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

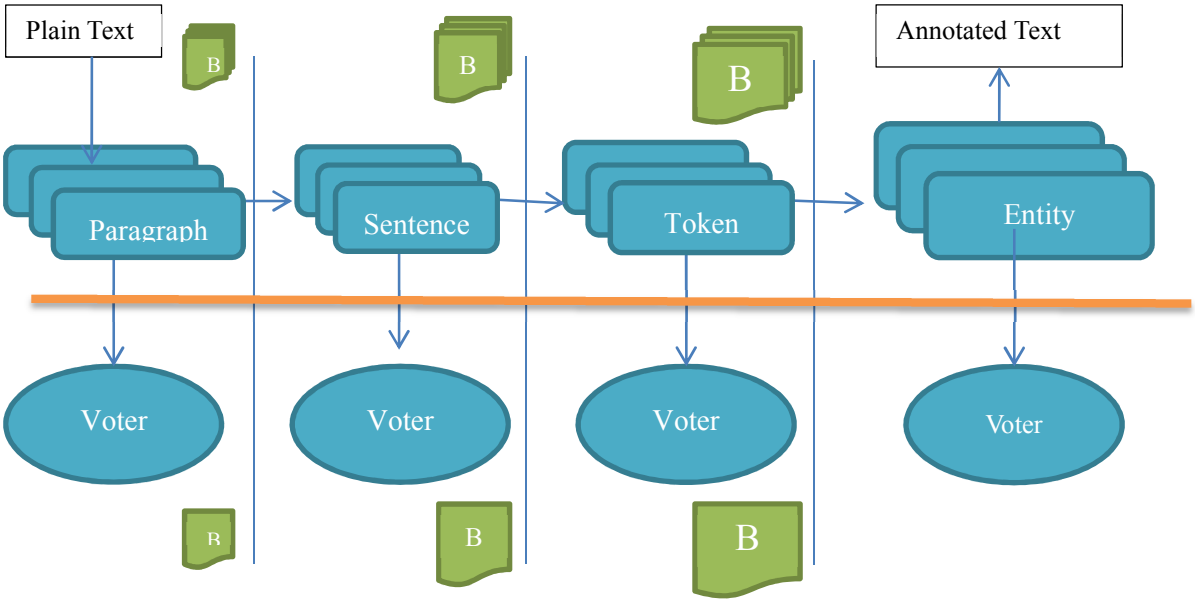

Figure 1. System Architecture

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.

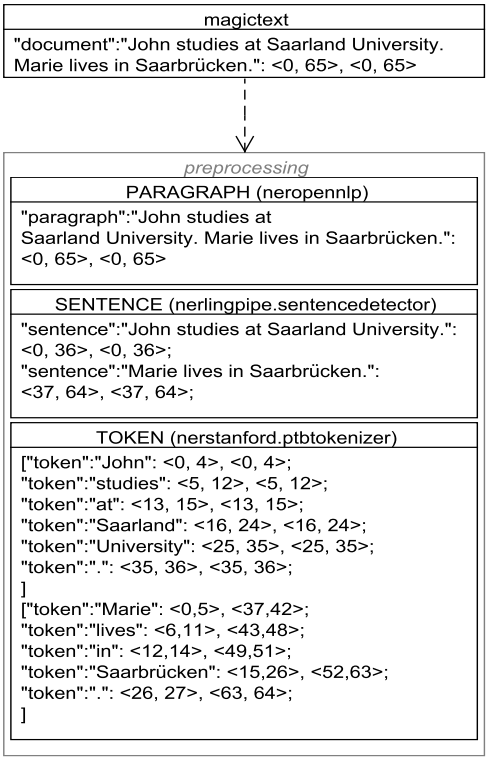

Figure 2. Blackboard data representation.

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,