Multiservice documentation v. 0.2

Mateusz Kopeć

Institute of Computer Science, Polish Academy of Sciences

m.kopec@ipipan.waw.pl

August 31, 2015

Contents

Chapter 1

General Information

MULTISERVICE is a robust linguistic Web service for Polish, combining several mature offline linguistic tools in a common online platform. Packaged TEI P5-based annotation is used as representation and interchange format for the service. In contrast to most frequent approaches, the architecture supports asynchronous handling of requests to enable processing large amounts of text.

Chapters 1-3 of this manual describe information valuable for the Multiservice users, Chapters 4-6 for Multiservice developers.

1.1 Usage

You have two ways to use MULTISERVICE:

1. use web service maintained by ICS PAS,
2. set up your own installation of the web service (manual in Chapter 5).

Whichever you choose, you may communicate with that web service in one of three ways:

1. via online web client (MULTISERVICEDEMO, see Chapter 2),
2. via console web client (see Chapter 3),
3. via web service API, creating your own client (start with example clients described in Chapter 3).

There is an online web client connected to the web service maintained by ICS PAS, available at http://multiservice.nlp.ipipan.waw.pl. This is probably the most simple way to test MULTISERVICE and see it in action. However, if you have to process texts automatically, you should use one of console clients or create one of your own. For performance reasons it may also be necessary to set up your own web service back-end locally, instead of using the one maintained by ICS PAS.

1.2 Integrated Tools

Offline versions of all integrated tools have been used by the linguistic community in Poland for several years and they proved their suitability and efficiency for linguistic engineering tasks. They constitute the basic building blocks of many local processing chains, but have never been made available online in a consistent manner (in a common infrastructure and format). Until now all integrated tools are open source and all are actively maintained and developed. Table 1.1 shows currently available tools. More detailed description of them is given later in this chapter.

Most tools are associated with their configurations/models. These are available to download at http://zil.ipipan.waw.pl/Multiservice.

Service name	Required layers	Provided layers	Available options
Pantera	-	segmentation, morphosyntax	useGuesser
Concraft	-	segmentation, morphosyntax	-
WCRFT	-	segmentation, morphosyntax	-
WMBT	-	segmentation, morphosyntax	-
PoliTa	-	segmentation, morphosyntax	-
Spejd	segmentation, morphosyntax	words, groups	-
Sentipejd	segmentation, morphosyntax	words	-
OpenTextSummarizer	-	summary	ratio
ŚwietlickaSummarizer	morphosyntax	summary	minPercent, maxPercent
LakonSummarizer	morphosyntax	summary	ratio, length
DependencyParser	morphosyntax	dependency parse	-
Ruler	segmentation, morphosyntax, mentions	coreference	-
Bartek	segmentation, morphosyntax, mentions	coreference	-
Nerf	segmentation, morphosyntax	names	-
MentionDetector	segmentation, morphosyntax	mentions	-

Table 1.1: Available tools

Chain construction rules

Tools presented in Table 1.1 may be combined in a processing chain. Processing chain is a list of NLP tools, which are applied in sequence to input data, augmenting it with additional information, stored in separate 'layers'.

An example of such chain would be to apply part of speech tagger PANTERA to raw text data and then apply DEPENDENCYPARSER. As seen in the Table 1.1, PANTERA doesn't have any required layers, which means it may be applied to raw text. PANTERA adds segmentation and morphosyntax layers (storing segmentation of the text to tokens and their morphosyntactic interpretations, respectively) to the text. This allows DEPENDENCYPARSER to be run, as it requires the morphosyntactic layer to be present in its input (parser won't work without any tool providing this layer run beforehand). The output of DEPENDENCYPARSER is dependency parse layer, which is added to the original input text.

1.2.1 Part of speech taggers

Several part of speech taggers are available in MULTISERVICE. They all rely on common morphosyntactic analyzer for Polish, namely MORFEUSZ[1]. It provides positional tags starting with part of speech information followed by values of morphosyntactic categories corresponding to the given part of speech. Currently we use version 1.0 of MORFEUSZ, using linguistic data coming from POLIMORF[2] dictionary.

Pantera

PANTERA is a morphosyntactic rule-based Brill tagger of Polish. It uses an optimised version of Brill's algorithm adapted for specifics of inflectional languages. The tagging is performed in two steps, with a smaller set of morphosyntactic categories disambiguated in the first run (part of speech, case, person) and the remaining ones in the second run. Due to free word order nature of Polish the original set of rule templates as proposed by Brill has been extended to cover larger contexts. PANTERA is freely available at http://code.google.com/p/pantera-tagger/, you may also see http://zil.ipipan.waw.pl/PANTERA.

Current PANTERA model was trained on version 1.2 of 1-million word subcorpus of the National Corpus of Polish, available at http://clip.ipipan.waw.pl/NationalCorpusOfPolish. Training was done using the corpus reanalyzed[3] with Polimorf morphological dictionary, with `threshold=6`.

Concraft

CONCRAFT is a morphosyntactic tagger for the Polish language. The tool combines the following components into a pipeline:

- a morphosyntactic segmentation and analysis tool MACA[4],
- a morphosyntactic disambiguation library CONCRAFT[5].

As for now, the tagger doesn't provide any lemmatisation capabilities. As a result, it may output multiple interpretations (all related to the same morphosyntactic tag, but with different lemmas) for some known words, while for the out-of-vocabulary words it just outputs orthographic forms as lemmas. More information at CONCRAFT webpage at http://zil.ipipan.waw.pl/Concraft.

Current CONCRAFT model was trained on version 1.2 of 1-million word subcorpus of the National Corpus of Polish, available at http://clip.ipipan.waw.pl/NationalCorpusOfPolish. Training was done using the corpus reanalyzed[6] with Polimorf morphological dictionary.

WCRFT

WCRFT (Wrocław CRF Tagger) is a simple morpho-syntactic tagger for Polish producing state-of-the-art results. The tagger combines tiered tagging, conditional random fields (CRF) and features tailored for inflective

[1] http://sgjp.pl/morfeusz/
[2] http://zil.ipipan.waw.pl/PoliMorf
[3] according to the procedure described at http://nlp.pwr.wroc.pl/redmine/projects/wcrft/wiki/Training
[4] http://nlp.pwr.wroc.pl/redmine/projects/libpltagger/wiki
[5] https://github.com/kawu/concraft
[6] according to the procedure described at http://nlp.pwr.wroc.pl/redmine/projects/wcrft/wiki/Training

languages written in WCCL. The algorithm and code are inspired by Wrocław Memory-Based Tagger. WCRFT uses CRF++ API as the underlying CRF implementation. Tiered tagging is assumed. Grammatical class is disambiguated first, then subsequent attributes (as defined in a config file) are taken care of. Each attribute is treated with a separate CRF and may be supplied a different set of feature templates. For details of the underlying algorithms, as well as tagger evaluation, please refer to the tagger website at http://nlp.pwr.wroc.pl/redmine/projects/wcrft/wiki/WCRFT1.

Current WCRFT model was trained on version 1.2 of 1-million word subcorpus of the National Corpus of Polish, available at http://clip.ipipan.waw.pl/NationalCorpusOfPolish. Training was done using the corpus reanalyzed[7] with Polimorf morphological dictionary.

WMBT

WMBT (Wrocław Memory-Based Tagger) is a simple morpho-syntactic tagger for Polish producing state-of-the-art results. WMBT uses TiMBL API as the underlying Memory-Based Learning implementation. The features for classification are generated by using WCCL. WMBT uses a tiered tagging approach. Grammatical class is disambiguated first, then subsequent attributes (as defined in a config file) are taken care of. Each attribute may be supplied a different set of features. More information available at tagger webpage: http://nlp.pwr.wroc.pl/redmine/projects/wmbt/wiki.

Current WMBT model was trained on version 1.2 of 1-million word subcorpus of the National Corpus of Polish, available at http://clip.ipipan.waw.pl/NationalCorpusOfPolish. Training was done using the corpus reanalyzed[8] with Polimorf morphological dictionary.

PoliTa

PoliTa is a meta-tagger, which uses many individual POS taggers to combine their decisions and produce a potentially better output than any of the individual methods by themselves. Its current implementation uses Multiservice itself to run several other taggers and then combine their results into single version. More information available at http://zil.ipipan.waw.pl/PoliTa.

1.2.2 Parsers/shallow parsers

Spejd

Spejd is an engine for shallow parsing using cascade grammars, able to co-operate with TaKIPI for tokenization, segmentation, lemmatization and morphologic analysis. Parsing rules are defined using cascade regular grammars which match against orthographic forms or morphological interpretations of particular words. Spejd's specification language is used, which supports a variety of actions to perform on the matching fragments: accepting and rejecting morphological interpretations, agreement of entire tags or particular grammatical categories, grouping (syntactic and semantic head may be specified independently). Users may provide custom rules or may use one of the provided sample rule sets. More information available at http://zil.ipipan.waw.pl/Spejd/.

Current Spejd configuration uses grammar of Polish (version 1.0), developed by K. Głowińska within NKJP, available at http://clip.ipipan.waw.pl/LRT?action=AttachFile&do=view&target=gramatyka_Spejd_NKJP_1.0.zip.

Sentipejd

Sentipejd is a Spejd grammar detecting sentiment carrying expressions in text. It requires a slightly modified Spejd version, currently not publicly available. Sentipejd consits of:

- shallow sentiment rules as described in an article: *Shallow parsing in sentiment analysis of product reviews* by Aleksander Buczyński and Aleksander Wawer.
- sentiment dictionary available from Polish sentiment dictionary, available at http://zil.ipipan.waw.pl/SlownikWydzwieku.

Sentipage webpage is avaialable here: http://zil.ipipan.waw.pl/Sentipejd.

Current Sentipejd version uses grammar available at http://zil.ipipan.waw.pl/Multiservice.

[7] according to the procedure described at http://nlp.pwr.wroc.pl/redmine/projects/wcrft/wiki/Training
[8] according to the procedure described at http://nlp.pwr.wroc.pl/redmine/projects/wcrft/wiki/Training

DependencyParser

DependencyParser is trained on the Polish Dependency Bank (PDB, Pol. Składnica zależnościowa[9]) with the the publicly available parsing system – MaltParser[10]. MaltParser is a transition-based dependency parser that uses a deterministic parsing algorithm. The deterministic parsing algorithm builds a dependency structure of an input sentence based on transitions (shift-reduce actions) predicted by a classifier. The classifier learns to predict the next transition given training data and the parse history. Information about parser performance and Polish dependency relation types available at http://zil.ipipan.waw.pl/PolishDependencyParser.

Current model of DependencyParser was taken from its webpage.

1.2.3 Summarizers

OpenTextSummarizer

OpenTextSummarizer is an open source tool for summarising texts. The program reads a text and decides which sentences are important and which are not. It ships with Ubuntu, Fedora and other linux distros. OTS supports many (25+) languages which are configured in XML files. Several academic publications have benchmarked it and praised it. More information available at its webpage http://libots.sourceforge.net/.

Current configuration of OpenTextSummarizer is simply selecting `pl` dictionary. We use default version from Ubuntu repositories.

ŚwietlickaSummarizer

ŚwietlickaSummarizer is a tool for creating short text summaries. It utilises text extraction method, i.e. the output consists of sentences from the original text. The tool uses a number of machine learning algorithms, including neural networks, linear regression, Bayesian networks and decision trees. The output sentences are chosen based on different signals, such as the length of the sentence, its position in the text structure and properties of the words it contains. The system was trained specifically for newspaper articles in Polish. It is possible, however, to adjust it for other kinds of documents and languages. More information avaialble at: http://clip.ipipan.waw.pl/Summarizer.

Current configuration of ŚwietlickaSummarizer is an ensemble of best classifiers, it was claimed best by the tool's author in her master's thesis.

LakonSummarizer

Lakon is part of Adam Dudczak's master thesis, aimed at the evaluation of newspaper article summarization techniques based on sentence selection. More information about it at its webpage: http://www.cs.put.poznan.pl/dweiss/research/lakon/.

Current configuration of Lakon uses location-based features for sentence selections, as it proved to be most effective choice in evaluation conducted by the tool's author.

1.2.4 Coreference resolvers

Ruler

Ruler facilitates the automatic clustering of mentions into coreferent clusters using a simple deterministic rule-based method. You may find more about it at http://zil.ipipan.waw.pl/Ruler.

Its current configuration in Multiservice uses version 1.2 of Ruler.

Bartek

Bartek is a statistical coreference resolver, it uses a machine learnt method. Bartek comes with default models trained on the full Polish Coreference Corpus and contains compiled resources extracted from Polish Wikipedia and plWordnet. You may read more about it at http://zil.ipipan.waw.pl/Bartek.

Bartek in Multiservice is in version 1.1. It uses default models attached to that version.

[9] http://zil.ipipan.waw.pl/Sk%C5%82adnica
[10] http://maltparser.org/

1.2.5 Named entity recognizers

Nerf

Nerf is a statistical named entity recognition tool based on linear-chain conditional random fields. It recognises embedded structures of named entities consistent with the type hierarchy used in the National Corpus of Polish. You may read more about it at: http://zil.ipipan.waw.pl/Nerf.

In Multiservice we use the Haskell implementation, with the model trained on version 1.1 of 1-million word subcorpus of the National Corpus of Polish.

1.2.6 Other

MentionDetector

MentionDetector, a tool for performing mention detection for coreference resolution. It uses output of tagger, shallow parser and named entity recogniser to collect and filter mentions. Named entity recogniser and shallow parser are only optional prerequisites, but provide better final results. Additionally, MentionDetector detects zero subject mentions by itself.

You may read more about MentionDetector at http://zil.ipipan.waw.pl/MentionDetector.

Its current configuration in Multiservice uses version 1.2, with default zero subject detection model trained using the full Polish Coreference Corpus, version 0.92.

Chapter 2

Multiservice Demo

Multiservice Demo is an example web-based human interface for Multiservice. It may be installed on any web server to provide interface for any Multservice back-end. There is an instance of it available at http://multiservice.nlp.ipipan.waw.pl/, connected to back-end hosted by ICS PAS.

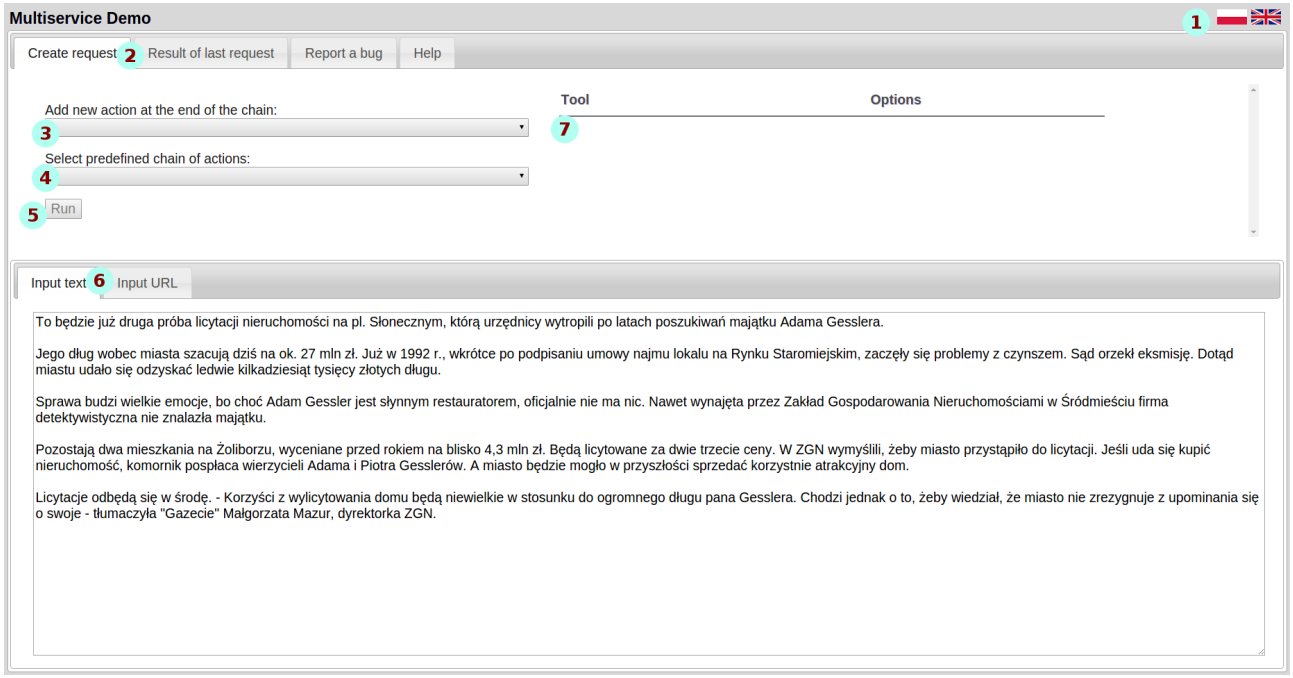

Figure 2.1: Multiservice Demo starting page

2.1 User manual

Home screen of the Multiservice Demo if presented in Figure 2.1. In fact it is the only one screen of this application, but it has several tabs which give different views. First of all, language of the interface may be changed by clicking chosen flag 1. Then, as this application is designed to demonstrate NLP tools, we need to state, what tools do want want to run and for which input, therefore we correctly are in the `Create request` tab 2.

The input may be raw text, which we are supposed to input in the bottom of the page, in the `Input text` lower tab 6, replacing the sample text starting with "To będzie...". Another option is to use the second lower tab named `Input URL` and paste the URL of the page we want to summarize.

The only thing left to do is to specify NLP tools to use. We may obey the rules presented in Section 1.2 during the chain construction. If you want to use one of the predefined chains, choose an option via `Select`

`predefined chain of actions` dropdown 4. If you do so, example chain will show up in the table in the right side of the dropdown 7. The tools in the chain are listed from the top, i.e. the first tool in the chain is in the top of the list. If a tool has optional options to specify, you may do so in the `Options` column. You may also remove the last element of the chain by clicking the `X` symbol next to it in the right part of the chain table. Via a dropdown menu named `Add new action at the end of the chain` 3. It obeys the chain construction rules, i.e. it shows only tools, for which requirements are already met in the existing chain.

If you want to construct a chain from scratch, simply start with `Add new action at the end of the chain` 3 dropdown and add each tool in correct sequence to the chain. If you don't know, what are prerequisites for a NLP tool, you may take a look at the `Help` tab 2, which contains information similar to Table 1.1.

Finally, we may press the `Run` button 5 and wait (usually up to few seconds) to get the processing results. In the meantime a popup will appear, showing current request status. If it succeeds, you will automatically be taken to the `Result of last request` tab, which is visible in the Figure 2.2.

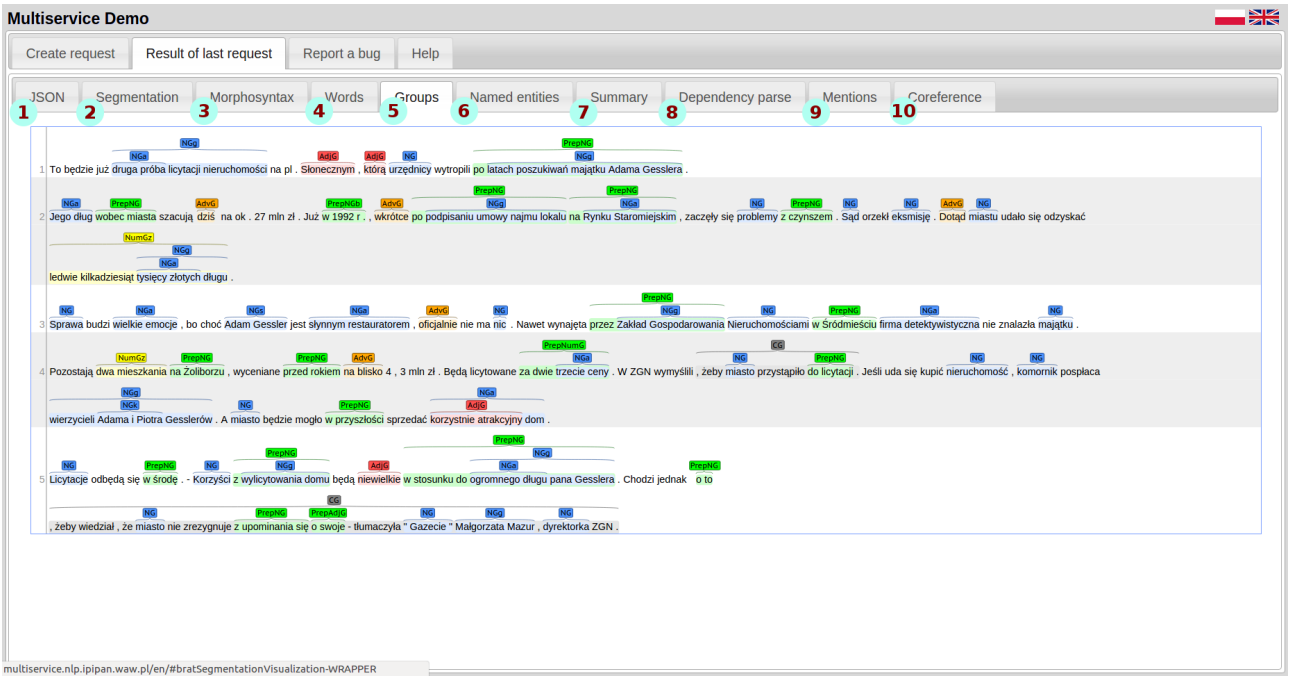

Figure 2.2: MULTISERVICE DEMO results page

This tab visualises outputs of selected NLP tools applied to given input. A set of tabs 1 - 9 allows to switch between various output layers. Number of layers present may vary depending on the chain of tools selected. One tab always present is the `JSON` tab, which shows raw output of the MULTISERVICE back-end, i.e. all output data in JSON format, which is more fancy visualized in the other tabs.

2 `Segmentation` layer presents segmentation of input text into sentences (`sent` annotation) and words (`seg` annotation).

3 `Morphosyntax` layer shows above each word its selected part of speech. Additionaly, when we hover over a word, we can see all morphosyntactic interpretations with the bold one selected by the tagger.

4 `Words` layer show so-called syntactic words, which connect words from segmentation into larger units. Broader part of speech tag is annotated above each unit.

5 `Groups` layer visualises syntactic groups, produced by a shallow parser. Group types are shown above groups.

6 `Named entities` layer shows named entities, with their types annotated above.

7 `Summary` layer presents final summary of the input as raw text.

8 `Dependency parse` layer presents dependency parse relations, together with relation types. Only one sentence is visible. To see a parse of selected sentence, it has to be clicked in the list below visualization.

9 `Mentions` layer shows mentions, which are input to coreference resolver.

10 `Coreference` layer presents coreference groups. To see, with which mention given mention is in a group, we have to hover over its annotation. The whole group will be highlighted.

Any bugs or problems with this demo are welcome and can be reported via `Report a bug` tab 2.

2.2 Implementation details

MULTISERVICE DEMO is a DJANGO[1] application. DJANGO is a high-level PYTHON framework. It may be run on a web server using FCGI or WSGI interfaces. User interface is created with JQUERY UI[2]. Visualization of linguistic data is done with help of BRAT[3] visualization tool, with some customizations. MULTISERVICE DEMO uses SQLLITE[4] database for internal storage.

[1] https://www.djangoproject.com/
[2] https://jqueryui.com/
[3] http://brat.nlplab.org/
[4] https://www.sqlite.org/

Chapter 3

Clients

There are currently three sample client applications for MULTISERVICE, available to download at http://git.nlp.ipipan.waw.pl/multiservice/clients. They should be used as a starting point for your own implementation of MULTISERVICE client. Two of them are written in Python and one in Java. They are presented in detail in the following sections.

3.1 Python Thrift client

THRIFT is a software framework for interchanging data between software written in different programming languages. It is used internally by MULTISERVICE to interchange data, therefore it is fully supported as either input or output format.

Python Thrift client is a single file: `thrift_client.py`. It consumes text to process from its standard input, constructs a Thrift object with that text and sends it to MULTISERVICE to be processed with a chain of tools specified as the program arguments. After processing, text comes back as a set of Thrift objects containing all the automatic annotation. They are converted to JSON format and printed to standard output.

If you want to have maximum capabilities, this client is the best way to start (or writing a Thrift client in another programming language), because you may sent partially annotated data to MULTISERVICE and receive output as objects in memory, which may be easily worked with. See the source code of the client, how the Thrift request is created. Thrift data structures are defined in the main repository at http://git.nlp.ipipan.waw.pl/multiservice/multiservice/tree/master/core/thrift.

3.1.1 Installation

This client requires Python 2.7 interpreter installed, as well as the following Python libraries:

- jsonpickle
- thrift
- multiservice-0.1

The first two are available in public Python repositories, the last one has its sources published in the MULTISERVICE back-end repository: http://git.nlp.ipipan.waw.pl/multiservice/multiservice/tree/master/core/PyUtils. For convinience, its compiled version is available also next to `thrift_client.py` file in the MULTISERVICE clients repository.

For example, in Ubuntu you may install all requirements using the following commands (we assume you downloaded `multiservice-0.1-py2.7.egg` file alongside `thrift_client.py`):

```
sudo apt-get install python-setuptools
sudo easy_install jsonpickle
sudo easy_install thrift
sudo easy_install multiservice-0.1-py2.7.egg
```

3.1.2 Usage

The client uses standard input, so we may for example use it like:

```
echo "Ala ma kota." | python thrift_client.py Concraft Nerf Spejd
```

or to process text from file named `input.txt`:

```
python thrift_client.py Concraft Nerf Spejd < input.txt
```

Space-separated arguments after `thrift_client.py` specify the target processing chain. Optionally, you may specify the MULTISERVICE back-end host and port, using `--host` and `--port` parameters. By default, requests are sent to ICS PAS back end.

3.2 Python SOAP client

Second type of Python client uses SOAP web service definition protocol, with the Web Services Description Language (WSDL) file present at http://ws.multiservice.nlp.ipipan.waw.pl/WebService-1.0-SNAPSHOT/ClarinWS?wsdl. There are libraries in many programming languages, which using such file generate the code of client applications.

Python SOAP client is a single file: `soap_client.py`. It consumes text to process from its standard input, constructs a SOAP request with that text and sends it to MULTISERVICE to be processed with a chain of tools specified as the program arguments. After processing, text comes back as XML response containing all the automatic annotation. The output comes in the Packaged TEI format (see Section 3.4). Unfortunately, current implementation only produces morphosyntax and segmentation layers, the other ones would not be returned.

Theoretically, using SOAP protocol it should be possible to sent partially annotated data in Packaged TEI format to MULTISERVICE and process it further, however given that only layers possible to output are morphosyntax and segmentation, it does not have much sense.

3.2.1 Installation

This client requires Python 2.7 interpreter installed, as well as the following Python libraries:

- suds[1]

It is available in public Python repositories. For example, in Ubuntu you may install all requirements using the following commands:

```
sudo apt-get install python-setuptools
sudo easy_install suds
```

3.2.2 Usage

The client uses standard input, so we may for example use it like:

```
echo "Ala ma kota." | python soap_client.py Concraft Nerf Spejd
```

or to process text from file named `input.txt`:

```
python soap_client.py Concraft Nerf Spejd < input.txt
```

Space-separated arguments after `soap_client.py` specify the target processing chain. Optionally, you may specify the MULTISERVICE back-end host and port, using `--host` and `--port` parameters. By default, requests are sent to ICS PAS back end.

[1] https://fedorahosted.org/suds/

3.3 Java SOAP client

Java SOAP client uses SOAP web service definition protocol, with the Web Services Description Language (WSDL) file present at http://ws.multiservice.nlp.ipipan.waw.pl/WebService-1.0-SNAPSHOT/ClarinWS?wsdl. There are libraries in many programming languages, which using such file generate the code of client applications. Such procedure was used to generate code of this client application, using Apache Axis2 library.

Similar to Python SOAP client, Java SOAP client consumes text to process from its standard input, constructs a SOAP request with that text and sends it to Multiservice to be processed with a chain of tools specified as the program arguments. After processing, text comes back as XML response containing all the automatic annotation. The output comes in the Packaged TEI format (see Section 3.4). Unfortunately, current implementation only produces morphosyntax and segmentation layers, the other ones would not be returned.

Theoretically, using SOAP protocol it should be possible to sent partially annotated data in Packaged TEI format to Multiservice and process it further, however given that only layers possible to output are morphosyntax and segmentation, it does not have much sense.

3.3.1 Installation

This client requires Maven and Java Development Kit in version at least 1.7 installed (for compilation, for usage of compiled `.jar` file, Java Runtime Environment is sufficient). To compile the `.jar`, execute the command:

```
mvn clean install
```

from the `java` directory, where `pom.xml` file is present. The `.jar` is in the `target` folder and is named `client-1.0-SNAPSHOT.one-jar.jar`.

3.3.2 Usage

The client uses standard input, so we may for example use it like:

```
echo "Ala ma kota." | java -jar client-1.0-SNAPSHOT.one-jar.jar Concraft Nerf Spejd
```

or to process text from file named `input.txt`:

```
java -jar client-1.0-SNAPSHOT.one-jar.jar Concraft Nerf Spejd < input.txt
```

Space-separated arguments after `client-1.0-SNAPSHOT.one-jar.jar` specify the target processing chain. Requests are sent to ICS PAS back end.

3.4 Packaged TEI format

Packaged TEI P5-based Linguistic Representation is mostly compatible with NKJP representation described here: http://nlp.ipipan.waw.pl/TEI4NKJP/. The main difference is that all the files (headers, `text.xml`, `segmentation.xml`, `ann_morphosyntax.xml` etc.) are ‘packed‘ into one file. That is instead of multiple files following this pattern:

```
<teiCorpus>
  <teiHeader>
    ... corpus ‘global‘ header ...
  </teiHeader>
  <TEI>
    <teiHeader>
      ... local header ...
    </teiHeader>
    <text>
      ... some annotation layer data (ex. segmentation)...
    </text>
  </TEI>
</teiCorpus>
```

you have one file like that:

```
<teiCorpus>
  <teiHeader>
    ... Multiservice ‘global‘ header (could be the same as in NKJP_header.xml) ...
  </teiHeader>
  <TEI>
    <teiHeader>
      ... header specific for single annotation layer (could be the same as in header.xml) ...
    </teiHeader>
    <text>
      ... some annotation layer data (ex. segmentation - the same as ann_segmentation.xml) ...
    </text>
  </TEI>
  <TEI>
    ... some other annotation layer ...
  </TEI>
</teiCorpus>
```

They easiest way to parse Packaged TEI P5-based Linguistic Representation is to use TeiAPI Java library. More information about TeiAPI (including Maven artifacts) is available at http://zil.ipipan.waw.pl/TeiAPI.

Chapter 4

Architecture

To be able to process large amounts of text, requests sent to the Web service are handled in asynchronous manner. Invoking one of the available methods results in returning the request token (identifier) which can be used to check the request status and retrieve the result when processing completes. This design was directly inspired by 'TaKIPI Web Service': http://nlp.pwr.wroc.pl/clarin/ws/takipi/ prepared by ICS PAS and WrocUT.

This chapter is under development.

4.1 Code organisation

MULTISERVICE code is distributed in two git repositories:

- Main (back-end, subservices, web service, demo web client): http://git.nlp.ipipan.waw.pl/multiservice/multiservice
- Command line clients: http://git.nlp.ipipan.waw.pl/multiservice/clients

4.1.1 Main repository contents

- `config` – contains sample MULTISERVICE configuration.
  - `config.xml` – file describing locations of available subservices.
- `core` – main components of MULTISERVICE.
  - `RequestManager` - najgłówniejszy serwis przyjmujący wszystkie zlecenia. Odbiera zlecenia ze świata (np. od Dema) i zleca je podserwisom. Jest napisany w Javie.
  - `thrift` - źródła thriftowe (do generowania kodu pośredniczącego). Zawiera thriftowe definicje obiektów do wymiany danych pomiędzy podserwisami i RequestManagerem. Więcej patrz tutaj: http://diwakergupta.gi missing-guide/
  - `MultiserviceDemo` - interfejs przeglądarkowy napisany w Django, wywołuje RequestManagera i ładnie pokazuje wyniki.
  - `WebService` - webserwis napisany w technologii JAX-RS, umożliwia odpalanie serwisu jako webserwisu typu SOAP.
  - `CppUtils` - thriftowy interfejs skonwertowany C++ (przez thrift/generate-all.sh) + różne wspomagacze (np. AnnotatingServer, który ułatwia tworzenie własnego serwera do anotacji)
  - `JavaUtils` - jw. ale w Javie
  - `PyUtils` - jw. ale w Pythonie (2.7)
- `doc` – documentation (this file).
- `scripts` – scripts for installing and running MULTISERVICE.

- `subservices` – annotating subservices (implementing interfaces from `core/thrift/subservices.thrift`).
  - `cpp` – subservices written in C++.
    - `PanteraService` – service implementing the Pantera tagger.
    - `SpejdService` – service implementing the Spejd shallow parser.
  - `haskell` – subservices written in Haskell.
    - `concraft-multiservice-master` – service implementing the Concraft tagger.
    - `nerf-multiservice-master` – service implementing the Nerf named entity recogniser.
  - `java` – subservices written in Java.
    - `BartekService` – service implementing the Bartek coreference resolver.
    - `MentionDetectorService` – service implementing the MentionDetector.
    - `ServiceWrapper` – Java wrapper for C++ subservices.
    - `ConversionService` – service offering conversions between plain text and TEI formats.
    - `OpenTextSummarizerService` – service implementing the OpenTextSummarizer.
    - `SwietlickaSummarizerService` – service implementing the SwietlickaSummarizer.
    - `DependencyParserService` – service implementing the DependencyParser.
    - `RulerService` – service implementing the Ruler coreference resolver.
    - `UrlParser` – service able to parse webpage at given URL and return it as a Thrift text.
    - `LakonSummarizerService` – service implementing the Lakon summarizer.
    - `SampleService` – sample service, not used in production.
  - `python` – subservices written in Python.
    - `PolitaService` – service implementing the PoliTa tagger.
    - `TEIWriterService` – deprecated service implementing conversion of annotated data to TEI format (replaced by `java/ConversionService`).
    - `WCRFTService` – service implementing the WCRFT tagger.
    - `WMBTService` – service implementing the WMBT tagger.
- `third_party` – sources and installation scripts for third party requirements.

4.1.2 Command line clients repository contents

- `java` – Java SOAP client (see Section 3.3).
- `python` – Python clients.
  - `multiservice-0.1-py2.7.egg` – compiled library for python Thrift client.
  - `soap_client.py` – python SOAP client (see Section 3.2).
  - `thrift_client.py` – python Thrift client (see Section 3.1).

Chapter 5

Installation and administration

This chapter present how the MULTISERVICE back-end with web service and web client can be installed and administered in Ubuntu Server 14.04 system. Superuser privileges are needed, however installation may be manually tweaked to run without it. Users of other systems have to go through scripts line by line and implement their own equivalent commands.

5.1 Installation

Installation is very straightforward: we first clone the git repository and run `installAll.sh` script.

```
sudo apt-get -y install git
git clone http://git.nlp.ipipan.waw.pl/multiservice/multiservice.git
cd multiservice/scripts
./installAll.sh
```

This may take some time, probably over an hour. First part of the script downloads subservice models and configuration settings from http://zil.ipipan.waw.pl/Multiservice webpage. Then, third party components are installed:

- MORFEUSZ
- PANTERA
- SPEJD
- MACA
- CONCRAFT
- WCRFT
- WMBT
- THRIFT

After that the script installs all subservices, the back-end managment system (`RequestManager`), SOAP webservice and MULTSERVICEDEMO (see Chapter 2). Regarding subservices, it performs:

- `./configure && make && sudo make install` command for all C++ subservices,
- `mvn package install -DskipTests=true` command for all Java subservices,
- `sudo python setup.py install` command for all Python subservices,
- `sudo cabal install --global` command for all Haskell subservices.

5.1.1 MultiserviceDemo configuration

We should adjust MultiserviceDemo settings (in file `core/MultiserwisDemo/MultiserwisDemo/settings.py` to reflect our server address, by changing following parameters:

- `ROOT_URL` – url at which demo will be hosted; for ICS PAS instance, it is `http://multiservice.nlp.ipipan.waw.pl/`.
- `STATIC_URL` – url at which static files for demo will be hosted; for ICS PAS instance, it is `http://multiservice.nlp.ipipan.waw.pl/static/`.
- `SECRET_KEY` – we should add a unique value.

5.2 Administration

5.2.1 Logging

Log files are placed in `log` folder. Each subservice or component has its own pair of log files.

5.2.2 Starting

To start everything, we should call:

```
cd scripts && ./runAll.sh
```

We should add this also to CRON, if we want the service to start automatically upon reboot.

5.2.3 Restarting

We can restart everything by running the same command as we used for starting: wystarczy:

```
cd scripts && ./runAll.sh
```

To restart a single subservice or component, we need to identify the script that starts it, called by the `runAll.sh` script. Then we can run this script to restart given service. Start script first kill any daemons occupying their ports. Important: environment `MULTISERVICE_LOG` and `PATH` should be initialised as in `runAll.sh`.

Chapter 6

Development of new subservices

6.1 General rules

Subservices should not contain the code generated by Thrift from the common data structures definition file. Because of that, they should import libraries with them during their build. Sections for all currently used subservice programming languages present that in more detail.

6.1.1 C++

C++ subservices should be placed in the `subservices/cpp` folder. They should be managed by `autotools`. Installation script executes for each one of them:

```
./configure && make && make install
```

PanteraService and ServiceWrapper

Because PANTERA tagger is not a very reliable tool (it often results in segmentation faults), the `pantera_service` C++ program is wrapped into a Java application called `ServiceWrapper`. `ServiceWrapper` is at the same time a standard subservice, as well a client, which calls another subservice (both implement `AnnotatingService` interface from `core/thrift/subservices.thrift`). If the execution of the called subservice ends with an error, `ServiceWrapper` restarts that subservice and try to use it once more.

6.1.2 Java

Java subservices should be placed in the `subservices/java` folder. They should be managed by `maven`. Installation script executes for each one of them:

```
mvn package install -DskipTests=true
```

6.1.3 Python

Python subservices should be placed in the `subservices/python` folder. They should be managed by `setuptools`. Installation script executes for each one of them:

```
python setup.py install
```

6.1.4 Haskell

Haskell subservices should be placed in the `subservices/haskell` folder. Installation script executes for each one of them:

```
cabal install --global
```

6.2 Adding new service

To add a new subservice we need to implement a Thrift server with the `AnnotatingService` interface from `core/thrift/subservices.thrift` file. Examples of simple thrift services and clients are available at http://thrift.apache.org/.

This chapter is under development.