WERTi - Easing Second Language Acquisition

Aleksandar Dimitrov

June 23, 2008

Contents

Preface

1 Introduction

Using tools and methods made available through new achievements in computational linguistics and related subjects like cognitive science to ease the process of second language acquisition has only recently gained focus in research projects. While adaptions of traditional data sources like dictionaries have been used for quite a while we have yet to discover new and effective ways of

WERTi tries to approach this probmlem from a more general perspective. Making use of the the momentum of the Internet, WERTi provides a platform for implementing linguistic analysis and subsequent input enhancement methods on user specified pages on the World Wide Web. Using Java Servlets technology for serving content and the UIMA framework for processing it in a dynamic and flexible session, we aim to provide a platform for linguistic processing of online content that can go beyond input enhancement.

1.1 Acknowledgements

I'd like to thank my advisor Detmar Meurers for providing me with the unique opportunity on working on the WERTi system. WERTi was originally thought out and developed in Python by him and his team at the Ohio State University. My job was to port the system he had already written to Java using UIMA and generalize the concept as to make it easier to implement new functions.

I would also like to thank Janina Radó, who provided me with invaluable feedback about the general style guidelines of papers of this kind.

Furthermore I'd like to thank , , , and for providing testing the system and for their suggestions, remarks and feature requests.

2 Explaining the Core Functionality

This section will explain the principles underlying WERTi's functionality by first looking at the data processing architecture and then showing how this is then integrated into the user interface of the web application.

2.1 UIMA Analysis Engines

The current processing line can be split up into three different parts that can interact autonomously but each depend on the output of their predecessor. The components are arranged in the following order:

HTML processing: annotates HTML tags, finds "relevant" text within the tags. This lays ground to later linguistic analysis by setting the margins of where the latter has to operate on in the web content.

Linguistic processing: performs linguistic tasks (tokenization, sentence boundary detection, part-of-speech tagging...) on the text-annotations from the previous processing step. This is also the most intensive step from a computational point of view. Optimizations to the code are most likely to yield visible results here.

Post processing: Performs all sorts of content enhancement on the annotations of the previous two processing steps. Note that this step depends on both annotation results: it also needs certain HTML markups in order to correctly organize all of the code that is going to be executed on client side.

We will now go on and explain those steps in further detail.

2.1.1 HTML Processing

The HTML processor method was designed to be primitive and efficient. While using a fully capable HTML parser was considered, we preferred to use a more

simple approach. Full and formally correct HTML markup was deemed unnecessary and too time intensive. Furthermore, changing the implementation of an analysis step even this fundamental to further processing should be easy and without side-effects as long as the requirements to preconditions and postconditions are met.

Preconditions We have a document represented as a singleton String[1] and stored in the UIMA-CAS that is passed to the `process(CAS)`-method of the `HTMLAnnotator`-class.

Postconditions The document contains annotations marking up the positions and spans of html tags in the document text. The names of tags[2] are also stored and a `isClosing`-flag is set, that denotes whether this tag is closing another[3].

2.1.2 Linguistic Processing

Linguistic processing goes through 3 major steps: *Tokenization*, *Sentence Boundary Detection*

2.1.3 Post Processing - Input Enhancement

3 The User Interface

3.1 Web Interface

3.2 Programming Interface

4 Conclusion

4.1 Loose Ends

[1] As usual in UIMA. For large documents, UMIA provides ways of splitting up documents and processing the chunks independently. However, UIMA only considers documents well beyond one megabyte to be "big enough" to be split and the plain HTML most web pages serve rarely exceeds this mark.

[2] E.g. "p" for the tag `<p>` or "div" for the dag `div`.

[3] A preceding slash in the tag name closes the tag.