as the URL of the linked resource via the property `itsrdf:taIdentRef`. Thereby, the identity of the literal string of the link text and the resource linked is made explicit, leading to the mentioned disambiguation of the link text by URL. An example resource can be found in listing 5.

```

1 <http://dbpedia.org/resource/Pizza/abstract#char=0,112>
2   a nif:String , nif:Context , nif:RFC5147String ;
3   nif:isString "'Pizza ([pittsa]) is an oven-baked flat bread typically
4     topped with a tomato sauce, cheese and various toppings.'"^^xsd:string;
5   nif:beginIndex "0"^^xsd:nonNegativeInteger;
6   nif:endIndex "112"^^xsd:nonNegativeInteger;
7   nif:sourceUrl <http://en.wikipedia.org/wiki/Pizza> .

8 <http://dbpedia.org/resource/abstract#char=24,28>
9   a nif:String , nif:RFC5147String ;
10  nif:referenceContext <http://dbpedia.org/resource/Pizza/abstract#char=0,112>
11  ;
12  nif:anchorOf "'oven'"^^xsd:string ;
13  nif:beginIndex "24"^^xsd:nonNegativeInteger ;
14  nif:endIndex "28"^^xsd:nonNegativeInteger ;
15  a nif:Word, nif:Phrase ;
16  itsrdf:taIdentRef <http://dbpedia.org/resource/Oven> .

17 <http://dbpedia.org/resource/abstract#char=40,45>
18   a nif:String , nif:RFC5147String ;
19   nif:referenceContext <http://dbpedia.org/resource/Pizza/abstract#char=0,112>
20   ;
21   nif:anchorOf "'bread'"^^xsd:string ;
22   nif:beginIndex "40"^^xsd:nonNegativeInteger ;
23   nif:endIndex "45"^^xsd:nonNegativeInteger ;
24   a nif:Word, nif:Phrase ;
25   itsrdf:taIdentRef <http://dbpedia.org/resource/Bread> .

```

Listing 5: Example resource of the abstract corpus

The final corpus contains 4,285,608 `nif:Context` resources, one for every Wikipedia article contained in the dump. It is published in form of 86 dump files, each containing around 50000 entities, ranging from 97 to 290 MB in size to a total of 15.7 GB, counting 276,326,693 triples.

Validation Due to the size of the data and the diversity of the HTML that had to be processed, a special effort was made to validate the data. Validation was 3-fold:

1. The Raptor RDF syntax parsing and serializing utility 2.0.13³⁵ (raptor2-utils) was used to make sure the turtle files are syntactically correct.

Raptor is a widely used RDF parsing tool implemented in c, that is capable of parsing and converting most RDF serializations, as well as counting triples. It was run for every of the 86 corpus files, ensuring that no errors were returned, considering the RDF valid after this step.

³⁵<http://librdf.org/raptor/>

2. The GNU tools `iconv`³⁶ together with `wc`³⁷ was used to make sure the files only contain valid unicode codepoints. This was necessary because the data of the English Wikipedia contains citations from other languages, articles about international topics and IPA sequences denoting phonology, that also heavily rely on unicode for presentation.

Iconv is typically used to convert text files from one format to another. However, starting it with the `-c` parameter makes it discard inconvertible (thus wrongly encoded) characters. Doing a character count before and after the `iconv` execution and comparing the number of characters would show that characters were dropped. This procedure again was run in a script on all files of the corpus, considering the unicode characters valid after this step.

3. RDFUnit³⁸ was used to ensure adherence to the NIF 2.0 standard. For an exhausting explanation of this process, see Section 5.

4.4 Wikilinks corpus

The Wikilinks corpus [26] is a coreference resolution corpus of very large scale. Coreference resolution is the task of connecting expressions, also known as mentions, to individual entities they refer to. The Wikilinks corpus contains over 40 million mentions and the respective Wikipedia entities (around 3 million) they refer to, as well as the surrounding text. Data was derived from a web crawl of over 10 million web pages that link to Wikipedia somewhere in the text. Mentions as well as links were aggregated together with the pages' source in the corpus I used³⁹, resulting in a corpus of over 180GB compressed size.

The corpus surpasses most free corpora in size by far and serves as a very good test bed for measuring scalability of RDF as well as performance of NER Disambiguation tools in a noisy and multi-domain environment. The authors of these pages being human, the coreference resolution was done manually by linking a certain word or phrase to a relevant Wikipedia page. This fact should result in the links being of relatively high quality. The corpus can be used for NER to disambiguate between entities, as the meaning of the text fragments is captured in the Wikipedia link set by the owner of the website. It also spans multiple domains, as the crawled websites are heterogeneous by nature. Furthermore, the links target the English Wikipedia and can be used to obtain corresponding links to the DBpedia, thus allowing seamless creation of a Linked Data NER corpus.

Source corpus format The corpus is divided into around 3 million items, consisting of the *website URI* of the crawled sites and a number of *mentions*, including the English Wikipedia link, the hyperlink anchor text, its byte offset and in most cases a *context string*, i.e. suffix and prefix around the anchor of variable length. Furthermore, every item has the complete DOM of the web page attached. Items can thus be understood as representing the document, while mentions are specific instances of phrases in the document. The corpus is distributed in the Apache Thrift⁴⁰ format and includes some methods for easy access of specific fields of the corpus data.

Conversion to NIF and Hosting as Linked Data. 15% of the items did not contain any mention with context strings and were therefore omitted. The rest of the items were converted

³⁶<http://www.gnu.org/savannah-checkouts/gnu/libiconv/documentation/libiconv-1.13/iconv.1.html>

³⁷https://www.gnu.org/software/coreutils/manual/html_node/wc-invocation.html

³⁸<http://aksw.org/Projects/RDFUnit.html>

³⁹<https://code.google.com/p/wiki-link/wiki/ExpandedDataset>

⁴⁰<https://thrift.apache.org/>

into one `nif:Context` resource per item. Every mention was converted into a `nif:Phrase` resource, referring to the context via `nif:referenceContext`, as usual. The entity the mention links was added via `itsrdf:taIdentRef`. The DBpedia ontology classes of the entity were extracted from DBpedia and added via `itsrdf:taClassRef`, as well as NERD ontology classes to ease mapping in a number of NER tools. In addition, the coarse grained NE class was identified by using the class of the entity that is a direct subclass of `owl:Thing`. For example, a Lighthouse is a `dbo:Building`, which is a subclass of `dbo:ArchitecturalStructure` which in turn is a subclass of `dbo:Place` which is a direct subclass of `owl:Thing` and thus the top level concept we want to extract. These classes equivalent NERD classes were annotated using `nif:taNerdCoreClassRef`. The direct surrounding text of each mention was extracted and converted into a `nif:Snippet` resource. Mentions link the Snippet resources via `nif:wasConvertedFrom`. In turn, the Snippet resources use `nif:wasConvertedFrom` to link the original resource, adding the Xpath of the HTML element they were extracted from by adding an `&xpath=` parameter and the relevant Xpath to the URI.

Every converted item was aggregated into one Turtle file. Filenames were computed using the MD 5 hashcode of the URI denoting the files two parent folder names and its filename. To make the documents Linked Data, they were assigned URIs of the form:⁴¹ `http://wiki-link.nlp2rdf.org/api.php?uri=$websiteURI#char=0,.` A PHP script extracts the URI, computes the MD 5 hash and redirects the browser via HTTP 303 response to the file itself.

```

1 <http://example.org/wikilinks#char=0,520>
2   a nif:String , nif:Context , nif:RFC5147String ;
3   nif:isString """English: Cana Island Light north of Baileys Harbor, Wisconsin
4   [...]"""^^xsd:string;
5   nif:beginIndex "0"^^xsd:nonNegativeInteger;
6   nif:endIndex "520"^^xsd:nonNegativeInteger;
7   nif:sourceUrl <http://example.org/source.html> .

9 <http://example.org/wikilinks#char=0,68>
10  a nif:String , nif:RFC5147String , nif:Snippet ;
11  nif:referenceContext <http://example.org/wikilinks#char=0,520> ;
12  nif:beginIndex "0"^^xsd:nonNegativeInteger ;
13  nif:endIndex "68"^^xsd:nonNegativeInteger ;
14  nif:wasConvertedFrom <http://example.org/source.html&xpath=/html/body/div[3]>
15  .

16 <http://example.org/wikilinks#char=9,26>
17  a nif:String , nif:RFC5147String ;
18  nif:referenceContext <http://example.org/wikilinks#char=0,520> ;
19  nif:anchorOf """Cana Island Light""""^^xsd:string ;
20  nif:beginIndex "9"^^xsd:nonNegativeInteger ;
21  nif:endIndex "26"^^xsd:nonNegativeInteger ;
22  nif:wasConvertedFrom <http://example.org/wikilinks#char=0,68> ;
23  a nif:Phrase ;
24  itsrdf:taClassRef <http://dbpedia.org/ontology/Lighthouse> ;
25  nif:taNerdCoreClassRef <http://nerd.eurecom.fr/ontology#Location> ;
26  itsrdf:taClassRef <http://nerd.eurecom.fr/ontology#Lighthouse> ;
27  itsrdf:taClassRef <http://dbpedia.org/ontology/Building> ;
28  itsrdf:taClassRef <http://dbpedia.org/ontology/ArchitecturalStructure> ;
29  itsrdf:taClassRef <http://dbpedia.org/ontology/Place> ;
30  itsrdf:taClassRef <http://nerd.eurecom.fr/ontology#Location> ;
31  itsrdf:taIdentRef <http://dbpedia.org/resource/Cana_Island_Light> .

```

Listing 6: Example resource of the Wikilinks corpus

The corpus resulted in 10.626.762 files hosted in an Apache2 file system `http://wiki-link.nlp2rdf.org/` containing 533.016.300 triples and 31.542.468 million links to DBpedia in 12GB

⁴¹e.g. `http://wiki-link.nlp2rdf.org/api.php?uri=http://phish.net/song/on-green-dolphin-street/history\#char=0,`

gzipped turtle dumps (79 GB uncompressed). Listing6 shows the format of an example resource.

4.5 Other NIF corpora

Besides the corpora converted and released for this work, five other NIF corpora have been developed. In [27], three corpora have been used for evaluation of three Named Entity Disambiguation tools. One of the corpora was an early version of the NIF Wikilinks corpus presented in 4.4 that did not contain the surrounding text of the mention. The other two corpora were already well known NERD corpora, DBpedia Spotlight and KORE50. The authors used NIF as a corpus pivot format, converting the non-NIF corpora into NIF for easier processing and comparability. Instead of three parsers for the different corpus formats, only one parser was needed. The converted resources could also be published, allowing easier replication of the results and creating a valuable resource for other researchers.

The DBpedia Spotlight corpus first presented in [17] contains 60 sentences from ten New York Times articles in seven different categories, from business to fashion. Specific words in the article texts were manually annotated with links to relevant DBpedia entities. Overall, there are 249 annotated DBpedia entities. The corpus was converted by the authors of [27] into NIF and published as a turtle file⁴². Every original article was converted into one `nif:Context` as well as a number of `nif:Sentences` and one `nif:Phrase` resource for every entity annotated string. The corpus contains 3425 triples and 325 links to DBpedia using the `itsrdf:taIdentRef` property.

KORE50 is a corpus created for the AIDA⁴³ project, which also aims to map natural language strings to canonical entities. The KORE50 corpus [11] is an openly licensed subset of the larger AIDA corpus and contains 50 hand-crafted sentences from different domains with ambiguous mentions and DBpedia entity links. The format is a CoNLL 2003 style TSV format. The corpus was also converted by the authors of [27] and also published as a turtle file⁴⁴. Every sentence was converted into a `nif:Context` resource, linking a common `nif:Context` resource using `nif:broaderContext`. This concept was used in the early iteration of the NIF Wikilinks corpus but is not considered valid NIF any more. Thus, a corrected version of the corpus following the structure of the DBpedia Spotlight NIF corpus was published later⁴⁵. The corpus contains 1410 triples and 144 links to DBpedia using the `itsrdf:taIdentRef` property.

Three further NERD corpora have been released in [25], called News-100, Reuters-128 and RSS-500. News-100 contains 100 German news articles containing the homonym *Golf*, which can refer to a car model, a gulf or a sport. Reuters-128 is based on a subset of the Reuters 21578 corpus⁴⁶ corpus, that consists of Reuters news documents from 1987. Entity links were automatically annotated using FOX [19] and manually corrected by two scientists. RSS-500 is based on a list of 1,457 RSS feeds initially compiled by [8]. Feeds include a range of international newspapers as well as a large number of topics. The complete corpus contains 11.7 million sentences of which 500 have been selected for manual annotation.

For all corpora, documents were manually annotated with links to DBpedia entities via `itsrdf:taIdentRef`. In addition, the DBpedia version of the linked entities was denoted using the `itsrdf:taSource` property and a string describing the version, "DBpedia_en_3.9" for English corpora and "DBpedia_de_3.9" for the German corpus. The corpora were published as Turtle

⁴²<http://www.yovisto.com/labs/ner-benchmarks/data/dbpedia-spotlight-nif.ttl>

⁴³AIDA: Accurate Online Disambiguation of Named Entities in Text and Tables, <http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/aida/>

⁴⁴<http://www.yovisto.com/labs/ner-benchmarks/data/kore50-nif.ttl>

⁴⁵<http://www.yovisto.com/labs/ner-benchmarks/data/kore50-nif-2014-11-03.ttl>

⁴⁶<http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html>

files on github⁴⁷. They contain 12289 (News-100), 6967 (Reuters-128) and 10038 (RSS-500) triples, featuring 1547, 650 and 524 DBpedia links respectively.

5 Validation of NIF corpora

Being designed as an interchange format, the NIF ecosystem depends on all instances of NIF application to comply with the standard. Because production of NIF-conform corpora is not trivial and especially large corpora can be heterogeneous in data and structure, formal validation is crucial. At the time of writing, the best tool to exhaustively validate a corpus using a test-driven approach was RDFUnit, which will be introduced in the following section. Following the introduction, test case implementation for NIF will be addressed in 5.2, before the tests will be used for the validation of the converted corpora, described in 5.3.

RDFUnit has been applied for validating NIF corpora before in [13], a paper I co-authored. The tests run in this thesis provide a deeper insight into the procedure and expand the tests by including more and larger datasets, as well as including a scalability test not contained in [13]. All tests were completely redone for this work, as both, the RDFUnit version as well as the corpora have changed.

5.1 RDFUnit

RDFUnit [14] is a Java framework for test-driven evaluation of Linked Data quality. It uses test cases containing two SPARQL queries each, which test if a dataset is in accordance to the axioms of its ontology. The queries the tests contain query for constraint violations as well as for the prevalence of the violations, if applicable.

1	# Violation Query		# Prevalence Query
2	SELECT DISTINCT ?s WHERE {		select count(distinct ?s) WHERE {
3	?s dbo:birthDate ?v1.		?s dbo:birthDate ?v1 .
4	?s dbo:deathDate ?v2.		?s dbo:deathDate ?v2 . }
5	FILTER (?v1 > ?v2) }		

Because the ontology already provides criteria for data correctness, most of the testing SPARQL queries can be generated automatically from the ontology, using a pattern based approach as well as derivation from OWL axioms themselves. For example, tests for domain and range restrictions of properties, type as well as cardinality restrictions can be automatically tested. In addition manual tests can be defined to specifically test for other criteria. These manual tests can be used to formalize correctness criteria that are either too complex to be derived from the ontology or that relate directly to data expressed by specific instances in the dataset. For example, one of the NIF tests uses the SPARQL SUBSTR⁴⁸ function to make sure that the `nif:anchorOf` string of each `nif:String` resource (that is not a `nif:Context`) equals the part of the respective `nif:referenceContext`'s string indicated by begin and end offset of said string. RDFUnit is able to either test dataset in a database, using a SPARQL-endpoint, or datasets dump files directly on the system. Besides the name and location of the data, the system needs the names of the schemas being tested against. It tries to locate and download the schemas using LOV⁴⁹ and automatically generates tests from the ontologies. It also tries to find manual tests in its test library before running the tests on the dataset.

⁴⁷<https://github.com/AKSW/n3-collection/>

⁴⁸substring

⁴⁹Linked Open Vocabularies: <http://lov.okfn.org/dataset/lov/>

5.2 Test Case Implementation for NIF

By running the existing RDFUnit Test Auto Generators (TAG) on the *NIF* ontology, 199 test cases were automatically generated. The RDFUnit Suite, at the time of writing, does not provide full OWL coverage and thus, complex `owl:Restrictions` cannot be handled yet. In the frame of the examined ontology, RDFUnit did not produce test cases for unions of (`owl:unionOf`) restrictions with `owl:allValuesFrom`, `owl:someValuesFrom` and `owl:hasSelf`. NIF further has defined semantic constraints that can not be captured. For example, the strings found in the `nif:isString` property of the `nif:Context` are referred by other `nif:String` resources and must match their referred offsets. The constraints are largely expressed in natural language text in the `rdfs:comment` properties of the NIF ontology file, as well as further documentation and specification documents. Formalizing the constraints is a complicated knowledge extraction task in itself. With NIF, the developers know how the data is supposed to be structured. It is however not clear what errors can occur when the data is produced, which in turn makes it hard to check for these errors. Writing test cases is thus not a straightforward task because the knowledge of the desired output has to be transferred from the developers mind into a set of applicable tests. Furthermore, evaluating the coverage of the tests in terms of test-case adequacy, i.e how many of all possible errors are discovered by the test suite, is complicated, because of the heterogeneous nature of the data processed.

In [13], 10 legacy tests from a previous NIF validator were being used. Later, two tests were merged into one, leaving 9 tests. I was involved in debugging the tests for that work. However, during the testing procedure for this thesis, 2 new tests relating to newl< discovered correctness constraints were implemented. In the following, the correctness criteria beyond the ontology definitions will be presented, as well as selected test cases to demonstrate the formalization procedure. The additional test cases will be described in more detail, because they exemplify the mentioned difficulty of the task.

Besides ontological criteria, NIF imposes the following constraints. Constraints marked with * were discovered during this work. Constraints marked with + were debugged and edited during this work.

1. + a `nif:anchorOf` string must match the substring of the `nif:isString` from `nif:beginIndex` to `nif:endIndex`
2. the old namespace shall not be used
3. the `nif:beginIndex` of a `nif:Context` must be equal to zero (0)
4. + the length of `nif:isString` inside a `nif:Context` must be equal to the resource's `nif:endIndex`
5. the RFC of the URI scheme (RFC5147) must be spelled correctly
6. every `nif:String` that is not a `nif:Context` must have a `nif:referenceContext`
7. `nif:Context` must have a type that is a subclass of `nif:CString`
8. `nif:CString` is abstract and may not be serialized without another subclass
9. * the length of a string must not be zero
10. * the `nif:endIndex` of a `nif:String` must not be larger than the `nif:endIndex` of the `nif:referenceContext`

The most important test is test 1. As stand-off annotation format, the correctness of the annotation strings' offsets in reference to the document text is the most crucial information. It is used to pinpoint the position of the annotated string in the text itself, as well as in the URI's `char` fragment, allowing for the merge of multiple annotations. The original test case for this constraint in [13] was:

```

1 SELECT DISTINCT ?s WHERE {
2   ?s nif:anchorOf ?anchorOf ; nif:beginIndex ?beginIndex ;
3     nif:endIndex ?endIndex ;
4     nif:referenceContext [ nif:isString ?referenceString ] .
5   BIND (SUBSTR(?referenceString,
6             ?beginIndex , (?endIndex - ?beginIndex) ) AS ?test ) .
7   FILTER (str(?test) != str(?anchorOf) ) . }

```

However, this test is incorrect for what it tries to test. As specified in [9], SPARQL treats the first character of a string as 1, while NIF counts offsets and thus begins with zero. The test has to correct for this and increment the `?beginIndex` in the `SUBSTR` function. During testing of an early version of the Wikipedia abstracts corpus, this test failed sometimes, because the Virtuoso 7 implementation of the `SUBSTR` could not return a valid result for strings of length zero. Thus, constraint 9 was introduced and incorporated into an own test case. To keep test 1 from failing because of insufficient string length, a filter condition was added. However, during comparative in-memory and SPARQL-endpoint validation (see Section 5.3), another error was discovered. Substrings in Virtuoso 7 SPARQL queries must not be longer than the length of the string in question. So for the string "My favourite actress is Natalie Portman" with a length of 39 characters and trying to select the string "Portman" with offsets 32 and 39 using the function

```

1 SUBSTR("My favourite actress is Natalie Portman",33,8)

```

with a string length that's too long results in an error using Virtuoso 7. The SPARQL implementation of the RIOT framework used by RDFUnit for in-memory validation of dataset files is robust against this error. Robustness in this case means, that a substring that is "correct" will be returned, even if its `nif:endIndex` is larger than the length of the string. So one could not find it using the data file validation. The datasets tested in [13] also did not contain this error, but in the Wikipedia abstracts corpus, there were 17 strings that had `nif:endIndexes` larger than the string length of the `nif:referenceContext`.

Accounting for these errors, text case 1 was extended and two additional test cases introduced:

```

1 SELECT DISTINCT ?s WHERE {
2   ?s nif:anchorOf ?anchorOf ; nif:beginIndex ?beginIndex ;
3     nif:endIndex ?endIndex ;
4     nif:referenceContext ?context .
5   ?context nif:isString ?referenceString.
6   FILTER(xsd:integer(?endIndex) <= xsd:integer(?contextEnd))
7   FILTER((xsd:integer(?endIndex) - xsd:integer(?beginIndex)) > 0)
8   FILTER (substr(?isString, xsd:integer(?beginIndex)+1 , (xsd:integer(?
    endIndex) - xsd:integer(?beginIndex)) ) != str(?anchorOf)) .}

```

The `FILTER` in Line 6 filters case 10, the `FILTER` in Line 7 filters case 9. The `BIND` statement was abolished and instead included in the `FILTER` because `BIND` is executed at the end of the query, before the filtering. To be able to filter the results that are then evaluated using the problematic `SUBSTR` function, `BIND` could not be used because it returned the mentioned `BAD SUBRANGE` error.

Finally, the new tests could be implemented. To test case 9, the `nif:beginIndex` and `nif:endIndex` of every `nif:String` including subclasses are selected and subtracted to test if they describe a string of length zero. Type casts to `xsd:integer` are necessary, because Virtuoso may misinterpret the values as strings, thus making comparison fail.

```

1 SELECT DISTINCT ?s WHERE {
2   ?resource rdf:type/rdfs:subClassOf* nif:String ;
3   nif:beginIndex ?beginIndex ;
4   nif:endIndex ?endIndex .
5   FILTER((xsd:integer(?endIndex) - xsd:integer(?beginIndex))=0)}

```

To test case 10, `nif:endIndex` and `nif:referenceContext` of a resource are selected. The respective `nif:Context` resource's `nif:endIndex` is queried and tested against the other resources `nif:endIndex`. If the context `nif:endIndex` is smaller than the `nif:endIndex` of the referring resource, the erroneous resource is part of the results.

```

1 SELECT DISTINCT ?s WHERE {
2   ?resource nif:endIndex ?endIndex ;
3   nif:referenceContext ?context .
4   FILTER EXISTS {
5     ?context nif:endIndex ?contextEnd .
6     FILTER(xsd:integer(?contextEnd)<xsd:integer(?endIndex))
7   }

```

5.3 Validation of collected corpora

For each corpus, correctness and accordance to the NIF format had to be ensured. For this purpose, RDFUnit was used to iteratively validate the corpora, analyze the errors, pinpoint the source of error in the conversion algorithms, regenerate the data and validate it again until no errors are found. This paradigm is called "test-driven evaluation of linked data quality" [14] but may also be called test-driven data development in the context of this work, as the data contained in the corpora is actively shaped and developed in a process that is motivated by test outputs. Contrary to the evaluation of an already established data source on the web, the data being tested in this work is transient as long it does not test correctly. This is procedurally analogous to test-driven development, where tests are written first and their failure to execute successfully motivates the implementation of new methods, guaranteeing the accordance of software features to already formalized standards. Similarly, tests that formalize NIF already exist, as described in 5.2 and their failure to execute successfully can be used to correct errors in conversion algorithms. Table 3 presents a complete replication and expansion of the evaluation of corpora presented in [13], adding 4 corpora. Please note that the table does not present the final state of evaluation but a snapshot of a late development stage. The final corpora all evaluate without error.

Methodology RDFUnit was run for every single corpus. The small corpora KORE, Spotlight, and the N3 corpora were validated using the in-memory validation of dump files RDFUnit provides. The larger corpora converted for this work were validated with both, the in-memory validation as well as the SPARQL endpoint validation. For this purpose, the datasets were loaded in an open-source edition of Virtuoso server (version 7.0)⁵⁰. The results of the dataset evaluation are provided in table 3. Table 4

Due to its size, only one of the 109 Wikilinks corpus dump files was loaded and validated. This can be justified by the even distribution of errors throughout the corpus, as it is very heterogeneous in nature and does not contain any topical clusters. Furthermore, the corpus was converted using a custom algorithm, so any systematic error of the conversion algorithm can also be identified using only a subset of converted data. The design of NIF as a general conversion and interoperability format is robust against any smaller fluctuations, such as string content of the converted documents, making a complete validation a nice feature of the corpus but not strictly

⁵⁰<http://virtuoso.openlinksw.com>

Dataset	Size	SC	FL	ER	AEErrors	MEErrors
Wikipedia abstracts	276M	208	1	2	0	17
Tiger	17.4M	209	0	0	0	0
Brown	13.1M	207	2	0	0	1,196,783
Wikilinks	4.9M	208	1	0	0	85,804
News-100	13K	209	0	0	0	0
RSS-500	10K	209	0	0	0	0
Reuters-128	7K	209	0	0	0	0
Spotlight	3K	207	2	0	0	70
KORE50	2K	199	10	0	501	0
KORE50 corrected	2K	209	0	0	0	0

Table 3: Overview of the NLP datasets test execution. Columns denote size in triples count, the number of successful test cases (SC), failed (FL) and did not complete due to an error (ER). Additionally, the total number of the individual violations from automated test cases (AEErrors) and manual test cases (MEErrors).

necessary. In contrast, the complete Wikipedia abstracts corpus was validated to function as a scalability benchmark.

Contrary to the validation presented in [13], the errors of the test case checking if `nif:beginIndex` and `nif:endIndex` have `rdfs:range xsd:nonNegativeInteger` were excluded from the results. These errors are the result of Virtuoso not correctly implementing this datatype, so tests testing for it always fail in Virtuoso triple stores. Using the in-memory validation, the tests don't fail. Therefore, I deem these results not meaningful and exclude them from the table to not obscure analysis of more interesting cases.

Error analysis Excluding KORE50 that was based on a former version of NIF and thus had systematic errors of a different nature, as explained in 4.5, the main source of error was a violation of constraint 1.⁵¹ This was the case for the Brown corpus, the Wikilinks corpus and the Spotlight corpus, for different reasons. In case of the Brown and the Wikilinks, quotation marks in the text were not properly treated. Generally, `nif:anchorOf` strings are encased by triple quotation marks to enable arbitrary strings in the literals. Yet, in case of the tested version of the Brown corpus, quotation marks were escaped by `\` to ensure a valid turtle string. This results in a string that is, in turtle, per quotation mark one character longer, due to the escaping `\` being counted as character. When one imports such a turtle file into Virtuoso, the escape characters are removed, resulting in shorter strings that don't match the test. Besides violating constraint 1,

⁵¹As defined in 5.2: a `nif:anchorOf` string must match the substring of the `nif:isString` from `nif:beginIndex` to `nif:endIndex`

Dataset	Size	Dump Eval	Endpoint Eval	
		Time	Import time	Eval time
Wikipedia abstracts	276M	427m 7.60s	19m 44s	42m 7.02s
Tiger	17.4M	16m 50.525s	1m 12s	16m 59.561s
Brown	13.1M	16m 43.68s	40s	16m 23.238s
Wikilinks	4.9M	14m 40.920s	36s	15m 16.363
News-100	13K	22.74s	652 ms	15m 3.70s

Table 4: Overview of the NLP datasets validation time

it also violates constraint 4.⁵² The solution for Brown was, not to escape the quotation marks, as they can be contained in triple quotation marks.

Wikilinks presents a special case, because its size as well as data source amount to completely arbitrary and highly heterogeneous strings that have to be represented. In some cases, the literals end in a quotation mark, resulting in 4 quotation marks terminating the literal, which causes turtle validators like rapper to throw a **syntax error**. In other cases, the strings will contain triple quotation marks, prompting the same error. The string may contain escape characters as well, serving as another source of errors, if they end the strings and thus escape one of the enclosing quotation marks. This problem is hard to address and is impossible to resolve without changing the original string. The solution selected was removing all instances of `\` and replacing all instances of quotation marks with double `'`.

The Spotlight corpus showed errors in the same text, that showed a systematic failure of the conversion algorithm to count the last period of a sentence as character, producing `nif:Sentence` resources with a `nif:endIndex` one character too small. The error was manually corrected, because the problem was not extensive enough to merit a scripted correction and the source code was not available.

The Wikipedia Abstracts corpus produced the most interesting errors so far. The failure of the test for constraint 1 to execute, prompted by the error of the Virtuoso `SUBSTR` function described in 5.2, was analyzed and resulted in discovery of constraints 9⁵³ and 10⁵⁴. It is notable that the very small number of cases affected ($6.152 \times 10^{-6}\%$) could never have been discovered using manual testing methods or other, less formalized testing procedures.

Another example is the discovery of the `<kbd>` element in the abstract texts, that was already mentioned in 4.3. It has pure styling functionality, yet was dropped due to SAX-parser being configured to skip non-link elements by default. This resulted in `Link` resources that again had strings of length 0, thus presenting no meaningful NIF string that could be discovered using `RDFUnit`.

All error sources, empty strings as well as strings whose `nif:endIndex` is larger than the one of the `nif:referenceContext`, were addressed by corrections of the conversion algorithm.

Because lots of datasets of different sizes were available now, the chance was used to evaluate

⁵²the length of `nif:isString` inside a `nif:Context` must be equal to the resource's `nif:endIndex`

⁵³the length of a string must not be zero

⁵⁴the `nif:endIndex` of a `nif:String` must not be larger than the `nif:endIndex` of the `nif:referenceContext`

the runtime of RDFUnit. The results, as presented in table 4 can be used to derive general hints on when to use the in-memory validation and when a local SPARQL endpoint is faster. Runtime values are direct results of the previous validation. Generally, in-memory validation of dataset dumps is a quick and easy way to use RDFUnit. No other software needs to be installed and set up, the software is easily build and executed on a specific dataset file with the NIF schema. Larger datasets profit from a triple store, like the massive difference between runtimes of the Wikipedia abstract corpus validation shows - in-memory validation takes 7 times longer than the combined triple store evaluation. Incidentally, the selection of tested datasets seems to contain the turning point between the two ways of validation, i.e. the point where in-memory validation and combined SPARQL validation time (import+validation) take the same time to execute. Datasets larger than the Tiger corpus will most likely be quicker to validate using a database.

However, databases provide additional benefits over data dump evaluation. They allow to easily run the testing SPARQL query against the data to specifically check which resources are faulty for which reasons. Alternatives for dump files are rather sparse, containing tools like Twinkle⁵⁵ or Jena ARQ Command line⁵⁶, running `grep` over the data for various patterns or inspecting the datasets manually using editors, like `vim`. However, tools to statically query RDF files are slow for repeated queries on larger datasets, because they have to parse the data for every query. Still, for many small datasets, debugging the file directly is preferable.

Another benefit of direct validation is the correctness of testing. It was already mentioned that most errors found in [13] resulted from the datatype `xsd:nonNegativeInteger` not being correctly implemented in Virtuoso. These errors are absent in direct validation. Some automatically generated test cases' queries also crashed Virtuoso by somehow filling the complete memory (16GB) of the test machine. These were cases using a `Group by()` `Having()` construct for filtering results. Because the query itself is relatively general in these cases, it seems that Virtuoso tried to load the complete result set into memory, prompting a crash in case of the very large Wikipedia abstracts dataset. This error was absent in direct validation as well, because it was done one file at a time, thus lowering the memory footprint of these queries.

6 Implementation and scalability of NIF tools

6.1 OpenNLP NIF Wrapper

The implementation of the OpenNLP wrapper follows the reference wrapper implementation of the Stanford Core NLP tools⁵⁷ using Java and Apache Jena. It allows annotating texts using OpenNLP models, generating NIF output. Sentence boundary detection, tokenization and part of speech tagging are implemented and can be applied to 6 languages (Danish, German, English, Dutch, Portuguese and Swedish). In accordance to the reference implementation, the wrapper can be called via the command line to annotate text input locally, or set up as a web service. Furthermore, the wrapper class serves as Java API to generally add OpenNLP spans to NIF Jena models, providing NIF RDF output to developers using OpenNLP.

General Architecture and parameters In general, NIF wrappers consist of three Java classes. One class represents the wrapper itself and offers methods for textual input, annotation using the respective NLP tool's functions, like tokenization or POS-tagging, and conversion of the annotated

⁵⁵A GUI tool to SPARQL query RDF files: <http://www.ldodds.com/projects/twinkle/>

⁵⁶A CLI query, parse and process RDF files and endpoints: <https://jena.apache.org/documentation/query/cmds.html>

⁵⁷`src/main/java/org/nlp2rdf/implementation/stanfordcorenlp`

text into NIF. This wrapper class is usually called either via a CLI class, that allows use of the wrapper via the command line of a Unix terminal and offers a number of NIF parameters, or via a web service class, that can be used to deploy the wrapper as a web service implementing the same parameters.

A core set of NIF parameters is specified in the NIF public API specification⁵⁸. It contains a set of mandatory parameters every NIF implementation must implement. Table 5 lists the core, as well as the OpenNLP specific parameters. Additional parameters can be implemented according to the needs of the specific implementations. In case of the OpenNLP wrapper, such additional parameters are `modelFolder`, which describes the location of the pretrained statistical models OpenNLP needs to operate, and `language`, which describes the input language and thereby serves to choose the correct pretrained models in the specified location. The language parameter accepts a 2 letter ISO 639-1 code, in compliance with the models OpenNLP distributes⁵⁹. However, a more granular and interoperable approach is needed for language specification. Language URIs, for example using `lexvo`⁶⁰, are preferable to reduce ambiguousness of language identification.

Parameter	shorthand	function
NIF core parameters		
input	-i	Input depending on informat and intype, usually a string to be annotated
informat	-f	Format of the input, usually text or turtle
intype	-t	The type of the input; direct, url or file (only CLI)
outformat	-o	The RDF serialization format of the output; default turtle
urischeme	-u	The URI scheme NIF should use; default RFC5147String
prefix	-p	The URI prefix of the NIF resources created in the annotation process
OpenNLP wrapper specific parameters		
language	-	Language of the pretrained OpenNLP models to choose as ISO 639-1 (2 letters) language code
modelFolder	-	folder the pretrained OpenNLP models are found in

Table 5: NIF core parameters and extensions for OpenNLP

The OpenNLPWrapper class is instantiated with three mandatory arguments. The NIF context resource as Jena Individual, an instance of NIFParameters holding the aforementioned parameters and a Jena OntModel that is later filled with the annotated resources. The constructor of the

⁵⁸<http://persistence.uni-leipzig.org/nlp2rdf/specification/api.html>

⁵⁹<http://opennlp.sourceforge.net/models-1.5/>

⁶⁰<http://www.lexvo.org/>

wrapper makes sure the parameters are complete and correct, checking for the existence of the model folder as well as the model files according to the language parameter.

The main method of the wrapper is `processText()`, that also expects the context resource as Jena Individual as well as the NIFParameters. The method handles the annotations and fills the Jena OntModel with the annotated resources. First, the sentences are annotated using the sentence boundary detection model. The model is loaded and run on the input text using `sentPosDetect()`, producing an array of OpenNLP `Span` objects, which contain start and end offsets in relation to a covered text.

These are converted into a List of Jena Individuals using the method `addSpans()`. `addSpans()` simply iterates over the array of `Spans`, creating new NIF resources using their start and end offsets to extract the anchor text from the document text, adding the essential NIF properties like `nif:beginIndex`, `nif:endIndex`, `nif:referenceContext`, `nif:anchorOf` as well as the ontology class of the resource that is referenced via one argument of the method. In this way, `addSpans()` provides an easy way to generally convert OpenNLP `Span` objects, which are ubiquitous in OpenNLP, into NIF resources of a Jena model and thus allows OpenNLP developers to use the wrapper class as a NIF API.

After sentence detection, tokenization and POS tagging are executed in a similar way. The tokenization and pos tagging models are loaded and used to instantiate OpenNLP `POSTagger` and `Tokenizer` classes. Then the list of sentence individuals previously created is iterated over. The sentence strings are used as input for the tokenizer and POS tagger, annotating the individual words of the sentences. Again, a list of Jena Individuals is created using `addSpans()`, representing the individual `Word` resources. Each list of words is again iterated to add the POS tags. Because all Jena Individuals are part of the Jena OntModel, it is filled now and can be processed further or put out.

Usage via command line interface The wrapper can be used via the command line, using the `OpenNLPWrapperCLI` class. This class only contains a `main()` method and serves to accept and parse parameters, check their validity and accept and preprocess the input text. A NIF ParameterParser is used to parse the input parameters. Besides the standard parameters, implementation specific parameters like `modelFolder` and `language` are added here. If any mandatory parameters are missing, the parser either logs an error or tries to use a default value.

Using the input, a Jena OntModel is created that either contains the existing NIF data in case of turtle input, or an initial `nif:Context` resource itself containing the document text. The algorithm then iterates over the `nif:Context` resources, initializing a new Annotator for each and executing the annotators `processText()` method. After annotation, the model is written to disk in the chosen format or simply put out on the console.

Usage as web service The `OpenNLPWS` class can be used to deploy the annotator as a RESTful web service. The class extends the standard `NIFServlet` class which in turn extends the Java `HttpServlet` class. `NIFServlet` provides the necessary `doGet()` and `doPost()` methods to handle HTTP requests as well as returning the annotated text. So the OpenNLP implementation only has to overwrite the `init()` method to add the specific necessary input parameters and the `execute()` method that handles the annotation. The `execute()` method itself works like the `main()` of the CLI class, except not having to handle parameter parsing. The web service can be easily deployed using the Maven Jetty plugin⁶¹ and the command `mvn jetty:run`.

⁶¹<http://www.eclipse.org/jetty/>

6.2 OpenNLP NIF parser

Besides wrapping OpenNLP annotators for them to produce NIF output, NIF input into OpenNLP is another use case that has a number of interesting applications. For one, it OpenNLP to be easily trained on arbitrary NIF corpus data and thus allows a wide range of possible inputs without conversion problems. At the time of writing, the OpenNLP framework solves this problem by providing a number of corpus converters for different corpus formats. The overhead produced by the need of writing a converter for every single corpus format can be reduced by using a pivot format, as stated in 1.3. Besides training, working with annotations created by other NLP tools in OpenNLP is another interesting use case at the core of NIF functionality. Finally, a NIF parser in general is an important missing step towards a full round-trip that shows the feasibility of NIF as an input format for NLP processing.

NIF parser implementation The NIF parsers are implemented according to OpenNLP code conventions and integrate into the OpenNLP framework. Three parsers have been created, able to parse sentences into `SentenceSamples`, named entity annotations into `NameSamples` and POS tags into `POSSamples` from NIF sources. All parsers employ a `factory` class that serves for parser initialization and extends native OpenNLP `StreamFactories`. These factories implement the NIF API parameters, parse them and initialize the Jena `OntModel`. They also read the input NIF files and create the specific NIF parsers.

All NIF OpenNLP parsers are OpenNLP `ObjectStreams`. However, as the RDF model is a graph and thus cannot simply be streamed, the whole model is parsed into memory first and the specific resources are simply read from the in-memory model. Although this conforms with the OpenNLP conventions, a true streaming solution would be preferable to have a smaller impact on memory consumption and scale better. Resources streamed are in each case of type `nif:Sentence`, because besides sentences themselves, OpenNLP `POSSample` and `NameSample` objects expect annotated sentences.

NIF parser usage example A nice example according to the training use case is reading OpenNLP `SentenceSamples` from a NIF file and using them to train a model:

```
1 //create a JENA model
2 Model model = ModelFactory.createDefaultModel();
3 //read the NIF file
4 model.read(new BufferedReader(new FileReader("corpusFile")), null, "TURTLE");
5 //create a new stream that reads the all sentences from context resources
6 TypedRDFResourceStream res = new TypedRDFResourceStream(model, NIFOntClasses.
7     Context.getOntClass(model));
8 //create a SampleStream from the NIF corpus
9 ObjectStream<SentenceSample> sampleStream = new NIFSentenceSampleStream(res);
10 //train the model
11 char[] eos = ".?!".toCharArray();
12 SentenceDetectorFactory sdFactory = SentenceDetectorFactory.create(null, "en",
13     true, null, eos);
14 SentenceModel sentmodel = SentenceDetectorME.train("en", sampleStream, sdFactory,
15     null);
```

Evaluation To assess the performance of the Parser, the NIF version of the Tiger corpus converted in 4.2 was parsed via the OpenNLP parser, producing a list of `SentenceSamples` that were then simply put on via `System.out`. The results can be seen in Figures 6.1 and 6.2. Figures were created using Java VisualVM⁶². In all evaluation figures using VisualVM, the used heap is shown in the lower part of the diagram, confined by the lower line, while the maximum heap

⁶²<http://visualvm.java.net/>

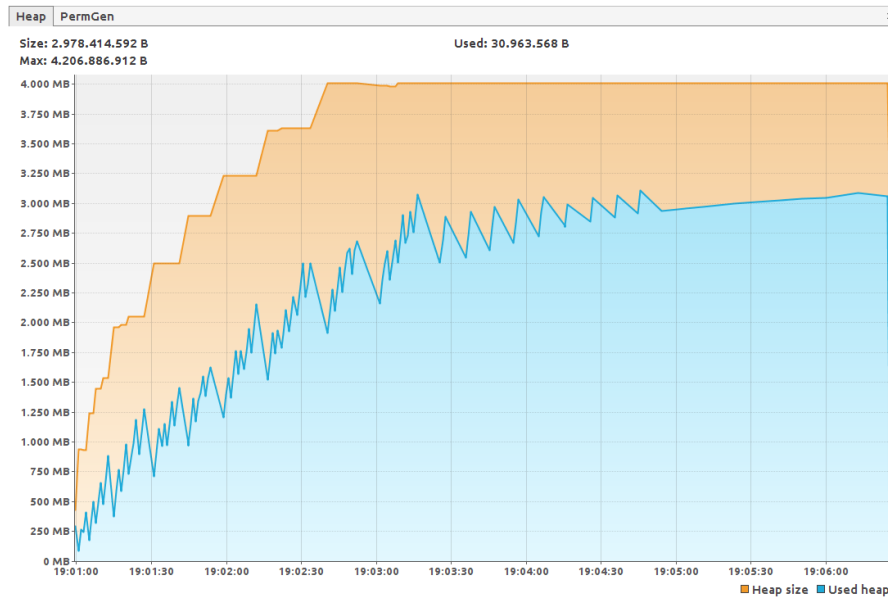


Figure 6.1: Heap size diagram of parsing sentences from the NIF TIGER corpus - Standard Jena model, 4 GB memory - finished in 5:30m

size is shown in the top part, confined by the upper line. The first test seen in figure 6.1 was started using the standard JAVA VM settings with 4 GB of memory. You can see that, after 2 minutes and 30 seconds, the HeapSpace consumption pattern changes, as the garbage collection activity is raised due to a maxed out heap. This increase garbage collection activity significantly decreases runtime. A second test (figure 6.2) was made, increasing the maximum HeapSize to 6 GB. Accordingly, the memory consumption pattern stayed stable and the whole corpus was parsed in half the time.

Taking into account that the NIF version of the Tiger corpus is one rather large corpus file, featuring 17,428,613 triples with 50,472 sentences in one very large context resource, the runtime is very appropriate for regular use. However, memory consumption is a large hindrance to scalability. While the original Tiger ConLL corpus was 60 MB in size, the NIF conversion already was 917.8 MB, over 15 times larger. Parsing it into a Jena model increased the memory needed to at least 3857.8 MB. This suggests that NIF corpora can only be feasibly used for small corpora and that the convenience of using Jena models in corpus conversion and consumption creates prohibitive memory consumption for work with larger corpora, like the Wikipedia Abstract corpus converted in 4.3.

It has to be noted that in these experiments, the whole RDF graph is parsed into memory. This serves as a convenient implementation also used in the reference NIF implementation of the Stanford Core NLP annotation. Most resources and small texts should be able to be parsed using this procedure. For larger corpora *Apache Jena TDB* can serve as a way to keep the Jena dependence and dramatically decrease runtime while reducing memory consumption by preprocessing, indexing and more specific queries.

Apache Jena TDB⁶³ is a RDF storage and query component for Jena in form of an indexed

⁶³<http://jena.apache.org/documentation/tdb/>

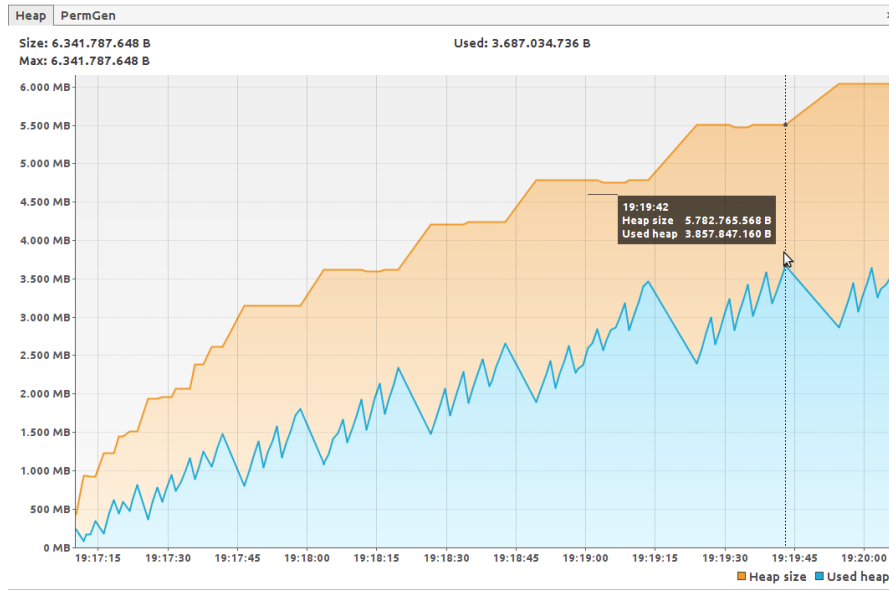


Figure 6.2: Heap size diagram of parsing sentences from the NIF TIGER corpus - Standard Jena model, 6 GB memory - finished in 2:49m

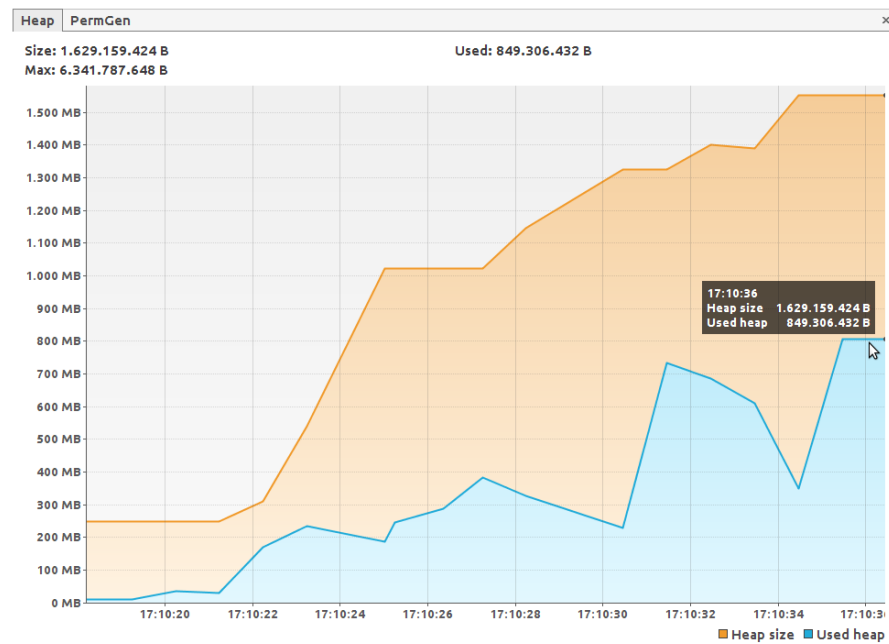


Figure 6.3: Heap size diagram of parsing sentences from the NIF TIGER corpus - TDB version - finished in 0:21m

RDF store materialized in the file system. It is not a fully featured triple store. The advantage of TDB is that, for single user, single process access to a dataset, it provides a quick way to use an indexed version of a dataset, thus providing reduced query times and less overhead than in-memory representations of RDF graphs. The disadvantage compared to triple stores like virtuoso is, that it does not provide ACID⁶⁴.

Creating a TDB dataset from a NIF corpus can be done via the command line tool `tdbloader` that comes with Jena TDB. The TDB representation of the NIF Tiger corpus is 1.6 GB large. To read the sentences from this version of the dataset, only a specific query for resources that have the type `nif:Sentence` has to be done, so only part of the corpus is consumed. This shows in a vastly improved runtime of 21 seconds and less then half the memory consumed (1629 MB). The diagram for this test can be seen in figure 6.3.

6.2.1 Creating NIF annotations via OpenNLP

Using NIF for its designated purpose of converting NLP tool output into RDF, this section will evaluate the performance of the implemented OpenNLP wrapper in comparison to the existing reference implementation of Stanford Core NLP tools. Testing was done using a 6.5 MB file containing English text for the first test and a 13 MB file containing the same text twice. There were 25902 and 51804 sentences respectively. Both annotators were set to detect sentence boundaries, tokenize and annotate part of speech. See table 6 for a comparative overview of the results.

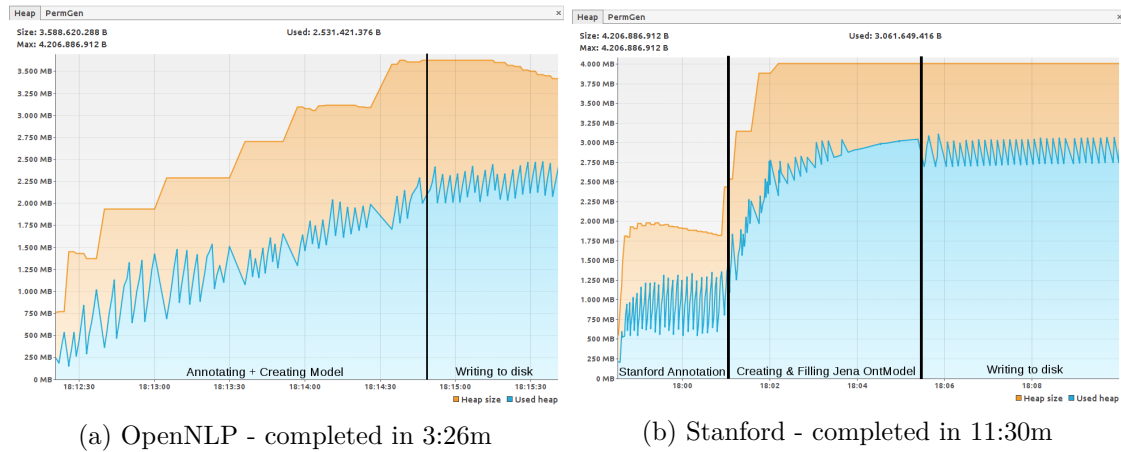
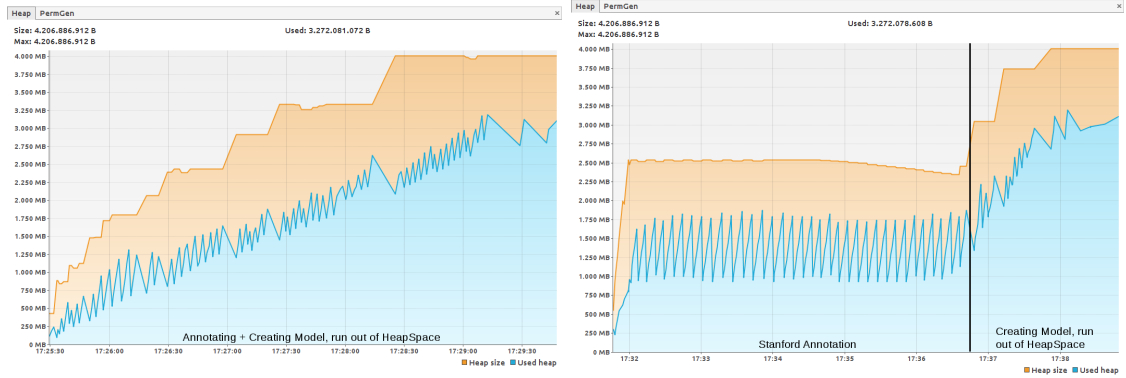


Figure 6.4: Annotation using OpenNLP and Stanford Core NLP for sentence detection, tokenization and POS tagging - 6.5 MB corpus size

⁶⁴corruption may occur when using multiple JVMs



(a) OpenNLP - 4GB HeapSpace full after 4:20m (b) Stanford - 4GB HeapSpace full after 7:05m

Figure 6.5: Annotation using OpenNLP and Stanford Core NLP for sentence detection, tokenization and POS tagging - 13 MB corpus size

Both tools successfully annotate the small test file. The large difference in runtime is due to the different runtime of the annotators and can not be attributed to wrapper implementation. File size and number of triples are different due to the OpenNLP implementation not adding `nif:nextWord` and `nif:previousWord` properties. Trying to annotate the larger test file caused the annotators to crash because of a full HeapSpace. Figures 6.4 and 6.5 show a closer examination via VisualVM.

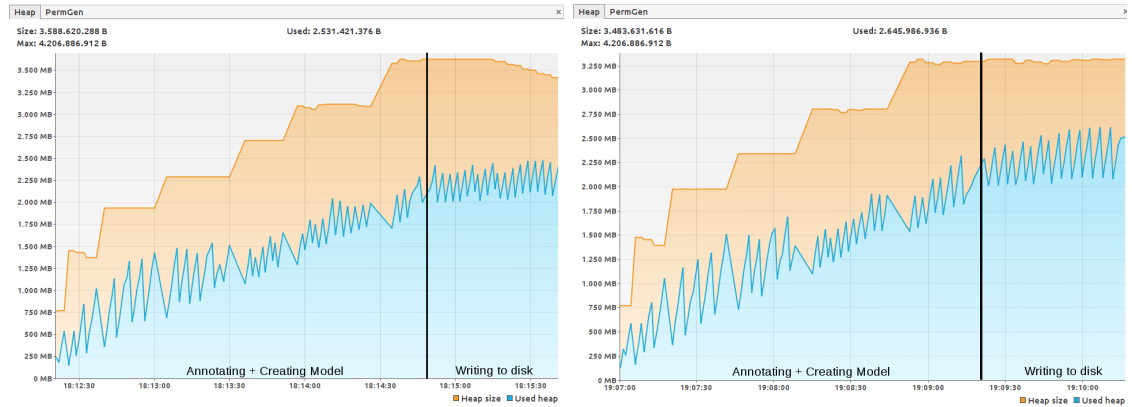
	OpenNLP	Stanford Core NLP
6.5 MB test file		
Runtime	3:26m	11:30m
Triples produced	13,772,917	17,282,490
Filesize	760 MB	1084 MB
13 MB test file		
Runtime	4:20m	7:05m
Triples produced	- (HeapSpace full)	- (HeapSpace full)
Filesize	- (HeapSpace full)	- (HeapSpace full)

Table 6: Results of annotation tests of OpenNLP and Stanford Core NLP wrappers

The figure first and foremost shows the different ways the annotators use to create and populate the Jena models. The Stanford Core NLP (figure 6.4b) implementation does the annotation step first. In this step, the memory consumption is stable and, other than total runtime, shows no significant difference between the small and the large file. The wrapper then populates the Jena model, shown by the significant increase in memory consumption, before writing it to disk. On the other hand, the OpenNLP wrapper annotates and adds to the model sentence by

sentence before writing the complete model to disk, shown by the relatively smooth increase in memory consumption (figure 6.4a). Due to the large memory overhead already diagnosed in the previous section, both approaches run out of HeapSpace during the larger test. Both implementations use a Jena `OntModel`. This is a reasonable choice as it enables inference on the annotations and provides a complete serialization of the model, including class and property definition axioms in the final RDF. However, these capabilities might come with increased memory consumption further complicating the task. To test this, the OpenNLP wrapper was rewritten to use a standard Jena `RDF Model` instead and then again tested for the small and large files. See figure 6.6. The comparison shows that there is no significant difference between the two implementations regarding memory consumption or runtime.

A solution for the lacking scalability in regard to file size would be a stream based approach, parsing and annotating sentence by sentence, writing the annotated resources to disk until the corpus is fully consumed. This procedure was employed with success for the Wikipedia Abstract corpus. However, it is hard to realize for texts that can not be divided into smaller logical units and thus have only one large `nif:Context` resource. As the length of the Context string is unclear until the file is fully consumed with a stream based approach, the end offset of the Context resource stays unknown as well. As long as the end offset is unknown, so is the final URI of the Context resource. However, all other strings, like sentences or words, refer to that resource via `nif:referenceContext`, which is not possible with knowing the URI. This means that, after having written the resources to disk and being done with the Context resource, the other string resources would have to be read again, adding a correct `nif:referenceCorpus` property with the Context URI as object. It is very likely that this procedure would only make sense for large corpora, which are a minority of the use cases at the moment. For this reason, the in-memory variant was kept.



(a) OpenNLP - 6.5 MB file, Jena `OntModel` - completed in 3:26m (b) OpenNLP - 6.5 MB file, Jena `RDFModel` - completed in 3:21m

Figure 6.6: Comparison between OpenNLP annotation memory consumption and runtime using Jena `OntModel` and Jena `RDFModel`

6.2.2 Round-trip combining both resources

To demonstrate the feasibility of a full round-trip from text to NIF annotations and back to text, the Wrapper and the Parser were used in a combined JUnit test. It first reads a test document⁶⁵ into a string using a StringBuilder. This string will serve as the "original" which the end result will be tested against. The test then runs the OpenNLP wrapper with the English sentence boundary detection model, transforming the string into a NIF model. This model is then written into another String in the turtle format via a StringWriter. This serves as input for the Parser, that reads the NIF turtle string, extracts the sentences' `nif:isString` property and concatenates them over the set of sentences. This serves as our output string that is finally compared to the input. The test results in two equal strings, proving the correctness of our round-trip and in turn the correctness of both, the annotation and the parse of the annotated strings.

7 Interoperability and In-Use

7.1 The Unicode Text Segmentation use case

The Unicode Localization Interoperability Technical Committee⁶⁶ (ULI) is an institution that aims to ensure the interoperable interchange of interoperable localization assets. Among these assets are *segmentation rules*, rules that define how to segment text into smaller logical units, such as words and sentences. These rules are then published as Unicode Standard Annexes making them part of the Unicode standard. The Unicode Standard Annex 29 [6], section 5 presents the following rules for sentence boundary segmentation that are valid for most Indo-European languages.

1. Break at the start and end of text
2. Do not break between carriage return and line feed. These are control characters that mark up line breaks. Different combinations are used in different operating systems, with both of them being used together in Windows.
3. Break after paragraph separators, like carriage return, line feed or the paragraph separator character.
4. Don't break after terminators like the period, if they are directly followed by a number of lower case letters, a number or if they are between upper case letters. These might be abbreviations. Don't break after terminators like the period, if they are followed by a comma, a colon or semicolon.
5. Break after sentence terminators, such as full stops, question marks and exclamation marks. An exhaustive list of what Unicode characters are sentence terminators is found in an auxiliary file to the Annex document⁶⁷.
6. Don't break otherwise.

While a large number of cases is covered by these general rules, the ULI committee started to collect exceptions from these rules, i.e. tokens that, if found at a sentence boundary, signify that this boundary was most likely erroneously detected. In most cases, these tokens are abbreviations,

⁶⁵English, one sentence per line, no special formatting

⁶⁶<http://uli.unicode.org>

⁶⁷<http://www.unicode.org/Public/UCD/latest/ucd/auxiliary/SentenceBreakProperty.txt>

defines as a number of characters ending on a sentence terminator, thus leading to incorrect sentence boundaries. The following listing shows an example of such an exception⁶⁸.

```
1 The quick brown Mr. | Fox jumped over the lazy dog. |
```

Listing 7: Example of erroneous sentence boundary detection. Boundaries marked up with red pipes.

In the example, *Mr.* is found to end the sentence, according to the rules above. The issue is with rule 4, that does not break the sentence at some abbreviations like *Y.M.C.A* or *i.e.* and numbers, like *1.000*, but not strings like *Mr.*, *Ph.D.* or *St.*. Special cases like company names ending with sentence terminators, such as *yahoo!* are also not included. This motivated the collection of lists of abbreviations, defined as strings of characters ending with a sentence terminator, as described. However, because Unicode is an international standard, these lists should be collected for as many languages as possible. Linked Open Data and especially the DBpedia, containing structured knowledge extracted from 119 Wikipedia language versions, provides a very appropriate, accessible and open source of this data.

The task therefore was to (1) extract multilingual abbreviation lists from DBpedia and (2) test the extracted abbreviations against a corpus to find out if the precision of the sentence boundary detection algorithm can be increased using these abbreviations.

7.2 Obtaining abbreviation lists from LOD

Abbreviation lists were obtained by using the open and multilingual knowledge base DBpedia. There is a number of advantages this approach has in favour of pre-compiled lists or using classical dictionaries as only data source:

- DBpedia as a resource derived from Wikipedia is a free and open data source that leverages the crowd as a means to ensure comprehensiveness and quality of the data
- Using those resources provides a solid base of relevant (as per Wikipedia standards) data for free
- The approach is inherently multilingual, as the DBpedia exists for 119 languages
- Additional information besides the abbreviation string is provided and can be used for disambiguation

The data flow of the extraction process is shown in 7.1

In the DBpedia, like in the Wikipedia, *redirect* links serve to link alternative names of resources to a *meaning* resource or to a *disambiguation* resource, that links different meanings of the string in question. *Meaning* resources can be understood as those that directly contain more information about the abbreviation, like its full entry name (that is the full string it abbreviates) in form of its `rdfs:label`; its type, represented by `rdf:type`; its Wikipedia categories (`dcterms:subject`) as well as its equivalent in other languages (`owl:sameAs`). *Disambiguation* resources contain a number of `http://dbpedia.org/ontology/wikiPageDisambiguates` properties that in turn link meaning resources.

To extract abbreviations from the DBpedia, the redirects for each language were downloaded in a dumpfile⁶⁹ and triples whose subject ends with a sentence terminator character (period, an

⁶⁸Segmentation according to the demo implementation here: <http://unicode.org/cldr/utility/breaks.jsp>

⁶⁹like http://downloads.dbpedia.org/3.9/en/redirects_en.ttl.bz2 for the English language

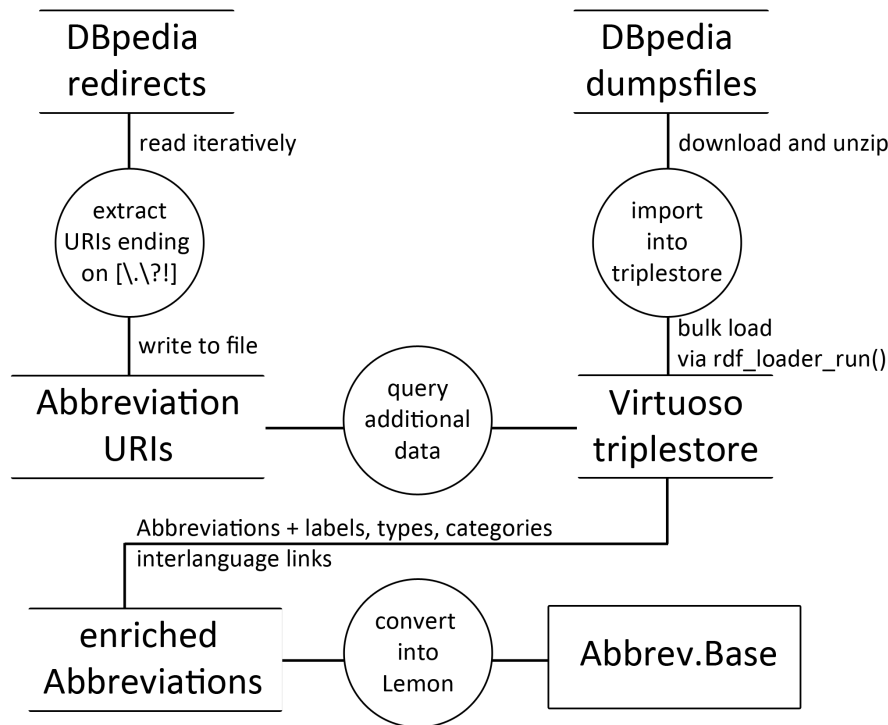


Figure 7.1: Data flow diagram showing the extraction of rich abbreviation data from DBpedia

exclamation or question mark) were extracted into a single file using a simple regular expression. The resulting file consisted (for the English language) of 46625 lines of this form:

```

1 <http://dbpedia.org/resource/A.I.> <http://dbpedia.org/ontology/
  wikiPageRedirects> <http://dbpedia.org/resource/AI>.

```

The abbreviation resource itself is the subject, a resource either containing a disambiguation resource or a meaning resource as the object of the triple. A number of the resulting resources had to be discarded to not unnecessarily enlarge the final list of abbreviations. For example, most strings containing one or more whitespaces are proper names like “Apple Computer Co.” or “Monsters Inc.” were discarded. Only the last part of the string is a valid abbreviation and often already covered in itself. Resources ending with ellipsis were also discarded.

To obtain more data on the abbreviation, additional DBpedia data for each language was imported into a local Virtuoso triple store. Official DBpedia endpoints were not used, because only a fraction of languages for which DBpedia data exists have own endpoints. Each abbreviation resource was then queried via the Virtuoso SPARQL endpoint to either get the information of the linked meaning resource, or all URIs the disambiguation resource links, to get their meanings and associated information in a second step. The abbreviations, the full names and abbreviation URIs were then compiled in .tsv files. Furthermore, every language’s abbreviations were converted into a lexicon according to the Lemon Ontology. An example of this modeling can be found in listing 8.

```

1 <http://nlp.dbpedia.org/abbrevbase/lexicon/en>
2   a lemon:Lexicon ;
3   lemon:language "en" ;
4   lemon:entry <http://nlp.dbpedia.org/abbrevbase/lexicon/en/entry/A.I.> ,
5     [...] .

7 <http://nlp.dbpedia.org/abbrevbase/lexicon/en/entry/A.I.>
8   lemon:form <http://nlp.dbpedia.org/abbrevbase/lexicon/en/entry/A.I.#form> ;
9   lemon:sense <http://nlp.dbpedia.org/abbrevbase/lexicon/en/entry/A.I.#sense1>,
10     <http://nlp.dbpedia.org/abbrevbase/lexicon/en/entry/A.I.#sense2>,
11     [...]
12   a lemon:LexicalEntry .

14 <http://nlp.dbpedia.org/abbrevbase/lexicon/en/entry/A.I.#form>
15   lemon:writtenRep "A.I."@en ;
16   a lemon:LexicalForm .

18 <http://nlp.dbpedia.org/abbrevbase/lexicon/en/entry/A.I.#sense1>
19   lemon:definition [
20     lemon:value "Aromatase inhibitor"@en
21   ] ;
22   rdfs:label "Aromatase inhibitor" ;
23   dcterms:subject <http://dbpedia.org/resource/Category:Aromatase_inhibitors>
24     ;
25   owl:sameAs <http://pl.dbpedia.org/resource/Inhibitory_aromatazy> ,
26     [...] ;
27   lemon:reference <http://dbpedia.org/resource/Aromatase_inhibitor> ;
28   a lemon:LexicalSense .

29 <http://nlp.dbpedia.org/abbrevbase/lexicon/en/entry/A.I.#sense2>
30   lemon:definition [
31     lemon:value "Amnesty International"@en
32   ] ;
33   rdf:type <http://schema.org/Organization>,
34     <http://dbpedia.org/ontology/Non-ProfitOrganisation>,
35     <http://dbpedia.org/ontology/Organisation> ;
36   rdfs:label "Amnesty International" ;
37   dcterms:subject <http://dbpedia.org/resource/Category:Erasmus_Prize_winners>,
38     <http://dbpedia.org/resource/Category:
39       International_human_rights_organizations> ;
40   owl:sameAs <http://cs.dbpedia.org/resource/Amnesty_International> ,
41     [...] ;
42   lemon:reference <http://dbpedia.org/resource/Amnesty_International> ;
43   a lemon:LexicalSense .

```

Listing 8: Example resource of the abbrevbase

Structurally, every language is one `lemon:Lexicon`, linking all its abbreviations via `lemon:entry`. Every abbreviation is a `lemon:lexicalEntry` resource linking the abbreviation string itself as its `lemon:form`. Every entry is linking to a number of `lemon:lexicalSense` resources containing the abbreviation meanings i.e. the full name, the `rdfs:types` and `dcterms:subjects` and a link to the respective DBpedia page. This knowledge base makes it possible to tackle a number of use cases:

- querying for every abbreviation of a distinct language, thus obtaining the abbreviation dictionaries of the single languages for text processing in a specific language
- querying for the meanings of a single abbreviation for disambiguation purposes
- querying only abbreviations of a certain type, like organizations or place names
- querying abbreviations by category, using the Wikipedia category system to restrict the abbreviations by topic.

The last two use cases are especially interesting with regard to NLP techniques. Texts differ in grammatical and lexical structure depending on their type or domain. One example of this

heterogeneity is mentioned in 2.2.2 and becomes apparent by the significantly different percentage of abbreviations used in the Brown (10%) and the Wall Street Journal (47%) corpus. Thus, it may be helpful to use domain-adjusted models and dictionaries to process the texts. The inclusion of types and categories for abbreviation implies that domain specific abbreviation dictionaries are just subsets of the complete dictionary that can easily be retrieved by SPARQL queries.

The extraction was run against all DBpedia languages, yielding 22,859 abbreviations with 78,197 meanings. Depending on the language, a large number of different concepts can be abbreviated using the same abbreviation, thus leading to a large number of meanings per abbreviation. Another reason for this imbalance is that the data is retrieved from Wikipedia and abbreviations can be freely added, thus possibly introducing a large number of “exotic” abbreviations not widely in use.

The number of abbreviations retrieved per language strongly depends on the size of the Wikipedia edition for that language. While the larger Wikipedias, like English, Spanish, French or German provide a decent number of abbreviations, there is a long tail of languages where no abbreviations could be extracted at all. Figure 7.2 shows the number of abbreviations and meanings per language in a linear scale, visualizing the long tail. No abbreviations could be extracted for 19 languages, 55 languages have under 10 abbreviations, 95 languages have under 100 abbreviations.

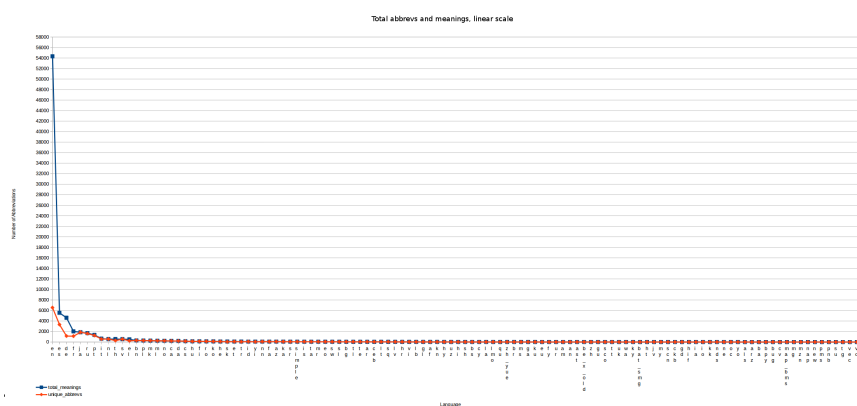


Figure 7.2: Abbreviations per language, linear scale

There are two major reasons for this long tail. First, the abbreviations may not be in the source data, i.e. the Wikipedia edition for that language. The Wikipedia language edition might not be large enough to have entries for abbreviations or local Wikipedia criteria might exclude abbreviations from notability. The languages that did not have abbreviations on average only had 32543.5 articles⁷⁰ and a very limited number of contributors, reinforcing this point. Second, the concept of an abbreviation defined as a string of characters ending with a sentence terminator may not exist in that language. Most East-Asian languages are for example fundamentally different from Indo-European languages regarding the existence of punctuation. “Chinese” Wikipedias for example (comprised of the three Wikipedias **zh**, **zh-yue**, **zh-min-nan**, containing around 850,000 articles) contained only 12 abbreviations in total, although the Wikipedia should be large enough to harbour much more.

⁷⁰according to http://meta.wikimedia.org/wiki/List_of_Wikipedias

Still, a good coverage could be achieved for the European languages. Figure 7.3 provides the abbreviations and meanings for all official EU languages in a logarithmic scale for added resolution. The long tail is much less pronounced, but still visible. On average there are 668 abbreviations per EU language, with the median being 153. Only Maltese contains no abbreviations. So the data is still skewed to the right but due to the higher average size of EU language Wikipedia editions and the bias of our abbreviation rule towards Indo-European languages, much less so. This is especially important because the European languages cover over 1.3 billion of native speakers.

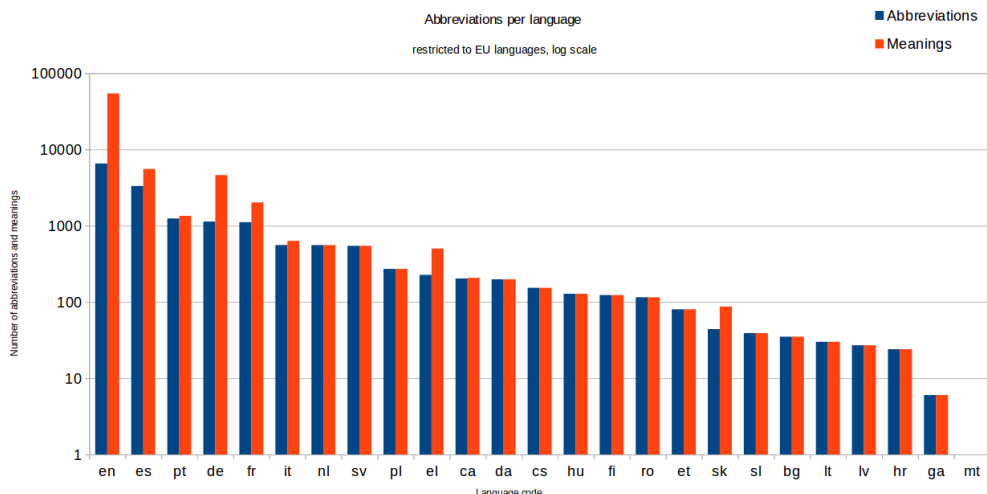


Figure 7.3: Abbreviations per language, only EU languages, logarithmic scale

Interestingly, most abbreviations follow a pattern: They are often chains of upper case letters followed by periods. Of the English abbreviation entries, 41.7% follow this pattern, 53.6% of German entries and 66.0% of Dutch abbreviation entries can also be expressed by this rule. A possible application of this connection will be presented in Section 7.3.

A major caveat of extracting lexical information from the DBpedia is, that case information for first letters of article names gets lost due to the MediaWiki software. All Wikipedia articles begin with an upper case letter, which is attributed to a technical restriction of the MediaWiki itself⁷¹. No reason is given for this restriction. The workaround used is a template called `{{Lowercase title}}`, that, if added to the article, displays the title correctly. However, this template is not mapped in the current version of the DBpedia and may only be used in a later version. For this reason, all abbreviations extracted begin with upper case letters. This severely restricts the usability of the extracted abbreviations, because case information is an important part of lexical data. This point and its practical consequences will be expanded upon in the following section.

7.3 Corpus-based evaluation of sentence boundary detection algorithms

In this section, the experiment run to prove the feasibility of using abbreviations as exceptions to improve sentence boundary detection quality will be described and results presented. The methodology used can be seen in figure 7.4.

⁷¹https://en.wikipedia.org/wiki/Wikipedia:Naming_conventions_%28technical_restrictions%29#Lowercase_first_letter

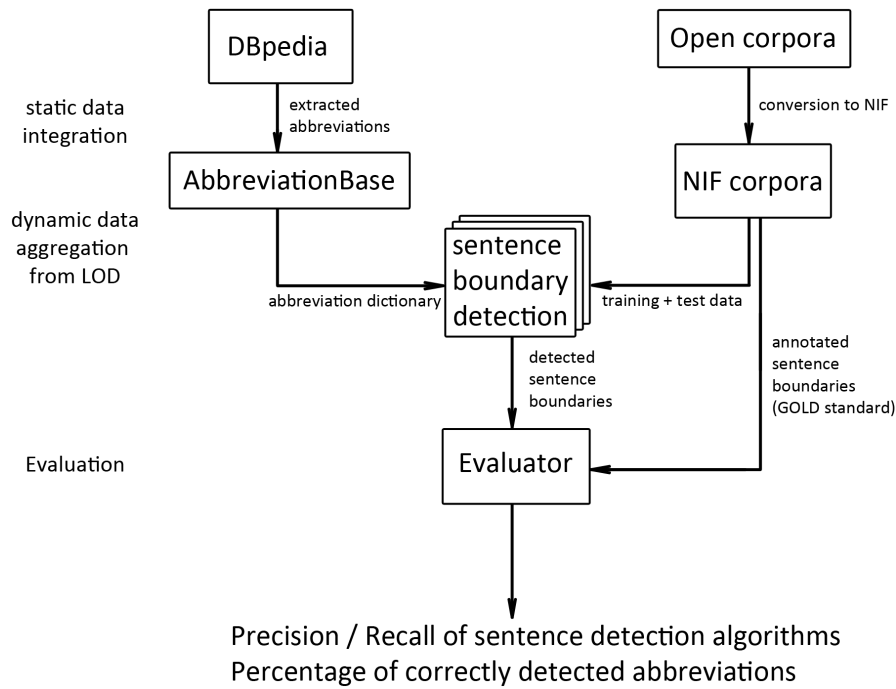


Figure 7.4: Data flow diagram showing the data conversion process

The abbreviations extracted in Section 7.2 provide one input, while both the Brown as well as the first 5000 sentences of the Tiger corpus converted in 4 will provide the second input. Sentences of the corpus will be parsed using the parser presented in 6.2. A subset of these sentences will be used to train an OpenNLP sentence detector using the OpenNLP wrapper presented in Section 6.1. This sentence detector and the Unicode rule-based sentence detector shall be run against the test corpus, one time with and one time without using abbreviation lists.

The International Components for Unicode (ICU)⁷² Java implementation of the Unicode sentence break rules will be used as implementation for the rule-based detector. The ICU project is a set of open source libraries for Java and C/C++ developed by IBM. It provides Unicode and globalization support like formatting according to locale data, time calculations, Unicode support and text boundaries, such as words, sentences and paragraphs in larger parts of text. In regards to sentence boundary detection, it relies on the Unicode sentence boundary specification, as described in 7.1. When the abbreviations are used, the last word of each detected sentence is compared to the abbreviation list. If it is found in the list, the sentence is concatenated with the next sentence and the procedure is repeated. Thus, every abbreviation is regarded as exception to the detection rules.

At this point, the problem with missing case information using DBpedia data becomes apparent again. Simple string identity means that abbreviations where case information is important, like in the case of *z.B.* (zum Beispiel; in example) in German, can not be found in the list, because all abbreviations start with upper case letters. To alleviate the issue, all abbreviations in the list were simply transformed to lower case and the last word of the supposed sentence was

⁷²<http://site.icu-project.org/>

transformed to lower case, prior to checking, too. Although there was no significant difference in correctness between the two approaches (literal identity and lower case transformation), the test data was too limited regarding domain, amount of languages and total amount of text, to make a recommendation in this case.

Evaluation results Table 7 shows the evaluation results for both corpora using no abbreviation list as baseline, an abbreviation list manually compiled by ULI (1233 english resp. 2052 german entries) and the abbreviation lists extracted from DBpedia (6536 english resp. 1134 german entries).

Algorithm	Abbreviation list	Precision	Recall
NIF Brown corpus			
ICU	-	83.78%	96.53%
ICU	ULI	92.89%	99.74%
ICU	DBpedia	85.68%	94.98%
OpenNLP	-	82.86%	83.56%
OpenNLP	ULI	83.25%	83.76%
OpenNLP	DBpedia	83.39%	81.64%
NIF Tiger corpus - first 5000 sentences			
ICU	-	84.97%	74.02%
ICU	ULI	84.87%	73.42%
ICU	DBpedia	84.97%	74.02%
OpenNLP	-	87.65%	75.92%
OpenNLP	ULI	87.35%	75.22%
OpenNLP	DBpedia	87.65%	75.92%

Table 7: Results for sentence boundary detection algorithms for Brown and the first 5000 sentences of the Tiger corpus using different abbreviation lists

As expected, using abbreviations as exceptions to rule-based sentence boundary detectors can provide a very effective way to increase precision. Especially the ICU detector gains significantly in precision and recall on the English Brown corpus. The maximum entropy implementation of Brown is less affected by the abbreviation lists but can still gain by the inclusion.

However, the results are less clear for the Tiger corpus. Both algorithms find nearly no abbreviations and in case of the ULI abbreviation list, false detection of abbreviations even causes less precise boundary detection. For example, the ULI abbreviation list contains abbreviations like “Haft.” or “eng.”, although these represent a regular noun and a regular adjective that can

occur at the end of a sentence and are thus not abbreviations.

In general, the part of the Tiger corpus analyzed is not well suited to test abbreviation lists, because it does not contain many abbreviations. In fact, only one example (“K.o.-Tropfen-Bande”) can be found, which does not present a common abbreviation.

Regarding the quality of the different abbreviation lists, the lists extracted from DBpedia are of lower quality than lists that are manually curated or extracted from higher quality, for example linguistic, sources. Although the absolute number of abbreviations that can be extracted is sufficiently large and data is available for a large number of languages, they have less of a positive impact on the sentence boundary detection algorithms. The better score of the OpenNLP boundary detector on the Tiger corpus with the DBpedia list can be explained because the DBpedia list did not contain any ambiguous abbreviations that can be confused for regular words, like the ULI list did. However, some important German abbreviations, like “z.B.” (such as) are not in the list. Others, like “bspw.” (for example) are contained in the list, but start upper case due to the Wikipedia casing constraints mentioned before and thus cannot be used effectively.

Additional properties of DBpedia abbreviations, like their categories that could be used to compile domain specific abbreviation lists could not be tested, because the test data is too sparse and the definition of a textual domain in terms of DBpedia categories is unclear. Typical sub-domains of corpora are newspaper texts in various topics, like sports or economy, as well as very specific domains, like scientific papers or government documents. In the second case, DBpedia data is too sparse, as abbreviations in the scientific domain are very specific and not well reflected in the Wikipedia. In case of newspaper texts, one could build a custom abbreviation list limited to only those abbreviations which have a category whose transitive closure of the `skos:broader` relationship, which denotes a broader category, connects them with a category relevant to the text domain. Additionally, a core list of most common abbreviations (like “Mr.”, “etc.”) would have to be used.

This can be achieved by processing multi-domain corpora and examining them for the number of abbreviation occurrences. Abbreviations can be ranked by occurrence or impact on boundary detection quality, and thus be used to compile abbreviation dictionary fragments containing only the most important abbreviations per language. Figure 7.5 shows a typical power law distribution of abbreviation impact on sentence boundary detection precision and recall, demonstrating that a small core of abbreviations contributes most of the quality gains reported. Extracted from multi-domain corpora, these are likely candidates for a common core of abbreviations used throughout different text types in one language.

Limitations Evaluation is limited by the corpora used. Comparing the evaluation results with similar evaluations using the Brown corpus, like [21] or [12] shows a significantly lower precision and recall in the evaluation executed here. Error analysis shows, that sentences not ending with a sentence terminator in the Brown corpus as well as the Tiger corpus are plenty. These are mostly headlines of texts and paragraphs that usually end with a paragraph mark which usually is also used by the boundary detectors. However, they are not included in the NIF corpora. In the Brown corpus, this is a gap in the conversion, as there are paragraph annotations in the TEI/XML source. In the Tiger CoNLL corpus, paragraphs are not annotated and can thus not be converted, leading to a lot of errors in detecting sentences that are not terminated.

As the Tiger example shows, another limitation is obtaining enough suitable corpora to evaluate the abbreviations in multiple languages. Trained statistical models for the machine learning sentence boundary detection algorithms only exist for a limited amount of languages and are domain dependent, increasing the need for more corpus data to train them. NIF alleviates some of these limitations by providing an interfacing format that supports corpora as well as NLP components, allowing integration of data and tools from many different sources. This

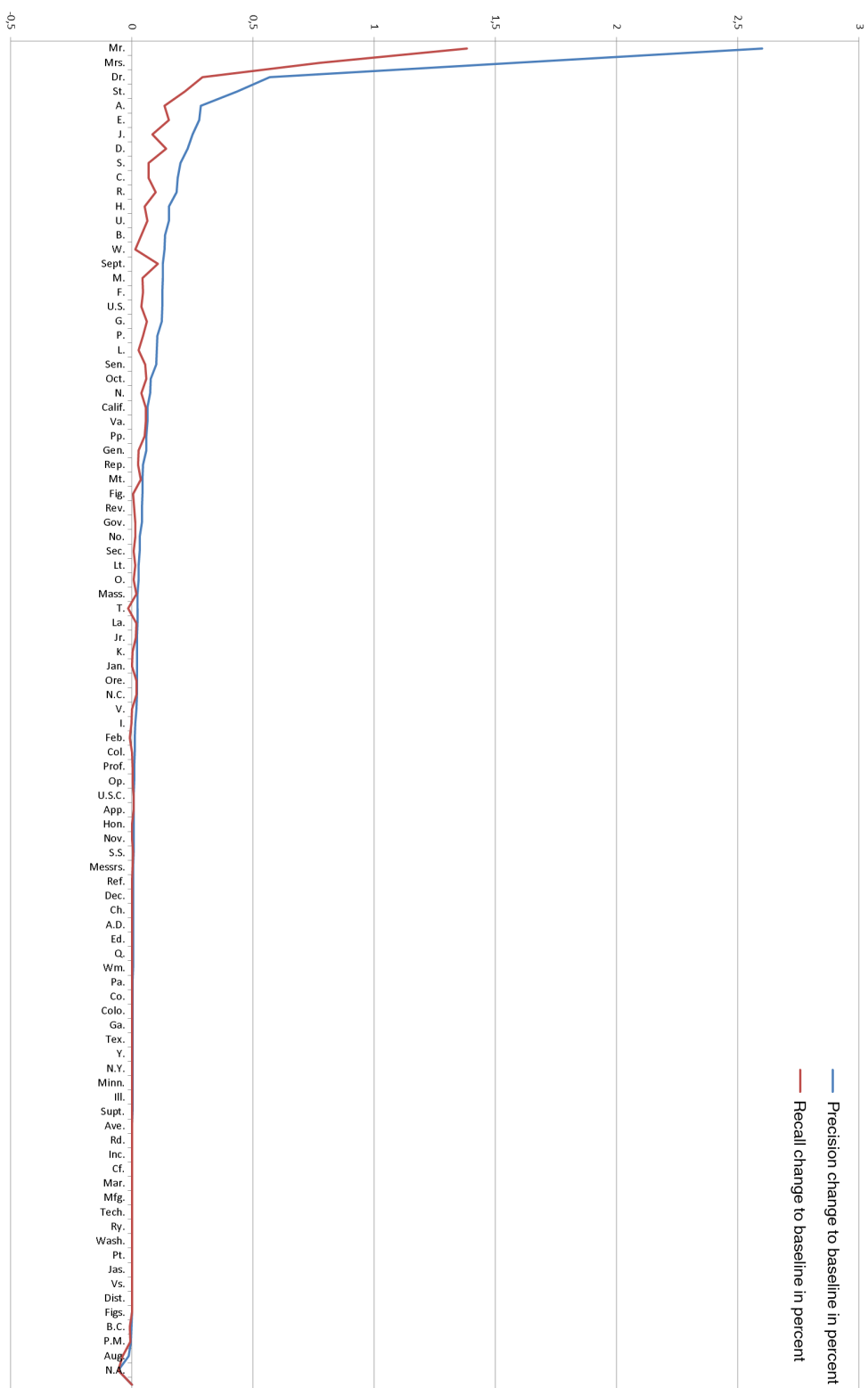


Figure 7.5: Precision and recall impact per abbreviation in percent

inspired GERBIL⁷³, the General Entity Annotator Benchmark, that leverages NIF to provide an interoperable evaluation framework for named entity recognition tools.

7.4 Gerbil

GERBIL [28] was developed because many researchers in NLP work at solving the same tasks at the same time, publishing proof of their progress using evaluations of their tools against the state-of-the-art and standard corpora. However, results vary and cannot be directly compared, as the intricacies of data preparation, corpus formats, tool versions and evaluation measures result in differences between the analyses. As a result, it is not quite clear which tool is best suited for a specific task or type of text or represents the state-of-the-art. Preparing the evaluation experiments, finding and formatting test data and setting up an evaluation pipelines has to be redone over and over again, which means a large overhead to researchers. Finally, tuning ones one tools is much easier if the performance can be reliably assessed against other state-of-the-art software. To help scientific progress and open up evaluation of NLP tools in an interoperable and comparable way, Gerbil was developed.

GERBIL allows the comparison of annotators on multiple datasets with uniform evaluation measures. It provides an easy web-based platform that is used to publish, persist and view evaluation results, which can also be used to add URLs of own services that abide by the NIF standard. Corpora in NIF can be added in the same way. Natively, GERBIL contains 9 annotators and 11 datasets for evaluation, providing a solid state-of-the-art. The ease of expansion, comparability and interoperability using NIF as a pivot format contribute to usage incentives and enable a broader base for testing NLP tools. Especially in regard to multilinguality, the GERBIL approach may fill the gaps that this evaluation had to leave unfilled.

8 Discussion and Future Work

Discussion. This work presented a thorough examination and expansion of the NIF ecosystem. Besides the original use case as format for NLP tool integration, NIF was identified as a possible corpus pivot format and accordingly, a large range of diverse existing corpora were converted into NIF. The complete contribution of corpora presented in this work adds up to 841,106,737 triples. The converted corpora were thoroughly validated to ensure data quality. RDFUnit presents an adequate and scalable solution for this purpose, easily extensible by the means of SPARQL test cases. Although writing these tests can pose unexpected problems, as discussed in Section 5.2, it is not the main obstacle for test-driven data development. Specifying correctness criteria of heterogeneous datasets, especially for NIF that contains arbitrary strings, proves to be the main hurdle for data quality.

A set of RDFUnit test cases, together with tests automatically generated from the original ontology, presents a formal specification of data quality. At the point of testing it with RDFUnit, this data already is result of a chain of tools. The source data might have been a structured data corpus, but it might also have been unstructured strings with few annotations. Some features of the data might cause problems with the conversion. It could, for example, contain escape characters incompatible with the serialization format. The data is then converted or annotated with a tool trying to address these errors, but there may be bugs or unexpected results. Now the output of this tool has to comply with the specification of the RDFUnit test cases, addressing errors in programming or source data quality that might not have existed when the specification was developed.

⁷³<http://aksw.org/Projects/GERBIL.html>

This is in contrast to test-driven software development, where the test is written first and the implementation of the feature allowing the test to validate comes second. Formalizing exhaustive correctness criteria in RDFUnit implies writing tests for all possible shortcomings of the input data. The incomplete state of NIF RDFUnit tests before expansion in this work shows that correctness specification coverage can not be guaranteed.

However, the refinement of the criteria based on expansive data testing, the correction of erroneous test cases and the expansion of the evaluation presented in [13] with more, larger datasets provide an important contribution to the NIF ecosystem and again prove the RDFUnit approach as an important part of state-of-the-art data development.

Besides the contributed corpora, this work also addressed the core of NIF with the implementation of a NIF Wrapper for the OpenNLP framework that also contains a parser for NIF, allowing the native use of NIF annotations in OpenNLP-based tools. A CoNLL converter was also implemented, allowing automatic conversion of a large number of evaluation datasets into NIF.

The implemented tools, as well as the reference implementation were thoroughly evaluated. This raised the problem of memory consumption of the tools implemented using the existing Jena-based NIF framework. In general, the memory overhead using Jena models is large and sometimes prohibitive for processing large corpora. Although this is not the standard case, solutions like using TDB or streaming already converted resources to reduce the Jena models' size were proposed but their implementation was outside this work's focus.

Finally, an industry use case was addressed by extracting abbreviation lists from linked data, using them to increase sentence boundary detection quality. NIF acted as moderator between evaluation and training corpora as well as the NLP tools itself, not unlike in GERBIL, but much less sophisticated.

The abbreviation extraction from DBpedia proved to be complicated in practice. Because the official SPARQL endpoint of DBpedia does not contain all the relevant data, the datasets themselves had to be downloaded and processed. However, it is not clearly documented which dataset files contain which properties, so one has to guess which of the 49 files are needed. After processing, it quickly became clear that the quality of DBpedia multilingual, lexical data is not really adequate for the use case at hand. The evaluation using two of the larger abbreviation lists, English and German, showed that even if there is a significant amount of data available, which is the case for most official EU languages, sentence boundary detection quality can only be slightly improved using these lists. Looking at the power law distribution of abbreviation impact on precision and recall, a small list of abbreviations should be enough to account for a large improvement of quality, as seen when the manually compiled ULI list is used.

The main problem here is, of course, that Wikipedia does not try to be a dictionary. Trivial abbreviations like "z.b." in German are not often not included because they generally don't fulfil Wikipedia's relevance criteria for encyclopedic articles. In contrast, Wiktionary provides an open, crowd-sourced dictionary. Its data conversion, DBnary⁷⁴, now is a very high-quality resource but was not available during the collaboration with ULI. In fact, using DBnary, the complicated data extraction is reduced to a number of SPARQL queries, because the SPARQL endpoint contains all 13 languages extracted in DBnary. Being a dictionary, casing information is also contained in DBnary, eliminating this shortcoming of DBpedia data. One of the steps to further this use case is therefore integrating DBnary data.

This use case not only showed the strengths of NIF to ease the burden of tool and data integration many NLP developers have to face. It also showed the real world value of contributions made by this work, tying together a very broad scaled effort of NIF development into a tangible application.

⁷⁴<http://kaiko.getalp.org/dbnary>

This “burden of development” is hard to evaluate without asking developers. From my perspective, having implemented all these tools and having converted the data, developing NIF itself is not easy, spawning non-obvious problems and gaps, suffering under poor documentation and scalability issues. The existing Jena implementation has to be reworked, as shown in the tool evaluation. RDFUnit eases NIF data creation significantly, though. However, NIF developers are not the target group of the project, NLP developers in general are. In turn, implementing the use case was relatively easy to do with NIF. Adding further corpora for evaluation, for example in other languages, or training the statistical tools on new corpus data, for example from a different domain, are two of the tasks that would be time consuming without NIF and only took a minor part in the implementation of the use case.

Future work. Besides solving some problems, this thesis also brought up many new ones that have to be addressed in the future. Ensuring scalability of NIF and improving the already implemented wrappers to provide a less resource-intensive framework will of course be on the agenda. Compliance with other projects is another important point. Integration of the OpenNLP wrapper directly into the Apache OpenNLP project is a mainly bureaucratic task that can help disseminate NIF and provide the OpenNLP community with interoperable RDF output. Providing a Python implementation could open up the large number of Python NLP libraries, such as NLTK⁷⁵ and gensim⁷⁶.

Finally, projects like GERBIL that constitute rich, open, interoperable tool chains, producing machine-readable and persistent Linked Data output to address substantial needs of the NLP community shall be the constant long-term goal in expanding the NIF ecosystem.

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] Tim Berners-Lee. Linked data, 2006.
- [3] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.
- [4] Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. The tiger treebank. In *Proceedings of the workshop on treebanks and linguistic theories*, volume 168, 2002.
- [5] Maria Sukhareva Christian Chiarcos. Olia – ontologies of linguistic annotation. *Semantic Web Journal - Special Issue on "Multilingual Linked Open Data (MLOD) 2012 Data Post Proceedings"*, 2015.
- [6] Unicode Localization Interoperability Technical Committee. Unicode standard annex #29 - unicode text segmentation. Technical report, Unicode, Inc., 2014.
- [7] W. N. Francis and H. Kucera. Brown corpus. *TEI XML version via NLTK data repository*, revised 1979.

⁷⁵<http://www.nltk.org/>

⁷⁶<https://radimrehurek.com/gensim/>

- [8] Dirk Goldhahn, Thomas Eckart, and Uwe Quasthoff. Building large monolingual dictionaries at the leipzig corpora collection: From 100 to 200 languages. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, 2012.
- [9] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. Sparql 1.1 query language, 2013.
- [10] Sebastian Hellmann, Jens Lehmann, Sören Auer, and Martin Brümmer. Integrating nlp using linked data. In *12th International Semantic Web Conference, 21-25 October 2013, Sydney, Australia*, 2013.
- [11] Johannes Hoffart, Stephan Seufert, Dat Ba Nguyen, Martin Theobald, and Gerhard Weikum. Kore: Keyphrase overlap relatedness for entity disambiguation. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, 2012.
- [12] T. Kiss and J. Strunk. nsupervised multilingual sentence boundary detecti. *Computational Linguistics*, 32:485–525, 2006.
- [13] Dimitris Kontokostas, Martin Brümmer, Sebastian Hellmann, Jens Lehmann, and Lazaros Ioannidis. Nlp data cleansing based on linguistic ontology constraints. In *Proc. of the Extended Semantic Web Conference 2014*, 2014.
- [14] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven evaluation of linked data quality. In *Proceedings of the 23rd International Conference on World Wide Web, WWW '14*, pages 747–758, Republic and Canton of Geneva, Switzerland, 2014. International World Wide Web Conferences Steering Committee.
- [15] Ora Lassila and Ralph R. Swick. Resource description framework (rdf) model and syntax specification, 1999.
- [16] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *COMPUTATIONAL LINGUISTICS*, 19(2):313–330, 1993.
- [17] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. Dbpedia spotlight: Shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems, I-Semantics '11*, pages 1–8, New York, NY, USA, 2011. ACM.
- [18] Ide N. and Suderman K. Graf: A graph-based format for linguistic annotations. In *Proceedings of The Linguistic Annotation Workshop (LAW) 2007*, 2007.
- [19] Axel-Cyrille Ngonga Ngomo, Norman Heino, Klaus Lyko, René Speck, and Martin Kaltenböck. Scms - semantifying content management systems. In *ISWC 2011*, 2011.
- [20] Dat P.T. Nguyen, Yutaka Matsuo, Aidan Hogan, and Mitsuru Ishizuka. Relation extraction from wikipedia using subtree mining. In Robert C. Holte and Adele Howe, editors, *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 1414–1420. The AAAI Press, 2007.
- [21] J. Read, R. Dridan, S. Oepen, and L. Solberg. Sentence boundary detection: A long solved problem? In *Proceedings of COLING 2012*, 2012.
- [22] J. C. Reynar and A. Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, 1997.

- [23] Giuseppe Rizzo, Raphaël Troncy, Sebastian Hellmann, and Martin Brümmer. NERD meets NIF: Lifting NLP extraction results to the linked data cloud. In *LDOW, 5th Workshop on Linked Data on the Web, April 16, 2012, Lyon, France*, Lyon, FRANCE, 04 2012.
- [24] Giuseppe Rizzo and Raphaël Troncy. Nerd: Evaluating named entity recognition tools in the web of data. In *10th International Semantic Web Conference, 23-27 October 2011, Bonn, Germany; Workshop on Web Scale Knowledge Extraction (WEKEX'11)*, 2011.
- [25] Michael Röder, Ricardo Usbeck, Sebastian Hellmann, Daniel Gerber, and Andreas Both. N3 - a collection of datasets for named entity recognition and disambiguation in the nlp interchange format. In *The 9th edition of the Language Resources and Evaluation Conference, 26-31 May, Reykjavik, Iceland*, 2014.
- [26] Sameer Singh, Amarnag Subramanya, Fernando Pereira, and Andrew McCallum. Wikilinks: A large-scale cross-document coreference corpus labeled via links to Wikipedia. Technical Report UM-CS-2012-015, 2012.
- [27] Nadine Steinmetz, Magnus Knuth, and Harald Sack. Statistical analyses of named entity disambiguation benchmarks. In *Proceedings of 1st International Workshop on NLP and DBpedia*, 2013.
- [28] R. Usbeck, M. Röder, and A. Ngonga. Gerbil - general entity annotator benchmark. In *Proceedings of WWW 2015 - forthcoming*.

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.

Ort

Datum

Unterschrift