Liner2 — a Customizable Framework for Proper Names Recognition for Polish

Michał Marcińczuk, Jan Kocoń and Maciej Janicki

Wrocław University of Technology, Wrocław, Poland
michal.marcinczuk@pwr.wroc.pl,janekkocon@gmail.com,macjan@o2.pl

Abstract. In the paper we present a customizable and open-source framework for proper names recognition called Liner2. The framework consists of several universal methods for sequence chunking which include: dictionary look-up, pattern matching and statistical processing. The statistical processing is performed using Conditional Random Fields and a rich set of features including morphological, lexical and semantic information. We present an application of the framework to the task of recognition proper names in Polish texts (5 common categories of proper names, i.e. *first names*, *surnames*, *city names*, *road names* and *country names*). The Liner2 framework was also used to train an extended model to recognize 56 categories of proper names which was used to bootstrap the manual annotation of KPWr corpus. We also present the CRF-based model integrated with a heterogeneous named entity similarity function. We show that the similarity function added to the best configuration improved the final result for cross-domain evaluation. The last section presents NER-WS — a web service for proper names recognition in Polish texts utilizing the Liner2 framework and the model for 56 categories of proper names. The web service can be tested using a web-based demo available at `http://nlp.pwr.wroc.pl/inforex/`.

Key words: Proper Names Recognition, Information Extraction, Liner2, Proper Name Similarity Function, NER Web Service

1 Introduction

Named entity recognition (NER) is one of the information extraction tasks. The goal of NER is to identify and classify multiword expressions in running text which refer to some objects from the world. Proper names are one of the classes of named entities. Proper names always denote unique (to some extent, because proper name can be also ambiguous) objects from the word.

Robust recognition of proper names is very important in many tasks from the natural language processing (NLP) domain, i.e. information extraction from medical documentation [1], text anonymization [2], machine translation [3] or forensic linguistics [4].

Recognition of named entities is a well studied task in the case of English [5]. However, only a few researches were conducted so far for Polish. Works for other

Slavic languages are not numerous either; Czech [6], Bulgarian [7] and Ukrainian [8]. Majority of approaches to NER for Polish are based on manual construction of rules, cf., applications in [1, 9–13], machine anonymization [2] and machine translation [3]. Only a few preliminary works were presented on the application of Machine Learning methods to NER, e.g. Memory Based Learning in [14], Decision Trees C4.5 and Naïve Bayes in [15], Conditional Random Fields in [16].

NER is not a trivial task and cannot be effectively performed using naive methods like simple dictionary look-up or pattern matching (see [17]). However, usage of external language resources can be very important to make the NER robust. In the paper we present a NER method which combines statistical approach with various language resources like wordnet, lexicons (proper names and keywords), rules and tools like similarity function for proper names. As the result of the research we constructed a customizable framework for NER for Polish called Liner2, which is also presented in the paper.

The paper is organised as following. Section 2 contains the description of Liner2 framework including its architecture, available modes, supported feature categories and chunkers. Section 3 presents a lexicon of proper names called NELexicon. In section 4 we present two applications of Liner2 framework and NELexicon. In section 5 we present and discuss an approach to improve the lexicon recall by applying a similarity function for proper names. Section 6 presents a web service for proper name recognition build on top of the Liner2 framework.

2 Liner2 Framework

This section presents the technical and usability aspects of Liner2 framework. The technical aspect refers to the framework architecture and division into components (Section 2.1). The usability aspect refers to available functions and configuration options (Sections from 2.2 to 2.5).

2.1 Architecture

Liner2 is divided into 5 components (see Figure 1). They are:

- **main controller** — it controls the data flow between components. The controller behavior depends on Liner2 running mode (see Section 2.2),
- **data reader** — reads the data from file or standard input and transforms them into internal structure (see Section 2.3),
- **feature space generator** — extends the feature space by generating new features according to given configuration (see Section 2.4),
- **chunker controller** — creates a set of chunkers arranged into a pipeline (see Section 2.5),
- **result writer** — writes processed data to a file or computes and prints evaluation of the chunking result (see Section 2.3).

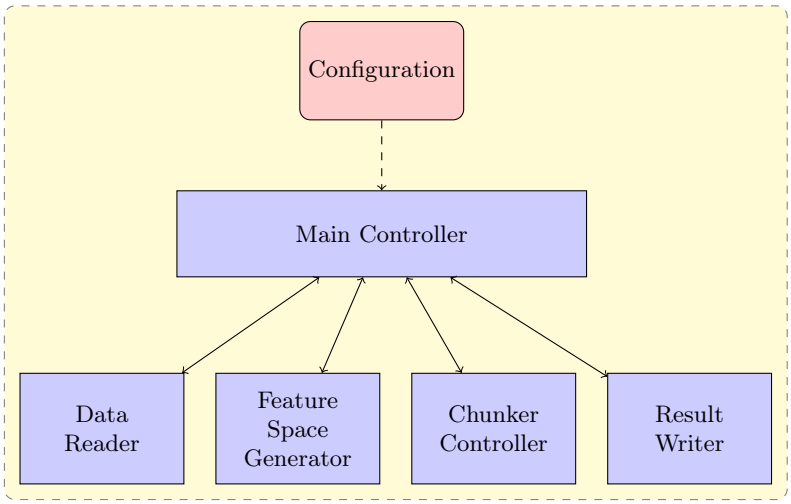

Fig. 1. Liner2 framework architecture.

2.2 Modes

Liner2 can be run in several modes suitable for different applications. Single document can be easily processed in `pipe` mode. Multiple documents and interactive processing can be done in `batch` mode. Feature generation can be done in `convert` mode. Quick evaluation can be performed using `eval` or `evalcv` mode. A brief summary of all modes is presented in Table 1. The following subsections describe them in details.

Mode	Brief description
`batch`	console interactive mode
`convert`	convert text from one format to another
`eval`	perform evaluation on given data
`evalcv`	perform n-fold cross validation on given data
`pipe`	processe single document in pipe-style
`train`	construct and serialize chunkers

Table 1. List of available Liner2 running modes

Mode *batch* is used to process multiple documents, test Liner2 in interactive manner and run Liner2 as subprocess. After loading data user is prompted to enter text to process. The text can be entered in CCL, IOB and plain format (see Section 2.3). For plain format Liner2 requires *maca* for morphological analysis

and optionally *wmbt* for tagging. Below is a sample interaction with Liner2 in the `batch` mode:

```
> $liner2 batch -ini model.ini -i plain -o tuples -maca -
# Loading, please wait...
# Enter a sentence and press Enter.
# To finish, enter 'EOF'.
> Jan Nowak mieszka w Krakowie.
(0,2,PERSON_FIRST_NAM,"Jan")
(3,7,PERSON_LAST_NAM,"Nowak")
(16,23,CITY_NAM,"Krakowie")
```

Mode *convert* is used to convert data between formats (IOB and CCL) and also to dump feature space generated for the input data. Below is an example of feature space dump generated for a sample sentence (`features.txt` contains feature definition presented in Figure 2).

Input (content of *input.xml* file):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cesAna SYSTEM "xcesAnaIPI.dtd">
<chunkList xmlns:xlink="http://www.w3.org/1999/xlink">
<chunk id="ch1" type="p">
  <sentence>
  <tok>
    <orth>Pan</orth>
    <lex disamb="1"><base>Pan</base><ctag>subst:sg:nom:m1</ctag></lex>
    <lex disamb="1"><base>pan</base><ctag>subst:sg:nom:m1</ctag></lex>
  </tok>
  <tok>
    <orth>Marek</orth>
    <lex disamb="1"><base>Marek</base><ctag>subst:sg:nom:m1</ctag></lex>
    <lex disamb="1"><base>marek</base><ctag>subst:sg:nom:m1</ctag></lex>
  </tok>
  <tok>
    <orth>Groszek</orth>
    <lex disamb="1"><base>groszek</base><ctag>subst:sg:nom:m3</ctag></lex>
  </tok>
  <tok>
    <orth>jest</orth>
    <lex disamb="1"><base>być</base><ctag>fin:sg:ter:imperf</ctag></lex>
  </tok>
  <tok>
    <orth>prezesem</orth>
    <lex disamb="1"><base>prezes</base><ctag>subst:sg:inst:m1</ctag></lex>
  </tok>
  <tok>
    <orth>Company</orth>
    <lex disamb="1"><base>company</base><ctag>subst:sg:nom:m1</ctag></lex>
  </tok>
  <tok>
    <orth>SA</orth>
    <lex disamb="1"><base>sa</base><ctag>subst:sg:nom:f</ctag></lex>
  </tok>
  </sentence>
</chunk>
</chunkList>
```

Command:

```
liner2 convert -i ccl -f input.xml -o iob -ini features.txt
```

Output:

```
-DOCSTART CONFIG FEATURES orth base ctag syn hyp1 class case number gender pattern
-DOCSTART FILE ch1
Pan      Pan     subst:sg:nom:m1   Pan     Pan     subst nom  sg m1   UPPER_INIT O
Marek    Marek   subst:sg:nom:m1   Marek   Marek   subst nom  sg m1   UPPER_INIT O
Groszek  groszek subst:sg:nom:m3   groszek groszek subst nom  sg m3   UPPER_INIT O
jest     być     fin:sg:ter:imperf być     być     fin   null sg null ALL_LOWER  O
prezesem prezes  subst:sg:inst:m1  prezes  głowa   subst inst sg m1   ALL_LOWER  O
Company  company subst:sg:nom:m1   company company subst nom  sg m1   UPPER_INIT O
SA       sa      subst:sg:nom:f    sa      sa      subst nom  sg f    ALL_UPPER  O
```

Mode *eval* is used to evaluate the performance of a model on a given corpus.

Command:

```
liner2 eval -i ccl -f input.xml -ini features.txt \
    -chunker c1:CHUNKER -use c1
```

Output (fragments):

```
(...skipped...)

Sentence #4924 from /nlp/corpora/pwr/wikinews4//annotated/0099843.txt

Text  : Budynek powstaje przy ulicy Władysława Reymonta w pobliżu skrzyżowania
        z  ulicą Józefa Chłopickiego ,  naprzeciwko cmentarza żydowskiego .
Tokens: 1______ 2_______ 3___ 4____ 5_________ 6_______ 7 8______ 9___________
        10 11___ 12____ 13__________ 14 15_________ 16_______ 17_________ 18

Chunks:
  TruePositive ROAD_NAM [5,6] = Władysława Reymonta
  TruePositive ROAD_NAM [12,13] = Józefa Chłopickiego
  FalsePositive PERSON_FIRST_NAM [5,5] = Władysława
  FalsePositive PERSON_LAST_NAM [6,6] = Reymonta
  FalsePositive PERSON_FIRST_NAM [12,12] = Józefa
  FalsePositive PERSON_LAST_NAM [13,13] = Chłopickiego

(...skipped...)

Annotation           &   TP &   FP &   FN & Precision & Recall  & F$_1$   \\
\hline
ROAD_NAM             &   11 &   10 &   45 &    52,38% &  19,64% &  28,57% \\
PERSON_LAST_NAM      &  824 &   30 &  813 &    96,49% &  50,34% &  66,16% \\
COUNTRY_NAM          & 1504 &   92 &  251 &    94,24% &  85,70% &  89,76% \\
PERSON_FIRST_NAM     &  825 &   28 &  287 &    96,72% &  74,19% &  83,97% \\
CITY_NAM             &  492 &  166 &  173 &    74,77% &  73,98% &  74,38% \\
\hline
*TOTAL*              & 3656 &  326 & 1569 &    91,81% &  69,97% &  79,42%
```

Mode *evalcv* is used to evaluate the performance of a model using n-fold cross validation on given set of documents. The documents must be a priori divided into n folds. The output contains evaluation for every fold and summary for all folds. The results are presented in the same way as in mode `eval`.

Mode *pipe* is used to process a single document. The document can be read form a file (CCL, IOB or plain format) or from standard input and saved to a file (CCL, IOB or tuple format) or printed to standard output.

Mode *train* is used to train and serialize chunkers defined with the `-chunker` parameter.

2.3 Input/Output Formats

Liner2 handles following data formats:

Plain text can be processed by Liner2 after providing the `-maca` parameter. The parameter value can be: `-` if *maca* is installed in the system or `PATH` to a `maca-analyze` program.

CCL is an XML-based format used by the *maca* tool [18]. Document can be read and write in CCL format.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cesAna SYSTEM "xcesAnaIPI.dtd">
<chunkList xmlns:xlink="http://www.w3.org/1999/xlink">
  <chunk id="ch1" type="p">
    <sentence>
      <tok>
        <orth>Ala</orth>
        <lex disamb="1"><base>Ala</base><ctag>subst:sg:nom:f</ctag></lex>
      </tok>
      <tok>
        <orth>mieszka</orth>
        <lex disamb="1"><base>mieszkać</base><ctag>fin:sg:ter:imperf</ctag></lex>
      </tok>
      <tok>
        <orth>w</orth>
        <lex disamb="1"><base>w</base><ctag>prep:loc:nwok</ctag></lex>
      </tok>
      <tok>
        <orth>Krakowie</orth>
        <lex disamb="1"><base>Kraków</base><ctag>subst:sg:loc:m3</ctag></lex>
      </tok>
      <ns/>
      <tok>
        <orth>.</orth>
        <lex disamb="1"><base>.</base><ctag>interp</ctag></lex>
      </tok>
    </sentence>
  </chunk>
</chunkList>
```

IOB is a format used in CoNLL-2003 shared task named entity recognition[1]. In this format every token is represented as a set of space-seperated values in a single row. Sentences are separated with empty lines. Every chunk is encoded using two symbols: `B-NAME` and `I-NAME`, where `NAME` is a chunk category, `B-NAME` (*begins*) is assigned to the first token and `I-NAME` (*inside*) to the other tokens of the chunk. Tokens which do not belong to any chunk are marked with `O` symbol (*outside*).

IOB without chunks

```
Ala         Ala          subst:sg:nom:f
mieszka     mieszkać     fin:sg:ter:imperf
w           w            prep:loc:nwok
Krakowie    Kraków       subst:sg:loc:m3
.           .            interp
```

IOB with chunks

```
Ala         Ala          subst:sg:nom:f       B-PERSON_NAM
mieszka     mieszkać     fin:sg:ter:imperf    O
w           w            prep:loc:nwok        O
Krakowie    Kraków       subst:sg:loc:m3      B-CITY_NAM
.           .            interp               O
```

Tuple format is used to display result of chunking in a compact way.

Tuple format

```
(1,3,PERSON_NAM,"Ala")
(11,18,CITY_NAM,"Krakowie")\end{verbatim}
```

2.4 Feature Generator

Liner2 contains a stand-alone feature generator which can generate 21 types of features. The features can be grouped into four categories, i.e. orthographic, morphological, lexicon-based and wordnet-based features.

1. **Orthographic features**
   - **orth** — a word itself, in the form in which it is used in the text,
   - n**-prefix** — n first characters of the encountered word form, where $n \in \{1, 2, 3, 4\}$. If the word is shorter than n, the missing characters are replaced with '_'.
   - n**-suffix** — n last characters of the encountered word, where $n \in \{1, 2, 3, 4\}$. If the word is shorter than n, the missing characters are replaced with '_'. We use prefixes to fill the gap of missing inflected forms of proper names in the gazetteers.
   - **pattern** — encode pattern of characters in the word:
     - ALL_UPPER — all characters are upper case letters, e.g. "NASA",

[1] Web page: `http://www.cnts.ua.ac.be/conll2003/ner/`

    - ALL_LOWER — all characters are lower case letters, e.g. “rabbit”
    - DIGITS — all character are digits, e.g. “102”,
    - SYMBOLS — all characters are non alphanumeric, e.g. “-_-”’,
    - UPPER_INIT — the first character is upper case letter, the other are lower case letters, e.g. “Andrzej”,
    - UPPER_CAMEL_CASE — the first character is upper case letter, word contains letters only and has at least one more upper case letter, e.g. “CamelCase”,
    - LOWER_CAMEL_CASE — the first character is lower case letter, word contains letters only and has at least one upper case letter, e.g. “pascalCase”,
    - MIXED — a sequence of letters, digits and/or symbols, e.g. “H1M1”.
  - **binary orthographic features**, the feature is 1 if the condition is met, 0 otherwise. The conditions are:
    - *(word) starts with an upper case letter*,
    - *starts with a lower case letter*,
    - *starts with a symbol*,
    - *starts with a digit*,
    - *contains upper case letter*,
    - *contains a lower case letter*,
    - *contains a symbol*
    - *contains digit*.

    The features are based on filtering rules described in [17], e.g., first names and surnames start from upper case and do not contain symbols. To some extent these features duplicate the *pattern* feature. However, the *binary features* encode information on the level of single characters, while the aim of the *pattern* feature is to encode a repeatable sequence of characters.
2. **Morphological features** — are motivated by the NER grammars which utilise morphological information [9]. The features are:
  - **base** — a morphological base form of a word,
  - **ctag** — morphological tag generated by tagger,
  - **part of speech**, **case**, **gender**, **number** — enumeration types according to tagset described in [19].
3. **Lexicon-based features** — one feature for every lexicon. If a sequence of words is found in a lexicon the first word in the sequence is set as *B* and the other as *I*. If word is not a part of any dictionary entry it is set to *O*.
4. **Wordnet-base features** — are used to generalise the text description and reduce the observation diversity. The are two types of these features:
  - **synonym** — word’s synonym, first in the alphabetical order from all word synonyms in Polish Wordnet. The sense of the word is not disambiguated,
  - **hypernym *n*** — a hypernym of the word in the distance of n, where $n \in \{1, 2, 3\}$

The feature space is defined using a set of `-feature` parameters. The generated features can be used by any chunker, like CRF or rule-based (see Section 2.5 for chunkers description). Figure 2 presents a sample definition of feature space.

```
-feature orth
-feature base
-feature ctag
-feature syn
-feature hyp1
-feature class
-feature case
-feature number
-feature gender
```

Fig. 2. Sample set of features definition.

```
-template t1:orth:-2:-1:0:1:2
-template t1:base:-2:-1:0:1:2
-template t1:syn:-2:-1:0:1:2
-template t1:hyp1:-2:-1:0:1:2
-template t1:hyp2:-2:-1:0:1:2
-template t1:hyp3:-2:-1:0:1:2
-template t1:class:-1:0:1
-template t1:case:0
-template t1:gender:-2:-1:0:1:2
```

Fig. 3. Sample template definition.

2.5 Sequence Chunking Methods

Liner2 offers several methods for sequence chunking, which can be divided into five categories:

1. **Supervised statistical models** — perform chunking on the basis of statistial model created on a training corpus.
2. **Rule-based chunkers** — perform chunking on the basis of rules.
3. **Gazetteer-based chunkers** — perform chunking on the basis of lexicons.
4. **Iterative** — the chunking process is performed in several iterations. The result from one iteration is used in the next iteration.
5. **Ensamble** — perform majority voting or union of other chunkers.

Most of the chunkers can be defined from command line or in configuration file and do not require any modifications in the source code. The only exception is the *heuristic* chunker, which requires to encode the rules as Java functions. Table 2 presents a summary of all available chunkers and the following subsections describe them in details.

The chunkers are defined using the `-chunker` parameter as following:

```
-chunker chunker_name:chunker_description
```

where `chunker_name` is an unique chunker name among all defined chunkers. The `chunker_description` defines the chunker. Format descriptions for all chunkers are presented in the following subsections.

Type	Symbol	Brief description
statistical	`crfpp-train`	train and serialize a CRF model
statistical	`crfpp-load`	deserialize an existing CRF model
rule-based	`heuristic`	set of rules defined as Java functions
rule-based	`wccl`	rules written in WCCL
lexicon-based	`dict-compile`	compile and serialize an unambigous dictionary look-up
lexicon-based	`dict-load`	deserialize an unambigous dictionary look-up
lexicon-based	`dict-full-compile`	compile and serialize a full dictionary look-up
lexicon-based	`dict-full-load`	deserialize a full dictionary look-up
iterative	`adu`	run given chunker twice, after first iteration update lexicon-base features
ensamble	`ensamble`	combine chunker results by applying a majority voting or merging

Table 2. List of available sequence chunking methods

Supervised statistical chunker

```
crfpp-train:p=P:template=T:(iob|ccl|data)=DATA:model=FILE
crfpp-load:PATH
```

Liner2 offers a trainable CRF-based chunker. The chunker utilises an existing open source implementation of the Conditional Random Fields called CRF++[2]. The chunker can be defined in two ways: `crfpp-train` — trains the model from scratch or `crfpp-load` — loads existing model.

Training model from scratch

```
crfpp-train:p=P:template=T:(iob|ccl|data)=DATA:model=FILE
```

where:

– **P** is a number of threads used to train the model (this parameters is passed to the CRF++),

[2] Web page: `http://crfpp.googlecode.com/svn/trunk/doc/index.html`

- **T** is a name of features template used to train the model. The feature template is defined in the configuration file and is transformed to a format accepted by CRF++ (see Figure 3),
- **DATA** is a path to a corpus used to train. The corpus can be defined directly by providing path to a file (`iob=PATH` or `ccl=PATH`) or by a name defined with `-corpus` parameter (`data=NAME`).
- **FILE** is a file name where the trained model will be saved. The saved model can be loaded using the `crfpp-load` chunker.

Loading existing model

```
crfpp-load:PATH
```

where **PATH** is a path to a serialized CRF model trained and saved with `crfpp-train` chunker.

Rule-based

```
heuristic:comma-separated-list-of-rules
```

This module chunks the text using Java-coded rules. Every rule is implemented as a Java function. The module requires changes in the Liner2 source code. The rules have access to complete feature space defined with the `-feature` parameters (see Section 2.4).

Sample rules implemented in the Liner2:

- **city** — set of rules which recognize city-related patterns, e.g. "*postal_ code upper_ case*",
- **general-ign-dict** — if given token's part of speech is *ign* (unknown) and any of the lexicon-based features has value *B* then mark the token with a symbol refering to the lexicon name,
- **general-camel-base** — if given token starts from an upper case letter and is marked by one lexicon-based feature as *B* then mark the token with a symbol refering to the lexicon name,
- **person** — set of rules which recognize patterns like "Mr. *first_ name surname*", "J. K. *surname*", etc.,
- **road** — if given token starts from upper case letter, previous token is present in lexicon of road name prefixes and following token is number, then mark the token as road name.
- **road-prefix** — if given token is present in lexicon of road names and previous token is present in lexicon of road name prefixes then mark the token as road name,

```
wccl:WCCL_FILE
```

This chunker requires an external tool called *wccl*[3]. The chunker creates annotations on the basis of rules written in WCCL format. A sample rule, that mark person name is presented below.

```
apply(
  match(
    optional(
      is('person_position_full'),
      optional( text(',') ),
    ),
    optional( is('person_title') ),
    is('person_first_nam_gaz'),
    optional( is('person_first_nam_gaz') ),
    is('person_last_nam_gaz'),
    optional( is('person_suffix') )
  ),
  actions(
    mark(:3, 'PERSON_FIRST_NAM'),
    mark(:4, 'PERSON_FIRST_NAM'),
    mark(:5, 'PERSON_LAST_NAM'),
    mark(:3, :5, 'PERSON_NAM')
  )
)
```

Lexicon-based — includes two types of chunkers: `dict` and `dict-full`. `dict` marks all unambiguous proper names that are not common words. `dict-full` marks every proper name that is present in the lexicon.

```
dict-compile:dict=DICT:common=COMMON:model=BIN
dict-load:BIN
```

where:

- **DICT** is a path to a lexicon of proper names,
- **COMMON** is a path to a list of common words,
- **BIN** is a path to a file where the compiled dictionary will be saved.

`dict-compile` is used to compile a lexicon from scratch and save it in a file. `dict-load` is used to load compiled lexicon from a file.

```
dict-full-compile:dict=DICT:model=BIN
dict-full-compile:dict=DICT
dict-full-load:BIN
```

[3] Web page: `http://nlp.pwr.wroc.pl/redmine/projects/joskipi/wiki`.

3 Lexicon of Proper Names

For the need of lexicon-based chunkers we have prepared a large lexicon of proper names called NELexicon[4]. We have processed several resources from the Internet containing lists of proper names of different categories. The NELexicon contains 1.4 millions unique entries "name:category" and 1.37 millions of unique names among all categories. Detailed statistics of NELexicon are presented in Table 3.

Count	Annotation	Count	Annotation	Count	Annotation
866	bay	8519	inst_org	371378	person_last
10	bridge	1930	island	61	political_ party
163	canal	2052	lake		
749	cape	1358	landscape	221	powiat
259	cave	3	mausoleum	385	province
107285	city	660	mountain	2725	river
14	coast	460	mountain	40862	road
415970	company	516	nation	131	sea
48	continent	65	oasis	1469	square
108	desert	11	ocean	272	strait
2	district	419039	organization	1137	subdivision
115	fortification	322	park	72	swamp
46	glacier	154	peninsula	47	temple
8	graveyard	66	periodic	22	title
327	institution	22434	person_first	71	waterfall

Table 3. Statistics of proper names in NELexicon.

4 Liner2 Applications

4.1 Recognition of Basic Proper Names

Liner2 has already been used to construct and evaluate a model for recognition 5 basic categories of proper names in Polish texts, i.e. first names, surnames, names of cities, countries and roads. The model was evaluated in two ways — single domain evaluation following 10-fold cross validation on a Corpus of Stock Exchange Reports (CSER) and cross-domain evaluation on a Corpus of Economic News from Wikinews (CEN). The model is a cascade of following chunkers (Figure 4 presents the chunkers configuration):

- CRF-based tagger (the complete list of features is presented in [20] and [21]),
- unambiguous dictionary look-up for country and city names,
- set of Java-coded rules.

```
-ini {INI_PATH}/features.ini
-chunker c1:crfpp-load:{INI_PATH}/crf_cicling.bin
-chunker c2:dict-load:{INI_PATH}/dict.bin:types=COUNTRY_NAM,CITY_NAM
-chunker c3:heuristic:person,road,road-prefix,city
-chunker en:ensamble:c1+c2+c3
-chunker f:adu:en
-use en
```

Fig. 4. Liner2 configuration for proper name recognition model.

In [20] we presented a basic model and in [21] we proposed and evaluated several ways of improving the performance of the model. The final configuration including all optimizations obtained in total 95.08% of precision and 96.07% of recall what is very high result (see Table 4 for detailed results). In the cross-domain evaluation the model obtained lower results on the level of 91.44% of precision and 70.53% of recall (see Table 5 for detailed results), but still acceptable performance.

Annotation	TP	FP	FN	Precision	Recall	F-measure
first names	654	13	29	98.05%	95.75%	96.89%
surnames	636	17	52	97.40%	92.44%	94.85%
countries	441	36	32	92.45%	93.23%	92.84%
cities	1947	128	34	93.83%	98.28%	96.01%
roads	377	16	19	95.93%	95.20%	95.56%
All	4055	210	166	95.08%	96.07%	95.57%

Table 4. 10-fold cross validation on CSER.

Annotation	TP	FP	FN	Precision	Recall	F-measure
first names	827	27	285	96.84%	74.37%	84.13%
surnames	831	31	806	96.40%	50.76%	66.51%
countries	1524	92	231	94.31%	86.84%	90.42%
cities	492	186	173	72.57%	73.98%	73.27%
roads	11	9	45	55.00%	19.64%	28.95%
All	3685	345	1540	91.44%	70.53%	79.63%

Table 5. Cross domain evaluation on CEN.

[4] Web page: `http://nlp.pwr.wroc.pl/en/tools-and-resources/nelexicon`

4.2 Proper Names Bootstrapping

Liner2 was also used to support the process of manual annotation of KPWr corpus [22] with 56 categories of proper names. The model was trained on 400 documents manually annotated by linguists using the configuration used for the basic model. The extended model was then run on another 400 unannotated documents. The results of automatic proper names annotation were manually verified by linguists. The verification was performed using a dedicated perspective in the Inforex system [23] (see Figure 5). For every recognized annotation linguist could choose to accept the annotation, change the proper name category (if the annotation boundary was recognized properly but the category was incorrect), discard the annotation (if both the annotation boundary and category were incorrect) or leave the evaluation for later (for example, if not sure at the moment).

The results of verification were stored in the database and used to measure the performance of bootstrapping. By applying this procedure we added more than 1700 new annotations. Evaluating the correctness of annotation boundaries and proper name categories, the model obtained 71% of precision and 42% of recall. When considering only the correctness of annotation boundaries, the model obtained 93% of precision and 54% of recall. This means, than more than half of annotations were automatically added to the documents and 71% of them had correct category. Only the other 30% required correction of the category and 46% of annotations had to be added manually from scratch.

5 Improving the Lexicon Recall

To improve the recall of lexicon-based features were applied a similarity function for named entities. The following subsections present the concept of heterogeneous named entity similarity function, the ways how the function could be combined with existing models for proper names recognition and evaluation of the presented approaches.

5.1 Introduction

The basic application of the heterogeneous named entity similarity function (henceforth *NamEnSim*) is to find for an input word w_I its lemma or an morphological word form in *NELexicon* if it exists. In practice, similar word set P returned by similarity function for w_I should contain its proper lemma w_L and the decision function value for all other pairs: $\langle w_I, w_O \rangle$ where $w_O \in P$ and $w_O \neq w_L$ should be lower than for $< w_I, w_L >$ [24]. This task is different than morphological guessing, e.g. [25] which is aimed at generation of lemmas for unknown words on the basis of an *a tergo* index.

Named entities similarity function is based on several string metrics, combined into a complex one using Logistic Regression. Similar approach as aggregation of different string metrics in named-entity similarity task (using Supported

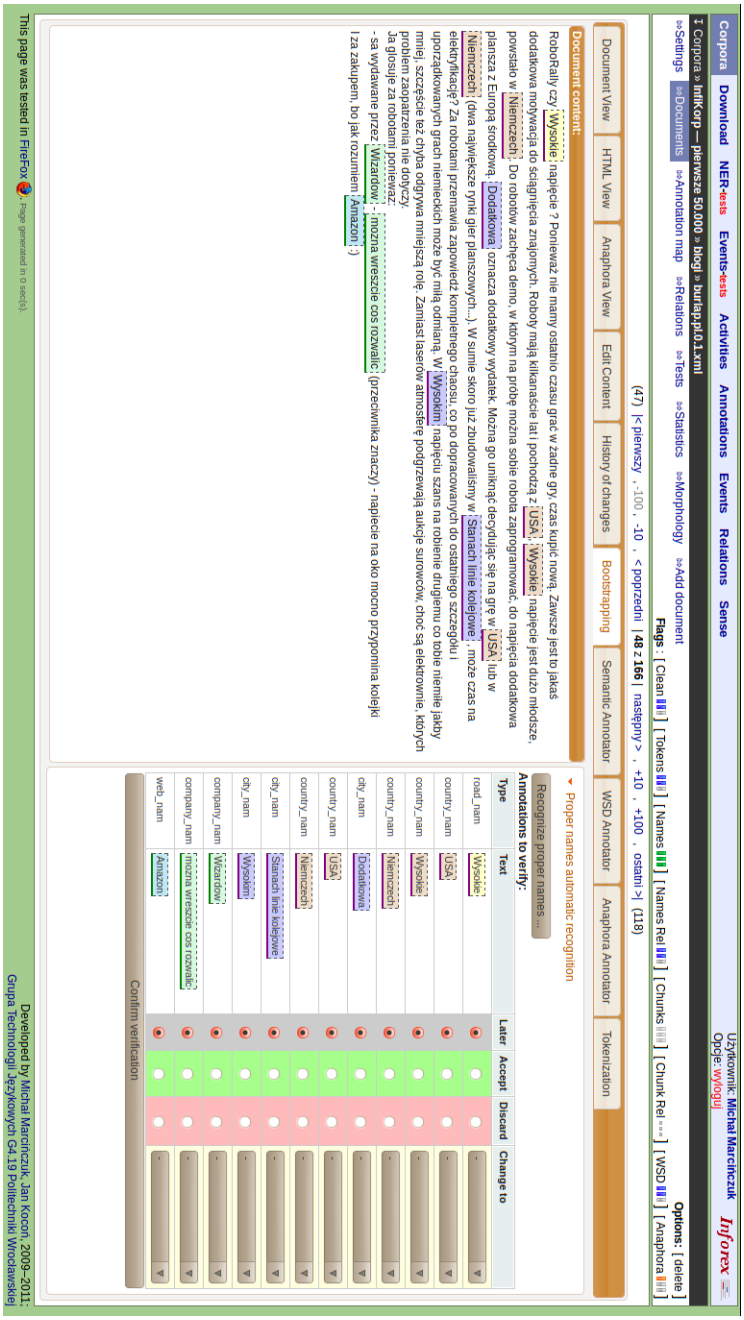

Fig. 5. Perspective for bootstrapping verification in the Inforex system.

Vector Machine) was presented in [26, 27]. A complex similarity function trained on the reduced set of single metrics (*NamEnSim5*) described in [24] was used as the additional feature in CRF-based model for proper name recognition in Polish texts.

5.2 Evaluation of CRF-based model

In evaluation process we used two corpora: CEN and CSER. Cross-domain evaluation on CEN trained on CSER was performed in 3 variants of different feature configuration and three versions of dictionaries representing search space for dictionary/similarity features:

- Variants:
  - IC – *Initial Configuration*, top features configuration for CRF-based model, described in [21].
  - SC – *Similarity Configuration*, dictionary features from IC are replaced by similarity features.
  - BC – *Best Configuration*, similarity features are added to IC.
- Versions:
  - NB – *No Bases*, search space for dictionary/similarity features contains only proper names from NELexicon.
  - B – *Bases*, base forms (lemmas) of NER (IC+NB configuration) results (only False Negative – unrecognized proper names) are added to NB.
  - BF – *Base and Forms*, unrecognized proper names (NER IC+NB False Negative results) added to B.

Table 6 contains the results of cross-domain evaluation on CEN trained on CSER for different test variants and versions.

Variant	**Version**	**TP**	**FP**	**FN**	**Precision** [%]	**Recall** [%]	**F-measure** [%]
NB	IC	3681	346	1544	91.41	70.45	79.57
NB	SC	3509	338	1716	91.21	67.16	77.36
NB	BC	3729	329	1496	**91.89**	**71.37**	**80.34**
B	IC	3998	347	1227	92.01	76.52	83.55
B	SC	3698	340	1527	91.58	70.78	79.84
B	BC	4031	328	1194	**92.48**	**77.15**	**84.12**
BF	IC	4107	348	1118	92.19	78.60	84.86
BF	SC	3733	345	1492	91.54	71.44	80.25
BF	BC	4141	333	1084	**92.56**	**79.25**	**85.39**

Table 6. Results of evaluation for different test variants and versions.

The analysis of the results showed that the best evaluation results can be obtained by adding similarity features to the top features configuration (baseline), described in [21]. Replacing the dictionary features from the baseline with

similarity features does not improve the results in the cross-domain evaluation. The best improvement (comparing to the baseline IC results in each test variant) can be observed with search space, extended by adding base forms of proper names, which were not recognized using baseline configuration (B variant). For each variant the difference between best configuration and initial configuration is: *NB – 0.77%, B – 0.91%, BF – 0.64%.*

6 NER-WS Web Service

The NER-WS web service makes the Liner2 available through Web. It utilises basic web services technologies [28]: SOAP protocol for data exchange and WSDL description language.

The service's implementation is based on a three-layer architecture, that was earlier successfully applied for other Natural Language Processing web services, e.g. the TaKIPI tagger [29]. This architecture was proved to be robust and well-scalable.

6.1 Usage

The web service is accessed through the WSDL description file, which contains URL addresses of endpoints, defines available functionalities and structure of request and answers, that are used for communication between the client and the server.

NER-WS has an *asynchronous* model of communication, i.e. it doesn't keep the connection with the client up within the time of computation. Instead, it allows two types of client requests, which are immediately answered and the connection is terminated:

- **Annotate** — client sends text to annotate, the WebService returns a unique *request identifier*,
- **GetResult** — client sends the request identifier, the WebService returns status of the request (one of: `QUEUED`, `PROCESSING`, `READY`, `ERROR`, `FINISHED`) and the result or error code if available).

The probing of WebService with GetResult requests (e.g. once per second) should be implemented on the client side.

6.2 Architecture

NER-WS has a three layer architecture (see Figure 6): the *SOAP server layer* communicates with clients, accepting requests and sending back responses. The texts to process, as well as processing results, are stored by the *database layer*. Text annotating is done by the *daemon layer*, which consists of many `liner2` daemons, running parallelly.

This architecture was chosen because of the following advantages:

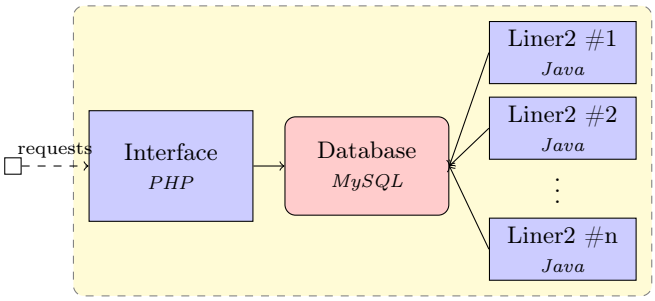

Fig. 6. NER-WS architecture

- **asynchronous communication** — the connection with client need not to be kept up during the computation,
- **distributed processing** — the daemons may run on many different machines and communicate with other layers through network,
- **parallel processing** — many requests may be handled simultaneously,,
- **scalability** — the number of daemons running may be adjusted to the needs and available resources; an increase of the number of daemons increases the processing speed and capacity of the WebService.

An additional mechanism was implemented to improve the server-side robustness: the daemons, that are inactive for a long period of time (e.g. because they were abnormally terminated or have lost connection to the server) are automatically detected and disconnected from the system.

6.3 Demo

A mockup instance of NER-WS is hosted at the Poznań Supercomputing and Networking Center (PSCN)[5]. The web service can be tested using a web-based application[6] that is a part of Inforex system (see Figure 7).

7 Summary

In the paper we presented a customizable framework for proper names recognition called Liner2. The framework was used to construct a model for recognition 5 basic categories of proper names and was also applied to bootstrap the annotation of KPWr corpus with 56 categories of proper names. We also presented a preliminary results of combining the Liner2 with similarity function for proper names.

The Liner2 framework was developed mainly for proper names recognition task for Polish. However, many of the existing components can be used for

[5] Home page: `http://www.man.poznan.pl/online/en/`
[6] Web page: `http://nlp.pwr.wroc.pl/inforex/index.php?page=ner`

other sequence labelling tasks, like syntactic chunks recognition. Most of the component are also language independent. The other, with some effort, could be also adopted to other languages than Polish.

License and Access

Liner2 — is available on a free license (GPL) and can be downloaded from the following page: `http://nlp.pwr.wroc.pl/liner2`,

Liner2 models — can be also downloaded from `http://nlp.pwr.wroc.pl/liner2`,

NER-WS demo — a web-based application utilizing NER-WS is available at `http://nlp.pwr.wroc.pl/inforex/index.php?page=ner`,

Acknowledgements

Work financed by NCBiR NrU.: SP/I/1/77065/10 (project SyNaT[7]) and Innovative Economy Programme project POIG.01.01.02-14-013/09 (project NEKST[8]).

References

1. Mykowiecka, A., Kupść, A., Marciniak, M., Piskorski, J.: Resources for Information Extraction from Polish texts, in Proceedings of the 3rd Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics, (LTC'2007), Poznań, Poland, 5-7.10.2007 (2007)
2. Graliński, F., Jassem, K., Marcińczuk, M.: An Environment for Named Entity Recognition and Translation, in L. Màrquez and H. Somers, editors, Proceedings of the 13th Annual Conference of the European Association for Machine Translation, pp. 88–95, Barcelona, Spain (2009)
3. Graliński, F., Jassem, K., Marcińczuk, M., Wawrzyniak, P.: Named Entity Recognition in Machine Anonymization, in M. A. Kłopotek, A. Przepiórkowski, A. T. Wierzchoń, and K. Trojanowski, editors, Recent Advances in Intelligent Information Systems., pp. 247–260, Academic Publishing House Exit (2009)
4. Marcińczuk, M., Zaśko-Zielińska, M., Piasecki, M.: Structure Annotation in the Polish Corpus of Suicide Notes, in Proceedings of TSD 2011, pages 419–426, Brno, Czech Republic, Springer (2011)
5. Marrero, M., Sánchez-Cuadrado, S., Lara, J. M., and Andreadakis, G.: Evaluation of Named Entity Extraction Systems, Research in Computing Science, 41:47–58 (2009)
6. Kravalová, J. and Žabokrtský, Z.: Czech named entity corpus and SVM-based recognizer, in Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration, pp. 194–201, ACL, Suntec, Singapore (2009)
7. Osenova, P. and Kolkovska, S.: Combining the Named-entity Recognition Task and NP Chunking Strategy for Robust Pre-processing., in Proc. of The 1st Workshop on Treebanks and Linguistic Theories, Sozopol, Bulgaria, pp. 167–182 (2002)

[7] Web page: `http://www.synat.pl`.
[8] Web page: `http://www.ipipan.waw.pl/nekst/`.

8. Katrenko, S. and Adriaans, P.: Named Entity Recognition for Ukrainian: A Resource-Light Approach, in Proceedings of the Workshop on Balto-Slavonic Natural Language Processing: Information Extraction and Enabling Technologies, pp. 88–93, ACL, Prague, Czech Republic (2007)
9. Piskorski, J.: Extraction of Polish named entities, in Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004, pp. 313–316, Association for Computational Linguistics, Prague, Czech Republic (2004)
10. Piskorski, J.: Named-Entity Recognition for Polish with SProUT, in L. Bolc, Z. Michalewicz, and T. Nishida, editors, Intelligent Media Technology for Communicative Intelligence, volume 3490 of LNCS, pp. 122–133, Springer (2004)
11. Urbańska, D. and Mykowiecka, A.: Multi-words Named Entity Recognition in Polish texts, in SLOVKO 2005 – Third International Seminar Computer Treatment of Slavic and East European Languages, Bratislava, Slovakia, pp. 208–215 (2005)
12. Abramowicz, W., Filipowska, A., Piskorski, J., Węcel, K., and Wieloch, K.: Linguistic Suite for Polish Cadastral System, in Proceedings of the LREC'06, pp. 53–58, Genoa, Italy, ISBN 2-9517408-2-4 (2006)
13. Savary, A. and Piskorski, J.: Lexicons and Grammars for Named Entity Annotation in the National Corpus of Polish, in Mieczysław A. Kłopotek, Małgorzata Marciniak, Agnieszka Mykowiecka, W. Penczek, and S. T. Wierzchnoń, editors, Intelligent Information Systems, pp. 141–154, Siedlce, ISBN 978-83-7051-580-5 (2010)
14. Marcińczuk, M. and Piasecki, M.: Pattern Extraction for Event Recognition in the Reports of Polish Stockholders, in Proceedings of the International Multiconference on Computer Science and Information Technology, volume 2, pp. 275–284, Wisła, Poland, October 15–17, 2007, ISSN 1896-7094 (2007)
15. Marcińczuk, M.: Pattern Acquisition Methods for Information Extraction Systems, Master's thesis, Blekinge Tekniska Högskola, Sweden (2007)
16. Savary, A. and Waszczuk, J.: Narzedzia do anotacji jednostek nazewniczych. In Adam Przepiórkowski, Mirosław Bańko, Rafał L. Górski, and Barbara Lewandowska-Tomaszczyk, editors, Narodowy Korpus Języka Polskiego [Eng.: National Corpus of Polish], pages 225–252. Wydawnictwo Naukowe PWN, Warsaw (2012)
17. Marcińczuk, M. and Piasecki, M.: Statistical Proper Name Recognition in Polish Economic Texts. In Control and Cybernetics, (2011)
18. Radziszewski, A., Śniatowski, T.: Maca: a configurable tool to integrate Polish morphological data, in Proceedings of FreeRBMT11, Barcelona, Spain (2011)
19. Przepiórkowski, A.: The IPI PAN Corpus: Preliminary version, Institute of Computer Science, Polish Academy of Sciences, Warsaw (2004)
20. Marcińczuk, M., Stanek, M., Piasecki, M., Musiał, A.: Rich Set of Features for Proper Name Recognition in Polish Texts. In Proceedings of the International Joint Conference Security and Intelligent Information Systems, 2011. Lecture Notes in Computer Science, Springer, ISBN 978-3-642-25260-0 (2011)
21. Marcińczuk, M., Janicki, M.: Optimizing CRF-based Model for Proper Name Recognition in Polish Texts. In: Proceedings of the 13th International Conference on Intelligent Text Processing and Computational Linguistics (to appear). Volume 7181 of Lecture Notes in Computer Science (LNCS)., Springer-Verlag (2012)
22. Broda, B., Marcińczuk, M., Maziarz, M., Radziszewski, A. and Wardyński, A.: KPWr: Towards a Free Corpus of Polish. In Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), Istanbul, Turkey. Ed(s). Nicoletta Calzolari (Conference Chair) and Khalid Choukri and Thierry Declerck and Mehmet Uğur Doğan and Bente Maegaard and Joseph Mariani and Jan

Odijk and Stelios Piperidis. European Language Resources Association (ELRA), ISBN 978-2-9517408-7-7 (2012)
23. Marcińczuk, M., Kocoń, J. and Broda, B.: Inforex — a web-based tool for text corpus management and semantic annotation. In Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), Istanbul, Turkey. Ed(s). Nicoletta Calzolari (Conference Chair) and Khalid Choukri and Thierry Declerck and Mehmet Uğur Doğan and Bente Maegaard and Joseph Mariani and Jan Odijk and Stelios Piperidis. European Language Resources Association (ELRA), ISBN 978-2-9517408-7-7 (2012)
24. Kocoń, J. and Piasecki, M.: Heterogeneous Named Entity Similarity Function. In Proceedings of the 15th International Conference TSD 2012, Brno, Czech Republic. Lecture Notes in Artificial Intelligence, Springer (2012).
25. Piasecki, M. and Radziszewski, A.: Polish Morphological Guesser Based on a Statistical A Tergo Index. In Proceedings of the International Multiconference on Computer Science and Information Technology — 2nd International Symposium Advances in Artificial Intelligence and Applications (AAIA'07), pages 247–256 (2007)
26. Cohen, W. W., Ravikumar, P. and Fienberg, S. E.: A Comparison of String Distance Metrics for Name-Matching Tasks. In Proceedings of IJCAI-03 Workshop on Information Integration, pages 73–78, (2003)
27. Cohen, W. W., Ravikumar, P. and Fienberg, S. E.: A Comparison of String Metrics for Matching Names and Records. In Proceedings of the KDD-2003 Workshop on Data, pages 13–18. Washington, DC. ISBN 3-540-29754-5 (2003)
28. Newcomer, E.: Understanding Web Services: XML, WSDL, SOAP and UDDI. Pearson (2002)
29. Broda, B., Marcińczuk, M. and Piasecki, M.: Building a Node of the Accessible Language Technology Infrastructure. Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10), Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odjik, Stelios Piperidis, Mike Rosner, Daniel Tapias (eds.), May 19-21, Valletta, Malta, ISBN 2-9517408-6-7 (2010)

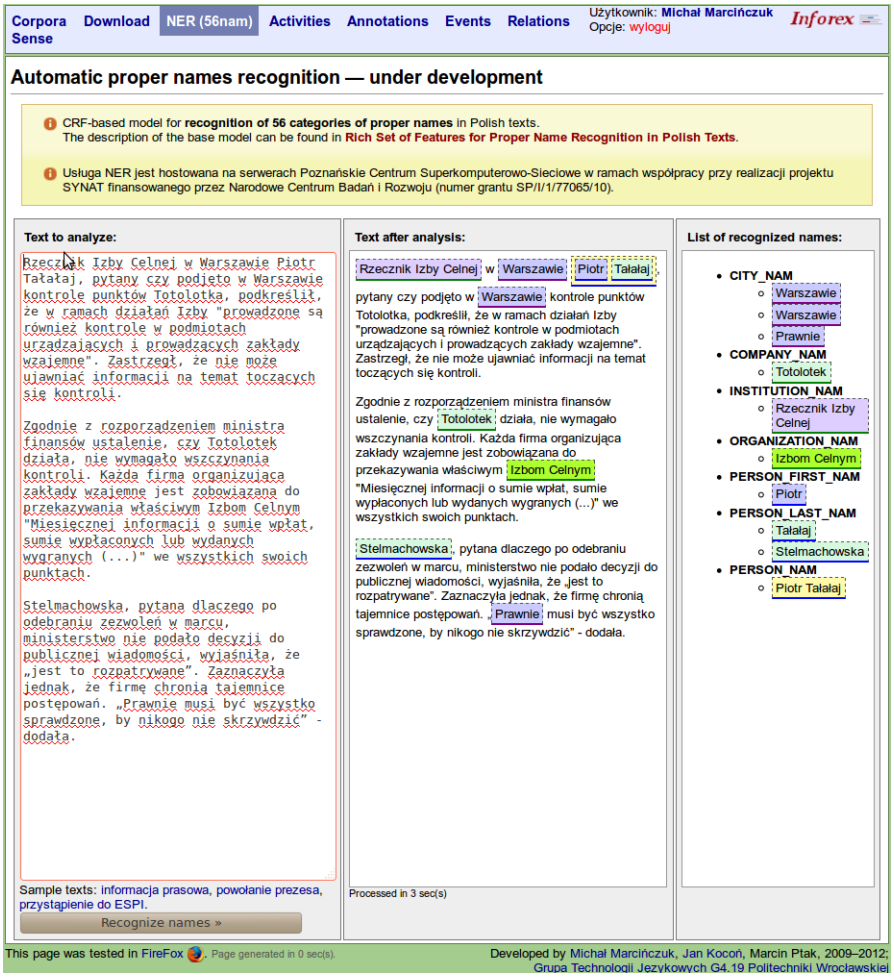

Fig. 7. Graphical interface using NER-WS