FP7-ICT-2011.4.2
Contract no.: 288342
www.xlike.org

Deliverable D2.1.1

Shallow linguistic processing prototype

Editor:	Marko Tadić, UZG
Author(s):	Marko Tadić, UZG; Željko Agić, UZG; Božo Bekavac, UZG; Xavier Carreras, UPC; Lluís Padró, UPC; Tadej Štajner, JSI; Esteban García-Cuesta, iSOCO; Juanzi Li, THU; Zhixing Li, THU; Alex Zhang, THU
Deliverable Nature:	Prototype
Dissemination Level: (Confidentiality)[1]	Public (PU)
Contractual Delivery Date:	M6
Actual Delivery Date:	M6.5
Suggested Readers:	All partners using XLike Toolkit.
Version:	1.0
Keywords:	shallow linguistic processing, pipeline, tokenisation, lemmatisation, stemming, POS-tagging, MSD-tagging, NERC, named entities

[1] Please indicate the dissemination level using one of the following codes:
• **PU =** Public • **PP =** Restricted to other programme participants (including the Commission Services) • **RE =** Restricted to a group specified by the consortium (including the Commission Services) • **CO =** Confidential, only for members of the consortium (including the Commission Services) • **Restreint UE =** Classified with the classification level "Restreint UE" according to Commission Decision 2001/844 and amendments • **Confidentiel UE =** Classified with the mention of the classification level "Confidentiel UE" according to Commission Decision 2001/844 and amendments • **Secret UE =** Classified with the mention of the classification level "Secret UE" according to Commission Decision 2001/844 and amendments

Disclaimer

Full Project Title:	Cross-lingual Knowledge Extraction
Short Project Title:	XLike
Number and Title of Work package:	WP2 – Multilingual linguistic processing
Document Title:	D2.1.1 – Shallow linguistic processing prototype
Editor (Name, Affiliation)	Marko Tadić, UZG
Work package Leader (Name, affiliation)	Marko Tadić, UZG
Estimation of PM spent on the deliverable:	

Executive Summary

This document presents the Shallow linguistic processing prototype developed during the first six months of the XLike project.

The document is describing the prototype that is aiming at providing the rest of the project teams with the shallow linguistic processing capabilities for each XLike language. The prototype offers two types of services: 1) Language identification (covering XLike languages); 2) Language analysis at shallow level (sentence splitting and tokenisation, POS/MSD-tagging and lemmatisation, named entity recognition and classification). All these subtasks are connected into a language specific pipeline. The output from this pipeline will provide the input for the following tasks in the project.

This prototype is bringing the most straightforward solution for a number of subtasks by using existing language tools and resources that could be combined in a pipeline hiding the complexity of internal operations to the end users. The more elaborate version of shallow linguistics processing pipeline will be developed by adapting this prototype, in order to accommodate also the needs of T2.2. At this moment the pipeline is oriented towards formal language register, while the informal language will be covered in further stages of WP2, i.e. tasks T2.3 and T2.4.

Table of Contents

List of Figures

List of Abbreviations

BIO	format of annotation for named entities
CoNLL	Conference on Natural Language Learning
HGC	Huge German Corpus
HMM	Hidden Markov Model
ISO	International Standard Organisation
POS	Part of Speech
MSD	Morphosyntactic Description
NERC	Named Entity Recognition and Classification
NE	Named Entity
NLP	Natural Language Processing
REST	web services protocol
STTS	Stuttgart-Tübingen Tagset for POS-tagging German corpora

1 Introduction

Each system aiming at cross-lingual extraction of knowledge data relies on language processing capabilities that represent the first step in the processing pipeline. In order to be most effective this language processing part has to be composed of smaller modules, each covering the individual subtask such as language detection, sentence splitting, tokenisation, lemmatisation, POS/MSD-tagging, named entity recognition and classification. This deliverable represents a report on producing the shallow linguistic processing pipeline and the pipelines for each language themselves.

The goal of this deliverable is to provide other research teams in XLike project with the uniform services for language processing of XLike languages. This represents the initial stage of the planned XLike Toolkit. Also, the goal was to avoid the reinvention of solutions for tasks that are already solved elsewhere – for some languages (e.g. English, German, and Spanish) even more times with different systems – but to examine and find the most appropriate existing tools for individual tasks. We were looking into tools, primarily published under different types of open source licences that would be easily connectable into a single-language pipeline. However, each single-language pipeline needs to adhere to the predefined common specification that would assure the uniform output for all of them. Therefore, the input, intermediary and output formats were adapted to the needs of the XLike Toolkit processing at further stages.

When in this deliverable we use the term “XLike languages” we will refer to English, German, Spanish, Chinese, Catalan and Slovenian.

2 General shallow linguistic processing prototype structure

The shallow processing prototype can be seen as a part of the full linguistic processing pipeline consisting of:

1. Shallow linguistic processing
2. Deep linguistic processing
3. Extraction of information in the form of relations.

We note that each of these operations encompasses a series of linguistic processing modules which are sometimes complex, and each of them requires some language-dependent configuration. For the sake of simplicity, at the moment each individual XLike language service offers a single operation called **`analyze_id`** that performs the full pipeline all-together. Section 4 below describes this operation in more detail.

A linguistic processing pipeline outputs linguistic structures that encode syntactic and semantic information for the input text. Figure 1 plots a sentence with linguistic annotations, and illustrates that the annotations can be quite complex.

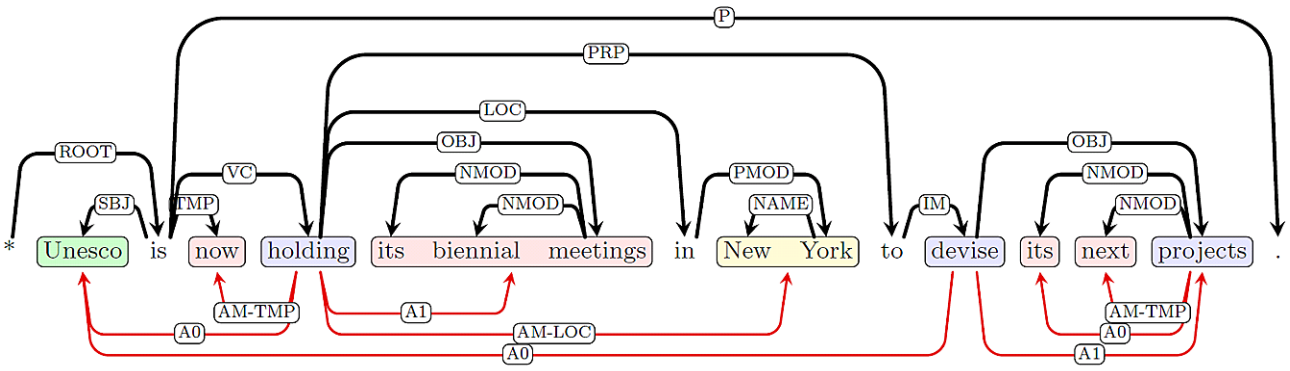

Figure 1: An example of a sentence with several linguistic structures in graph form

The general shallow processing prototype consists of the pipeline of different language processing modules each covering a specific subtask. The shallow linguistic processing prototype is offering two types of services:

1. Language identification
2. Language analysis.

The general shallow processing prototype can be represented schematically with:

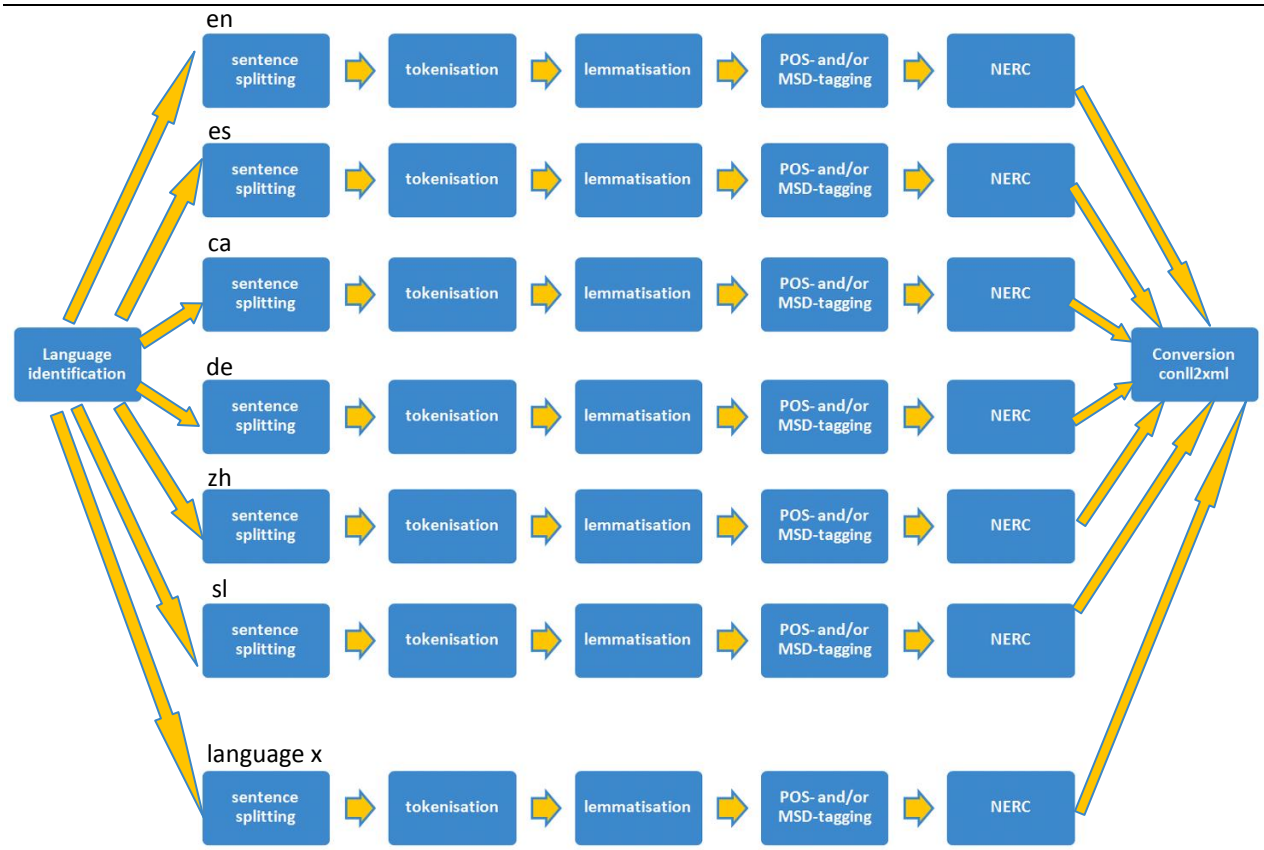

Figure 2: Schematic representation of XLike shallow linguistic processing prototype

2.1 Language identification

Since there is no assurance that the news streams, which are expected as the input source of texts to process, are accompanied by language identification meta-data, or the language identification could be derived from their URLs, the language identification module is needed at the very beginning of the shallow linguistic processing pipeline.

The language identification module takes as the input parameter the free text in utf8 encoding and outputs the language code following the ISO 639-2 specification:

- en: English
- es: Spanish
- ca: Catalan
- de: German
- zh: Chinese
- sl: Slovenian.

The language identification module is based on the well-known method that uses the character n-gram probability language model[2]. For XLike this model was trained on the dumps of Wikipedia data in XLike languages and a number of additional main world languages (e.g. Russian, Italian, etc.). In addition to that training, to avoid the possible language identification errors caused by genetical and typological closeness

[2] Padró & Padró (2004).

of languages or by similarities in their orthographic systems, language identification module was also trained on data from the set of additional languages:

- Galician, Portuguese and French vs. Spanish and Catalan;
- Croatian, Czech, Serbian (roman script), Slovak vs. Slovenian.

In this way we expect more precise language identification results. The evaluation of language identification precision and recall will be conducted in the next stage in WP2. This might result in additional training or other ways of refinement of language identification module.

If language identification module can't recognise the language, it returns the special value 'unk' (for unknown).

2.2 Language analysis

Language analysis part of the general shallow processing pipeline consists of six individual forks (or sub-pipelines) for each XLike language. Each of these pipelines is composed of several modules that cover tasks of:

- sentence splitting
- tokenisation
- lemmatisation
- POS- or MSD-tagging
- named entity recognition and classification

What is covered by the proposed specification of tasks in above list are the functionalities needed. Different XLike language pipelines don't need to have the exactly same number of modules. In some cases, one module can cover two (or more) tasks and we treat it as a black box as long as it provides the desired results following the predefined input and output formats. In this way we can take advantage of already existing and tested chains of modules that can function as parts of our shallow linguistic processing pipeline. This allowed us to avoid the need to build the whole pipeline and connect each module from the scratch for all languages. The details of technical implementation are in the Section 4.

2.3 Common output conversion module

All six pipelines for individual languages converge together at the end where a common converter between the intermediary format and output format is used.
This converter is used to convert the predefined intermediary CoNLL format into XLike output XML format defined and validated by XLike XML-schema.

3 Input, intermediary and output format specification

The shallow linguistic processing pipeline can function only if its input, intermediary and output formats of data are precisely defined.

3.1 Input format: free text

Since we expect that the news-streams will be encoded and formatted in a number of different ways that will not be easy to bring at the uniform level, our choice for input format was the free text encoded as utf8 character stream. This format is acceptable at this stage of processing, i.e. shallow linguistic processing prototype.
However, if needed, we can adapt the first module in the general pipeline, i.e. language identification module, to accept a different input format, e.g. XML, HTML or similar. This remains to be checked after the verification and evaluation of the shallow linguistic processing prototype at the further stages of processing.

3.2 Intermediary format: CoNLL

During the choice of the intermediary format we had to take care about all relevant information, not just at the shallow, but also at the deep linguistic processing level. In the NLP community the CoNLL format is a popular *markdown* format to represent linguistic structure. Its origins are in the Shared Tasks of the CoNLL conference (starting from 1997), which have provided benchmarks datasets and evaluation methods for a number of tasks, including Named Entity Recognition and Classification, Dependency Parsing and Semantic Role Labelling. In XLike, we will use this format to represent linguistic structure produced by modules in WP2. The main advantages of this format are:

- It is text-based.
- It represents linguistic structure in layers, and as a consequence appropriate for pipeline architectures like the one in XLike.
- It is relatively easy to inspect the linguistic structure of a sentence.
- It has no tags, thus reducing the size of annotated datasets.
- Texts with linguistic annotations can be directly saved in plain text files, or be embedded into an XML text element.

Next we describe the general CoNLL format, and then specify the actual format for XLike.

3.2.1 General rules of the CoNLL format

The general rules of the CoNLL format are:

- A sentence of n tokens is represented by an $n \times m$ matrix of string values. The i-th row contains the annotations for the i-th token. Each of the m columns corresponds to a layer of linguistic structure. The matrix is sometimes referred to as a block of columns.
- Columns are separated by spaces or tabs. The string values, or fields, should contain at least one non-space character, and should contain no spaces. There is no way to escape characters; so, for example, it is not valid to write a string field containing a space in quotes.
- All rows should have the same number of fields.
- Sentences (i.e. blocks of columns) are separated by one blank line.
- In some columns, a "_" character (without the quotes) stands for a null value. Exceptions are the columns representing word forms or lemmas, where that character is interpreted as the "_" string value.

3.2.2 The CoNLL format for XLike

The CoNLL format we will use for XLike closely follows that from the CoNLL-2009 Shared Task[3]. See Appendix A for an example of a sentence annotated in CoNLL format. At the moment, we will consider the following columns. Consider a sentence with *n* tokens:

1. **`id`**: Token position in the sentence, i.e. a number from 1 to *n*.
2. **`token`**: literal form of the token appearing in the text.
3. **`lemma`**: lemma.
4. **`pos`**: Part-of-speech tag.
5. **`msd`**: Morpho-syntactic description (In some languages a list of tags separated by the "|" character. In other languages it can be the full MSD-tag. An empty list is represented by "_").
6. **`ne`**: Named-entity tag in BIO form. Tokens which are not part of a named entity receive the O tag. For a named entity of type *k* spanning from tokens *i* to *j*, token *i* receives the **`B-`***k* tag, while tokens *i* + 1 to *j* receive the **`I-`***k* tag.
7. **`head`**: Dependency head. In a dependency tree, each token has a single head. This column contains the index of such head. Hence, if column *i* has index *h* it means that there is a syntactic dependency from token *h* to token *i*. A special value of *0* indicates the *root* of the sentence.
8. **`dlabel`**: Dependency label. The syntactic function of token *i* with its head. In unlabelled dependency trees, we use values "_".
9. **`pred`**: Predicate. If the token is a semantic predicate, the lemma of such predicate. If the token is not a predicate, a "_" value is inserted. The lemma of the predicate sometimes includes the *predicate sense*, i.e. a number indicating the semantic sense of the predicate according to some predicate catalogue.
10. (or more) **`p-role`**: Arguments of the *p*-th predicate. Rather than a single column, this is a sequence of columns (possibly empty), with one column for each predicate of the sentence. That is, if the pred column contains *P* non-null values, then the column 10 + *p* – 1 encodes the arguments of the *p*-th predicate. If token *i* is not a direct argument of *p* then it contains a null value "_"; if it is a direct argument, then it contains the *semantic role* between the *p*-th predicate and the syntactic subtree headed by token *i*.

ID	WORD--->	LEMMA-->	POS	MSD	NE-->	DEP-->	SRL---------->				
1	Unesco	unesco	NNP	_	B-ORG	2	SBJ	_	A0	A0	_
2	is	be	VBZ	_	O	0	ROOT	_	_	_	_
3	now	now	RB	_	O	2	TMP	_	AM-TMP	_	_
4	holding	hold	VBG	_	O	2	VC	hold.04	_	_	_
5	its	its	PRP	_	O	7	NMOD	_	_	_	_
6	biennial	biennial	JJ	_	O	7	NMOD	_	_	_	_
7	meetings	meeting	NNS	_	O	4	OBJ	_	A1	_	_
8	in	in	IN	_	O	7	LOC	_	AM-LOC	_	_
9	New	new	NNP	_	B-LOC	9	NAME	_	_	_	_
10	York	york	NNP	_	I-LOC	8	PMOD	_	_	_	_
11	to	to	TO	_	O	4	PRP	_	_	_	_
12	devise	devise	VB	_	O	11	IM	devise.01	_	_	_
13	its	its	PRP	_	O	15	NMOD	_	_	_	A0
14	next	next	JJ	_	O	15	NMOD	_	_	_	AM-TMP
15	projects	project	NNS	_	O	12	OBJ	project.02	_	A1	_
16	.	.	.	_	O	2	P	_	_	_	_

Figure 3: An example of a sentence from the Figure 1 with several linguistic structures in CoNLL format

[3] http://ufal.mff.cuni.cz/conll2009-st/task-description.html

3.3 Output format: XLike XML

The XML format is designed to represent the output of linguistic analysis that will be used at the end of the XLike shallow linguistic processing pipeline. Here we provide a brief description of the XML elements while the detailed description is defined by XLike XML schema in the Appendix. The output of an analysis is represented sentence by sentence. Each sentence is contained in an XML element **`sentence`**. This element contains three types of basic elements, namely words, entities and relations:

- **`word`**: Represents an occurrence of a word in a sentence. The attributes are:
  - **`token`**: the token as it occurs in the input text
  - **`lemma`**: the lemma of the word (disambiguated)
  - **`pos`**: the POS/MSD tag (disambiguated)
  - **`id`**: the position of the token in the sentence (starting at 1), which serves as an identifier for this token
  - **`start`**: the start character of the token in the input text
  - **`end`**: the ending character of the token in the input text

  Further attributes that may be included are:
  - A confidence value of the syntactic disambiguation for this word
  - Additional morpho-syntactic information, such as language dependent traits of the token (gender, number, case, WordNet sense, list of confidence-weighted alternative analyses,...).
- **`entity`**: (preliminary) The occurrence of a named entity in text. Entities correspond to contiguous tokens, and may have a type. The attributes are:
  - **`form`**: the entity mention as it occurs in the input text
  - **`lemma`**: a canonical value for this entity
  - **`type`**: the type of named entity (location, organization, person, time,...)
  - **`id`**: a unique identifier of the entity in the sentence
  - **`start`**: the **`id`** of the initial word forming the entity
  - **`end`**: the **`id`** of the final word forming the entity
- **`relation`**: (preliminary) The occurrence of a relation in a sentence. For now we assume relations in triple form, where the relation has a name and the two arguments are entities of the sentence. The attributes are:
  - **`predicate`**: the name of the relation (lemma of the predicate evoking the relation)
  - **`arg1`**: the **`id`** of the entity that is first argument
  - **`arg2`**: the **`id`** of the entity that is second argument.

```
<item>
 <sentences>
  <sentence id="1">
   <text>Unesco is now holding its biennial meetings in New York.</text>
    <tokens>
      <token pos="NP00SP0" end="6" lemma="unesco" id="1.1" start="0">Unesco</token>
      <token pos="VBZ" end="9" lemma="be" id="1.2" start="7">is</token>
      <token pos="RB" end="13" lemma="now" id="1.3" start="10">now</token>
      <token pos="VBG" end="21" lemma="hold" id="1.4" start="14">holding</token>
      <token pos="PRP$" end="25" lemma="its" id="1.5" start="22">its</token>
      <token pos="JJ" end="34" lemma="biennial" id="1.6" start="26">biennial</token>
      <token pos="NNS" end="43" lemma="meeting" id="1.7" start="35">meetings</token>
      <token pos="IN" end="46" lemma="in" id="1.8" start="44">in</token>
      <token pos="NP00G00" end="55" lemma="new_york" id="1.9" start="47">New_York</token>
      <token pos="Fp" end="56" lemma="." id="1.10" start="55">.</token>
    </tokens>
   </sentence>
 </sentences>
 <entities>
   <entity type="location" displayName="new_york" id="2">
    <mentions>
      <mention SentenceId="1" id="1.9" words="New York"/>
    </mentions>
   </entity>
   <entity type="person" displayName="organization" id="1">
    <mentions>
      <mention SentenceId="1" id="1.1" words="Unesco"/>
    </mentions>
   </entity>
  </entities>
  <relations>
    <relation subject="1.1" name="hold" object="1.9" id="3"/>
  </relations>
</item>
```

Figure 4: An example of a sentence from the Figure 1 with several linguistic structures in XML format

The exact XLike XML format schema listing is available in the Appendix.

4 Technical implementation

The shallow linguistic processing pipeline is technically realised as a series of web services that use REST protocol and each of them is covering one or several language processing tasks (see section 2.2 for details).

The one-stop entry point to individual language pipelines in this prototype stage is set to URL http://www.xlike.org/sandbox/ where each pipeline can be tested through the respective URL.

Minimal unit that will be processed by each service is defined at the level of each service. However, to avoid too many calls, that can slow down the processing time significantly, at this moment the minimal unit is the whole news article. Additional reasons for a unit of that size are that some modules require the whole text to be present in order to process it properly (e.g. anaphora resolution, NERC using the one-meaning-per-text strategy, etc.).

At this prototype stage no evaluation has been provided either for individual modules/services and/or for complete pipelines. The reasons for that are:

- there is a deliverable dedicated to that topic (D7.3.1) targeted for M12 when more processing material will be collected for testing.
- the relevant golden standards for individual modules and languages are not available at this stage. Also, whether the intrinsic evaluation of each module/service will be provided or only the extrinsic evaluation of the whole pipelines will be produced, remains to be decided during the work on aforementioned deliverable.

For XLike languages we selected the most prominent tools that represent the state of the art. Our main orientation was to use open source code as much as possible and to go for a proprietary solutions only if there was no other option. This principle was preserved practically in all pipelines.

In the following subsections we will describe the shallow linguistic processing pipelines for each language.

4.1 English, Spanish, Catalan

This section describes language-dependent linguistic services for English, Spanish and Catalan based on FreeLing[4]. For these three languages the same system is used, therefore a joint description. For each language, a service named **`analysis_id`** is offered. The service performs linguistic analysis for texts in the language **`id`** of the service. The input to the operation consists of the text to be analysed, plus a number of options that configure the behaviour of analysers themselves. The output consists of a structure encoding several layers of linguistic analysis for the input text as described in the Section 3.

The functional specification is:

- Input:
    - text: The input text to analyse, in UTF-8.
    - analysisOptions: A struct of options to control the operation. All these options have a default value, and they do not need to be specified. Except where noted, all options are boolean-valued.
        - input: A string-valued field indicating the format of the text input parameter.

          It is one value of the following:

            - plain: a plain string containing free text (default).
            - tokenized: a text split into sentences and tokenized, represented in CoNLL format.

[4] http://nlp.lsi.upc.edu/freeling

- tagged: a text in CoNLL format already split, tokenized and tagged.
- parsed: a text in CoNLL format already split, tokenized, tagged and parsed.

- conll: Whether to return the analysis in CoNLL format (default).
- taggingOptions A struct of options controlling the behaviour of the tagger:
  - punctuation: Assignment of special part-of-speech tags to punctuation tokens.
  - numbers: Recognition of numerical expressions (e.g. “five hundred and eighteen”) and assignment of lemmas encoding their numerical value.
  - dates: Detection of dates (e.g. “the fourth of January”)
  - multiwords: Detection of multiword expressions (e.g. “as long as”)
  - quantities: Detection of ratios, percentages, and physical or currency magnitudes (e.g. twenty per cent, 20%, one out of five, 1/5, one hundred miles per hour). This options requires activating the numbers option.
  - ner: Recognition (i.e. segmentation) of named entities.
  - nec: Classification of named entities. Requires activating the ner option.
- parsingOptions: A struct of options controlling the behaviour of the parser.
  - srl: Whether to perform semantic role labelling.
  - (more options to be specified)
- extractionOptions: A struct of options controlling the behaviour of the extraction method.
  - (options need to be specified)

- Output:
  - the linguistic analysis in CoNLL format (default). See Section 3 for a specification of this format.

4.2 German

The pipeline for processing German is combining several existing modules that deal with individual tasks, wrapped in a web services. The external modules used are:

Apache OpenNLP library (http://opennlp.apache.org/) is a machine learning based toolkit for the processing of natural language text. It supports the most common NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, and coreference resolution. This library is used by the service for tokenizing and sentence detection. The pre-trained models used by this library are available at http://opennlp.sourceforge.net/models-1.5/

Stanford POS Tagger (http://nlp.stanford.edu/software/tagger.shtml) is used by the service for determining part of speech token attributes.
There are three models for German language available:

- *german-hgc.tagger* model is trained on the first 80% of the Negra corpus, which uses the STTS tagset. The Stuttgart-Tübingen Tagset (STTS) is a set of 54 tags for annotating German text corpora with part-of-speech labels, which was jointly developed by the Institut für maschinelle Sprachverarbeitung of the University of Stuttgart and the Seminar für prachwissenschaft of the

University of Tübingen. This model uses features from the distributional similarity clusters built over the HGC.
- *german-dewac.tagger* model uses features from the distributional similarity clusters built from the deWac web corpus .
- *german-fast.tagger* model lacks distributional similarity features, but is several times faster than the other alternatives.

TreeTagger (http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html) and the Java wrapper (http://code.google.com/p/tt4j/) are used for determining the lemmas. The information about the models (in German) can be found at the TreeTagger web page.

Stanford Named Entity Recognizer (http://nlp.stanford.edu/software/CRF-NER.shtml) is used by the service for determining named entities. There are two classifiers for German NER available:
- *Huge German Corpus-generalized classifier:* This classifier has been trained on the CoNLL 2003 German data and has been generalized with the distributional similarity lexicon formed using the 175 million tokens of the HGC which have been clustered in 600 clusters. HGC is a collection of news-wire text.
- *deWaC-generalized classifier* **:** This classifier has been trained on the CoNLL 2003 German data and has been generalized with the distributional similarity lexicon formed using the 175 million tokens of the deWaC which have been clustered in 400 clusters. deWaC corpus has been created by collecting the content from the web, hence it is unclean and contains data from all genres.

The models and classifiers used by the web service can be configured using the application properties file. The service is currently configured using the following models/classifiers: german-hgc.tagger model for part-of-speech tagging, Huge German Corpus-generalized classifier used for named entity recognition, Apache OpenNLP German sentence and token model used for tokenization and German model used in Tree tagger for determining the lemma.

The functional specification for German pipeline is:

- Input:
    - Target: one of the following values, specifying the desired output elements
        - tokens: plain tokens
        - lemmas: tokens with lemma, pos and msd attributes
        - entities: tokens with lemma, pos, msd and ne attributes
    - InputType: one of the following values specifies the type of input. For the token target this parameter is optional.
        - text: the text parameter is free text
        - conll: the text parameter is CoNLL file contents
    - Text: depending on the input type the free text to analyze or the CoNLL file content in String form
- Output:
    - the linguistic analysis in CoNLL format (default). See Section 3 for a specification of this format.

4.3 Slovenian

This section describes the service for Slovene language processing, composed of a tokenizer, lemmatizer, morphosyntactic tagger and a named entity extractor.

The tokenizer, lemmatizer, and the MSD tagger belong to the "Obeliks" system, written in C#. The lemmatizer [Juršič et al., 2007] is based on supervised learning of ripple-down rules. The MSD tagger, a supervised approach using maximum entropy for sequence classification [Rupnik et al., 2008] works in concert with the lemmatizer, producing morphosyntactic tags in the format, specified in [Erjavec and Krek, 2008].

The named entity extractor detects entities of classes 'person', 'location' and 'other', where 'other' consists of the remaining common classes, such as 'event', 'product', 'organization', 'title' and similar, but not including 'time', 'quantity' or 'money'. The method is based on conditional random fields, implemented in the Mallet system [McCallum, 2002]. The features used in the component are the following: morphosyntactic features, memberships of lemmas in Wikipedia lexicons, and offset-based conjuctions of features. The model is trained on a corpus of 2173 annotated sentence [Krek and Erjavec, 2012] The aggregate performance of the system is measured as F1 score of 0.765, with slightly better performance on 'person' and 'location', but slightly lower on 'other', as it is a very wide and ill-defined class.

The functional specification is:

- Input:
    - Target: one of the following values, specifying the desired output elements
        - tokens: plain tokens
        - lemmas: tokens with lemma, and msd attributes
        - entities: tokens with lemma, msd and ne attributes
    - Text: the free text to analyze
- Output:
    - the linguistic analysis in CoNLL format (default). See Section 3 for a specification of this format.

4.4 Chinese

This section describes the service for Chinese language processing, composed of a tokenizer, PoS tagger and a named entity extractor.

The tokenizer and PoS tagger is written in C/C++ and then packed as a lib for the invoking in Java. It's based on a dual-level HMM model. The tokenizer can process 500K Chinese text per second at a high accuracy of 98.45%. The size of dictionary files is less than 3 Mb after compressed. The tag set used in the PoS tagger contains 95 tags. These 95 tags are in two levels. The top level contains 22 tags at the top level and 73 tags at the second level.

The named entity extractor detects entities of classes 'person', 'location', and 'organization'. We are capable to detect other entities classes such as 'time', 'money' and 'number' in future work.

External modules used by the service: ICTCALS library (http://www.ictclas.org/index.html) is a toolkit for the processing of Chinese language such as tokenization, PoS tagging and so on. This library is used by the service for tokenization and PoS tagging.

The functional specification is:

- Input:
  - text: The input text to analyse, in UTF-8.
  - analysisOptions: A struct of options to control the operation. All these options have a default value, and they do not need to be specified. Except where noted, all options are boolean-valued.
    - input: A string-valued field indicating the format of the text input parameter.

      It is one value of the following:
      - plain: a plain string containing free text (default).
      - tokenized: a text split into sentences and tokenized, represented in CoNLL format.
      - tagged: a text in CoNLL format already split, tokenized and tagged.
      - parsed: a text in CoNLL format already split, tokenized, tagged and parsed.
    - conll: Whether to return the analysis in CoNLL format (default).
    - ner: Recognition and classification of named entities.
    - parsingOptions: A struct of options controlling the behaviour of the parser.
      - (options need to be specified)
    - extractionOptions: A struct of options controlling the behaviour of the extraction method.
      - (options need to be specified)
- Output:
  - the linguistic analysis in CoNLL format (default). See Section 3 for a specification of this format.

5 Conclusions

This document presents the Deliverable 2.1.1 Shallow linguistic processing prototype. Its structure, functional specification and some details of technical specification are presented. Also, the definition of input, intermediary and output formats are given.

References

Erjavec, T., Krek, S. (2008) **Oblikoskladenjske specifikacije in označeni korpusi JOS**. Zbornik Šeste konference Jezikovne tehnologije, Ljubljana, 49-53.

FreeLing User Manual (v3.0), 2012-05 [http://nlp.lsi.upc.edu/freeling/doc/userman/userman.pdf, accessed 2012-06-28]

Juršič, M., Mozetič, I., Lavrač, N. (2007) **Learning Ripple Down Rules for Efficient Lemmatization**. Proceedings of the 10th International Multiconference Information Society, IS 2007, Ljubljana, 206–209.

Krek, S., and Erjavec, T. **Korpus 400.000 besed**. [http://www.slovenscina.eu/Vsebine/Sl/Kazalniki/K10.aspx, accessed 2012-05].

McCallum, A., (2004) **Mallet: A machine learning for language toolkit** [http://mallet.cs.umass.edu/, accessed 2012-05].

Padró, M., Padró, L. (2004) **Comparing Methods for Language Identification.** *Procesamiento del Lenguaje Natural*, n. 33, pg. 155--162. September, 2004.

Padró, L., Stanilovsky, E. (2012) **FreeLing 3.0: Towards Wider Multilinguality**. Proceedings of the Language Resources and Evaluation Conference (LREC 2012), ELRA, Istanbul, Turkey, May, 2012.

Rupnik, J., Grčar, M., and Erjavec, T. (2008) **Improving morphosyntactic tagging of Slovene language through meta-tagging**. Informatica Special Issue: Intelligent Systems, Costin Badica (eds.), 437-444.

Appendix A: XLike XML schema

```
<?xml version="1.0" encoding="utf-8"?>

<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema" >

  <xs:element name="item" type="item" />

  <!-- item contains information about a news item -->
  <xs:complexType name="item">
    <xs:sequence>
      <xs:element name="sentences" type="sentenceList" minOccurs="0" maxOccurs="1" />
      <xs:element name="entities" type="entityList" minOccurs="0" maxOccurs="1" />
      <xs:element name="conll" minOccurs="0" maxOccurs="1" type="xs:string" />
    </xs:sequence>
  </xs:complexType>

  <!-- List of sentences in the item -->
  <xs:complexType name="sentenceList">
    <xs:sequence>
      <xs:element name="sentence" minOccurs="0" maxOccurs="unbounded" type="sentence" />
    </xs:sequence>
  </xs:complexType>

  <!-- List of entities in the item -->
  <xs:complexType name="entityList">
    <xs:sequence>
      <xs:element name="entity" type="entity" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <!-- linguistic information for a sentence. Contains a list of tokens -->
  <xs:complexType name="sentence">
    <xs:sequence>
      <xs:element name="text" />
      <xs:element name="tokens" type="tokenList" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:string"/>
  </xs:complexType>

  <!-- List of tokens in a sentence -->
  <xs:complexType name="tokenList">
    <xs:sequence>
      <xs:element name="token" type="token" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
```



```
  <!-- linguistic information for a token -->
  <xs:complexType name="token" mixed="true">
    <xs:attribute name="id" type="xs:string" />
    <xs:attribute name="pos" type="xs:string" />
        <xs:attribute name="lemma" type="xs:string" />
        <xs:attribute name="start" type="xs:int" />
        <xs:attribute name="end" type="xs:int" />
  </xs:complexType>

  <!-- A named entity mentioned in the text. Contains a list of specific mentions -->
  <xs:complexType name="entity">
    <xs:sequence>
      <xs:element name="mentions" type="mentionList" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:int" />
    <xs:attribute name="displayName" type="xs:string" />
    <xs:attribute name="type" type="xs:string" />
    <xs:attribute name="concept" type="xs:anyURI" />
  </xs:complexType>

  <!-- List of tokens in a sentence -->
  <xs:complexType name="mentionList">
    <xs:sequence>
      <xs:element name="mention" type="mention" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <!-- A mention to one particular named entity -->
  <xs:complexType name="mention">
    <xs:sequence>
      <xs:element name="token" type="mentionToken" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:int" />
    <xs:attribute name="sentenceId" type="xs:string"/>
    <xs:attribute name="words" type="xs:string" />
  </xs:complexType>

  <!-- attributes for mention tokens -->
  <xs:complexType name="mentionToken">
    <xs:attribute name="id" type="xs:string" />
  </xs:complexType>

</xs:schema>
```

Appendix B: Examples of sentences in CoNLL format

An English sentence

```
1   Ms.      ms.      NNP   _   B-PER   2   TITLE   _          _          _
2   Waleson  waleson  NNP   _   I-PER   3   SBJ     _          _          _
3   is       be       VBZ   _   O       0   ROOT    _          _          _
4   a        a        DT    _   O       8   NMOD    _          _          _
5   free     free     JJ    _   O       7   HMOD    _          _          _
6   -        -        HYPH  _   O       5   HYPH    _          _          _
7   lance    lance    NN    _   O       8   NMOD    _          AM-MNR     _
8   writer   writer   NN    _   O       3   PRD     writer.01  A0         A1
9   based    base     VBN   _   O       8   APPO    base.01    _          _
10  in       in       IN    _   O       9   LOC     _          _          AM-LOC
11  New      new      NNP   _   B-LOC   12  NAME    _          _          _
12  York     york     NNP   _   I-LOC   10  PMOD    _          _          _
13  .        .        .     _   O       3   P       _          _          _
```

A German sentence

```
1   Der              der              ART     Nom|Sg|Masc              O   3   NK
2   texanische       texanisch        ADJA    Pos|Nom|Sg|Masc          O   3   NK
3   Milliardär       Milliardär       NN      Nom|Sg|Masc              O   6   SB
4   Ross             Ross             NE      Nom|Sg|Masc              B-PER   5   PNC
5   Perot            Perot            NE      Nom|Sg|Masc              I-PER   3   NK
6   hat              haben            VAFIN   3|Sg|Pres|Ind            O   0   ROOT
7   das              der              ART     Acc|Sg|Neut              O   9   NK
8   politische       politisch        ADJA    Pos|Acc|Sg|Neut          O   9   NK
9   Establishment    Establishment    NN      Acc|Sg|Neut              O   12  OA
10  in               in               APPR    _                        O   9   MNR
11  Washington       Washington       NE      Dat|Sg|Neut              B-LOC   10  NK
12  aufgeschreckt    aufschrecken     VVPP    Psp                      O   6   OC
13  .                _                $.      _                        O   6   PUNC
```

A Spanish sentence

```
1   Alcan        Alcan      n   postype=proper|gen=c|num=c                  B-ORG   2   suj
2   es           ser        v   postype=semiaux.|gen=c|num=s|person=3|mood=ind.|tense=pres.  O   0   sentence
3   una          uno        d   postype=indefinite|gen=f|num=s              O   2   atr
4   de           de         s   postype=preposition|gen=c|num=c            O   3   sp
5   las          el         d   postype=article|gen=f|num=p                O   7   spec
6   mayores      mayor      a   postype=qualificative|gen=c|num=p          O   7   s.a
7   empresas     empresa    n   postype=common|gen=f|num=p                  O   4   sn
8   del          del        s   postype=preposition|gen=m|num=s|contracted=yes  O   7   sp
9   mundo        mundo      n   postype=common|gen=m|num=s                  O   8   sn
10  dedicada     dedicar    v   postype=main|gen=f|num=s|mood=pastparticiple  O   3   S
11  a            a          s   postype=preposition|gen=c|num=c            O   10  creg
12  la           el         d   postype=article|gen=f|num=s                O   13  spec
13  producción   producción  n   postype=common|gen=f|num=s                 O   11  sn
14  de           de         s   postype=preposition|gen=c|num=c            O   13  sp
15  aluminio     aluminio   n   postype=common|gen=m|num=s                  O   14  sn
16  .            .          f   punct=period    O   2   f
```

A Catalan sentence

```
1   Unio_de_Pagesos      Unio_de_Pagesos  n   postype=proper|gen=c|num=c                  B-ORG   3   suj
2   va anar      v   postype=auxiliary|gen=c|num=s|person=3|mood=indicative|tense=present  O   3   v
3   demanar      demanar  v   postype=main|gen=c|num=c|mood=infinitive   O   0   sentence
4   als al       s   postype=preposition|gen=m|num=p|contracted=yes      O   3   ci
5   assistents   assistent   n   postype=common|gen=c|num=p              O   4   sn
6   que que      p   postype=subordinating     O   8   conj
7   participin   participar   v   postype=main|gen=c|num=p|person=3|mood=subjunctive|tense=present  O
    3   cd
8   a   a        s   postype=preposition|gen=c|num=c   O   7   creg
9   la  el       d   postype=article|gen=f|num=s       O   10  spec
10  manifestacio     manifestacio     n   postype=common|gen=f|num=s       O   8   sn
11  de  de       s   postype=preposition|gen=c|num=c   O   10  sp
12  Madrid       Madrid   n   postype=proper|gen=c|num=c                  B-LOC   11  sn
13  .   .        f   punct=period     O   3   f
```

A Chinese sentence

1	今天	今天	t	_	_	2	_	_	_
2	下午	下午	t	_	_	13	_	_	_
3	,	,	wd	_	_	13	_	_	_
4	20	20	m	_	_	5	_	_	_
5	余	余	m	_	_	6	_	_	_
6	名	名	q	_	_	7	_	_	_
7	清华大学	清华大学	nt	_	B-ORG	8	_	_	_
8	同学	同学	n	_	_	13	_	_	_
9	与	与	cc	_	_	9	_	_	_
10	美国	美国	nsf	_	B-LOC	11	_	_	_
11	国务卿	国务卿	n	_	_	12	_	_	_
12	希拉里·克林顿	希拉里·克林顿	nrf	_	B-PER	9	_	_	_
13	进行	进行	vx	_	_	0	_	_	_
14	了	了	ule	_	_	13	_	_	_
15	交流	交流	vn	_	_	13	_	_	_
16	。	。	wj	_	_	13	_	_	_

Appendix C: XLike web service definition

Overall web services definition

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:tns="http://localhost:9090/axis2/services/analysis_LANG/"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  name="analysis_LANG"
                  targetNamespace="http://localhost:9090/axis2/services/analysis_LANG/">

  <!-- WSDL - Types -->
  <wsdl:types>
    <xsd:schema targetNamespace="http://localhost:9090/axis2/services/analysis_LANG/">

      <!-- Analysis request. Contains a required parameter "text", plus other optional parameters -->
      <xsd:element name="analysisRequest">
        <xsd:complexType>
          <xsd:sequence>
         <!-- Required parameters. Common to all languages -->
            <xsd:element name="text" type="xsd:string" />
         <!-- Optional parameters. Common to all languages -->
            <xsd:element name="input" type="xsd:string" minOccurs="0" maxOccurs="1"  />
            <xsd:element name="target" type="xsd:string" minOccurs="0" maxOccurs="1" />
            <xsd:element name="conll" type="xsd:boolean" minOccurs="0" maxOccurs="1" />
         <!-- Tool parameters. May differ between languages or between language analysis tools -->
            <xsd:element name="numbers" type="xsd:boolean" minOccurs="0" maxOccurs="1" />
            <xsd:element name="punct" type="xsd:boolean" minOccurs="0" maxOccurs="1" />
            <xsd:element name="dates" type="xsd:boolean" minOccurs="0" maxOccurs="1" />
            <xsd:element name="multiw" type="xsd:boolean" minOccurs="0" maxOccurs="1" />
            <xsd:element name="quant" type="xsd:boolean" minOccurs="0" maxOccurs="1" />
            <xsd:element name="ner" type="xsd:boolean" minOccurs="0" maxOccurs="1" />
            <xsd:element name="nec" type="xsd:boolean" minOccurs="0" maxOccurs="1" />
          </xsd:sequence>
     </xsd:complexType>
      </xsd:element>

      <!-- Response to analysis request. Contains a structure with linguistic information on input text  -->
      <xsd:element name="analysisResponse">
         <xsd:element name="item" type="item" />
      </xsd:element>

      <!-- item contains information about a news item -->
      <xsd:complexType name="item">
     <xsd:sequence>
       <xsd:element name="sentences" type="sentenceList" minOccurs="0" maxOccurs="1" />
       <xsd:element name="entities" type="entityList" minOccurs="0" maxOccurs="1" />
       <xsd:element name="conll" minOccurs="0" maxOccurs="1" type="xsd:string" />
     </xsd:sequence>
      </xsd:complexType>
```



```
 <!-- List of sentences in the item -->
 <xsd:complexType name="sentenceList">
<xsd:sequence>
  <xsd:element name="sentence" minOccurs="0" maxOccurs="unbounded" type="sentence" />
</xsd:sequence>
 </xsd:complexType>

 <!-- List of entities in the item -->
 <xsd:complexType name="entityList">
<xsd:sequence>
  <xsd:element name="entity" type="entity" minOccurs="0" maxOccurs="unbounded" />
</xsd:sequence>
 </xsd:complexType>

 <!-- linguistic information for a sentence. Contains a list of tokens -->
 <xsd:complexType name="sentence">
<xsd:sequence>
  <xsd:element name="text" />
  <xsd:element name="tokens" type="tokenList" minOccurs="0" maxOccurs="1" />
</xsd:sequence>
<xsd:attribute name="id" type="xsd:string"/>
 </xsd:complexType>

 <!-- List of tokens in a sentence -->
 <xsd:complexType name="tokenList">
<xsd:sequence>
  <xsd:element name="token" type="token" minOccurs="0" maxOccurs="unbounded" />
</xsd:sequence>
 </xsd:complexType>

 <!-- linguistic information for a token -->
 <xsd:complexType name="token" mixed="true">
<xsd:attribute name="id" type="xsd:string" />
<xsd:attribute name="pos" type="xsd:string" />
<xsd:attribute name="lemma" type="xsd:string" />
<xsd:attribute name="start" type="xsd:int" />
<xsd:attribute name="end" type="xsd:int" />
 </xsd:complexType>

 <!-- A named entity mentioned in the text. Contains a list of specific mentions -->
 <xsd:complexType name="entity">
<xsd:sequence>
  <xsd:element name="mentions" type="mentionList" minOccurs="0" maxOccurs="1" />
</xsd:sequence>
<xsd:attribute name="id" type="xsd:int" />
<xsd:attribute name="displayName" type="xsd:string" />
<xsd:attribute name="type" type="xsd:string" />
<xsd:attribute name="concept" type="xsd:anyURI" />
 </xsd:complexType>

 <!-- List of tokens in a sentence -->
```

```
     <xsd:complexType name="mentionList">
    <xsd:sequence>
      <xsd:element name="mention" type="mention" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
     </xsd:complexType>

     <!-- A mention to one particular named entity -->
     <xsd:complexType name="mention">
    <xsd:sequence>
      <xsd:element name="token" type="mentionToken" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:int" />
    <xsd:attribute name="sentenceId" type="xsd:string"/>
    <xsd:attribute name="words" type="xsd:string" />
     </xsd:complexType>

     <!-- attributes for mention tokens -->
     <xsd:complexType name="mentionToken">
    <xsd:attribute name="id" type="xsd:string" />
     </xsd:complexType>

  </xsd:schema>
</wsdl:types>

<!-- WSDL - Messages -->
<wsdl:message name="analyzeRequest">
  <wsdl:part name="part1" element="tns:analysisRequest" />
</wsdl:message>
<wsdl:message name="analyzeResponse">
  <wsdl:part name="part1" element="tns:analysisResponse" />
</wsdl:message>

<!-- WSDL - portType -->
<wsdl:portType name="analysis_LANG">
  <wsdl:operation name="analyze">
    <wsdl:input message="tns:analyzeRequest" />
    <wsdl:output message="tns:analyzeResponse" />
  </wsdl:operation>
</wsdl:portType>

<!-- WSDL - Binding -->
<wsdl:binding name="analysis_LANG_SOAP" type="tns:analysis_LANG">
  <soap:binding style="document"
                transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="analyze">
    <soap:operation soapAction="analysis_LANG#analyze" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
```



```
    </wsdl:operation>
  </wsdl:binding>

  <!-- WSDL - Service -->
  <wsdl:service name="analysis_LANG">
    <wsdl:port binding="tns:analysis_LANG_SOAP" name="analysis_LANG_SOAP">
      <soap:address location="http://localhost:9090/axis2/services/analysis_LANG" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

An example of FreeLing-based web service

```
<service name="analysis_LANG">
  <parameter name="ServiceClass" locked="xsd:false">analysis_LANG</parameter>
  <description>WS providing access to all FreeLing funcionalities for LANG</description>
  <operation name="analyze">
    <messageReceiver class="wsf_cpp_msg_recv" />
    <parameter name="RESTMethod">GET</parameter>
    <parameter name="RESTLocation">analyze</parameter>
    <!--
     Operation 'analyze' has the following parameters:

     Required parameters:
          text : Text to analyze.

     Optional parameters:
       target : Desired analysis {tokens,lemmas,entities,relations}
                Default value: entities
       input  : Input analysis level {text, tokens, lemmas, entities}
                Default value: text
       conll  : Produce output in conll format. Boolean (def: false)

     Analyzer parameters (may differ from one language to another)
       numbers: Activate numbers detection. Boolean (def: true)
       punct  : Activate punctuation detection. Boolean (def: true)
       dates  : Activate dates detection. Boolean (def: true)
       multiw : Activate multiword detection. Boolean (def: true)
       ner    : Activate NE detection. Boolean (def: true)
       nec    : Activate NE classification. Boolean (def: true)
       quant  : Activate quantities detection. Boolean (def: true)
       parser : Select parser to use {txala,treeler}
                Default value: txala
    -->
  </operation>
</service>
```