Collaborative Project

LOD2 - Creating Knowledge out of Interlinked Data

Project Number: 257943 Start Date of Project: 01/09/2010 Duration: 48 months

Deliverable 3.2.1
NLP2RDF

Dissemination Level	Public
Due Date of Deliverable	Month 8, 30/4/2011
Actual Submission Date	31/07/2011
Work Package	WP3, Knowledge Base Creation, Enrichment and Repair
Task	Task T3.2
Type	Report
Approval Status	Approved
Version	1.0
Number of Pages	25
Filename	deliverable-3.2.1.pdf

Abstract: *NLP2RDF* is a reference implementation of the NLP Interchange Format (NIF). NIF is an RDF/OWL-based format that allows to combine and chain several NLP tools in a flexible, light-weight way. The core of NIF consists of a vocabulary, which can represent Strings as RDF resources. A special URI design is used to pinpoint annotations to a part of a document. These URIs can then be used to attach arbitrary annotations to the respective character sequence. Based on these URIs, annotations can be interchanged between different NLP tools. A web site *http://aksw.org/Projects/NIF* was created to serve as a reference point for the new format.

History

Version	Date	Reason	Revised by
0.1	2011-03-13	Created first version of Olia Connectors	Sebastian Hellmann
0.2	2011-04-20	Finished research on existing URI recipes	Sebastian Hellmann
0.3	2011-05-13	Introduced Maven and Mercurial	Sebastian Hellmann
0.4	2011-05-20	Content-negotiation for Ontologies	Sebastian Hellmann
0.5	2011-06-13	Implemented basic Web service architecture	Sebastian Hellmann
0.6	2011-07-10	Implemented fox.aksw.org	Sebastian Hellmann
0.7	2011-07-17	OLiA namespace was changed	Sebastian Hellmann
0.8	2011-07-24	Improved NIF project page	Sebastian Hellmann
0.9	2011-07-27	documentation	Sebastian Hellmann
1.0	2011-07-30	typos	Sebastian Hellmann
1.1	2011-08-15	incorporated first feedback from several mailing lists	Sebastian Hellmann
1.2	2011-08-20	incorporated feedback from developers of FOX	Sebastian Hellmann
1.3	2011-08-22	incorporated feedback from KAIST	Sebastian Hellmann

Author list

Organisation	Name	Contact Information
ULEI	Sebastian Hellmann	hellmann@informatik.uni-leipzig.de

Aknowledgements

The progress documented in this deliverable is based on discussions, feedback and aide provided by the following people (in no particular order):

LOD2 consortium: Sören Auer (AKSW, University of Leipzig), Jens Lehmann (AKSW, University of Leipzig), Claus Stadler (AKSW, University of Leipzig),

Axel Ngonga (AKSW, University of Leipzig, developer of FOX[1]), Key-Sun Choi (SWRC, KAIST), Pablo Mendes (Free University Berlin, developer of DBpedia Spotlight[2]), Andraz Tori (Zemanta[3]), Andreas Blumenauer (Semantic Web Company), Orri Errling (OpenLink), William Waites (OKFN), Friedrich Lindenberg (OKFN), Michael Hausenblas (DERI, LATC),

Other affiliations: Christian Chiarcos (University of Potsdam, developer of OLiA[4]), Gerhard Heyer and his research group (ASV[5], University of Leipzig), Iryna Gurevych and her research group (UKP[6], Technical University of Darmstadt), Johanna Völker (University of Mannheim, developer of LExO[7]), Erik Wilde (UC Berkeley, developer of RFC 5147[8]), David Booth (http://dbooth.org), Yury V. Katkov (AILAB[9]), Christian Ehrlich (Ontos[10]), Open Linguistics Group[11] and the participants of the workshop[12] at OKCon 2011.

Executive summary

The NLP Interchange Format (NIF) is an RDF/OWL-based format that allows to combine and chain several NLP tools in a flexible, light-weight way. The core of NIF consists of a vocabulary, which can represent Strings as RDF resources. A special URI design is used to pinpoint annotations to a part of a document. These URIs can then be used to attach arbitrary annotations to the respective character sequence. Based on these URIs, annotations can be interchanged between different NLP tools. NIF consists of several components:

- *URI recipes* to anchor annotation in documents
- Ontologies to describe the relations between these URIs
- Best practices for choosing annotations for these URIs
- Documentation on how to integrate NLP tools and adapt them to NIF
- Descriptions of how to create *NIF Web Services*
- A reference implementation (*NLP2RDF*), that can be freely exploited.

[1] http://fox.aksw.org
[2] http://dbpedia.org/spotlight
[3] http://zemanta.com
[4] http://nachhalt.sfb632.uni-potsdam.de/owl/olia.owl
[5] http://asv.informatik.uni-leipzig.de/
[6] http://www.ukp.tu-darmstadt.de/
[7] http://code.google.com/p/lexo/
[8] http://tools.ietf.org/html/rfc5147
[9] http://ailab.ifmo.ru
[10] http://ontos.com
[11] http://linguistics.okfn.org/
[12] http://okcon.org/2011/programme/open-linguistics-workshop

A web site *http://aksw.org/Projects/NIF* was created to serve as a reference point for the new format.

Contents

List of Figures

List of Tables

1 Introduction

This deliverable provides the documentation for the design and the prototypical implementation of an integration approach for NLP tools, formats and annotations based on RDF.

The **NLP Interchange Format (NIF)** is an RDF/OWL-based format that allows to combine and chain several NLP tools in a flexible, light-weight way. The format is presented in chapter 2. The core of NIF consists of three parts: 1. a recipe for unique and stable URIs used to anchor annotations. 2. a vocabulary, which can represent Strings, Words and Sentences as RDF resources. 3. transformations for the programmatic usage of the Ontologies of Linguistic Annotations (OLiA). NIF enables to annotate documents with overlapping, multi-tier annotations that potentially remain stable over the life-time of a document. Especially, the introduced recipe (Context-Hash) for URIs possesses advantageous properties, which withstands comparison to other URI naming approaches.

In chapter 3, we showcase the implementation of a system that is using NIF. Several tools were integrated and exposed via Web Services. These NIF web services can be downloaded and deployed locally to create ad-hoc NLP chains. We named this project **NLP2RDF** and it essentially contains implementations and wrappers that produce NIF output. The instructions on how to install and run the NLP2RDF 1.0 release can be found in the Google Code download area[1]. Furthermore, it provides the infrastructure to participate in generalising and extending NIF.

This deliverable concludes with a road map and future work in chapter 4.

A web site *http://aksw.org/Projects/NIF* was created to serve as a reference point for the new format.

[1] `http://code.google.com/p/nlp2rdf/downloads/list`

2 NLP Interchange Format (NIF)

The NLP Interchange Format (NIF) is an RDF/OWL-based format that allows to combine and chain several NLP tools in a flexible, light-weight way. The core of NIF consists of three parts:

1. a set of URI recipes, used to create unique and potentially stable URIs to anchor annotations in documents.

2. a vocabulary, which can represent Strings, Words and Sentences as RDF resources.

3. transformations for the programmatic usage of the Ontologies of Linguistic Annotations (OLiA).

2.1 Referencing Strings with URIs

In a similar way as a fragment identifier[1] references a sub resource of a primary resource, we aim to reference a SUBSTRING of a given TEXT with the help of a URI. This means, that we need to design a string, that serves as an IDENTIFIER for a substring when related to a DOCUMENT[2]. Although, many ways can be envisioned for such an identifier, the most direct is to use either the offset relative to the document combined with the length of the substring or the start and end position of the substring relative to the document. The identifier string "5-10" for example can either relate to a substring that is 10 characters long and starts at position 5 of a given document or it references the substring from position 5 until position 10. When creating a URI this "identifier string" needs to be appended to a URI *base* to form a full URI. In the case of a web document the URL can be used as the base. Let's say for example that we would like to annotate the substring "Semantic Web" in the section "The four rules" in the sentence "If it doesn't use the universal URI set of symbols, we don't call it Semantic Web." in the web document: `http://www.w3.org/DesignIssues/LinkedData.html` as shown in Figure 2.1.

The first problem, we would have to address would be UNIQUENESS, as the substring "Semantic Web" occurs 9 times in the document.

[1] `http://tools.ietf.org/html/rfc3986#section-3.5`

[2] We will use the term DOCUMENT to refer to whole unit and TEXT to refer to the actual content, e.g. a HTML document on the web containing text (the HTML code)

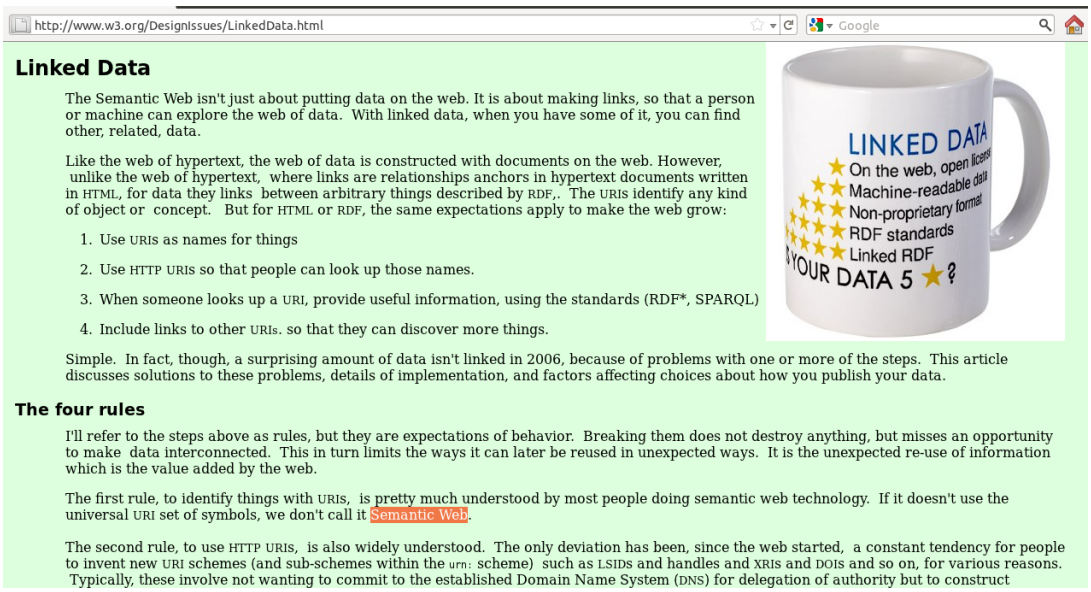

Figure 2.1: Annotating "Semantic Web"

A fragment identifier such as `#Semantic+Web` would definitely be insufficient and lack the necessary information to distinguish the different substrings. If we look at the text of the document (the underlying HTML source), we can discover that the substring we want to annotate is on line 223, starts at position 14406 has 12 characters and ends at position 14418. The left context "`don't\n call it` " and the right context "`.<br>\n <br>\n` " of the substring can also be helpful to distinguish it from other occurrences. If the HTML tags and new lines are removed the context becomes more readable (Left: "`symbols, we don't call it` ", right: "`. The second rule, to use`").

There are three ways to append the identifier to the URL:

1. separated by a "/", so it becomes part of the URL.
2. using a query parameter, such as "?nif=".
3. separated by a "#", so it becomes the fragment identifier.

For our purposes it is of minor relevance, which of the three possibilities is used. Note that the fragment identifier is normally not included in an HTTP request and thus invisible to the server. The choice is left open to the implementing application, as option one and two can produce de-referencable URIs, where the server handles the logic. A browser extension on the other hand could highlight the referenced substring.

On Table 2.1, we included examples for all approaches mentioned in this deliverable. For more approaches, we refer the reader to Wikipedia[3].

[3] `http://en.wikipedia.org/wiki/Fragment_identifier`. Note: to the best of our knowledge this list seems to be complete.

Name	http://www.w3.org/DesignIssues/LinkedData.html
Offset (Start-End)	#14406-14418
Offset w/ recipe	#offset_Semantic+Web_14406-14418
Offset-Length w/ #	#14406-12
Offset-Length w/ /	/14406-12
Offset-Length w/ ?	?nif=14406-12
Context-Hash w/recipe	#hash_md5_4__12_Semantic+Web_abeb272fe2deadd2cd486c4cea6cddf1
Word w/o context	#word_Semantic+Web
Yee (Context)[8]	#:words:we-dont-call-it-(Semantic Web).-The-second-rule,-to-use
Wilde and Dürst[6]	#char=14406-12
LiveURL (Content-Based) [3]	#8Semantic12+0x206A73ED

Table 2.1: URI anchors to annotate the substring "Semantic Web" in the section "The four rules" in the sentence "If it doesn't use the universal URI set of symbols, we don't call it Semantic Web." in the web document: `http://www.w3.org/DesignIssues/LinkedData.html`. All examples append a suffix to the URL of the web document `http://www.w3.org/DesignIssues/`+suffix.

2.2 Criteria for URI Recipes

As the suitability of the string identifiers highly depends on the specific task, we supply a list of criteria in this section, which allows to evaluate and design such identifiers. We call an algorithm to create identifiers URI RECIPE. Appending the identifier to the URL base is the last step in such a recipe. The annotation task at hand can be described as follows: For a given text t (a sequence of character) of length l_t (# number of characters), we are looking for a *URI recipe* to create a URI, that can serve as a *unique* identifier for a substring s of t (hence $l_s \leq l_t$). We generally assume that this substring consists of adjacent characters only and is therefore a unique character sequence within the document, if we account for parameters, such as context and position. Such a *URI recipe* can have the following properties:

UNIQUENESS The URIs, produced by the recipe, must uniquely identify the substring, i.e. there is no ambiguity. Uniqueness for a URI is generally required by definition (the 'U' in URI).

ARBITRARINESS The URI recipe must be able to produce URIs for arbitrary substrings, while still producing valid URIs. Valid URIs must not contain invalid characters. Also a URI needs to be finite in length. Furthermore, most browser limit the size of the URIs, they can handle[4]

XML COMPATIBILITY The identifier part for the generated URIs should not start with a number, as it might interfere with RDF/XML serialisation. XML tag elements can not begin with a number, so there is potentially the danger that the prefix namespace will be chosen badly, resulting in a tag such as $< 14406-14418 >$. Normally this happens for OWL classes and properties. Although, it is a minor detail, it still poses a restriction on the usage of the produced URIs .

TRANSPARENCY The URI should only become invalid, if the referenced string is changed in a significant way, thus rightfully rendering the annotations void. It should not become invalid through unrelated changes. This property implies *justified stability* (see below).

ADDRESSABILITY The produced URIs can be utilized to algorithmically find the annotated substring within the text, i.e. calculate the start and end position.

[4]Microsoft Internet Explorer has a maximum uniform resource locator (URL) length of 2,083 characters. `http://support.microsoft.com/kb/q208427/`

Self-Description Some URI recipes require certain parameters to find the appropriate substring in the document. The URIs should contain encoded information that can be used to identify the recipe and that can be used to reproduce the *configuration* of the recipe.

Implementation Effort Although this is not a strict functional criterion, it is important that the URI recipe is easy to implement. The created URIs are to be handled by multiple inter-operable tools. A correct implementation is necessary to allow the creation of tool chains.

Readability Although the readability is non-functional as well, it aids developers as well as users (annotators) in quickly identifying the referenced substring. Readability is enhanced, if the URI contains at least some characters of the referenced string and improves, if it additionally contains the context.

Functionality The produced URIs potentially reference strings that are calculated by a certain function (e.g. a case-insensitive regex addresses all occurrences of a string). The referenced strings are not required to be adjacent in this case. Functionality measures the complexity and expressiveness of the used uri recipe.

Notes on Justified Stability

Although stability of identifiers is in general a desirable property [2], unconditional stability can lead to erroneous annotations. Therefore, identifiers should just remain stable, if the annotated reference string stays the same and the semantics of any annotations remain valid. In case, for example, an offset was used, the insertion of a single character at any position prior to the referenced string breaks up the whole index and renders the annotation void. Such a change is often completely unrelated to the referenced string. We give an example here. For the sentence "My favorite actor is Bruce Willis", we would like to create an annotation that links "Bruce Willis" to DBpedia using the offset-length identifier. The respective URI could be either `ex:offset-length__21-12`, or `ex:word__Bruce+Willis`. If a "u" is inserted into "favourite" the first identifier will become invalid, although it could have been adjusted to match the new text. Such an instability is therefore unjustified. The second identifier remains stable, which can easily be justified. If the sentence, however, changes to "My favorite actress is Natalie Portman." both identifiers become invalid. It is easy to see that the second identifier is more *transparent* w.r.t. justified stability.

2.3 Existing Approaches

LiveURLs [3] [5] offers two different ways to produce string identifiers a content-based and a position based. Both can be appended to the URL as a fragment identifier. The user can select a text in the browser and the provided Firefox plugin creates the URL with the corresponding fragment. This URL can be shared and the referenced string is highlighted. The fragment is created using `sSl+c`, where "s is the length of the starting word of the selection, S is the starting word of selection, l is the length of the total selection, + is a delimiter, c is the checksum. As the identifier starts with a number, it can create a conflict with XML serialisation. Furthermore, the identifier does not contain enough information to uniquely distinguish duplicates, i.e. it would match several occurrences. The position based method uses a combination of the parent node's id and index in the DOM tree alongside an identifier for the child position. As all position based methods it is highly susceptible to change.

Wilde and Dürst [6] filed an RFC in April 2008[6] proposing a parameter-like syntax using hash-fragments that refer to statistics about the characters in the string (e.g. offsets, line, length), e.g. `ftp://example.com/text.txt#line=10,20;length=9876,UTF-8`. As all position based methods it is highly susceptible to change. The parameter syntax is highly extensible and could easily incorporate other approaches.

Yee [8] proposed *Text-Search Fragment Identifiers*, which pinpoint the wanted substring with a fragment that includes the string and its context. Before the creation of the identifier string, however, the original HTML source is manipulated and all HTML tags are removed and special characters are normalized. The resulting URL for our example is: `''#:words:we-dont-call-it-(Semantic Web).-The-second-rule,-to-use''`. This recipe produces the URIs with the best readability. The problem is that the proposed normalization (i.e. remove HTML and tokenise context), can not be standardized easily as it it relies on complex NLP methods. Therefore there is no real guarantee to reproduce the manipulation bi-directionally (e.g. to find the annotated substring.). If the selected substring becomes longer, so does the URI. It is for example impossible to annotate a large paragraph due to size restrictions.

Wilde and Baschnagel [7] propose to use regex patterns following the parameter "matching" as fragment identifiers, i.e. `matching=Semantic\sWeb` would match all nine occurrences of "Semantic Web" at once. Although, this recipe is very powerful, it is not straight-forward to implement an algorithm that produces

[5] `http://liveurls.mozdev.org/index.html`
[6] `http://tools.ietf.org/html/rfc5147`

unique regular expressions for URIs. Considering that it is possible to include the context in such a URI this recipe is a superset of the previous approach by Yee. Other disadvantageous are the unpredictability respective uniqueness and bad readability.

2.4 NIF - URI Recipes

Based on this analysis, we designed several URI recipes, that on the one hand counteract some of the disadvantages of the existing approaches, while on the other hand incorporating several advantages. The decision whether the URIs use '#' or '/' or '?nif=' is left open to the implementing application. Therefore, each URI recipe presented here starts with a variable $base that ends with one of {'#', '/','?nif='} . Furthermore, the substring identifiers following the $base always starts with a letter to avoid XML incompatibility. They also contain a part ($configuration) that is describing the configuration of the used recipe as well as naming the used recipe (Self-Description) and a second part($identifier) that serves as the actual identifier. The parts are separated by a double underscore "__" . In total, each recipe produces URIs in the form of: $base$configuration__$identifier. Any documentation about subsequent URI recipes should start first with a description of how the $identifier is created, followed by a description on how $configuration is encoded. The criterion Self Description evaluates, whether it is possible to reproduce the configuration of the URI Recipe from the URI itself.

2.4.1 Offset URI Recipe

The $identifier contains the first 20 characters (URL encoded) followed by two positive integers giving the starting and ending position of the substring, separated by a '-'. The $configuration consists of a single word 'offset' resulting for example in $prefix."offset__14406-14418" or `http://www.w3.org/DesignIssues/LinkedData.html#offset__Semantic+Web_14406-14418` This recipe is easy to implement and decode and the referenced string can be found efficiently. The identifier is rendered invalid, however, as soon as a single character is inserted or removed prior to the referenced substring, as the offset is changed.

An implementation of this recipe can be found in the NLP2RDF project[7].

2.4.2 Context-Hash URI Recipe

The $identifier contains the total length of the referenced substring, followed by the first 20 characters of the referenced substring. These 20 characters should be URL-encoded and less characters should be given if the total length is less than 20. This is followed by a digest produced by a hash function such as md5. The message for

[7] `http://code.google.com/p/nlp2rdf/source/browse/nif/src/main/java/org/nlp2rdf/core/impl/OffsetBased.java?r=f5110f9d2d61db9b64b6af841bbbc3774153c11e`

	Uniq	Arb	XML	Trans	Addr	Self	Impl	Func
Context-Hash (NIF)	+	+	+	+	+	+	o	o
Offset	++	++	-	- - -	++	-	++	o
Offset w/ recipe (NIF)	++	++	+	- - -	++	+	++	o
Yee (Context)	+	- -	+	+	- -	- -	- -	o
Wilde and Dürst (Param)	++	++	+	- - -	++	++	+	+
LiveURL (Content-Based)	- -	+	-	+	+	-	++	o
LiveURL (Position-Based)	+	+	-	- -	+	-	-	o
Wilde et.al. (Regex)	o	++	+	+	+	+	- -	++

Table 2.2: Comparison of URI recipes

this digest is made up of $leftcontext$_e$."(".$substring.")"$rightcontext$_e$, where e (environment) is the length of the included context before and after the referenced substring. The size of e can be chosen for a document so that each message inserted into the hash function is unique, therefore the produced digest will be unique – albeit for collisions. This follows from the fact that e can be chosen so large that the context covers the whole document. The separators "(" and ")" change the message, so that it does not coincide with longer or shorter strings in the same document.

The $configuration consists of the word "hash", the used function, in this case "md5" and an integer for e . An example would be `''#hash_md5_4__12_Semantic+Web_79edde636fa` where the message for the digest is made up of " it (Semantic Web).⟨br"

An implementation of this recipe can be found in the NLP2RDF project[8] .

2.4.3 Example Annotation

Based on the provided URI recipes, it is now possible to attach arbitrary annotations to any document that is based on Strings. Such an annotation could then look like this in Turtle syntax:

```
@prefix : <http://www.w3.org/DesignIssues/LinkedData.html#> .
@prefix rev: <http://purl.org/stuff/rev#> .
:hash_md5_4__12_Semantic+Web_abeb272fe2deadd2cd486c4cea6cddf1
   rev:hasReview  [
      rev:hasComment "Hey Tim, good idea that Semantic Web!" .
  ] .
```

2.4.4 Comparison

Table 2.2 gives a comparison of URI recipes. Both recipes used in NIF fix shortcomings of other methods with respect to the discussed criteria. Better URI recipes can be envisioned, e.g. a combination of the Offset and Context-Hash

[8] http://code.google.com/p/nlp2rdf/source/browse/nif/src/main/java/org/nlp2rdf/core/impl/MD5Based.java?r=49ee55b199db54219a2b00b564c9347988becbb9

recipe. The NLP Interchange Format provides a framework for the definition of further recipes and they can be easily included[9]. The evaluation is argumentative only, in the future we will quantify the judgment.

2.5 NIF - Ontologies

NIF is built upon two Ontologies the String Ontology and the Structured Sentence Ontology (SSO). As the ontology documentation should suffice to explain the ontologies (see next section), we will give two example of the usage here in Figure 2.2 and Figure 2.3[10]. In the first figure, we can see how the Context-Hash URIs are used for each word in the sentence "*My favorite actress is Natalie Portman!*" and the sentence itself, which coincides with the whole document here. *Word* and *Sentence* are OWL classes in the Structured Sentence Ontology and they are modelled as a subclass of *String*. Each *String* has one mandatory property `anchorOf`, which contains the referenced substring. In Figure 2.2, a simple annotation is added to each word, i.e. the stem of the referenced string using the Porter algorithm[4]. In most cases, the stem and the original string overlap, except for example for the stripped "*e*" of "*favorite*". We provide an implementation that produces the output shown in Figure 2.2. [11]) .

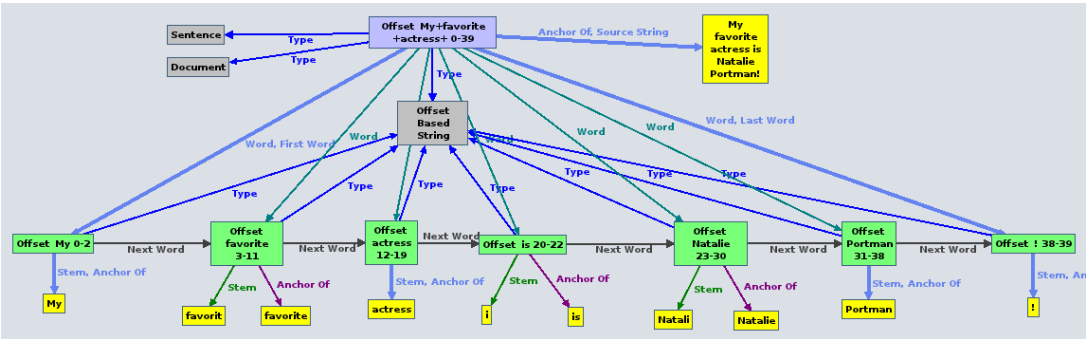

Figure 2.2: An annotation for the stem is added to each word.

2.5.1 OLiA Ontologies

Furthermore, NIF uses the Ontologies of Linguistic Annotations (OLiA, [1][12]) to provide a stable ontological interface for applications. In Figure 2.3 we show how this interface is used. The annotations are provided by the Stanford POS Tagger[5], which uses the Penn Tag Set[13]. OLiA provides an *Annotation Model*

[9]Please send suggestions to nlp2rdf-discussion@googlegroups.com
[10]Visualisation with Gruff `http://www.franz.com/agraph/gruff/index.lhtml`
[11]Documentation `http://code.google.com/p/nlp2rdf/wiki/SnowballStemmer`
[12]overview at `http://nachhalt.sfb632.uni-potsdam.de/owl/`
[13]see `http://www.comp.leeds.ac.uk/amalgam/tagsets/upenn.html`

for the most frequently used tag sets, such as Penn [14]. These *Annotation Models* are then linked to a *Reference Model*, which provides the interface for applications. Queries such as "Return all Strings that are annotated (i.e. typed) as `http://nachhalt.sfb632.uni-potsdam.de/owl/olia.owl#PersonalPronoun`" are therefore possible, regardless of the underlying tag set. This also guarantees language independence. We provide an implementation that produces the output shown in Figure 2.3. [15].

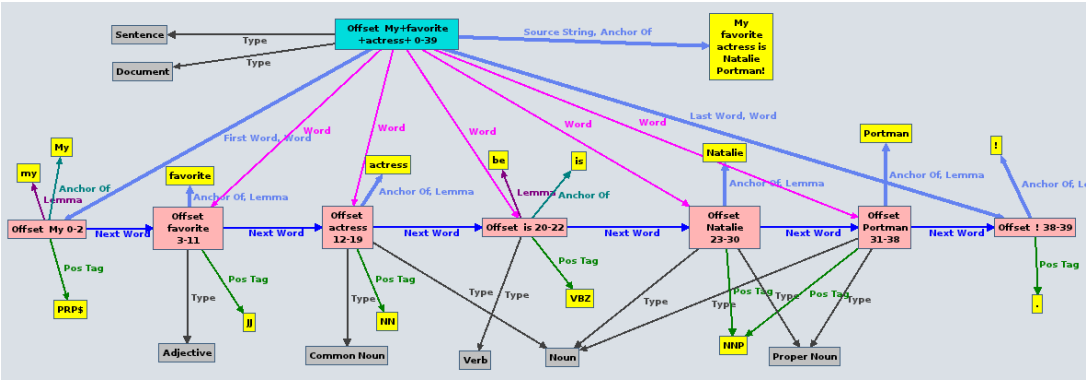

Figure 2.3: A visualisation of NIF using POS tags produced by the Stanford Core NLP framework. The yellow boxes on the bottom show the actual annotations produced by the Stanford Tagger, while the grey OWL classes below are provided by the OLiA Reference Ontology `http://nachhalt.sfb632.uni-potsdam.de/owl/`.

2.5.2 Creation and Documentation

The ontologies alongside with the documentation are available online. In this deliverable, we included information on how we published the ontologies on the web, so the process can be repeated. The String Ontology is available at `http://nlp2rdf.lod2.eu/schema/string/` and the Structured Sentence Ontology can be found here: `http://nlp2rdf.lod2.eu/schema/sso/`. The location of the ontology uses content negotiation and it is possible to explore them in an HTML browser. The documentation was created with the OWLDoc Protégé Plugin[16] as shown in Figure 2.4. The resulting documentation can be seen in Figure 2.5, the ontology can be downloaded with a blank accept-header (default), an accept-header with "application/rdf+xml" or directly at `http://nlp2rdf.lod2.eu/schema/string/string.owl` or `http://nlp2rdf.lod2.eu/schema/sso/sso.owl`. Below we included the Apache .htaccess file used to enable content negotiation.

[14] `http://nachhalt.sfb632.uni-potsdam.de/owl/penn.owl`
[15] Documentation `http://code.google.com/p/nlp2rdf/wiki/NIFWebServices`
[16] see `http://protegewiki.stanford.edu/wiki/OWLDoc` for the plugin and `http://code.google.com/p/ontology-browser/` for recent developments.

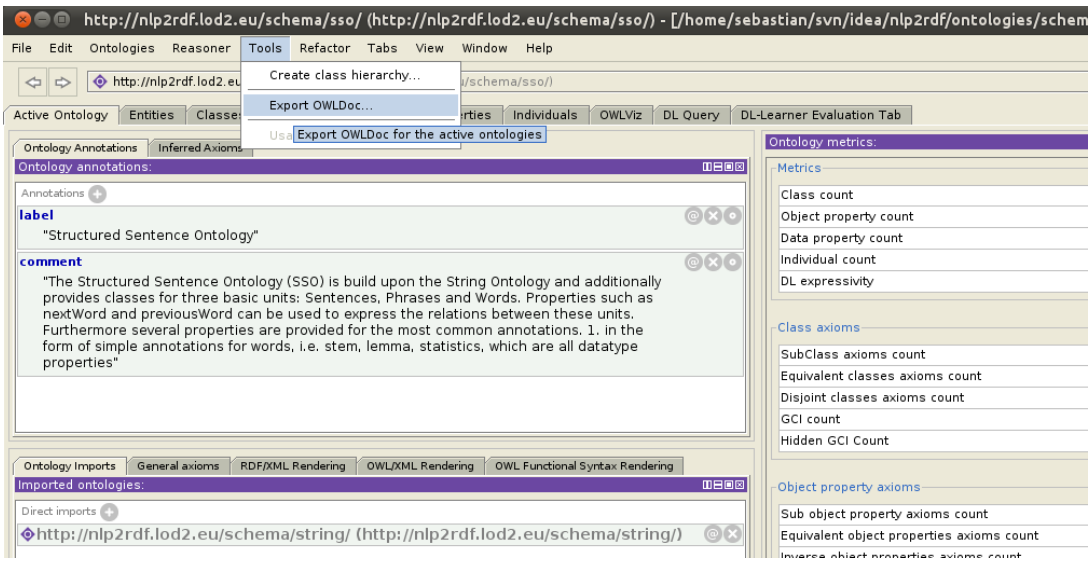

Figure 2.4: Generation of the HTML Documentation with Protégé

```
# From http://www.w3.org/TR/swbp-vocab-pub/#recipe4
Options -MultiViews
AddType application/rdf+xml .rdf .owl
RewriteEngine On

RewriteRule string/string.owl - [L]
RewriteRule sso/sso.owl - [L]
RewriteRule ^nlp2rdf.lod2.eu.schema/stringDoc/index.html - [L]
RewriteRule ^nlp2rdf.lod2.eu.schema/ssoDoc/index.html - [L]

# Rewrite rule to serve HTML content from the vocabulary URI if requested
RewriteCond %{HTTP_ACCEPT} !application/rdf\+xml.*(text/html|application/xhtml\+xml)
RewriteCond %{HTTP_ACCEPT} text/html [OR]
RewriteCond %{HTTP_ACCEPT} application/xhtml\+xml [OR]
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/.*
RewriteRule string/(.*)$ http://nlp2rdf.lod2.eu/schema/stringDoc/index.html [R=303]

# Rewrite rule to serve HTML content from the vocabulary URI if requested
RewriteCond %{HTTP_ACCEPT} !application/rdf\+xml.*(text/html|application/xhtml\+xml)
RewriteCond %{HTTP_ACCEPT} text/html [OR]
RewriteCond %{HTTP_ACCEPT} application/xhtml\+xml [OR]
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/.*
RewriteRule sso/(.*)$ http://nlp2rdf.lod2.eu/schema/ssoDoc/index.html [R=303]


# Rewrite rule to serve RDF/XML content if requested
RewriteCond %{HTTP_ACCEPT} application/rdf\+xml
RewriteRule string/(.*)$ http://nlp2rdf.lod2.eu/schema/string/string.owl [L,R=303]

RewriteCond %{HTTP_ACCEPT} application/rdf\+xml
RewriteRule sso/(.*)$ http://nlp2rdf.lod2.eu/schema/sso/sso.owl [L,R=303]
```

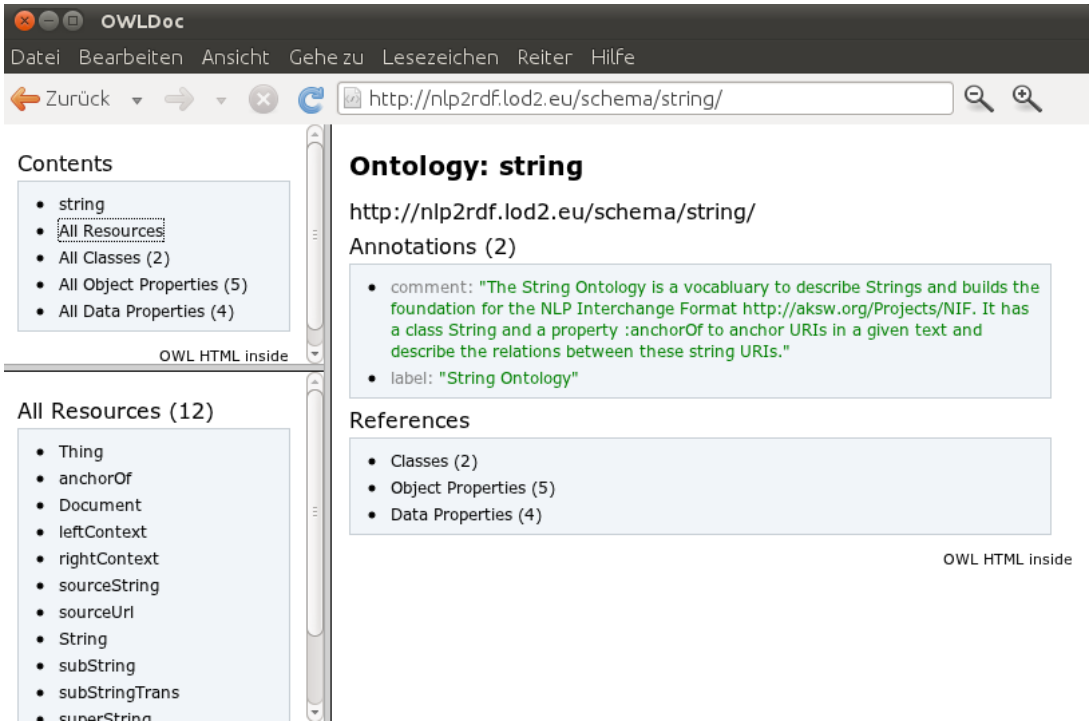

Figure 2.5: The HTML Documentation of the String Ontology

```
# Choose the default response
# Rewrite rule to serve RDF/XML content by default
RewriteRule string/ http://nlp2rdf.lod2.eu/schema/string/string.owl [L,R=303]
RewriteRule sso/ http://nlp2rdf.lod2.eu/schema/sso/sso.owl [L,R=303]
```

3 NLP2RDF

As mentioned earlier, the NLP2RDF project provides reference implementations and demo showcases to support the adoption of NIF. The project is hosted at Google Code and provides an issue tracker available at `http://code.google.com/p/nlp2rdf/issues/list` and a discussion mailing list `http://groups.google.com/group/nlp2rdf-discussion` or email *nlp2rdf-discussion@googlegroups.com*.

The Wiki contains useful information on how to deploy, install and extend the software, we forward the reader to `http://code.google.com/p/nlp2rdf/wiki/Welcome` as an entry point for further reading. The instructions on how to install and run the NLP2RDF 1.0 release can be found in the Google Code download area[1].

3.1 How NLP Tools are integrated with NIF Models

NLP tools can be integrated with NIF, if an adapter is created, that is able to parse a *NIF Model* into the internal data structure and also to output the NIF as a serialization. The effort for this integration is normally very low; just a parser and a serializer have to be written. A NLP pipeline can then be formed by either:

- passing the NIF RDF Model from tool to tool
- passing the text to each tool and then merge the NIF output to a large model

The URI recipes of NIF are designed to make it possible to have zero overhead and only use one triple per annotation, e.g.:

```
@prefix ld: <http://www.w3.org/DesignIssues/LinkedDataLinkedData.html#>
@prefix revyu: <http://purl.org/stuff/rev#>
ld:offset__Semantic+Web_14406-14418
      rev:hasComment  "Hey Tim, good idea that Semantic Web!".
```

To ease programmatic access however, a NIF model requires the following:

3.1.1 Requirement 1

All URIs created by the mentioned URI recipes should be typed with the respective OWL Class.

So in the above case the following additional triples are needed:

[1] `http://code.google.com/p/nlp2rdf/downloads/list`

```
@prefix str: <http://nlp2rdf.lod2.eu/schema/string/>
@prefix ld: <http://www.w3.org/DesignIssues/LinkedData.html#>
ld:offset__Semantic+Web_14406-14418
rdf:type  str:OffsetBasedString .
```

We have created two OWL Ontologies available at:

- `http://nlp2rdf.lod2.eu/schema/string/`
- `http://nlp2rdf.lod2.eu/schema/sso/`

3.1.2 Requirement 2

In each returned NIF model there should be at least one uri that relates to the document as a whole.

In our case this would be:

```
@prefix str: <http://nlp2rdf.lod2.eu/schema/string/>
@prefix ld: <http://www.w3.org/DesignIssues/LinkedData.html#>
ld:offset__%3Chtml%20xmlns%3D%22http%3A%2F%2F_0-25482
      rdf:type  str:OffsetBasedString .
ld:offset__%3Chtml%20xmlns%3D%22http%3A%2F%2F_0-25482
      rdf:type  str:Document .
ld:offset__%3Chtml%20xmlns%3D%22http%3A%2F%2F_0-25482
      str:sourceUrl  <http://www.w3.org/DesignIssues/LinkedData.html> .
```

3.1.3 Requirement 3

Each other annotated String should be related to the URI given to the Document with a property that is a sub poperty of str:subString

There are several properties available auch as str:subString, sso:word or sso:sentence. Custom properties can be used as well. The rationale behind this is that it is possible to relate each annotation to each document, in case there are several documents in one NIF Model.

```
@prefix str: <http://nlp2rdf.lod2.eu/schema/string/>
@prefix ld: <http://www.w3.org/DesignIssues/LinkedData.html#>
ld:offset__%3Chtml%20xmlns%3D%22http%3A%2F%2F_0-25482
  str:subString ld:offset__Semantic+Web_14406-14418 .
```

3.1.4 Requirement 4

For each annotation, a reference model should be used, so the annotations are machine-interpretable.

This part poses the most challenges to the interchange format. Normally each NLP tool speaks its own dialect (also called domain-specific language - DSL - or vocabulary or tagset), i.e. does the annotation in a very specific way. NIF offers a framework to add arbitrary annotations. It therefore allows that each tool can use whatever dialect seems suitable. However, NIF requires that this dialect is mapped to a reference Ontology. This mapping is not always straight-forward and for most domains and tasks such reference models do *not* exist yet.

DBpedia for Entity Linking

For Knowledge Extraction[2] from text, DBpedia[3] is frequently used and can be considered a de-facto standard in the Semantic Web community. Without guaranteeing that the list is exhaustive, we would like to mention: DBpedia Spotlight[4], OpenCalais[5], Zemanta API[6], Extractiv[7], AlchemyAPI[8], FOX[9], Ontos[10], the Wiki Machine[11]. .

OLiA and POWLA for Morpho-Syntax

Another example for such a reference model are the Ontologies of Linguistic Annotations (OLiA), which provide a reference framework to interlink several annotation systems and tagsets. OLiA allows to build applications against this ontological interface. See `http://nachhalt.sfb632.uni-potsdam.de/owl/` for more information. OLiA and POWLA[12] are developed in collaboration with Christian Chiarcos[13])). POWLA is a full-fledged Annotation model, which can be used to group the annotations into layers and structure whole documents. It can be used to provide clean modelling of Linguistic Annotations even if the annotations are complex. Linguistic annotation schemes for Corpora can be very complex and thus it is necessary to use a more elaborate way to represent annotations.

3.2 NIF Webservices

To achieve a true ad-hoc integration and flexibility, we can encapsulate the NLP tools in Webservices. The integration is then simply done by merging the resulting RDF statements, as we will show in the following example. Note that we have also implemented the scenario and the webservices are actually running and can be tested. For a complete documentation see the developers wiki[14].

3.2.1 Example concatenation

The SnowballStemmer in Figure 2.2 and Stanford Core Figure 2.3 can easily be integrated with NIF. The output models just need to be combined and put into

[2] `http://en.wikipedia.org/wiki/Knowledge_extraction`
[3] `http://dbpedia.org`
[4] `http://spotlight.dbpedia.org`
[5] `http://www.opencalais.com/node/9501`
[6] `http://www.zemanta.com/`
[7] `http://www.extractiv.com/demo.html`
[8] `http://www.alchemyapi.com`
[9] `http://fox.aksw.org`
[10] `ontos.com`
[11] `http://thewikimachine.fbk.eu/`
[12] `http://www.sfb632.uni-potsdam.de/~d1/paula/doc/`
[13] `http://www.ling.uni-potsdam.de/~chiarcos/`
[14] `http://code.google.com/p/nlp2rdf/wiki/NIFWebServices`

the same model. With Jena[15] the code would look like:

```
String param =
    "?input=My%20favorite%20actress%20is%20Natalie%20Portman!&type=text";
Model model =
    ModelFactory.createDefaultModel();
URL stemmer =
    new URL("http://nlp2rdf.lod2.eu/annotator-stanford/NIFStemmer"+param);
URL stanford =
    new URL("http://nlp2rdf.lod2.eu/annotator-stanford/NIFStanfordCore"+param);

model.read(
    new BufferedReader(
        new InputStreamReader(stemmer.openConnection().getInputStream())),
    null);
model.read(
    new BufferedReader(
        new InputStreamReader(stanford.openConnection().getInputStream())),
    null);
```

The resulting model merges naturally because the URIs are either the same or can be transformed. The result can be seen in Figure 3.1

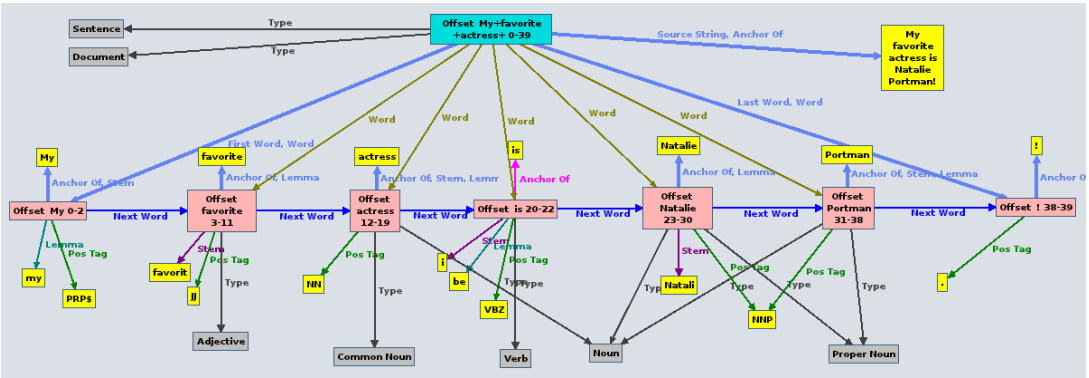

Figure 3.1: SnowballStemmer and Stanford combined.

[15] http://jena.sourceforge.net/

4 RoadMap and Future Work

4.1 Further Horizontal and Vertical Integration

For this prototype release we integrated NLP tools from diverse areas such as morphology (stemming), POS-tagging and Entity Disambiguation (FOX[1]). Future work will include a broader coverage of tools from the same areas. Especially the area of entity disambiguation is of high importance. We will study the parameters of all available tools and create a reference ontology. For other areas of NLP, UIMA and Gate will be integrated and the web service wrapping the Stanford Core Framework will receive more options to expose the multitude of languages supported. Ultimately, we will inform the developers and maintainers of these tools and provide the source code and support to integrate NIF-output directly into their projects.

4.2 Measurements of URI stability

Although we propose a method that seems to produce stable identifiers for strings embedded in documents, the question is still open how to measure and quantify this stability. Therefore, we will conduct measurements on a changing document collection, i.e. Wikipedia articles to see how stable context-hash identifiers are.

4.3 Community Feedback

During the project time of LOD2, feedback will collect feedback from consortium members and the Semantic Web community. This will guarantee to produce a solution that is generic enough to cope with numerous use cases and application scenarios. Although, several issues such as the context-hash URI naming approach and the ontologies have been discussed with many people and therefore received several iterations, further feedback from a wider variety of people will be required to finalize the NLP Interchange Format.

[1] `http://fox.aksw.org`

4.4 Versioning

After the publication of this deliverable, we will produce a version 1.0 of NIF, which will be stable and which can be implemented by the LOD2 consortium. During the implementation of NIF 1.0 feedback will be gathered and result in a draft for NIF 2.0, which will serve as a proposal for standardisation (W3C or ISO) .

Bibliography

[1] C. Chiarcos. An ontology of linguistic annotations. *LDV Forum*, 23(1):1–16, 2008.

[2] Martin Hepp, Daniel Bachlechner, and Katharina Siorpaes. Harvesting wiki consensus - using wikipedia entries as ontology elements. In Max Völkel and Sebastian Schaffert, editors, *Proceedings of the First Workshop on Semantic Wikis – From Wiki to Semantics, co-located with the 3rd Annual European Semantic Web Conference (ESWC 2006)*, Workshop on Semantic Wikis. ESWC2006, June 2006.

[3] Natarajan Kannan and Toufeeq Hussain. Live urls: breathing life into urls. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 879–880, New York, NY, USA, 2006. ACM.

[4] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[5] Kristina Toutanova and Christopher D. Manning. Enriching the knowledge sources used in a maximum entropy part–of–speech tagger. In *In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)*, pages 63–70, 2000.

[6] E. Wilde and M. Duerst. URI Fragment Identifiers for the text/plain Media Type. `http://tools.ietf.org/html/rfc5147`, 2008. [Online; accessed 13-April-2011].

[7] Erik Wilde and Marcel Baschnagel. Fragment identifiers for plain text files. In *Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, HYPERTEXT '05, pages 211–213, New York, NY, USA, 2005. ACM.

[8] K. Yee. Text-Search Fragment Identifiers. `http://zesty.ca/crit/draft-yee-url-textsearch-00.txt`, 1998. [Online; accessed 13-April-2011].