An Adaptive Framework for Named Entity Combination

Bogdan Sacaleanu[1], Günter Neumann[2]
[1]IMC AG, [2]DFKI GmbH
[1]New Business Department, [2]Language Technology Department
Saarbrücken, Germany
E-mail: Bogdan.Sacaleanu@im-c.de, Günter.Neumann@dfki.de

Abstract

We have developed a new OSGi-based platform for Named Entity Recognition (NER) which uses a voting strategy to combine the results produced by several existing NER systems (currently OpenNLP, LingPipe and Stanford). The different NER systems have been systematically decomposed and modularized into the same pipeline of preprocessing components in order to support a flexible selection and ordering of the NER processing flow. This high modular and component-based design supports the possibility to setup different constellations of chained processing steps including alternative voting strategies for combining the results of parallel running components.

Keywords: Named Entity recognition, voting, meta-model

1. Introduction

We describe a flexible and dynamic architecture for developing platform independent Named Entities (NE) processors based on existing open source tools. It provides a lightweight framework, called NER-Hub, for integrating and combining reusable text processing components like sentence detectors, tokenizers and NE extractors. Compared to similar platforms like GATE (Cunningham, 2011), the system provides an extended functionality of pulling together the result of comparable components for producing better precision and higher recall than the best of the individual components. This distinctive feature uses a voting mechanism to combine the results of several NE extractors and it can be easily extended to cover other combination approaches.

The NER-Hub framework described in this paper covers both processing tools mandatory for NE recognition and a voting mechanism for combining results at different stages in the workflow. Along with the framework come implementations of every processing step, namely paragraph selection, sentence detection, tokenization and NE extraction wrapped around components from open-source projects like LingPipe [1], OpenNLP [2] and Stanford NLP [3]. The voting framework delivers a simple implementation of statistics-based methods for combining the results of different NE extraction systems and it can be easily extended to the other stages of processing.

2. System Architecture

The NER-Hub framework is based on a blackboard architectural model (Figure 1), similar to systems such as Hearsay-II (Erman, 1980), where the communication between modules happens through a shared knowledge structure that is iteratively updated by a group of specialist

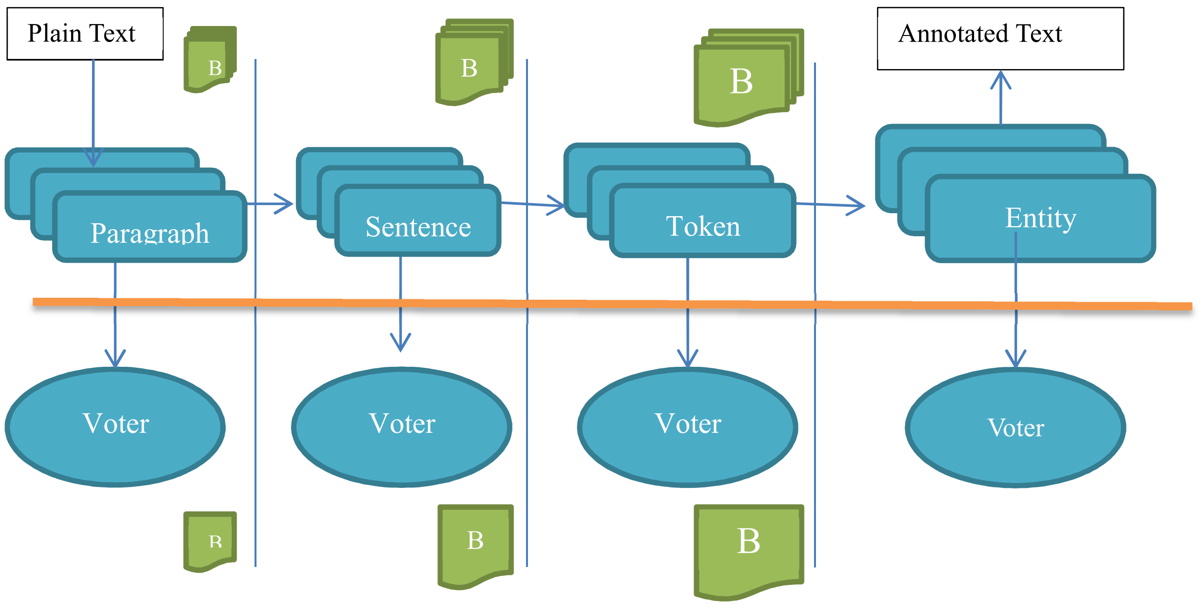
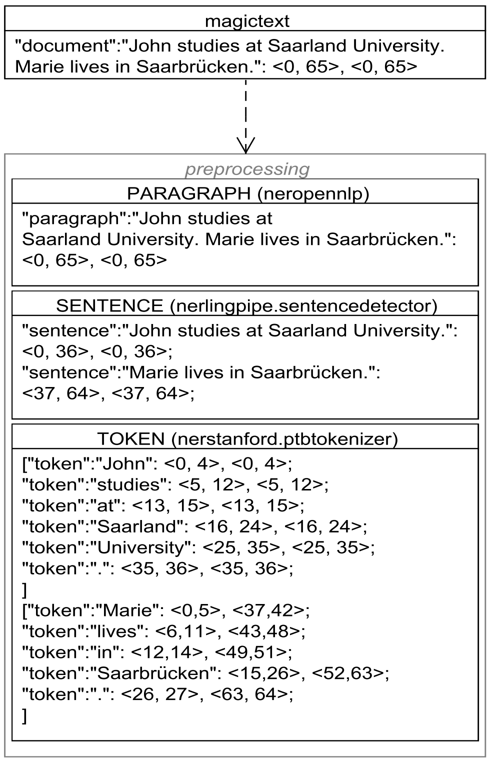


Figure 1. System Architecture

knowledge sources. As such, the framework consists of two major components:

- the specialist modules, also called knowledge sources, which provide specific linguistics expertise (i.e., paragraph and sentence detectors, tokenizers, NE recognizers) and voting functionality;
- the blackboard, representing a shared structure of partial solutions and contributed information.

We distinguish between two types of specialist modules: those enriching the information structure of the blackboard with linguistic knowledge and those reducing partial solutions to a final solution. These last modules, also called voters, combine the output of similar processes (several NE recognizers, for example) into a single one with better evaluation figures such as precision and recall.


magictext
preprocessing
PARAGRAPH (neropennlp)
SENTENCE (nerlingpipe.sentencedetector)
TOKEN (nerstanford.ptbtokenizer)

```
"document":"John studies at Saarland University.
Marie lives in Saarbrücken.": <0, 65>, <0, 65>
```

```
"paragraph":"John studies at
Saarland University. Marie lives in Saarbrücken.":
<0, 65>, <0, 65>
```

```
"sentence":"John studies at Saarland University.":
<0, 36>, <0, 36>;
"sentence":"Marie lives in Saarbrücken.":
<37, 64>, <37, 64>;
```

```
["token":"John": <0, 4>, <0, 4>;
"token":"studies": <5, 12>, <5, 12>;
"token":"at": <13, 15>, <13, 15>;
"token":"Saarland": <16, 24>, <16, 24>;
"token":"University": <25, 35>, <25, 35>;
"token":".": <35, 36>, <35, 36>;
]
["token":"Marie": <0,5>, <37,42>;
"token":"lives": <6,11>, <43,48>;
"token":"in": <12,14>, <49,51>;
"token":"Saarbrücken": <15,26>, <52,63>;
"token":".": <26, 27>, <63, 64>;
]
```

Figure 2. Blackboard data representation.

The shared data structure (blackboard) is a pointer-based representation of linguistic information provided by individual specialist modules for the input text (Figure 2). Every knowledge source enriches the blackboard with character-based positional annotations relative to the whole document and to the previous processing stage where necessary (tokenization). This general representation of annotations makes it straightforward to extend the processing line with further modules like co-reference resolution and the corresponding voters, and allows for using the framework in the context of other languages as well.

The NER-Hub framework and its reference implementation for the defined modules are realized as Java OSGi bundles leveraging the benefits of a very mature component system. As the most beneficial aspects of using such a system in the context of text engineering are its high adaptability and dynamicity. Regarding adaptability, OSGi is designed from ground up to allow mixing and matching of components. For our framework mixing available voters it benefits the quick development of complex voting methods and evaluating the contribution of each individual based on the whole result. Matching of components is relevant when adding new implementation of a given specialist module (i.e., NE recognizer) that are automatically found and bound in the voting process. The OSGi model is a dynamic model as it allows installing, starting, stopping, updating and uninstalling components without bringing down the whole system. For our framework it means that new system configurations can be run simultaneously without being required to have several instances of the same whole environment run in different memory spaces (JVM) or even different machines, as next releases are announced to support a distributed model as well.

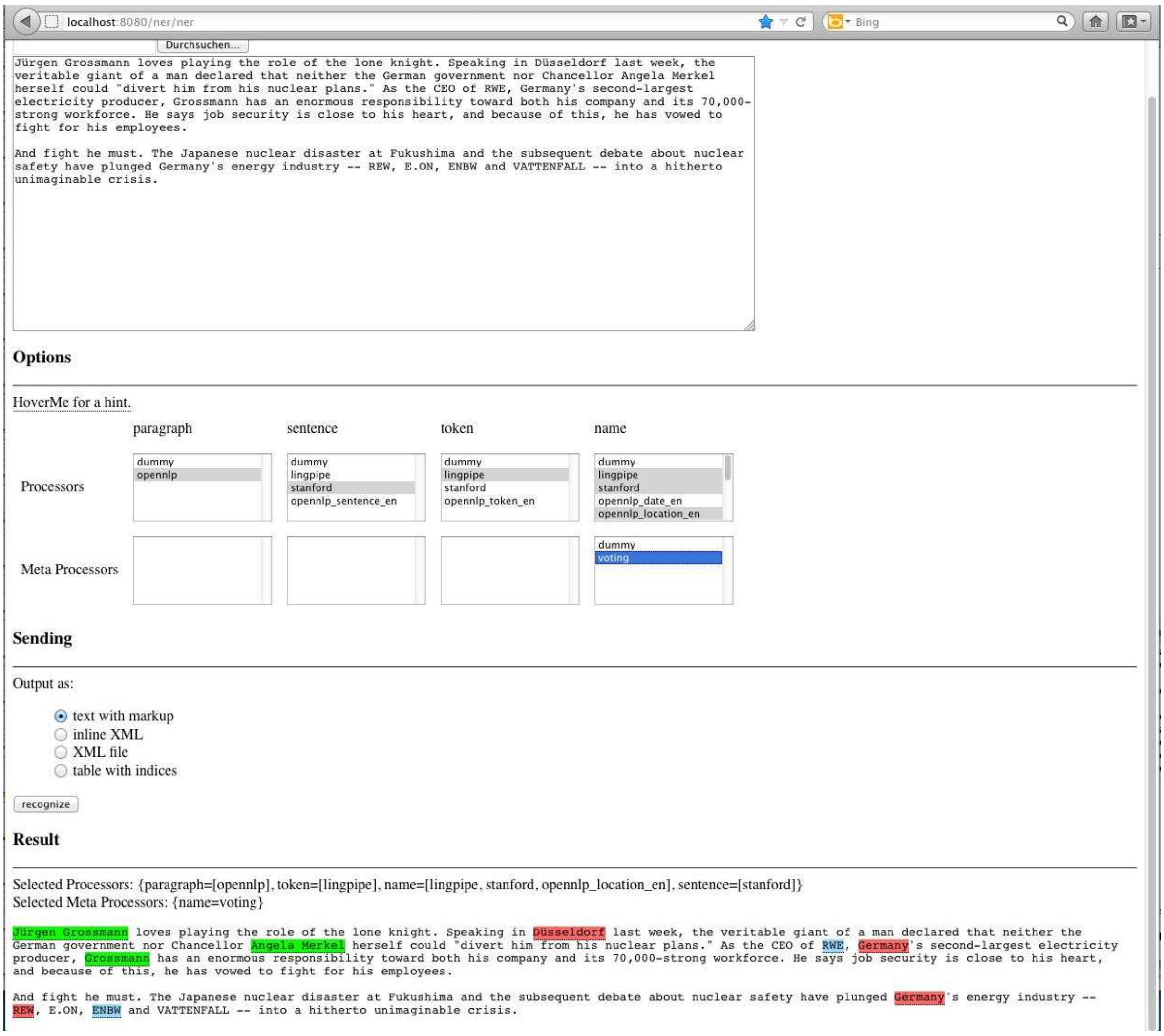
3. User interface

The user interface (Figure 3) is divided into three sections: for providing the input text from which NEs are to be extracted, selecting the different pre-processing and NER tasks, and the format in which the output will be presented. The interface offers two possibilities for specifying the input: it can be provided in the form of a plain text file, or directly entered in a text field.

There are four different options for displaying or returning the result. The first two of them (see below) involve displaying the original text with some mark-up denoting the NEs which were recognized on the web page itself. Therefore they are more suitable for smaller texts.

1. text with mark-up - NEs are highlighted in different colours with the help of XSLT transformations. The actual NE label can be seen when the mouse cursor is over the highlighted span (Figure 3).
2. inline XML - NEs are marked with the inline XML tag <label>, which contain the attribute “value” to denote the type of NE this label represents, for instance: <label value=“person”> John Smith</label>.
3. XML file - an XML file offered as download, containing the original text represented as described in 2.
4. table with indices - a table containing only the NEs found in the text with their corresponding begin and end indices and the name of the named entity processor which produced this output. The indices are relative to the sentence in which the named entity occurs. The start index is inclusive, the end index is exclusive.

The middle panel of the GUI provides options for choosing processors for the different tasks: paragraph detection,



```
Jürgen Grossmann loves playing the role of the lone knight. Speaking in Düsseldorf last week, the
veritable giant of a man declared that neither the German government nor Chancellor Angela Merkel
herself could "divert him from his nuclear plans." As the CEO of RWE, Germany's second-largest
electricity producer, Grossmann has an enormous responsibility toward both his company and its 70,000-
strong workforce. He says job security is close to his heart, and because of this, he has vowed to
fight for his employees.

And fight he must. The Japanese nuclear disaster at Fukushima and the subsequent debate about nuclear
safety have plunged Germany's energy industry -- REW, E.ON, ENBW and VATTENFALL -- into a hitherto
unimaginable crisis.
```

Options

HoverMe for a hint.

	paragraph	sentence	token	name
Processors	dummy opennlp	dummy lingpipe stanford opennlp_sentence_en	dummy lingpipe stanford opennlp_token_en	dummy lingpipe stanford opennlp_date_en opennlp_location_en
Meta Processors				dummy voting

Sending

Output as:

- [✓ text with markup]
- inline XML
- XML file
- table with indices

recognize

Result

Selected Processors: {paragraph=[opennlp], token=[lingpipe], name=[lingpipe, stanford, opennlp_location_en], sentence=[stanford]}
Selected Meta Processors: {name=voting}

```
Jürgen Grossmann loves playing the role of the lone knight. Speaking in Düsseldorf last week, the veritable giant of a man declared that neither the
German government nor Chancellor Angela Merkel herself could "divert him from his nuclear plans." As the CEO of RWE, Germany's second-largest electricity
producer, Grossmann has an enormous responsibility toward both his company and its 70,000-strong workforce. He says job security is close to his heart,
and because of this, he has vowed to fight for his employees.

And fight he must. The Japanese nuclear disaster at Fukushima and the subsequent debate about nuclear safety have plunged Germany's energy industry --
REW, E.ON, ENBW and VATTENFALL -- into a hitherto unimaginable crisis.
```

Figure 3. User Interface.

tokenization, sentence detection, named entity recognition, and voting (meta-processor). Typically only one processor can be selected per task. In the case of named entity recognition, one or more options can be selected, but when only one NER is chosen, no voting is performed.

4. Combining Results by Voting

System combination as a way of improving performance over the individual systems has been reported for different tasks: (Van Halteren et al., 1998) describe successful experiments for part-of-speech tagging, (Kim Sang, 2000) presents improved performance for noun phrase recognition, (Florian et al., 2002) demonstrate significantly lower error rate for word sense disambiguation. Improved results for NE recognition have been reported by (Florian et al., 2003) and (Sigletos et al., 2005) announced better performance for information extraction. A common factor of these system combination approaches is the use of a voting framework that allows different instantiations like count-based voting, confidence-based voting and probability-based voting. Of these voting methods the probability-based combination of system is significantly delivering improved performance, while simpler statistics-based methods only improve either precision or recall.

The current version of our NER-Hub framework includes reference implementations for three basic voting methods, namely length-based, count-based and confidence-based voting. The first method returns the candidate(s) with the longest length, whereby the empty candidates are considered to have a length of zero. The count-based method groups candidates by label, start index, and end index and computes a tally. The group with the highest counts are returned as winners (Figure 4).

The confidence-based method allows setting priority values to individual processing units, such that the results of a component are preferred over other results in case of an non-deterministic case (prefer *comp1* over all others if output is *comp1:x*, *comp2:y* and *comp3:z*). Due to the high adaptability feature of OSGi-based systems, these voting methods can be called individually or mixed into a new linear workflow.

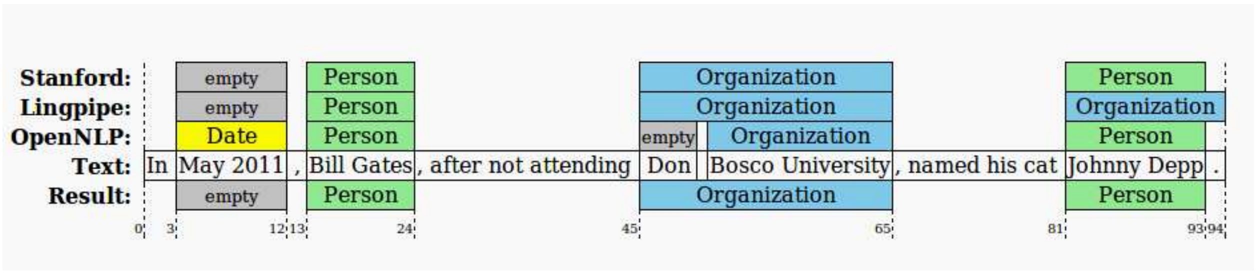


Figure 4. Example of Count-based Voting.

5. Status and Running Example

Currently the NER-Hub framework is fully implemented and offering reference implementation for all modules based on components from open-source projects like LingPipe, Open NLP and Stanford NLP.

We demonstrate the main workflow of the system with a practical example, namely what happens with a sample input text in the different processing stages.

Example settings:

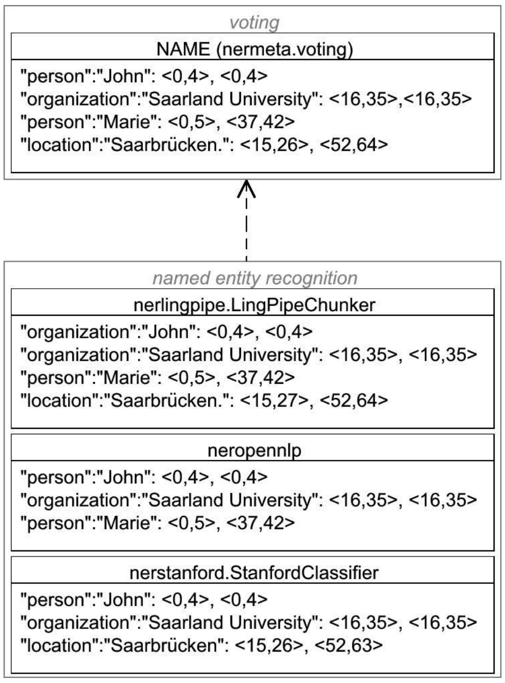
- *Input text*: John studies at Saarland University. Marie lives in Saarbrücken.
- *Processors*:
  - o paragraph = opennlp
  - o sentence = lingpipe
  - o token = Stanford
  - o entity = lingpipe, stanford, opennlp
- *Meta processors*: voting

The input gets passed from the user interface to the Controller and wrapped in a NerResultObject at the level "document" with begin and end indices spanning the whole input. The Controller then calls the different available processors (paragraph, sentence, token, and finally name) with it. The wrapped input is first passed to the three components performing text pre-processing tasks. Paragraph detection, sentence splitting and tokenization processors each generate an intermediate NerResultObject representation encoding the additional information they provide. All of these results are combined in a hierarchical structure, which gets passed to every next processor.

For this particular example, the intermediate results generated by the pre-processors are displayed in Figure 2. The first processor which is called, the OpenNLP paragraph detector, returns only one paragraph, due to the small text example. LingPipe sentence detector recognizes the two sentences, whose NerResultObject parent is set to the paragraph. Finally, the tokens found by the Stanford tokenizer are assigned to their corresponding parent sentences in the hierarchy.

Each NerResultObject in Figure 2 is additionally described by two sets of indices indicating its position in the text. The first set denotes the position with respect to its parent in the hierarchy, and the second – to the original input. This is visible at the token level, where due to the fact that the first six tokens belong to the first sentence, and the rest to the second, the two sets of indices describing the tokens in the second sentence differ.

After calling all of the pre-processing components, the Controller calls the meta processor for named entity recognition. This processor is responsible for calling the actual named entity recognizers, collecting the results from them and voting on the final result. The input NerResultObject which it receives from the Controller already contains all the information provided by the paragraph, sentence and token processors. The different levels of the hierarchy can be used by each NER individually, based on the kind of input it accepts.



```
"person":"John": <0,4>, <0,4>
"organization":"Saarland University": <16,35>,<16,35>
"person":"Marie": <0,5>, <37,42>
"location":"Saarbrücken.": <15,26>, <52,64>
```

```
"organization":"John": <0,4>, <0,4>
"organization":"Saarland University": <16,35>, <16,35>
"person":"Marie": <0,5>, <37,42>
"location":"Saarbrücken.": <15,27>, <52,64>
```

```
"person":"John": <0,4>, <0,4>
"organization":"Saarland University": <16,35>, <16,35>
"person":"Marie": <0,5>, <37,42>
```

```
"person":"John": <0,4>, <0,4>
"organization":"Saarland University": <16,35>, <16,35>
"location":"Saarbrücken": <15,26>, <52,63>
```

Figure 5. Result of Named Entity Voting.

The result produced by the three NERs and the final result after voting can be seen in Figure 4. The two example sentences illustrate the advantage of using the voting strategy, since in this case it manages to avoid all of their individual errors and produces the desired output.

6. Evaluation

Also part of the NER-Hub framework is an automated testing environment based on CoNLL-2003 English news test data for NE extraction that has been used for evaluating various combinations of pre-processing for NE extraction (Table 1) and the voting methods previously presented (Table 2).

Config	NE	Sent	Token	F-Measure
Standard	OpenNLP	OpenNLP	OpenNLP	56.24
Standard	Stanford	Stanford	Stanford	87.12
Standard	LingPipe	LingPipe	LingPipe	51.72
Best	OpenNLP	OpenNLP	LingPipe	57.42
Best	Stanford	OpenNLP	OpenNLP	89.48

Table 1. Standard vs. Best Module Combination

Evaluation of using different preprocessing modules (sentence detector, tokenizer) for NE extraction has shown that combining the components from different sources can boost the performance of the whole system.

Configuration	Precision	Recall	F-Measure
Best Individual	88.10	86.15	87.12
Best Combined	**91.06**	75.89	82.79

Table 2. Results of combining all NE-Recognizers

Using OpenNLP for pre-processing and combining the results of all three NE recognizers by way of voting (count-, confidence- and length-based in linear order) improved the precision over the best individual module (Stanford), but dropped in the recall score.

Compared to the best outcomes registered in the CoNLL-2003 shared task with a precision of 88.99, a recall of 88.54 and an F-measure of 88.76 we conclude that both a combination of pre-processing components from different NLP tools and a combination of the recognition results can improve the overall performance of an integrated system.

7. Framework Extensions

We have already begun to extend NER-Hub framework with a further component layer for the resolution of NE co-reference. Such component is able to resolve the referents of pronouns to corresponding Named Entities in texts like “Peter loves Mary. He is very lucky.” We have chosen the corresponding coref-component of the OpenNLP toolkit, because we had already hands on experimentation with the integration of major NE components of OpenNLP into the framework. Later, we will also integrate additional alternative coref-components via the voting mechanism.

The major challenge with the integration of a coref-component was that the sentence-based streaming approached followed so far in NER-Hub cannot easily expanded to the co-reference layer in the same way, as the coref-component has to process actually any sentence which contain a pronoun and has to check a window of previously processed sentences. The strategy we are currently following is a kind of compromise. Since the co-reference algorithm is the last in the processing order it is called once for every recognized named entity. So to be able to process a meaningful chunk of the text, the coref-component stores named entities, tokens and sentences until the end of a paragraph is reached. Only then the actual co-reference resolution takes place. With this paragraph oriented structure the coref-component can adapt the original streaming in a way that still yields meaningful results without completely abandoning the streaming-based approach. Note that the coref-component is defined as a OSGi bundle in the same way as the other NE components, and as such, can be selected and exchanged (also from the user interface) in the same way.

8. Summary

We have described a framework (NER-Hub) for the design and the implementation of a meta-system for NE recognition as a combination of several available state-of-the-art systems for this task - OpenNLP, Stanford, and LingPipe. In order to combine them, a voting strategy, aiming at achieving higher overall accuracy, was used over their individual results. The general representation of the shared data makes the framework suitable for other languages for which components are readily available.

The goal of designing a flexible and easily-expandable framework was reached with the help of OSGi, a service platform and component model for Java. As such the final result of our work is an easy-to-use and easy-to-expand system, which can be accessed via a web-based user interface and run as a web service.

Next steps regarding the presented framework are to provide advanced voting methods (probability-based), make it available as an open source to the community and extend it with further modules for co-reference resolution.

9. Acknowledgements

This work has been partially funded by the EU project Excitement (FP7-IST). The development of the system described here has been started as part of a software project course which took place in the winter semester 2010 at the Saarland University. We would like to thank the participating students Jens Illig, Charles La Rosse, Iliana Simova, and Dominikus Wetzel. We also thank Sebastian Barth for his support during the implementation of the coref-layer which he did as part of a follow-up software project in 2011.

10. References

Cunningham, H. et al. (2011). Text Processing with GATE (Version 6). University of Sheffield Department of Computer Science. ISBN 0956599311.

Erman. L. (1980). The hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty. In ACM Computer Surveys, volume 12.

Florian, R., Cucerzan, S., An, I, Yarowsky, D., Schafer, C.

(2002). Combining Classifiers for Word Sense Disambiguation. In Natural Language Engineering 8, 4, 327-341.

Florian, R., Ittycheriah, A., Jing, H., Zhang, T. (2003). Named entity recognition through classifier combination. In Proceedings HLT-NAACL 2003 - Volume 4 (CONLL '03), 168-171.

van Halteren, H., Zavrel, J., Daelemans, W. (1998). Improving data driven wordclass tagging by system combination. In Proceedings of ACL - Volume 1, 491-497.

Tjong Kim Sang, E. F. (2000). Noun phrase recognition by system combination. In Proceedings of NAACL. San Francisco, CA, USA, 50-55.

Sigletos, G., Paliouras, G., Spyropoulos, C. D. and Hatzopoulos, M. (2005). Combining information extraction systems using voting and stacked generalization. In Journal of Machine Learning Research, 6:1751-1782.

Systems

[1] Alias-i. 2011. LingPipe 4.1.0.

http://alias-i.com/lingpipe (accessed February, 2011).

[2] Apache Incubator. 2011. OpenNLP 1.5.0.

http://incubator.apache.org/opennlp/index.html

(accessed February, 2011).

[3] The Stanford Natural Language Processing Group. (2011). Stanford Named Entity Recognition (NER) 1.1.1. http://nlp.stanford.edu/software/CRF-NER.shtml (accessed February, 2011).