RACAI's Linguistic Web Services

Dan Tufiş, Radu Ion, Alexandru Ceauşu, Dan Ştefănescu
Research Institute for Artificial Intelligence, Romanian Academy
13, "Calea 13 Septembrie", Bucharest 050711, Romania
tufis@racai.ro, radu@racai.ro, aceausu@racai.ro, danstef@racai.ro

Abstract

Nowadays, there are hundreds of Natural Language Processing applications and resources for different languages that are developed and/or used, almost exclusively with a few but notable exceptions, by their creators. Assuming that the right to use a particular application or resource is licensed by the rightful owner, the user is faced with the often not so easy task of interfacing it with his/her own systems. Even if standards are defined that provide a unified way of encoding resources, few are the cases when the resources are actually coded in conformance to the standard (and, at present time, there is no such thing as general NLP application interoperability). Semantic Web came with the promise that the web will be a universal medium for information exchange whatever its content. In this context, the present article outlines a collection of linguistic web services for Romanian and English, developed at the Research Institute for AI for the Romanian Academy (RACAI) which are ready to provide a standardized way of calling particular NLP operations and extract the results without caring about what exactly is going on in the background.

1. Introduction

Software application interoperability is a long standing goal in its own. When referring to data exchange, it is not always easy to couple two different authored programs which admittedly complement each other and obtain the combined result because one will have to make sure that the output from one program fits as input to the second (and the problem grows in complexity with the number of programs that have to be combined). Another issue to take into consideration is the format of different software resources that are used by programs. It may be the case that different developers are creating custom resources that are to be used by their own applications but that, given the appropriate changes to the format, could be also used by other applications (the best example here is the existence of different ontologies each coded in different knowledge representation languages but which could be used by a variety of applications only if their format would be the same[1]).

Natural Language Processing (NLP) applications and resources make no exception. The typical NLP researcher would search the Internet for the necessary tools and resources for the language of interest and would then take the time to mix all these into a functional system. When the tools and/or resources are not to be found in the public domain or are too expensive to acquire, he or she will implement the necessary tools and/or resources, with the primary concern on the correct operation and, usually, ignoring the interoperability issues. Then the cycle is resumed and another NLP researcher would use these developed tools and/or resources and waste some more time to make them work in other situations and/or environments.

It seems that one way out from this endless collect, adapt, test and use loop is provided by the Semantic Web idea of web services. According to Tim Berners-Lee, a web service provides for "program integration across application and organizational boundaries" (Berners-Lee, 2003). The integration is achieved via a standardized RPC call interface implemented with SOAP which is essentially built over XML. Apart from that, the software API of the web services can be formally described with WSDL which is also a XML language.

The great advantage of the web service concept is that the machine hosting the actual service need not be the same with the machine of the user of the service. This is a concept borrowed from RPC and CORBA and extended such that the message transport can be implemented according to any Internet protocol such as HTTP, SMTP, POP or TCP. One of the protocols for the message encoding is SOAP (the latest version is 1.2) which is "a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment" (citation taken from W3C SOAP 1.2 specifications[2]).

Together, SOAP and WSDL assure the user that the web service is readily available to use directly in any application (provided that the user knows the URL of the WSDL file of the web service). Thus, the time spent to collect, adapt and test a standalone application is reduced to minimum because the user need not: **i)** download the application but merely use its interface; **ii)** adapt the application because its interface is well described with WSDL and there are software tools that given the WSDL file, import all the semantics into the current project (see Microsoft Visual Studio for example) and **iii)** test the application because it was well tested by its creators and it is running in the proper environment.

The present article will describe several linguistic web

[1] Of course, there is also the problem of ontology interconnection which involves concept mapping but on a first level, common representation formalisms should be adhered to.

[2] http://www.w3.org/TR/soap12-part1/

services for English and Romanian developed at the Research Institute for Artificial Intelligence of the Romanian Academy (RACAI) implementing NLP operations such as POS tagging (with its prerequisites sentence and token splitting), lemmatization, chunking, word linking, WordNet lookup, languages identification, diacritics insertion (for Romanian) and Romanian Wikipedia indexing and searching.

2. General Architecture

The POS tagging, lemmatization and chunking operations are implemented by **TTL** (Ion, 2007), a Perl module which was intended as an API for NLP applications requiring these operations. Word linking is achieved through **LexPar** (Ion 2006; Ion 2007), a lexical attraction model linker with syntactic filtering of possible links also written in Perl. WordNet lookup, language identification and diacritics insertion (**DIAC+**, (Tufiş & Ceauşu, 2008)) are implemented in C# and were developed as standalone applications. The indexing and search engines were developed for the CLEF series [3] of QA evaluation exercises.

All applications were adapted as web services using SOAP::Lite[4] in the case of Perl applications and native C# capabilities for the ones written in C#. While Microsoft Visual Studio supports WSDL generation on the fly, we needed to encode function return values and signatures using POD::WSDL[5] in the case of Perl applications and manually generate WSDL web service description.

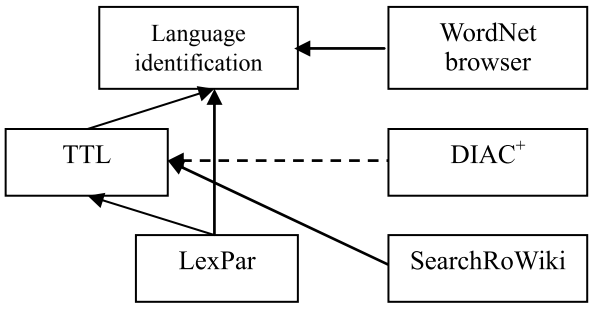

Figure 1: A general operation dependency graph of the web services

Figure 1 depicts a dependency graph of the web services operation. Both TTL and LexPar have been trained to process either English or Romanian and, therefore, their invocation requires the language code for the appropriate selection of the linguistic resources. Thus, in an unsupervised scenario, a generic application calling the TTL or LexPar services should first call the Language Identification service (described in section 7) and use the result as an input parameter for TTL or LexPar (actually this is a generic requirement for any of our multilingual services). LexPar needs the XML output formatting of TTL in order to add word links. WordNet lookup, searching through Romanian Wikipedia and DIAC+ are standalone web services that do not require any further preprocessing. The dashed line between DIAC$^+$ and TTL shows that the diacritics recovery web service can also work on TTL processing output.

In what follows, we will describe each of the web services concentrating on their programming interfaces. For the details of their algorithms, the reader may consult the references.

3. TTL and LexPar

TTL (Ion, 2007) is a text preprocessing module developed in Perl. Its functions are: Named Entity Recognition (by means of regular expressions defined over sequences of characters), sentence splitting, tokenization, POS tagging, lemmatization and chunking. The NER function is included as a preprocessing stage to sentence splitting because end of sentence markers may constitute parts of an NE string (i.e. a period may be a part of an abbreviation). POS tagging is achieved through the HMM tagging technology. The POS tagger of TTL follows the description of HMM tagger given in (Brants, 2000) but it extends it in several ways allowing for tiered tagging, for a more accurate processing of unknown words and also for tagging of named entities (which are practically labeled by the NER module before actual POS tagging). The TTL's tag-set is the MSD[6] with its smaller superset CTAG. (TTL tagging methodology follows the tiered tagging approach (Tufiş, 1999) where MSDs are recovered from an initial CTAG annotation). Lemmatization is achieved after POS tagging by lexicon lookup (in general, a word form and its POS tag uniquely identify the lemma). In the case of out-of-lexicon word forms the lemmatization is performed by a statistical module which automatically learns normalization rules from the existing lexical stock (for details see (Ion, 2007)). Finally, chunking is implemented with regular expressions over sequences of POS tags. It is not recursive and it does not perform attachments (PPs to NPs for instance).

The TTL web service offers the following remote procedures (these are the actual names from the WSDL file which is located at http://ws.racai.ro/ttlws.wsdl):

1. `SentenceSplitter` which takes as parameters the language of the text to process (currently either "`en`" or "`ro`") and a SGML entity encoded text and returns another string which is a list of sentences separated by carriage return/line feed sequence ("`\r\n`");
2. `Tokenizer` which has as parameters the language code and a sentence and returns a list of tokens separated by "`\r\n`" each token possibly carrying its NE tag (added to the token with the tab character "`\t`") given by the NER module of the `SentenceSplitter` in the case the token is a NE

[3] http://www.clef-campaign.org/
[4] http://www.soaplite.com/
[5] http://search.cpan.org/~tareka/Pod-WSDL-0.05/

[6] http://nl.ijs.si/ME/V2/msd/

(i.e. a real or integer number, a roman number, percents, abbreviations, dates, clock times, etc.);

3. `Tagger` which takes the language code and a tokenized sentence from `Tokenizer` and returns a MSD POS tagged sentence which is a string with triples of token, "`\t`", MSD separated by "`\r\n`";
4. `Lemmatizer` uses the POS tagged sentence along with the language code and returns a lemmatized sentence which resembles the one from the `Tagger`'s output except that the token annotation is enriched with its lemma which is separated again by a "`\t`" from the MSD tag;
5. `Chunker` is the final operation of TTL and, beside the language code, it takes a lemmatized sentence and returns the same sentence with chunk information added after the lemma annotation;
6. `XCES` is a helper function which calls all the previously mentioned operations and returns an XML representation of the result.

In principle, TTL operations are to be pipe-lined from 1 to 5, `SentenceSplitter` which takes the actual text as parameter being the first function call, `Tokenizer` the second function call, and so on till the `Chunker` operation. Since TTL operates with SGML entities and not UTF-8 representation of the text, the user is required to transform the input text from UTF-8 to SGML by calling `UTF8toSGML` helper function of the TTL web service and convert the response back to UTF-8 with the reverse function `SGMLtoUTF8`. The conversion cannot be automatically made because the web service cannot know how many calls are stacked and thus, when to convert back to the UTF-8 encoding.

To get a feel of how the TTL web service is invoked, here is a short example written in Perl using the SOAP::Lite package. We exemplify the process with the English sentence "*This is a simple example of a web service remote execution.*".

```
use SOAP::Lite;
my( $soap ) = SOAP::Lite->new()->
 uri( 'http://ws.racai.ro/pdk/ttlws' )->
 proxy( 'http://ws.racai.ro/' );
print(
 $soap->Chunker( "en",
  $soap->Lemmatizer( "en",
   $soap->Tagger( "en",
    $soap->Tokenizer( "en",
     $soap->SentenceSplitter( "en", "This is
          a simple example of a web service
          remote execution." )
     ->result()
    ) #end Tokenizer call.
    ->result()
   ) #end Tagger call.
   ->result()
  ) #end Lemmatizer call.
  ->result()
 ) #end Chunker call.
  ->result()
 ); #end print
```

We have set the URI of the TTL web service which is its universal identifier over the Internet with the `uri` method of the newly created SOAP object. Then, we have specified the physical URL of the web server which hosts the service with the `proxy` method. We are now ready to call all the TTL's public procedures. By writing `$soap->Chunker(...)` for instance, the SOAP::Lite package does all the hard work for us: it encodes the method call and its input parameters into a SOAP message, it sends the message to the service web server and receives from it another SOAP message which encodes the procedure's return value. It parses the SOAP message response and extracts the result which then it presents to us with the `result()` method call. So, the result of running this sample code looks like this:

```
This          Pd3-s    this
is            Vmip3s   be        Vp#1
a             Ti-s     a         Np#1
simple        Afp      simple    Np#1,Ap#1
example       Ncns     example   Np#1
of            Sp       of        Pp#1
a             Ti-s     a         Pp#1,Np#2
web           Ncns     web       Pp#1,Np#2
service       Ncns     service   Pp#1,Np#2
remote        Afp      remote    Pp#1,Np#2,Ap#2
execution     Ncns     execution     Pp#1,Np#2
.             PERIOD   .
```

Information on each line was added from left to right: token, MSD tag, lemma and chunking information. Regarding the chunks, every token has a list of the chunks it belongs to. The order of the chunks in the list signifies chunk inclusion (e.g. the token 'service' belongs to the noun phrase no. 2 which is embedded into the prepositional phrase no. 1).

`XCES` is another function of TTL which turns the vertical text format exemplified above into an XML encoding, resembling XCES format[7]. For our recurrent example, the result of invoking the `XCES` function is suggested below:

```
<seg lang="en">
 <s id="example.1">
  <w lemma="this" ana="Pd3-s">
     This</w>
  <w lemma="be" ana="Vmip3s" chunk="Vp#1">
     is</w>
  <w lemma="a" ana="Ti-s" chunk="Np#1">
     a</w>
  <w lemma="simple"ana="Afp"chunk="Np#1,Ap#1">
     simple</w>
  <w lemma="example" ana="Ncns"chunk="Np#1">
     example</w> ...
```

Lexpar (Ion 2006; Ion 2007) is a word linker. A link between two syntactico-semantic related words in a

[7] http://www.cs.vassar.edu/XCES/

sentence is an approximation of a dependency relation as described in (Mel'čuk, 1988) with the difference that the orientation and labeling are missing. A link structure of a sentence (called a linkage) is constructed with a Lexical Attraction Model (Yuret, 1998). We have improved the convergence properties of a LAM with a syntactic filter that rejects links are not syntactically valid (e.g. a link between an adverb and a determiner).

The LexPar web service is hosted on the same machine as TTL (its WSDL file is at http://ws.racai.ro/lxpws.wsdl) and it provides only one function: `LinkSentence`. This function generates the linkage of the tokenized, tagged and chunked sentence. The input parameters of this function are the XCES encoding of the sentence to be processed and the language code. and returns the same XML encoding enriched with the linkage information. The output in the case of our example is given below:

```
<seg lang="en">
 <s id="example.1">
  <w lemma="this" ana="Pd3-s" head="1">
     This</w>
  <w lemma="be" ana="Vmip3s" chunk="Vp#1">
     is</w>
  <w lemma="a" ana="Ti-s" chunk="Np#1" head="5">
     a</w>
  <w lemma="simple" ana="Afp" chunk="Np#1,Ap#1"
     head="5">simple</w>
  <w lemma="example" ana="Ncns" chunk="Np#1"
     head="1">example</w>
 ...
```

We can see that for all but one tokens there is a `head` attribute. This attribute has an integer as its value which indicates the position in the sentence (0 based numbering) to which the token is linked (the naming of the attribute does not imply that the token with the head information is actually the head of the relation). The token without this attribute (in our example the verb *be*) is the root of the linkage. The linkage of the sentence is almost always a connected graph (and it is always a planar and acyclic graph). The cases in which the graph is not connected occur whenever the syntactic filter wrongly rejects correct links. However, this rarely happens because in the vast majority of cases the rejected links are indeed incorrect.

4. DIAC^{+}

The main task of the DIAC^{+} web service is diacritics recovery in Romanian texts (Tufiş & Ceauşu, 2008). For Romanian, automatic restoration of the diacritics is a real challenge, both because of their frequency (every third word might contain at least one diacritical character) and due to their significant contribution to the morpho-lexical and semantic disambiguation of the words. As the majority of Romanian texts published on the web don't use the diacritics, for the researchers (and not only) relying on web data this is a long-time expected service.
The diacritics recovery web service is an integrated processing flow including tokenization, sentence splitting, tiered tagging, lemmatization, etc. The pre-processing steps are not the ones used in TTL (although they could be), but were adapted for a better integration with MS Office for which the diacritics recovery service was designed and optimized for speed and memory. These steps are implemented in C# exploiting very useful pieces of code existing in the .NET libraries. Among the major differences with respect to TTL, we can mention: tokenization does not merge compound lexical items, tiered tagging is a two steps maximum entropy classifier, the lexicon is indexed by the non-diacritical form of the words, etc.

The morpho-syntactical descriptions, as encoded in the MSD tags, are used to disambiguate between several possible word forms that may or may not contain diacritics. This approach relies on a wide-coverage dictionary and an accurate tagging method (tiered tagging) to ensure high precision since overlooking or improperly adding the diacritical signs may change the meaning of the sentence. There are also other approaches, most of them based on a character language model, but they cannot provide the same degree of precision (for further details see (Tufiş & Ceauşu, 2008) in this volume).

The web service description is available at http://nlp.racai.ro/WebServices/TextProcessingWebService.asmx?WSDL. The web service exposes only one function - `process` that takes three arguments: the first identifies the text to be processed, the second specifies whether lemmatization is requested or not and the last argument determines whether the spelling correction will be applied or not. The result of the web service is provided as a vertical text, each line containing in a tab-delimited format a word-form, its associated morpho-syntactic description, and the corresponding lemma. For example, for the non-diacritical text "*Nu poti spala cu lacrimi un rau profund*" (approx: "*Tears cannot make good a profound damage*") the web service returns:

```
Nu          Qz        nu
poţi        Vmm-2s    putea
spăla       Vmnp      spăla
cu          Spsa      cu
lacrimi     Ncfp-n    lacrimă
un          Timsr     un
rau         Ncms-n    rău|râu
profund     Afpms-n   profund
```

In the exemplified output, the word "*rau*" remained unchanged (it could be recovered either as *râu* ("*river*") or *rău* ("*damage*") as the lemma column shows). This is a semantic ambiguity (the morpho-syntactic descriptors are identical for the two interpretations) and is beyond the ability of the current system (and of the most existing systems).

A sample client of the web service is available at http://nlp.racai.ro/WebServices/TextProcessing.aspx. Via this interface a user can send archived files to be processed and retrieve the archived results. Archived content is used in order to minimize bandwidth requirements.

5. WordNet Browser

The Wordnet browser is a web service that allows browsing through aligned wordnets. For now, only the Princeton 2.0 and the Romanian WordNets are available but the web service can be easily extended with wordnets for different languages or even with wordnets of different versions. Table 1 shows the main figures for the statistics of the aligned wordnets available in the wordnet browser web service. As the figures for Romanian wordnet are continuously changing, one should check the latest statistics at http://nlp.racai.ro/wnbrowser/ (click on the *RoWordnet Statistics* tab).

	Princeton WordNet 2.0	Romanian WordNet
Synsets:	115424	47797
- nouns	79689	36017
- verbs	13508	9555
- adjectives	18563	1391
- adverbs	3664	834
Number of literals	203147	72532
Unique literals	153236	42499
Relations:		
- hypernym	94842	46487
- holo_part	8636	4302
- category_domain	6166	2956
- holo_member	12205	1519
- near_antonym	7642	2642
- be_in_state	1296	646
- holo_portion	787	362
- also_see	3240	692
- verb_group	1748	1464
- causes	218	181
- subevent	409	348
- similar_to	22196	1337

Table 1: Princeton and Romanian wordnets statistics

The data of both wordnets is stored in XML files with records like:

```
<SYNSET><ID>ENG20-12977363-n</ID>
<POS>n</POS>
<SYNONYM>
<LITERAL>cvintilion<SENSE>1</SENSE></LITERAL>
</SYNONYM>
<DEF>un milion de cvadrilioane</DEF>
<ILR>ENG20-12969974-n<TYPE>hypernym</TYPE>
</ILR>
<DOMAIN>number</DOMAIN>
<SUMO>PositiveInteger<TYPE>@</TYPE></SUMO>
<SENTIWN><P>0.0</P><N>0.0</N><O>1</O></SENTIWN>
</SYNSET>
```

Each record is indexed by literal and synset id. This simple representation allows the client of the web service to search for both literals and synsets.

As the example above shows, the Romanian WordNet contains not only the Princeton WordNet specific data but also the IRST DOMAIN (Bentivogli et al, 2004), SUMO (Niles & Pease, 2001) and SentiWordnet (Esuli & Sebastiani, 2006) annotations. Currently these annotations can be visualized on the web browser available at http://nlp.racai.ro/wnbrowser/.

A common usage scenario for the current wordnet web service is to translate a word from Romanian to English or vice-versa: **i)** the client application queries the web service for all ids of the synsets containing a given literal in the source language; **ii)** the client queries for all the synsets labeled by the returned ids in the target language; **iii)** the client application extracts the literals from the target language synsets. The literals from the last processing stage are the possible translations of the given literal.

For example, one can first check what wordnet sources are available using the function `GetSources()`. The web service will return the array of strings which, for the moment, contains "wn20-en" and "wn20-ro". If we want the translation of the Romanian word "*biografie*" the first step would be to call the function: `GetLiteral(string source, string literal, int level)`. If the argument `level` is bigger than 0, then the synset relations will be recursively expanded with the related synsets for *n* levels. The call `GetLiteral(`"wn20-ro", "biografie", 0`)` will have the following result:

```
<Result source="wn20-ro" literal="biografie">
<SYNSET><ID>ENG20-06113482-n</ID>
<POS>n</POS>
<SYNONYM>
<LITERAL>biografie<SENSE>1</SENSE></LITERAL>
<LITERAL>viaţă<SENSE>16</SENSE></LITERAL>
</SYNONYM>
<DEF>Expunere (scrisă şi comentată) a vieţii unei
persoane.</DEF><BCS>1</BCS>
<ILR>ENG20-06111883-n<TYPE>hypernym</TYPE>
</ILR>
<DOMAIN>telecommunication</DOMAIN>
<SUMO>Biography<TYPE>+</TYPE></SUMO>
<SENTIWN><P>0.0</P><N>0.0</N><O>1</O></SENTIWN>
</SYNSET>
</Result>
```

The function `GetSynset(string source, string ili, int level)`, allows for ILI-code based synsets retrieval. The call to `GetSynset(`"wn20-en", "ENG20-06113482-n",0`)` will return the following result:

```
<SYNSET><ID>ENG20-06113482-n</ID>
<POS>n</POS>
<SYNONYM>
<LITERAL>biography<SENSE>1</SENSE></LITERAL>
<LITERAL>life<SENSE>8</SENSE></LITERAL>
<LITERAL>life story<SENSE>1</SENSE></LITERAL>
<LITERAL>life history <SENSE>1</SENSE></LITERAL>
</SYNONYM>
<ILR><TYPE>hypernym</TYPE>ENG20-06111883-n
</ILR>
<DEF>an account of the series of events making up
```

```
a person's life</DEF><BCS>1</BCS>
<SUMO>Biography<TYPE>+</TYPE></SUMO>
<DOMAIN>telecommunication</DOMAIN>
<SENTIWN><P>0.0</P><N>0.0</N><O>1</O></SENTIWN>
</SYNSET>
```

The web service description is available at http://nlp.racai.ro/wnbrowser/Wordnet.asmx?wsdl and a user friendly interface implementing a web service client can be found at http://nlp.racai.ro/wnbrowser/. In the future versions of this service, new functions will be added to allow queries on aligned wordnets using different criteria like synset relations, domains, SUMO information, etc.

6. SearchRoWiki

SearchRoWiki (Search Romanian Wikipedia) is a web service originally developed for the Romanian shared task at CLEF 2007. The web service searches through the collection of 43000 Romanian documents available on Wikipedia[8] and it is based on a C# port of the Lucene search engine[9]. The web service was designed to use the results of a query analysis (specified as a disjunction of weighted Boolean terms) to retrieve a list with documents/sections that best match the query.

On the indexing side, SearchRoWiki uses the TTL and LexPar web services (presented in section 3) to annotate the available documents. The functions controlling the indexing stage were not made public. There were considerable improvements when we used the Romanian tokenizer instead of Lucene's default tokenizer because most of the words with hyphen and the abbreviations were handled in a consistent manner. Since lexical normalization (stemming or lemmatization) improves the recall of an information retrieval system this web service invokes TTL lemmatization. Instead of filtering the index terms using a stop words list, SearchRoWiki uses the information from POS-tagging to keep only the content words (nouns, main verbs, adjectives, adverbs and numerals). In addition, the web service uses the sentence and chunk annotation to insert phrase boundaries into the term index; a phrase query cannot match across different chunks or sentences.

In the SearchRoWiki index there are different fields for the surface form of the words and their corresponding lemmas. This kind of distinction applies to titles and document text resulting in four different index fields: title word form (`title`), title lemma (`ltitle`), document word form (`text`) and document lemma (`ltext`).

To achieve better precision in ranking the documents and their content, the web service uses two indexes: **i)** one for the documents (43486 documents, 694467 terms) and **ii)** one for the sections of the documents (90819 sections, 700651 terms). The hit list returned from the web service is a list of sections that match the query. The sections (paragraphs) are sorted and ranked using the documents index.

Another feature of the SearchRoWiki web service is the possibility to find the maximal conjunctive query given a set of Boolean terms. The web service will first try to match all of the query terms against the document index. If the search does not have a result, the system will recursively try to match n - 1 of the conjunctive terms until the query returns at least one result from the document index. The returned documents from the conjunctive Boolean query are used to select the corresponding sections in which the query terms occur. The sections are ranked using a new query with terms of the maximal conjunctive Boolean query. The terms of this new query are joined with the disjunction operator.

For example, the call to `GetResults(`"**ltitle**:\"Twin Peaks\" AND **ltext**:\"Twin Peaks\""`)` will search for all the documents with words "Twin Peaks" both in the title and in the document body. The web service returns an XML document containing all the sections where the query terms appear. The sections are grouped and ranked based on the score of the documents.

The web service description is available at http://nlp.racai.ro/WebServices/SearchRoWikiWebService.asmx?WSDL and a sample client can be found at http://nlp.racai.ro/WebServices/SearchRoWiki.aspx.

7. Language Identification

The Language Identification web service is derived from a stand alone application that was initially aimed at autonomously collecting web data for English and Romanian. It was also meant to check whether all the paragraphs/sentences in a given, presumably, monolingual corpus where indeed written in the respective language. This is, for instance, how we clean-up the Romanian part of the 22-language parallel corpus JRC-Acquis (Steinberger et al., 2006). Currently, the web service distinguishes among the 22 languages of the European Union., present in the JRC-Acquis parallel corpus. The function takes as its sole parameter a string consisting of a fragment of text and returns a string in which one can find the language code for that text and a confidence score for the classification. The Language Identification service can be easily extended with arbitrary new languages because the implementation is language independent, trainable on language specific data. The training module is not available as a web service, but an user interested in having a new language included in the Language Identification engine may contact the administrator (ws-admin@racai.ro) of the web service platform and send training data. The new language will be added as soon as possible.

Most developers construct their applications for language identification using N-gram or Markov chains approaches. We have taken a different approach. Given training texts in different languages (approx. 1.5Mb of text for each language), a training module counts the prefixes (the first

8 http://ro.wikipedia.org/
9 http://lucene.apache.org/

3 characters) and the suffixes (4 characters endings) for all the words[10] in the texts, for each language. Thus, for every language two models are constructed. The models will contain the weights (percentages) of prefixes and suffixes in the texts representing a language. In the prediction phase, for a new text, two models are built on the fly in a similar manner. These models are then compared with the stored models representing each language for which the application was trained. The comparison is performed using the following functions:

$$S_p = \left(\frac{g_P}{t_P}\right)^4 * \sum_{p \in M_T} \left(p_L * \left(1 - \left|p_L - p_T\right|\right)\right)$$

$$S_s = \left(\frac{g_S}{t_S}\right)^4 * \sum_{s \in M_T} \left(s_L * \left(1 - \left|s_L - s_T\right|\right)\right)$$

where:
g_p (g_s) – the number of prefixes (suffixes) in the input text model (M_T) which also exist in the language model M_L;
t_p (t_s) – the number of prefixes (suffixes) in M_T;
p_L (s_L) - the weight of prefix p (suffix s) in M_L;
p_T (s_T) - the weight of prefix p (suffix s) in M_T;

The total score for a language is:

$$S_t = \frac{\alpha * S_p + \beta * S_s}{\alpha + \beta}$$

We used $\alpha = 1$ and $\beta = 2$. The best score wins. The WSDL description of the Language Identification web service is to be found at the following URL: http://nlp.racai.ro/webservices/LangIdWebService.asmx?WSDL. There is also a web application that uses the Language Identification web service which is located at http://nlp.racai.ro/webservices/LanguageId.aspx.

8. Future Work and Conclusions

Recently (January 2008) it was launched a very large European project aimed at constructing a research language resources and tools infrastructure devoted primarily to scholars in Humanities and Social Sciences. A crucial work-package of this project, called CLARIN[11], is dedicated to establishing language resources and tools (LRT) federations, building registry infrastructure and promoting the generalization of linguistic web services and workflow services. Although for its construction phase, the CLARIN project is supposed to deal mostly with the infrastructural aspects, a series of experiments are planned in order to assess the validity of major future development lines and estimate the costs for an operational and persistent multilingual LRT infrastructure for all European languages and several other languages in the world.

There are several other language-processing tools (a collocation identifier, a predicate-argument extractor, a sentence aligner of parallel corpora, two different word aligners for parallel texts, an advanced search engine and a question answering system for Romanian) that are already implemented as stand-alone applications and which we plan to include into the web services platform. The future developments of our present linguistic web services platform as well as the assurance of the interoperability with other existing web service platforms will be our major responsibilities in the CLARIN project.

The access to the web services is research license-based and up to now it was used by various colleagues from Bulgaria, Canada, Denmark, France, Italy, The Netherlands, Romania and USA for processing Romanian texts totalizing more than 2 billion words.

[10] For words with 4 characters or less, the considered prefix and suffix will be the word itself.
[11] http://www.clarin.eu/

9. References

Bentivogli, L., Forner, P., Magnini, B., Pianta, E. (2004). Revising WordNet Domains Hierarchy: Semantics, Coverage, and Balancing. In *Proceedings of COLING 2004 Workshop on "Multilingual Linguistic Resources"*. Geneva, Switzerland, pp. 101--108.
Berners-Lee, T. (2003). Web Services and Semantic Web: Integrating Applications. Talk at the Twelfth International World Wide Web Conference. Budapest, Hungary.
Brants, T. (2000). TnT – A Statistical Part-Of-Speech Tagger. In *Proceedings of the 6th Applied NLP Conference ANLP-2000*. Seattle, WA, pp 224--231.
Esuli, A., Sebastiani, F. (2006). SentiWordNet: A publicly Available Lexical Resourced for Opinion Mining, In Proceedings of LREC2006. Genoa, Italy, pp. 417--422.
Ion, R., Barbu Mititelu, V. (2006). Constrained Lexical Attraction Models. In *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference*. Menlo Park, Calif.: AAAI Press, pp. 297--302.
Ion, R. (2007). Word Sense Disambiguation Methods Applied to English and Romanian. PhD thesis (in Romanian). Romanian Academy, Bucharest.
Mel'čuk, I.A. (1988). *Dependency Syntax: Theory and Practice*. Albany, NY: State University of New York Press.
Niles I., Pease, A. (2001) Towards a Standard Upper Ontology. In *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*. Ogunquit, Maine, pp. 2--9.
Steinberger, R., Pouliquen, B., Widiger, A., Ignat, C., Erjavec, T., Tufiş, D., Varga D. (2006). The JRC-Acquis: A multilingual aligned parallel corpus with 20+ languages. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC'2006)*. Genoa, Italy, pp.2142--2147.
Tufiş, D. (1999). Tiered Tagging and Combined Classifiers. In F. Jelinek, E. Nth (Eds.), *Text, Speech and Dialogue, Lecture Notes in Artificial Intelligence*. Springer, pp. 28--33.
Tufiş, D., Ceauşu, A. (2008). DIAC+: A Professional Diacritics Recovering System. In this volume.
Yuret, D. (1998). Discovery of linguistic relations using lexical attraction. PhD thesis, Department of Computer Science and Electrical Engineering, MIT.