

Shallow Parsing with Apache UIMA

Graham Wilcock

University of Helsinki

Finland

graham.wilcock@helsinki.fi

Abstract

Apache UIMA (Unstructured Information Management Architecture) is a framework for linguistic annotation and text analytics. Its support for standards, interoperability and scalability makes UIMA attractive for NLP researchers.

The paper describes shallow parsing as an example of configuring existing NLP tools to perform a task in the UIMA framework. First, part-of-speech tagging is done using the OpenNLP tagger. Next, full syntactic parsing by the OpenNLP parser is shown. UIMA has ready-made configurations for these tasks. Of course, tagging is fast and full parsing is slow.

Shallow parsing was enabled by adding a UIMA wrapper for the OpenNLP chunker and by extending the UIMA type system to include chunk labels. Shallow parsing with the chunker is fast, like tagging. The chunks are displayed in UIMA Annotation Viewer by re-using phrase types already defined for the full parser.

1 Apache UIMA

UIMA (Unstructured Information Management Architecture) is a Java framework for large-scale annotation and analysis of texts and other modes of unstructured information. Its design supports interoperability of annotations and scalability of applications (Webster et al., 2008; Hahn, 2008).

UIMA originated at IBM (Ferrucci and Lally, 2004) but is now an open-source Apache project (<http://incubator.apache.org/uima>) with an active community of users and developers. UIMA can be used as an Eclipse plugin (Chase, 2005) or its tools can be used independently.

In UIMA, annotations are made by *annotator* components running in *analysis engines*. New an-

notators can be written in Java or other languages, and existing annotation tools can be used in UIMA by means of *wrappers*. Configuration details for analysis engines and their annotators are specified by XML *descriptor* files.

Applications are created by combining analysis engines for the required types of annotators into an appropriate sequence. The sequence is specified in another XML descriptor file and is executed in an *aggregate analysis engine*. Each annotator in the sequence adds its annotations to the *common analysis structure* (CAS) (Götz and Suhre, 2004), a data structure containing the original text and the annotations made by previous annotators.

2 Part-of-speech tagging

A comprehensive set of robust open-source Java tools for natural language processing is available from the OpenNLP project (<http://openlp.sourceforge.net>). Some examples of using the OpenNLP tools for text annotation, both by themselves and as plugins for other tools, are given by Wilcock (2009).

UIMA includes example wrappers and descriptors for several of the OpenNLP tools, so they can easily be used as UIMA components. UIMA also provides an example descriptor for an aggregate analysis engine (`OpenNLPAggregate.xml`) that performs part-of-speech tagging by running a sequence of three OpenNLP tools: sentence detector, tokenizer, and POS tagger.

These components add annotations to the CAS. The sentence detector adds `Sentence` annotations, giving the begin and end points of each sentence. The tokenizer adds `Token` annotations, giving the begin and end points of each token. By contrast, the tagger does not add new annotations to the CAS; it updates the `posTag` features of the existing `Token` annotations. When `Token` annotations are created by the tokenizer, their `posTag` features initially have the value `Null`. The tag-

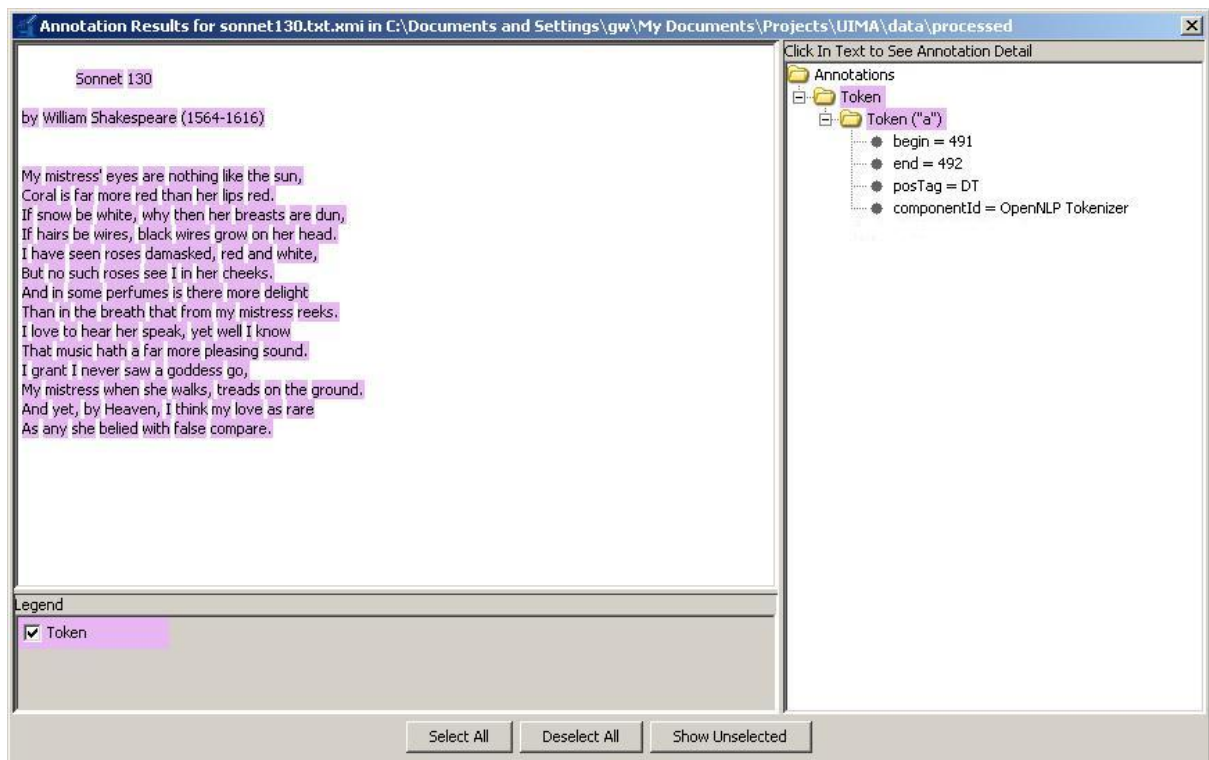


Figure 1: UIMA Annotation Viewer showing part-of-speech tagging by OpenNLP Tagger

ger subsequently updates this feature in the CAS with a part-of-speech tag from the Penn Treebank tagset. For example in Figure 1 the token “a” has a posTag value of DT (determiner).

When the annotations are displayed in UIMA Annotation Viewer as in Figure 1, they are all the same colour because they are all the same type, namely Token. The different posTag values can only be seen by inspecting individual tokens, such as the token “a” in Figure 1.

3 Full syntactic parsing

UIMA also provides a wrapper and descriptor for the OpenNLP parser. The aggregate analysis engine for tagging described in Section 2 is easily extended to include full syntactic parsing by adding the descriptor for the parser’s analysis engine.

When the extended aggregate analysis engine is run, the parser adds annotations to the CAS for the syntactic constituents (S, NP, VP ...) that it identifies. The annotations give the begin and end points of each constituent. The constituent labels are taken from the Penn Treebank set of syntactic labels used by the OpenNLP Parser.

Unlike the Token annotations in Figure 1, the syntactic annotations in Figure 2 (ADVP, ADJP, NP

...) are displayed in different colours because they are of different types. Annotations of type ADVP are yellow, annotations of type SBAR are red, and so on. The types (not the colours) are defined in an application-specific type system, which can be edited as described in Section 4.

Of course, full syntactic parsing is much slower than part-of-speech tagging, and this is a serious practical problem for large-scale text annotation. The rest of paper shows how to do shallow parsing, which is almost as fast as tagging. Shallow parsing uses chunk labels made by OpenNLP chunker, as described in Sections 5 and 6.

4 Editing the type system

An essential feature of the UIMA architecture is that all annotations are defined in an appropriate type system. This supports interoperability of annotations created by different annotation tools. The type system makes it possible to check automatically that the annotation types that are output by one component are the appropriate types to be input to the next component.

UIMA provides an example type system for use with OpenNLP tools. This ready-made type system includes types for the syntactic constituents

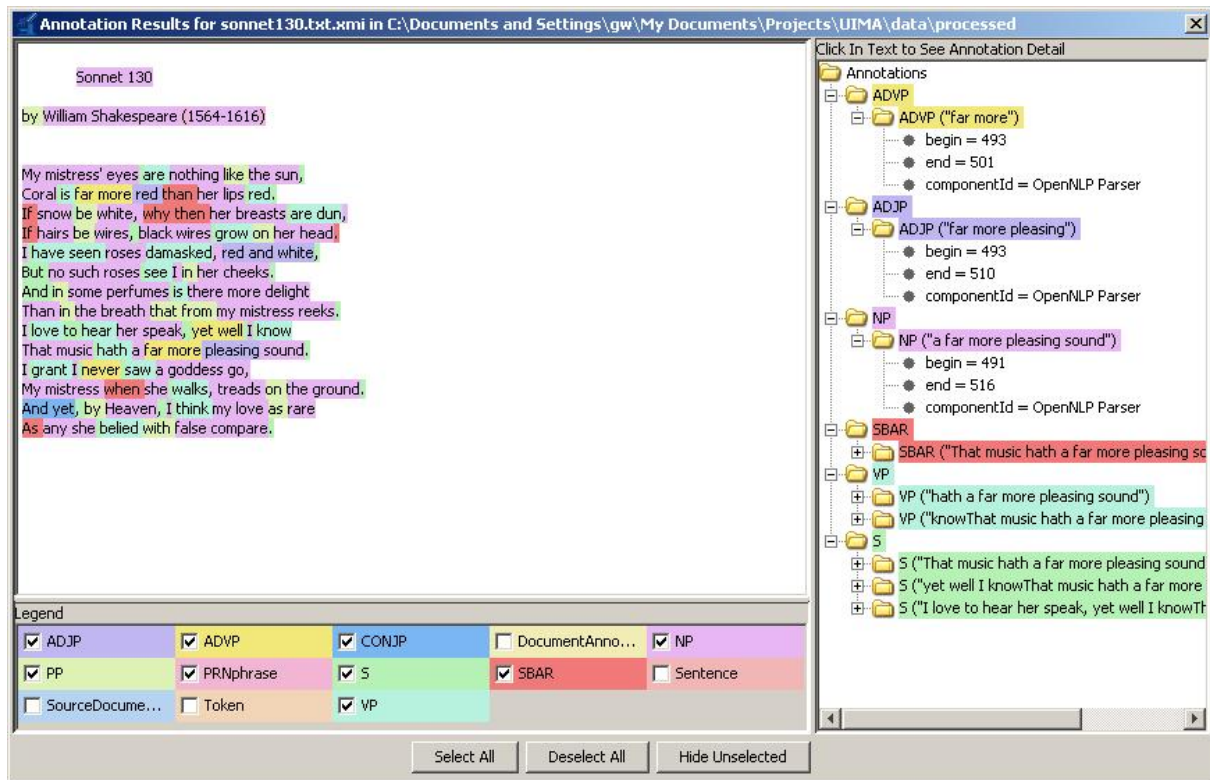


Figure 2: UIMA Annotation Viewer showing full syntactic parsing by OpenNLP Parser

used by the OpenNLP parser. However, it does not include types for chunk labels, as UIMA does not provide a wrapper for the OpenNLP chunker. Before making a wrapper for the chunker (see Section 5), the type system needs to be edited.

The example type system is specified by a descriptor file `OpenNLPexampleTypes.xml`. When UIMA is used with Eclipse, the type system can be edited with UIMA Component Descriptor Editor. Otherwise, the XML descriptor can be edited using any text editor.

The OpenNLP chunker works by adding chunk labels to tagged tokens. Therefore it is not necessary to define new annotation types for the chunk labels, it is only necessary to add a new feature to the existing Token type. The new feature will be called `chunkLabel`, as shown in Figure 3.

5 Chunk labeling

The OpenNLP chunker takes as input the tokens already found by the OpenNLP tokenizer and the tags already assigned to the tokens by the OpenNLP tagger. The chunker adds a new chunk label to each tagged token.

The chunk labels are in the IOB format used at CoNLL-2000 (Tjong Kim Sang and Buchholz,

2000). The first token in an NP chunk is labelled B-NP (Begin NP). The other tokens up to the end of the NP are labelled I-NP (Inside NP). Other chunk types have similar labels. Tokens that are not in any chunk are labelled O (Outside).

UIMA does not provide a wrapper for the OpenNLP chunker, so a new wrapper was written in Java. The wrapper uses the chunk labels assigned by the chunker to update the `chunkLabel` features of the Token annotations in the CAS. For example in Figure 3 the token “a” has been given a `chunkLabel` value of B-NP.

The wrapper for the OpenNLP chunker was easy to write as it is very similar to the example wrapper provided by UIMA for the OpenNLP tagger. The tokenizer creates Token annotations in the CAS with `posTag` and `chunkLabel` features both initialized to `Null`. The wrapper for the tagger updates the `posTag` feature with the part-of-speech tag, and the new wrapper for the chunker updates the `chunkLabel` feature with the chunk label in the same way.

6 Shallow parsing

As already noted in Section 2 in the case of tagging, when the Token annotations are viewed in

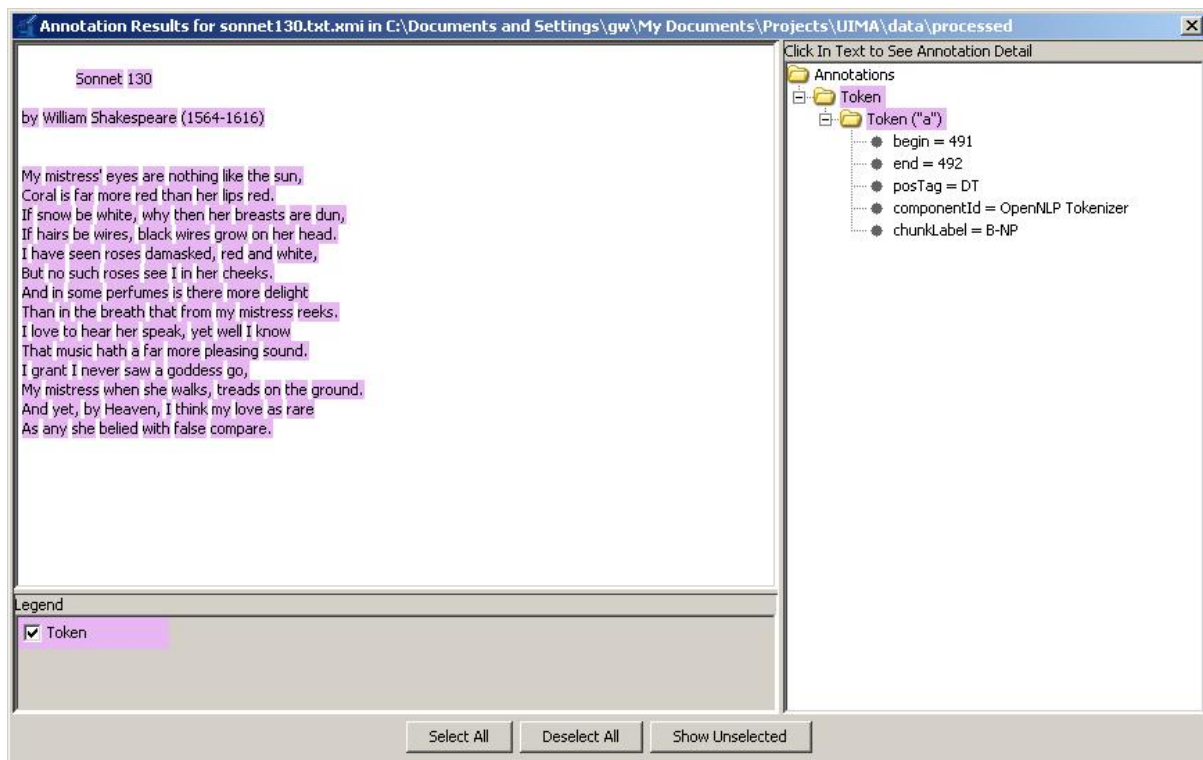


Figure 3: Chunk labeling by OpenNLP Chunker

UIMA Annotation Viewer (Figure 3), they are all the same colour because they are all the same type. The different chunkLabel values can only be seen by inspecting individual tokens, such as the label B-NP for the token “a” in Figure 3.

In order to see where the chunks begin and end, and to distinguish the different chunk types, they should be displayed in different colours like the constituents found by the parser in Figure 2. This is quite easy. It requires an annotator that creates a new annotation in the CAS for each chunk, giving the begin and end points of the chunk, and also specifying the chunk type.

The begin point of the chunk is the begin point of the first token in the chunk. In the case of an NP chunk, this is the token with chunkLabel value B-NP. The end point of the chunk is the end point of the last token in the chunk. In an NP chunk, this is the last consecutive token with chunkLabel value I-NP. This is found with a simple loop.

The chunk types can be specified by exploiting the fact that the chunk types (NP, PP, VP ...) are also syntactic constituents used by the OpenNLP parser. These types are therefore already defined in the example type system, as noted in Section 4. Annotations for these types can be added to the CAS in the same way as in the example wrapper

for the OpenNLP parser. These new annotations for different chunk types are displayed in different colours by UIMA Annotation Viewer, as shown in Figure 4.

This form of shallow parsing by the OpenNLP chunker is fast. Its speed is comparable to part-of-speech tagging by the OpenNLP tagger. It is much faster than full syntactic parsing by the OpenNLP parser.

7 Conclusion

Apache UIMA is an attractive framework for NLP researchers, but it is not yet widely known. The paper therefore presents an introduction to UIMA by describing how existing NLP tools can be configured to perform an annotation task in the UIMA framework. First, part-of-speech tagging was shown using the OpenNLP tagger, and full syntactic parsing was shown using the OpenNLP parser. These are tasks for which UIMA provides ready-made configurations.

In order to demonstrate how to perform a task for which UIMA does not provide a ready-made configuration, shallow parsing was implemented. This required adding a UIMA Java wrapper for the OpenNLP chunker. In addition, the UIMA type

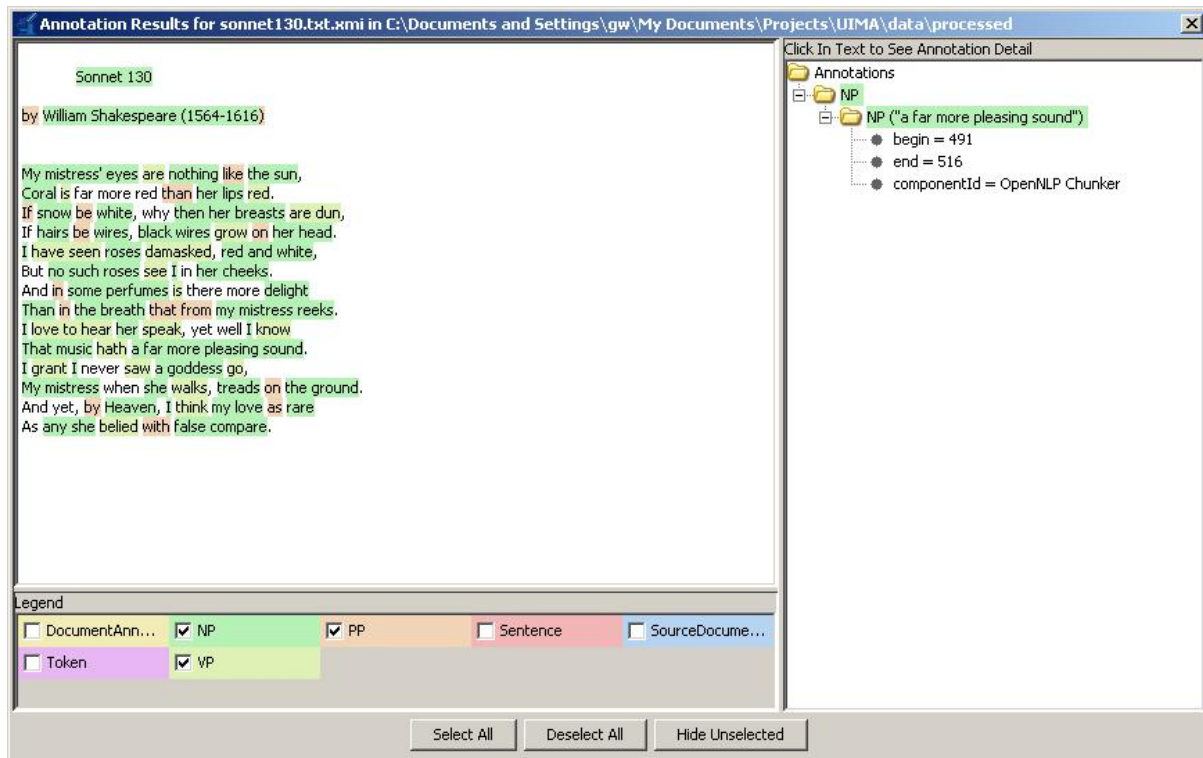


Figure 4: Shallow parsing by OpenNLP Chunker

system was extended to include chunk labels. The practical motivation for doing shallow parsing is that it is much faster than full syntactic parsing, in fact it is comparable in speed to tagging.

There are also several reasons why this form of shallow parsing with the OpenNLP chunker is a good example for learning about UIMA. The new Java wrapper is easy to write as it is similar to the existing wrapper for the OpenNLP tagger. The extension to the type system is easy to understand as the new `chunkLabel` feature is similar to the existing `posTag` feature. Finally, the chunked phrases are easy to display in UIMA Annotation Viewer as the phrase types are already defined for the full syntactic parser.

Acknowledgments

The author is a recipient of an IBM Innovation Award for Unstructured Information Analytics.

References

- Nicholas Chase. 2005. Create a UIMA application using Eclipse. <http://www-128.ibm.com/developerworks/edu/x-dw-xml-i.html>.
- David Ferrucci and Adam Lally. 2004. Building an example application with the Unstructured In-

formation Management Architecture. *IBM Systems Journal*, 43(3):455–475.

- Thilo Götz and Oliver Suhre. 2004. Design and implementation of the UIMA Common Analysis System. *IBM Systems Journal*, 43(3):476–489.
- Udo Hahn, editor. 2008. *Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP*. Marrakech. Workshop at LREC-2008.
- Erik Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*. Lisbon.
- Jonathan Webster, Nancy Ide, and Alex Chengyu Fang, editors. 2008. *Proceedings of the First International Conference on Global Interoperability for Language Resources*. Hong Kong.
- Graham Wilcock. 2009. *Introduction to Linguistic Annotation and Text Analytics*. Morgan and Claypool.