Computational Linguistics in the Netherlands Journal 4 (2014) 71-84 Submitted 06/2014; Published 12/2014

Building an NLP pipeline within a digital publishing workflow

Hans Paulussen* HANS.PAULUSSEN@KULEUVEN-KULAK.BE
Pedro Debevere** PEDRO.DEBEVERE@UGENT.BE
Francisco Bonachela Capdevila* FRANCISCO.BONACHELACAPDEVILA@KULEUVEN-KULAK.BE
Maribel Montero Perez* MARIBEL.MONTEROPEREZ@KULEUVEN-KULAK.BE
Martin Vanbrabant* MARTIN.VANBRABANT@KULEUVEN-KULAK.BE
Wesley De Neve** WESLEY.DENEVE@UGENT.BE
Stefan De Wannemacker* STEFAN.DEWANNEMACKER@KULEUVEN-KULAK.BE

* *iMinds-ITEC, KU Leuven, Kortrijk, Belgium*

** *iMinds-MMLab, UGent, Gent, Belgium*

Abstract

Outside the laboratory environment, NLP tool developers have always been obliged to use robust techniques in order to clean and streamline the ubiquitous formats of authentic texts. In most cases, the cleaned version simply consisted of the bare text discarded of all typographical information, tokenised in such a way that even the reconstruction of a simple sentence resulted in a displeasing layout. In order to integrate the NLP output within the production workflow of digital publications, it is necessary to keep track of the original layout. In this paper, we present an example of an NLP pipeline developed to meet the requirements of real-world applications of digital publications.

The NLP pipeline presented here was developed within the framework of the iRead+ project, a cooperative research project between several industrial and academic partners in Flanders. The pipeline aims at enabling automatic enrichment of texts with word-specific and contextual information in order to create an enhanced reading experience on tablets and to support automatic generation of grammatical exercises. The enriched documents contain both linguistic annotations (part-of-speech and lemmata) and semantic annotations based on the recognition and disambiguation of named entities. The whole enrichment process, provided via a web service, can be integrated into an XML-based production flow. The input of the NLP enrichment engine consists of two documents: a well-formed XML source file and a control file containing XPath expressions describing the nodes in the source file to be annotated and enriched. As nodes may contain a pre-defined set of mixed data, reconstruction of the original document (with selected enrichments) is enabled.

1. Introduction

In this paper, we introduce the iRead+ enrichment pipeline. This natural language processing (NLP) pipeline was developed as part of the iRead+ project, that investigated how e-texts can automatically be enriched with supporting information (pictures, additional articles, etc.) through a single framework that addresses the needs of a wide range of target audiences (ranging from general readers and language learners to children with reading problems such as dyslexia). The enrichment pipeline had to be developed in such a way that it could easily be integrated into the production workflow of a digital publication. The central idea behind the iRead+ project is the belief that enriching texts with supporting material makes for a more immersive, compelling and motivating reading experience. The reader gets information at his fingertips, and hence it is no longer necessary to search for extra information in another application.

Current solutions for e-text enrichment have two major drawbacks, however: (i) they heavily rely on manual programming, and (ii) no single solution can accommodate the needs of different target

audiences requiring different types of e-text enrichments. In response to those shortcomings, the iRead+ consortium firstly investigated the development of a text enrichment solution that enables a high degree of automation, thus removing an important barrier to the technology's cost-effective roll-out. Secondly, the team pursued the creation of a single framework that could accommodate the requirements of a variety of target audiences. In fact, first an enrichment engine (containing both an enrichment pipeline and an enrichment delivery service) was developed by the research teams, and then the feasibility of the enrichment webservice was tested in three different use cases, whereby three industrial partners[1] implemented each a use case specific demonstrator, showing how the enrichment pipeline can be used to select use case specific enrichments.

In the remainder of this paper, we will first explain why NLP tools are important in production environments, and why some adaptations are required (Section 2). The next two sections will give further details on the main topics of the enrichment pipeline: i.e. the linguistic annotation (Section 3) and the semantic annotation in the enrichment procedure (Section 4). In Section 5, we will explain the general architecture of the enrichment pipeline and the XML format of the enriched document, which is based on TEI specifications. This is followed by a description of a first test of the tool, consisting of the implementation of three demonstrators using the enrichment engine in three application specific ways (Section 6) and a general conclusion (Section 7).

2. NLP tools for production environments

The iRead+ project aimed at developing an enrichment engine that can be integrated in a real-world production environment such as a digital printing service of a publishing house. This approach is quite different from most existing NLP tools. Usually, the first task in processing texts in language technology consists in cleaning and even cleansing documents into raw text, so that the NLP tool is not blocked in linguistic processing. However, this implies that all markup is lost. In this sense, language technology can be compared to *lossy data* in compression technology, whereby the original data cannot be reconstructed from the compressed data. After preprocessing the data, only the content is left over, without any trace of the markup. For example, the string `"een <bold>rode</bold> fiets"` is transformed to `"een rode fiets"` after pre-processing. The markup `<bold>` is deleted and cannot be retrieved[2].

This processing approach was understandable in the context of a language laboratory, but it cannot be used in a production environment of a publishing house. For present-day production flows, it is important that extra processing, such as linguistic annotation and enrichment, can be directly integrated into the rest of the publication cycle. Therefore, NLP tools should be able to support textual data mixed with markup.

Compared to ten years ago, tools for language technology have become ubiquitous. One of the first NLP toolkits was GATE[3] (Cunningham et al. 2011), originally developed for English. Nowadays, most of them are usually presented as an NLP pipeline or an NLP toolkit, often distributed via a web interface. In most cases the input format is limited to raw text data. If HTML or XML is supported, the markup is usually commented out. Typical examples are PyPLN[4] (Coelho et al. 2013) and PSI-Toolkit[5] (Gralinski et al. 2013).

Probably, the first NLP tools using an XML pipeline, were introduced from about 2005. A good example is the LT-TTT package, originally presented in Grover et al. (2000)[6]. LT-TTT was specifically developed for English, although it can also be used for the development of other languages. A similar example for French, is MACAON (Nasr et al. 2010, Nasr et al. 2011). In both

1. The three firms involved are d!nk, Cartamundi and Sensotec.
2. This example is a simplification of the problem, since in this case it can be handled by offset annotation and appropriate mapping, but things can be more complicated.
3. https://gate.ac.uk/
4. http://pypln.org/
5. http://psi-toolkit.amu.edu.pl/
6. A more stable version was introduced in Grover and Tobin (2006).

MACAON and LT-TTT, raw text is first wrapped into an XML document, and for the rest of the pipeline, the XML format is used as chaining format between the different processes. Usually, an extra tool translates the verbose XML format back to ordinary text, but this is only for testing purposes. The advantage of the XML format is mainly a better control for the developers over the transfer of data between the different processes. However, the input format remains raw text: no markup is allowed.

Although the tools are assumed to be open for many languages, they are mainly limited to one language, or the adaptation of the tool requires specific knowledge not directly available. In theory, NLTK (Bird et al. 2009) is open for different languages, but it is unclear, for example, how to use NLTK for even simple preprocessing for French. Some tools are specially developed for different languages, such as the multilingual preprocessing toolkit described in van de Kauter et al. (2013). The NLP pipeline presented here is developed for annotating Dutch and French texts. The architecture is developed in such a way that extra languages can be added to the system, without disturbing the overall architecture of the pipeline.

However interesting the evolution of the NLP tools for the last years, most of them remain unidirectional, in the sense that they are able to annotate documents, but still discard markup in such a way that the annotated document cannot yet be returned back to the publishing production flow. In building the enrichment engine, we intended and succeeded in creating a first version of an NLP tool that does take into account the markup of the original document, and the demonstrators presented in Section 6, show how the enriched information can be inserted back into the original document.

3. Linguistic annotation

The enrichment engine has been developed to enrich Dutch and French texts. In this section we present the NLP tools used for the first part of the NLP pipeline, which consisted in the linguistic annotation: i.e. Part-of-Speech tagging and lemmatisation. For each language, a different tool was selected: TreeTagger for French and Frog for Dutch. Two issues required special attention. First of all, it was important to integrate the tools transparently into the workflow of the enrichment engine. But more importantly, some adaptations in pre- en post-processing were necessary to handle the mixed data, in order to keep track of linguistic content and markup.

3.1 Dutch

For Dutch, we opted for Frog[7], a tool based on memory-based language processing (Daelemans and Bosch 2005). In fact, Frog is an integration of memory-based NLP modules developed for Dutch in Tilburg and Antwerp. This tool is considered the default tagger for the annotation of Dutch texts, and has been used for different projects, including projects of the STEVIN programme (Spyns and Odijk 2013).

Notwithstanding the overall quality of the annotations of Frog, the tool had problems in handling markup tags, so that special post-processing was required. For example, a markup tag, such as `<bold>` will be segmented into three tokens: i.e. "`<`", "`bold`" and "`>`", as illustrated in Table 1, where a text of 5 tokens is presented as a text of 10 tokens. Since the segmentation and the annotation of the markup tags is consistently the same, it is not so difficult to fuse the segmented elements into one token again, after the linguistic annotation. But it remains unclear whether the segmented tokens have any influence on the annotation of the surrounding tokens. This should be further analysed.

7. http://ilk.uvt.nl/frog/

	Expected	Obtained
1	een	een
2	<bold>	<
3	rode	bold
4	</bold>	>
5	fiets	rode
6		<
7		/
8		bold
9		>
10		fiets

Table 1: Tokenisation of text containing SGML tags

3.2 French

For French, we used TreeTagger[8]. Instead of using the basic small PoS tag set, covering only 33 labels, the richer GRACE tag set (Paroubek 2000) was used[9]. Therefore, another language model—called parameter file in TreeTagger terminology—had to be selected. A language model based on the GRACE evaluation program (Adda et al. 1998) was provided by the LIMSI research team, who had created a corpus using the GRACE tag set (Allauzen and Bonneau-Maynard 2008)[10]. However, since that LIMSI language model had not been trained for lemmatised forms, the tagger has to be run twice, first with the original language model (containing also lemmatised forms), then with the LIMSI language model (containing no lemmatised forms). The output is fused afterwards.

token	TT	GRACE	lemma
Il	PRO:PER	Pp3msn-	il
a	VER:aux:pres	Vaip3s-	avoir
lu	VER:pper	Vmps-sm	lire
deux	NUM	Dk-mp-	deux
livres	NOM	Ncmp	livre
pendant	PRP	Sp	pendant
la	DET:ART	Da-fs-d	le
dernière	ADJ	Ao-fs	dernier
semaine	NOM	Ncfs	semaine
.	SENT	F	.

Table 2: Sample TreeTagger basic tag set and GRACE tag set

Table 2 illustrates the differences between the general tag set of TreeTagger and the more fine-grained GRACE tag set. For example, the word *livres* (*books*) is only considered a noun (NOM) for TreeTagger, whereas GRACE gives more detailed information: i.e. a common noun, masculine and plural (Ncmp). The use of a reduced tag set is not uncommon in French taggers. Like TreeTagger,

8. The tool is described in (Schmid 1994, Schmid 1995). This approach was motivated by the fact that the basic PoS tag set of TreeTagger is rather limited for French. The French TreeTagger tagset can be found at: http://www.ims.uni-stuttgart.de/~schmid/french-tagset.html.
9. A similar approach was followed in the creation of the Dutch Parallel Corpus (Paulussen et al. 2013).
10. There are different versions of the GRACE tag set. The basic tag set is described in (Rajman et al. 1997).

MACAON and MElt also use a limited tag set[11]. This is mainly due to the fact, that in language technology, the subcategories are often less important.

The distinction between the basic tagging set and the extended set containing categories and subcategories is especially apparent in the case of verbs, which have the most complex morphology: e.g. the verb form *lu* (derived from *lire* (*read*)) is considered a past participle in the basic TreeTagger tagging set (`VER:pper`), whereas GRACE gives more details: i.e. `Vmps-sm` refers to a main verb form of the past participle in the singular and masculine form. In most applications the main category suffices for further processing, but this is not the case for applications involving computer assisted language learning (CALL). The additional information on subcategories were important in developing the exercise generator as part of the second demonstrator (See Section 6).

4. Semantic annotation

The output of the linguistic annotation is used as input for the semantic annotation. In this section, we describe the functioning of the semantic annotation engine. This engine is responsible for named entity detection, multi-word expression detection, named entity disambiguation, and dictionary linking. We first describe briefly the knowledge base used by the enrichment module. Then we explain the named entity detection procedure and the multi-word expression detection. This is followed by a description of the named entity disambiguation module, which is the core of the semantic annotation engine. Finally, we describe the dictionary linking module.

4.1 Knowledge base

Named entity disambiguation, dictionary linking, and enrichment make use of several data sets. These data sets together constitute the knowledge base of the enrichment pipeline. As the back-end of the enrichment pipeline is based on Semantic Web technologies, we use a Resource Description Format (RDF) representation for the data sets selected. The RDF data are stored in a triple store and can be queried using the SPARQL Protocol and RDF Query Language (SPARQL).

Figure 1 gives an overview of the data sets selected. A data set represented with a dashed circle means that we first had to create an RDF representation for that data set. A data set represented with a solid circle indicates that we used the corresponding data set from the LOD cloud[12]. A line between two data sets indicates that interlinking is present between resources of the two data sets (e.g. through the use of common interlinking/mapping properties such as `owl:sameAs` or `rdfs:seeAlso`).

4.2 Named entity detection

Named entities (NEs) are mainly *typed proper nouns*. Whereas common nouns are typically stored in a lexicon, proper nouns are stored in an encyclopaedic dictionary. The semantic annotation engine we developed uses the following common types of NEs: PER (person), ORG (organisation), LOC (location) and the rest category MISC (miscellaneous names). The study of Desmet and Hoste (2014) describes a more fine-grained set of NE categories (based on the analysis of data from the Dutch SoNaR[13] corpus), but the analysis is restricted to the recognition of NEs and their types; no NE disambiguation is carried out.

For named entity detection, we make use of the state-of-the-art Stanford Named Entity Recognition (NER) tool (Finkel et al. 2005), constructing both a Dutch and French language model for usage with this tool. For the construction of the Dutch language model, we made use of the Conference on Natural Language Learning (CoNLL) 2002 Dutch training data, as made available for

11. MElt, a tagger developed by the Alpage team (Denis and Sagot 2012), uses 28 tags which are listed in Crabbé and Candito (2008).
12. http://lod-cloud.net
13. http://lands.let.ru.nl/projects/SoNaR/

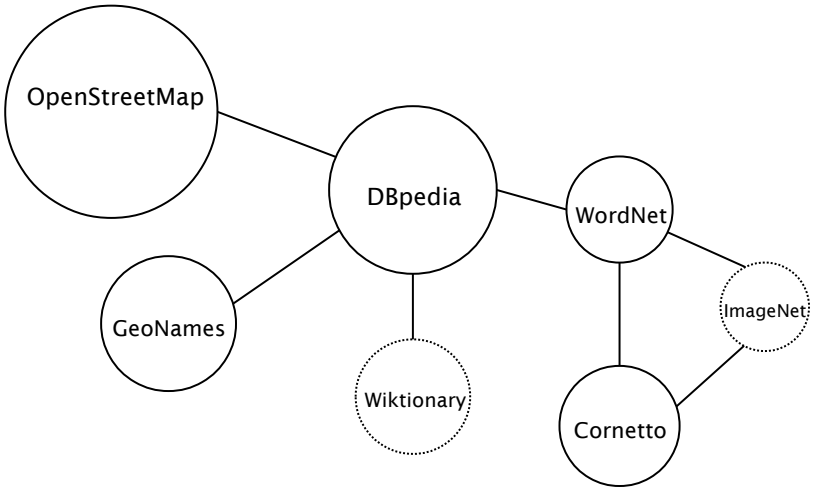

Figure 1: Data sets selected

the task on language-independent NER[14]. For the construction of the French language model, we manually created training data from a corpus of French news articles provided by Roularta Media Group (RMG), one of the industrial partners in the iRead+ project[15].

4.3 Multi-word expression detection

For multi-word expression detection, we make use of the LingPipe implementation of the dictionary-based string matching algorithm proposed by Aho and Corasick (1975). The time complexity of this algorithm is linear in the length of the patterns plus the length of the text to be searched plus the number of output matches. To construct a dictionary of multi-word expressions, we collected such expressions from data sets such as DBpedia[16], Wiktionary[17], and Cornetto[18].

4.4 Named entity disambiguation

For named entity disambiguation, we follow a two-step approach. The first step consists of collecting a set of candidate resources from the knowledge base for each detected NE. The second step consists of computing a disambiguation score for each candidate resource collected. The resource having the highest score is then selected as the matching resource[19].

To that end, we make use of Bayesian inference, modeling the disambiguation score of a candidate resource as a posterior probability that is derived from a prior probability and a likelihood. We compute the prior probability of a candidate resource as the number of resources that link to the candidate resource in the knowledge base, divided by the total number of resources that link to all candidate resources. This can be expressed as follows:

$$prior(r) = \frac{link\_in\_count(r)}{\sum_{r' \in C} link\_in\_count(r')}, \quad (1)$$

14. http://www.cnts.ua.ac.be/conll2002/ner/data/
15. Due to copyright reasons this training data is not publicly available
16. http://dbpedia.org
17. http://www.wiktionary.org/
18. http://www2.let.vu.nl/oz/cltl/cornetto/
19. Note that this approach assumes that for each NE that is mentioned in a text, a corresponding resource is present in the knowledge base. As this is often not the case, detection of these so called NIL mentions is a possible improvement of the disambiguation algorithm.

with r denoting a candidate resource for the NE that is currently the subject of disambiguation, C denoting the set of all candidate resources collected for the NE under consideration, r' denoting a candidate resource in C, and $link\_in\_count(r)$ denoting the number of links pointing to r.

To compute the likelihood of a candidate resource, we construct a graph where each node in the graph represents a candidate resource that was collected for a detected NE. In addition, we extend this graph with additional nodes that represent words appearing within a context window around the NE that is disambiguated (in our implementation, we set the context window to the sentence in which the NE appears). The links between the nodes in the graph are the links that are present in the knowledge base between these resources. The likelihood of a candidate resource for a detected NE is then computed based on the number of links between the resource and other resources in the graph (Note that we do not take into account links between candidate resources for the NE that is currently being disambiguated). This can be expressed as follows:

$$link\_count(r) = link\_in\_count(r) + link\_out\_count(r) + 1, \tag{2}$$

$$link\_count\_score(r) = link\_count(r)^{\alpha}, \tag{3}$$

$$likelihood(r) = \frac{link\_count\_score(r)}{\sum_{r' \in C} link\_count\_score(r')}. \tag{4}$$

Note that $link\_count(r)$ is the sum of the number of incoming and outgoing links for the candidate resource r in the context graph, and where this sum is Laplace smoothed. Moreover, $link\_count\_score(r)$ applies an exponential weight to $link\_count(r)$. In our implementation, we set α to two.

Using Bayesian inference, we can subsequently compute the disambiguation score of a candidate resource as follows:

$$p(r) = \frac{likelihood(r)prior(r)}{\sum_{r' \in C} likelihood(r')prior(r')}. \tag{5}$$

Finally, we select the candidate resource with the highest value for equation (5) as the disambiguation for the NE under consideration.

4.5 Dictionary linking

For words that are not considered to be NEs, the semantic annotation engine generates links to dictionary entries. To that end, the lemma and PoS information outputted by the linguistic annotation engine are used to consult a so-called dictionary index. This dictionary index consists of resources present in the knowledge base that have information available that is typically found in a dictionary. For Dutch, this dictionary index consists of the Dutch Wiktionary dataset and Cornetto, augmented with surface forms from the Dutch chapter of DBpedia. For French, this dictionary index consists of the French Wiktionary dataset, augmented with surface forms from the French chapter of DBpedia.

5. General architecture and XML format

The iRead+ enrichment engine consists of two parts. First of all, there is the enrichment pipeline which creates an enriched document, containing so-called *subjecturi*'s representing unique keys used for retrieving different types of enrichments, as will be explained further. Secondly, there is an enrichment delivery service that translates the subjecturi into the required enrichment (e.g. an image or textual information). This section first describes the general architecture of the web services, and then explains the XML format of the enrichment document. This section is concluded by a description of the enrichment delivery service.

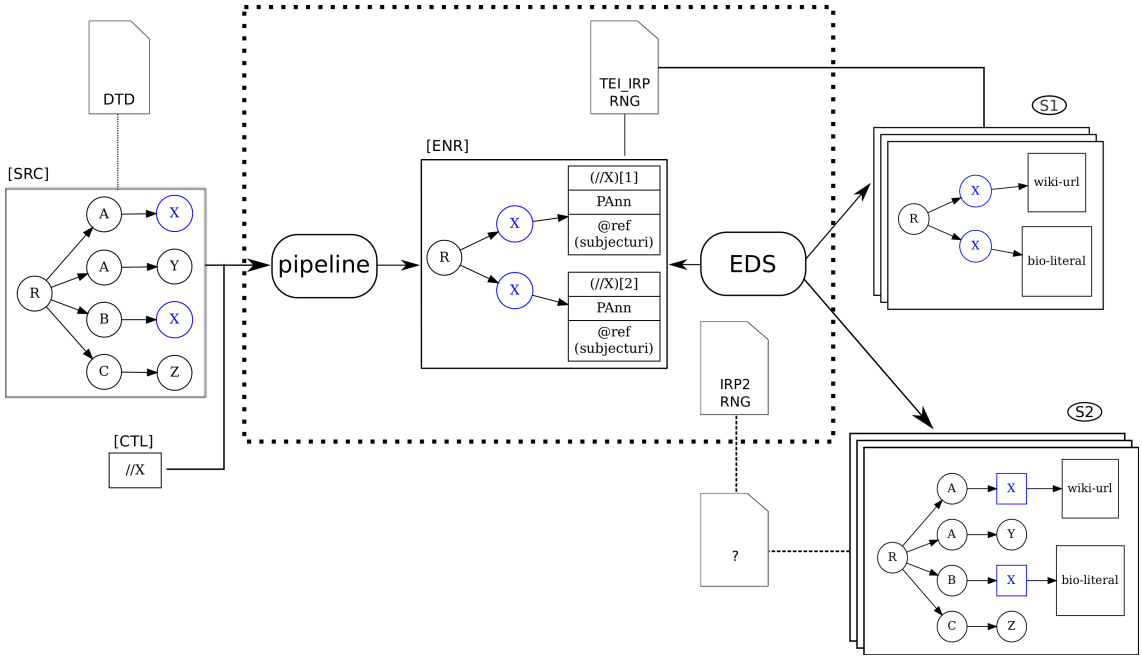

Figure 2: NLP pipeline and enrichment delivery service workflow

5.1 General architecture

The enrichment engine consists of an enrichment pipeline and an enrichment delivery service used for resolving the enrichment keys (i.e. the subjecturi's) in the enriched document. Moreover, two schemas have been created to parse an enriched document. Figure 2 gives an overview of the architecture of the enrichment engine, where a clear distinction is made between the core elements of the web services workflow (represented by the dotted box in the middle of the figure) and all other resources used or developed by the application developers.

The core elements consist of a pipeline creating an enriched document (ENR), and an enrichment delivery service (EDS) which developers can call to resolve the enrichment keys. Next, there are two schemas (TEI_IRP.RNG and IRP2.RNG), that will be explained further in this section.

The enrichment pipeline first of all enables the creation of an ENR document, starting from any well-formed XML document, the so-called source document (SRC), in combination with a control document (CTL), which contains XPath expressions referring to all possible nodes in the source document to be annotated. As an illustration, we see in Figure 2 that the control document selects all `X` elements in the source document. All other nodes are left untouched. Figure 3 shows a more realistic example, whereby all XPath expressions allowed for further processing are stored in `<xpathline>` elements. In this example, enrichment is allowed for all `<title>` and `<p>` elements.

The enrichment pipeline consists of two parts: first, there is the linguistic annotation process (covering part-of-speech tagging and lemmatisation) followed by the semantic annotation process, whereby named entities are detected and disambiguated. The enrichment pipeline allows any well-formed XML document as input (and is therefore structure-agnostic), which makes it flexible for various input formats. Mixed data (i.e. textual data interspersed with markup) are allowed, but at present this is limited to a restricted set of markup elements (such as `<bold>`, `<italic>` and

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ctldoc>
  <srcgrp>itec</srcgrp>
  <ctldocid>ctldocid-irp-ctl</ctldocid>
  <xpathlines>
    <xpathline>//title</xpathline>
    <xpathline>//p</xpathline>
  </xpathlines>
</ctldoc>
```

Figure 3: Example of control document

`<underline>`[20]). Instead of stripping the markup elements, the preprocessor retains the markup, so that it can be integrated in the output.

The ENR document contains for each annotated node (selected from the original document) both references to the original document node (expressed in XPath patterns) and one or more *subjecturi* attributes. A *subjecturi* represents a resource present in the knowledge base (further explained in section 4.1). This unique identifier is then used to retrieve enrichments via the enrichment delivery service (EDS)[21]. The ENR document in Figure 2 contains two nodes, referring to the first and second X element of the original document[22]. Each node contains extra elements, such as <PAnn> (which will be explained further) and a ref attribute indicating the subjecturi.

Thanks to the enrichment delivery service (EDS), the application developer can translate the subjecturi into a specific enrichment: e.g. a Wikipedia link or a biographic description of a particular person. At the moment, there are 26 enrichment types: e.g. abstract, definition, homepage, wikilink. This list can be extended, if need be. The type of enrichments possible depends on the subject type, mainly grouped between different types of named entities and dictionary entries. For example, the enrichment biography is typically for a person, whereas definition and synonyms are related to dictionary entries. Table 3 shows some examples of enrichment types, followed by the kinds of subjects they can occur with.

Depending on the type of application required, the application developer can create two basic types of documents. In a basic application, it can be sufficient for the application developer to simply use the ENR document and translate the subjecturi to a particular resolved enrichment (See box S1 in Figure 2). However, if he wants to fuse the original XML document and the enrichments stored in the ENR document and thus create a new fully enriched document, he can do so by using the XPath addresses in the ENR document, in order to select the correct node in the original XML document (S2 in Figure 2).

It is in this respect that we have created two schemas, which are part of the core elements in the above architecture figure: the schema TEI_IRP.RNG is developed to parse an entire ENR document, and the schema IRP2.RNG is developed to parse only the main element of an ENR document: i.e. <PEnrich>. This second schema can be used by developers who want to integrate the annotated nodes into a new document based on the original input document. It is up to the application developer to create a new schema based on the original DTD of the SRC document and schema IRP2.RNG to validate document S2 in Figure 2[23].

20. This set can easily be extended to a generic list of elements. In order to handle attributes as well, an extra filter is required.
21. See Section 5.3
22. This is indicated by the XPath expressions `(//X)[1]` and `(//X)[2]`.
23. In Figure 2, this new schema is indicated by a question mark, since the engine has no knowledge of such a schema. The enrichment engine has only knowledge of the ENR document.

5.2 XML format

As explained above, the enrichment engine takes as input any well-formed XML file. This requirement gives full flexibility to document providers and application developers, who themselves are responsible for the validity of the input document, using a DTD or another schema. It also has the advantage that the enrichment engine is agnostic of the structure of the input document. It only selects nodes from the input file, matching the XPath patterns in the control document. On the other hand, the intermediate files and the final enrichment document (ENR) must match with a valid schema specifically developed for the engine. This requirement is important if we want to analyse any problem concerning the structure of the XML documents generated by the enrichment engine. Therefore, we created the TEI_IRP.RNG schema, as explained in the previous subsection. In this subsection, we explain how we decided on the XML format and how the schemas were created.

After comparing the main XML formats generally used for structuring and annotating linguistic data, we decided to follow the de facto standard of TEI (Text Encoding Initiative)[24]. The decision was mainly inspired by the flexible format of the TEI modules and the practical scheme editor tool ROMA[25], which is based on the generic notion of the ODD document, which stands for *One Document Does it all* (Burnard 2013). TEI in itself is not a schema, but a set of modules describing different types of documents. Depending on the type of document to be validated, certain modules are selected, and a schema is created. The ROMA tool is an editor which facilitates the creation and modification of schemas based on the TEI modules[26].

An ODD document is a schema-independent language which describes the specifications of the structure of an XML document. The ODD can be compiled and translated into a number of schema formats, such as DTD, XML Schema or RELAX NG. An ODD itself is a TEI document, and can thus be validated accordingly. The ROMA tool enables easy creation and modification of ODD files and compilation of different schema formats. Moreover, documentation can be directly integrated. However flexible the tool may be, there are some drawbacks related to the web interface, so that certain adaptations have to be done directly into the code of the ODD document. Moreover, following the recommendations of the ODD developers, we created a separate namespace in order to avoid conflicts with future TEI specifications for all elements we created specifically for the iRead+ project.

Figure 4 gives an example of a <PEnrich>[27] element, which is the core element of an ENR document. The main body of an ENR document contains a set of <PEnrich> elements. A <PEnrich> begin element contains a path attribute as standoff pointer indicating the address of the element in the original source document, as illustrated in the following example, where the attribute path refers to the first paragraph element of the first section element in the original document:

```
<PEnrich path="(//section/paragraph)[1]" n="irp-1"> ... </PEnrich>
```

Each <PEnrich> element in turn contains the following three elements: a <PBase> element showing a cleaned version of the original node, a <PAnn> element containing the linguistic annotation for each token, and a <PNerd> element containing the named entities[28]. Figure 4 is a minimal example of the <PEnrich> element, lacking a number of attributes, including the offset attributes and the lemmaRef attributes. Although the <PEnrich> element shows some redundancy (E.g. the named entities are stored in both <PAnn> and <PNerd>), this approach can be cost-efficient, depending on the application used.

24. Stührenberg gives an overview of a number of important competitors of XML formats for linguistic annotation, including TEI, CES, and newer formats: e.g. FS, LAF, SynAF, MAF (Stührenberg 2012).
25. http://tei.oucs.ox.ac.uk/Roma/
26. The predecessor of the ROMA tool was called the Pizza Chef, referring clearly to the fact that you can mix any module and seasoning of your liking. The Pizza Chef was developed for TEI P4. ROMA is available for TEI P5.
27. Strictly speaking, we should mention the namespace (i.e. <irp:PEnrich>), but for reasons of legibility, the namespace is left out.
28. <PAnn> stands for annotation, whereas <PNerd> refers to Named Entity Recognition and Disambiguation.

```
<irp:PEnrich irp:path="(//section/paragraph)[1]" n="irp-1">
  <PBase xmlns="http://kuleuven-kulak.be/itec/ns/irp/">J'ai vu un
     éléphant <italic>bleu</italic> dans mon jardin à Paris. "C'est phantastique",
     disait Jean.</PBase>
  <irp:PAnn>
    <s irp:s_id="1">
      <w ana="Pp1msn-" lemma="je">J'</w>
      <w ana="Vaip1s-" lemma="avoir">ai</w>
      <w ana="Vmps-sm" lemma="voir">vu</w>
      <w ana="Da-ms-i" lemma="un">un</w>
      <w ana="Ncms" lemma="éléphant">éléphant</w>
      <w ana="-SGML-" lemma="-SGML-">&lt;italic&gt;</w>
      <w ana="Afpms" lemma="bleu">bleu</w>
      <w ana="-SGML-" lemma="-SGML-">&lt;/italic&gt;</w>
      <w ana="Sp" lemma="dans">dans</w>
      <w ana="Ds1mss-" lemma="mon">mon</w>
      <w ana="Ncms" lemma="jardin">jardin</w>
      <w ana="Sp" lemma="à">à</w>
      <irp:ne ref="http://fr.dbpedia.org/resource/Paris" irp:confidence="0.99" type="LOC">
        <w ana="Npms" lemma="Paris">Paris</w>
      </irp:ne>
      <w ana="F" lemma=".">.</w>
    </s>
    <s irp:s_id="2">
      <w ana="F" lemma="&quot;">"</w>
      <w ana="Pd-ms--" lemma="ce">C'</w>
      <w ana="Vmip3s-" lemma="être">est</w>
      <w ana="Afpms" lemma="&lt;unknown&gt;">phantastique</w>
      <w ana="F" lemma="&quot;">"</w>
      <w ana="F" lemma=",">,</w>
      <w ana="Vmii3s-" lemma="dire">disait</w>
      <irp:ne ref="http://fr.dbpedia.org/resource/Jean" irp:confidence="0.96" type="MISC">
        <w ana="Npms" lemma="Jean">Jean</w>
      </irp:ne>
      <w ana="F" lemma=".">.</w>
    </s>
  </irp:PAnn>
  <irp:PNerd>J'ai vu un éléphant
    <italic xmlns="http://kuleuven-kulak.be/itec/ns/irp/">bleu</italic> dans mon jardin
    à <irp:ne ref="http://fr.dbpedia.org/resource/Paris" irp:confidence="0.99"
    type="LOC">Paris</irp:ne>. "C'est phantastique", disait <irp:ne
    ref="http://fr.dbpedia.org/resource/Jean" irp:confidence="0.96"
    type="MISC">Jean</irp:ne>.
  </irp:PNerd>
</irp:PEnrich>
```

FIGURE 4: Example of <PEnrich> element

Note that the XML format is open for further processing, so that, for example, the FoLiA[29] format can be used if required (van Gompel and Reynaert 2013). Although inspired by TEI, the FoLiA format does not follow the TEI guidelines strictly, but it is nevertheless a transparent format for linguistic annotations. If integration of markup is required, then probably the FoLiA format needs some extra adaptations.

One could compare the TEI based schemata proposed for the enrichment engine with similar projects, such as the Analyzed Layout and Text Object (ALTO) XML Schema[30]. The main difference between ALTO and the ENR schemata is related to the fact that ALTO is mainly a format for storing layout information and OCR recognised text of pages. Basically, ALTO starts from the image of a printed text, whereas the enrichment engine starts off from XML-based documents. Therefore, ALTO is more appropriate for adding annotation layers to archived documents.

5.3 Enrichment delivery service

As already mentioned, the ENR document contains the identifiers (*subjecturis*) for all detected and disambiguated NEs and for words that have a corresponding resource in the knowledge base. A collection of RESTful web services, the so-called enrichment delivery service, was developed to enable the retrieval of specific enrichments for a given resource.

In order to obtain a specific enrichment, a GET request can be made with the resource URI and requested enrichment type (as illustrated in Table 3) as parameters. The response is either XML or JSON, depending on the requested format by the client using content negotiation. As not all resources in the knowledge base have information available for all enrichment types, an additional web service allows to obtain information about which enrichment types a given resource currently supports.

Note that, as the knowledge base consists of several data sets, different ontologies are in use. To prevent application developers from having to know different ontologies in order to obtain enrichments for a particular resource, we defined a uniform Application Programming Interface (API). That way, application developers making use of the API have a uniform view on the knowledge base. When a user makes an API call, this call is internally translated to a specific SPARQL query. Using this architecture, we can add new data sets to the knowledge base without users having to know about the details of the data set such as the ontology and the SPARQL queries needed to obtain a specific enrichment.

6. Demonstrators

The iRead+ project not only consisted of the development of the enrichment engine. The technology was also made operational in three different reading applications (three demonstrators), which were developed by application developers for three different audiences: the general reader, the learning reader and the struggling reader. All demonstrators were developed specifically to enrich written text on a tablet computer. This enrichment consisted in making reading on tablet more interactive by providing readers with the possibility to access extra information while reading. In what follows, we discuss the three reading applications.

The iRead+ application for the general reader allows the general reader to read a magazine on a tablet computer, and get directly extra information on the named entities found in the article. The big advantage for the reading experience (i.e. reading magazines on tablet) is that the reader no longer has to access extra information via another app and can therefore be considered an inclusive application. In this demonstrator, the focus was on the extraction of named entities from the ENR document. First the ENR document was created and stored on the tablet computer. Then, whenever the user selected a named entity (e.g. a toponym), the enrichment delivery service was

29. http://proycon.github.io/folia/
30. http://www.loc.gov/standards/alto/

Enrichment type	Description	Named Entity (general)	Named Entity (person)	Named Entity (location)	Dictionary entry / MWE
`abstract`	Short description of the subject	✓	✓	✓	
`antonyms`	List of antonyms of the subject				✓
`biography`	Compound of enrichment types typical for a biography		✓		
`birthdate`	Date of birth of the subject		✓		
`definition`	Definition of the subject				✓
`derivedwords`	List of derived words of the subject				✓
`examplesentence`	Example sentence using the subject				✓
`geocoordinates`	Position on earth of the subject			✓	
`homepage`	URL of the homepage of the subject	✓	✓	✓	
`hypernyms`	List of hypernyms of the subject				✓
`image`	URL of an image of the subject	✓	✓	✓	✓
`label`	Default label of the subject	✓	✓	✓	✓
`pronunciation`	Phonetic transcription (IPA) and/or sound recording (Uri) of the subject				✓
`relatedwords`	List of related words of the subject				✓
`wikilink`	Weblink to a Wikipedia/ Wiktionary type of information about the subject	✓	✓	✓	✓

Table 3: Examples of enrichment types

called to return the required enrichment. For instance, by clicking on a toponym in the text, a short summary (based on Wikipedia material) was shown and images were made available within the reading application.

Second, the project also created a reading application which consisted of both a reading and an exercise environment and was destined for Dutch-speaking learners of French. In this application, the linguistic annotations supplied the main enrichments, although extra information on named entities could also be accessed (cf. application for general readers). While reading, learners using this application could access extra linguistic information. For instance, learners could click on a verb in order to receive information on the tense and infinitive. Similarly, when clicking on a noun, information on the gender of the substantive was provided in the enrichment. This application also provided an *exercise* environment. On the basis of the linguistic annotation, the exercise environment (i.e. exercise generator) automatically created exercises on the basis of a series of predefined morphosyntactic topics (e.g. verb conjugation). For each exercise topic (e.g. Present tense), three modes were available: an exploration mode, an exercise mode, and a play mode. The objective of the exploration mode is to draw students' attention to the linguistic forms while they process written input for meaning. For instance, learners are asked to click on the target forms (e.g. all verb forms in present tense) and receive immediate feedback while doing so. Next, the exercise mode is offered in which on the fly exercises are created (multiple choice, or fill gaps) focusing on the same morphosyntactic topics. This mode can be considered more difficult than the previous one since learners are asked to provide the input themselves. Finally, the play mode offers similar exercises as in the second mode but this time with a time constraint in order to challenge the learner.

The enrichment engine was also deployed to create a reading tool for struggling readers (e.g. persons suffering from dyslexia). In this reading tool, readers could ask to highlight the word categories in different colours, in order to easily recognise word groups. Technically, this use case was a bit different from the others, because the developers started from PDF files, containing lots of illustrations. In order to align enrichments with the original texts in the PDF files, the developers first converted the PDF file to XML, keeping track of the original character offsets[31]. After processing the XML file, the enrichments were displayed on an annotation layer presented over the existing PDF layout.

The three use cases have shown that the enrichment engine can be dynamically used in different contexts where content and layout play an important role. The extra service call for resolving the enrichments did not result in a delay. The enrichments can either be called on-the-fly, or they can be collected beforehand, so that a resolved document is created. Both approaches have been used successfully in the demonstrators. The three proofs of concept have shown that each application was capable of enriching the sample documents and visiualize the enrichments according to the requirements of the application.

7. Conclusion

The iRead+ project has shown in which way NLP tools together with semantic tools can be used in a publication production pipeline, without destroying the original documents. Although the enrichment engine has been developed as a proof-of-concept, and is therefore not yet optimised for scalability, the demonstrators have shown that the use of a web service is an approach worth to be explored further. Within this project, the focus was on testing the feasibility of such a tool. The next step will consist in evaluating the quality of the enrichments, on the basis of a comparison of the output with a gold standard. We also consider further optimisation of the different tools, including improving the semantic coverage of the knowledge base, and improvement of the disambiguation algorithm.

31. In the two other demonstrators, no conversion was required, because the original document was already structured in XML

8. Acknowledgement

The iRead+ project was financed as an iMinds ICON project for a duration of 2 years (2011-2013). iMinds is the digital research center and business incubator for Flanders, Belgium. Special thanks goes to Aleksandra Miletic who—as a trainee of Lille3—has implemented the TEI schemata. We also would like to thank the anonymous reviewers for their comments and constructive suggestions, that have helped us to improve the quality of this article.

References

Adda, Gilles, Joseph Mariani, Josette Lecomte, Patrick Paroubek, and Martin Rajman (1998), The GRACE french part-of-speech tagging evaluation task, *Proceedings of the First International Conference on Language Resources and Evaluation (LREC)*, pp. 433–441.

Aho, Alfred V. and Margaret J. Corasick (1975), Efficient string matching: An aid to bibliographic search, *Commun. ACM* **18** (6), pp. 333–340.

Allauzen, Alexandre and Hélène Bonneau-Maynard (2008), Training and evaluation of POS taggers on the french MULTITAG corpus, *LREC*, European Language Resources Association.

Bird, Steven, Ewan Klein, and Edward Loper (2009), *Natural Language Processing with Python*, 1st ed., O'Reilly Media, Inc.

Burnard, Lou (2013), Resolving the durand conundrum, *Journal of the Text Encoding Initiative*.

Coelho, Flávio Codeço, Renato Rocha Souza, Álvaro Justen, Flávio Amieiro, and Heliana Mello (2013), PyPLN: a distributed platform for natural language processing, *CoRR*.

Crabbé, Benoît and Marie Candito (2008), Expériences d'analyse syntaxique statistique du français, *Traitement automatique des langues naturelles - TALN 2008*, Avignon, France.

Cunningham, Hamish, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters (2011), *Text Processing with GATE (Version 6)*. http://tinyurl.com/gatebook.

Daelemans, Walter and Antal van den Bosch (2005), *Memory-based language processing*, Studies in natural language processing, Cambridge University Press.

Denis, Pascal and Benoît Sagot (2012), Coupling an annotated corpus and a lexicon for state-of-the-art POS tagging, *Language Resources and Evaluation* **46** (4), pp. 721–736.

Desmet, Bart and Véronique Hoste (2014), Fine-grained dutch named entity recognition, *Language Resources and Evaluation* **48** (2), pp. 307–343.

Finkel, Jenny Rose, Trond Grenager, and Christopher Manning (2005), Incorporating non-local information into information extraction systems by gibbs sampling, *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 363–370.

Gralinski, Filip, Krzysztof Jassem, and Marcin Junczys-Dowmunt (2013), PSI-toolkit: A natural language processing pipeline, *Computational Linguistics - Applications*, pp. 27–39.

Grover, Claire and Richard Tobin (2006), Rule-based chunking and reusability, *In Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC)*.

Grover, Claire, Colin Matheson, Andrei Mikheev, and Marc Moens (2000), LT TTT - a flexible tokenisation tool, *In Proceedings of Second International Conference on Language Resources and Evaluation*, pp. 1147–1154.

Nasr, Alexis, Frèdèric Bèchet, and Jean-François Rey (2010), MACAON: Une chaîne linguistique pour le traitement de graphes de mots, *Traitement Automatique des Langues Naturelles*, Montrèal.

Nasr, Alexis, Frédéric Béchet, Jean-François Rey, Benoît Favre, and Joseph Le Roux (2011), MACAON: An NLP tool suite for processing word lattices, *ACL (System Demonstrations)*, The Association for Computer Linguistics, pp. 86–91.

Paroubek, Patrick (2000), Language resources as by-product of evaluation: The MULTITAG example., *LREC*, European Language Resources Association, pp. 151–154.

Paulussen, Hans, Lieve Macken, Willy Vandeweghe, and Piet Desmet (2013), Dutch parallel corpus: a balanced parallel corpus for dutch-english and dutch-french, Essential speech and language technology for Dutch: results by the STEVIN-programme, Springer, pp. 185–199.

Rajman, Martin, Josette Lecomte, and Patrick Paroubek (1997), Format de description lexicale pour le français. Partie 2: Description morpho-syntaxique, GRACE GTR-3-2.1, Technical report, EPFL & INaLF.

Schmid, Helmut (1994), Probabilistic part-of-speech tagging using decision trees, *Proceedings of International Conference on New Methods in Language Processing*, Manchester.

Schmid, Helmut (1995), Improvements in part-of-speech tagging with an application to german, *Proceedings of the ACL SIGDAT-Workshop*, Association for Computational Linguistics, Dublin, Ireland.

Spyns, Peter and Jan Odijk, editors (2013), *Essential Speech and Language Technology for Dutch*, Theories and Applications of Natural Language Processing, Springer.

Stührenberg, Maik (2012), The TEI and current standards for structuring linguistic data, *Journal of the Text Encoding Initiative*.

van de Kauter, Marjan, Geert Coorman, Els Lefever, Bart Desmet, Lieve Macken, and Véronique Hoste (2013), LeTs preprocess: The multilingual LT3 linguistic preprocessing toolkit, *Computational Linguistics in the Netherlands Journal* **3**, pp. 103–120.

van Gompel, Maarten and Martin Reynaert (2013), FoLiA: A practical XML format for linguistic annotation - a descriptive and comparative study, *Computational Linguistics in the Netherlands Journal* **3**, pp. 63–81.