

Documentation for the

JULIE Lab UIMA Wrapper for MST Parser

Katrín Tomanek Oleg Lichtenwald

Jena University Language & Information Engineering (JULIE) Lab

Fürstengraben 30

D-07743 Jena, Germany

katrin.tomanek@uni-jena.de

lichtenwald@coling-uni-jena.de

1 Objective

The JULIE Lab UIMA Wrapper for MST Parser is an UIMA Analysis Engine that determines and annotates all syntactic dependencies between words within a sentence. It is part of the JULIE Lab NLP tool suite¹ which contains several NLP components (all UIMA compliant) from sentence splitting to named entity recognition as well as a comprehensive UIMA type system.

The MST Parser is a dependency parser that searches for maximum spanning trees over directed graphs. Models of dependency structure are based on large-margin discriminative training methods. The functionality of the engine is predicated on the base implementation by Ryan McDonald and Jason Baldridge. For in-depth description of the parsing algorithm see McDonald et al. [MPRH05].

The MST parser was originally obtained from sourceforge.net² and integrated in UIMA during a course on computational linguistics at the JULIE Lab. Some changes to the original source code have been made, which was necessary to neatly integrate the parser in UIMA. These changes had the effect, that the model can be loaded now during the initialization phase of the UIMA wrapper (in the original version for each document the model was loaded as it wasn't intended to be run in a pipeline with multiple documents). Also, the paths to the input and output file can be set, which makes the parser more flexible. The code in the class `ParserOptions.java` was modified and thus a constructor

¹<http://www.julielab.de/>

²<http://sourceforge.net/projects/mstparser/>

was added to this class in order to set the model file name, the input file (test file), the output file and the format.

During the processing of documents, the annotator takes sentence annotations from the CAS and writes dependency relation annotations back to it.

2 Requirements and Dependencies

The annotator is written in Java (Java 1.5 or above required) using Apache UIMA version 2.2.1-incubation³.

The input and output of an AE takes place by annotation objects. The classes corresponding to these objects are part of a *JULIE UIMA Type System* in the current version 2.1.⁴

3 Using the AE – Descriptor Configuration

In UIMA, each component is configured by a descriptor in XML. In the following we describe how the descriptor required by this AE can be created with the *Component Descriptor Editor*, an Eclipse plugin which is part of the UIMA SDK.

A pre-configured descriptor is already contained in this package (see `desc/MSTParserDescriptor.xml`).

A descriptor contains information on different aspects. The following subsection refers to each sub aspect of the descriptor which is, in the Component Descriptor Editor, a separate *tabbed page*. For an indepth description of the respective configuration aspects or tabs, please refer to the *UIMA SDK User's Guide*⁵, especially chapter 12 on “Component Descriptor Editor User's Guide”.

To define your descriptor go through each tabbed page mentioned here, make your respective entries and save the descriptor as, e.g., `MSTParserDescriptor.xml`.

Overview This tab provides general information about the component. For the JULES MST Parser you need to provide the information as specified in Table 1.

Aggregate Not needed here, as this AE is a primitive.

³<http://incubator.apache.org/uima/>

⁴The *JULIE UIMA type system* can be separately obtained from <http://www.julielab.de/>

⁵<http://incubator.apache.org/uima/>

Subsection	Key	Value
Implementation Details	Implementation Language	Java
	Engine Type	primitive
Runtime Information	updates the CAS	check
	multiple deployment allowed	check
	outputs new CASes	don't check
	Name of the Java class file	<code>de.julielab.jules.mstparser.MSTParserAnnotator</code>
Overall Identification Information	Name	MST Parser Annotator
	Version	2.3
	Vendor	JULIE Lab
	Description	you may keep this empty

Table 1: Overview/General Settings for AE.

Parameters See Table 2 for a specification of the configuration parameters of this AE. Do not check “Use Parameter Groups” in this tab.

Parameter Settings The specific parameter settings are filled in here. For each of the parameters defined in Table 2, add the respective values here (has to be done at least for each parameter that is defined as mandatory). See Table 3 for the respective parameter settings of this AE.

Type System On this page, go to *Imported Type* and add the *JULIE UIMA Type System* (Use “Import by Location”).

Capabilities Nothing needs to be done here.

Index Nothing needs to be done here.

Resources Nothing needs to be done here.

Parameter Name	Parameter Type	Mandt.	Multi-valued	Description
modelFileName	String	yes	no	Specifies the file name of the trained model which will be used by the parser.
temporaryPath	String	yes	no	Specifies the path where the temporary files will be stored.
format	String	yes	no	Determines whether the MST format or the CONLL format will be used.
posTagSetPref	String	no	no	Specifies the preferred pos tag set, if the CAS contains pos tag annotations from different pos tag sets. In order to achieve better results, make sure that the preferred pos tag set fits the model. Please also note that there must be pos tag annotations before running the parser, otherwise you'll get an error.
secondOrder	Boolean	yes	no	Determines, whether the parser should be run in second order mode. If set to "false", then the first order will be used, otherwise the second order will be used. Make sure that it fits to the model, otherwise you'll get an error.

Table 2: Parameters of this AE.

Parameter Name	Parameter Syntax	Example
modelFileName	set to a valid file name	src/test/resources/models/small.model
temporaryPath	set to a valid and reasonable path	src/test/resources/data/tmp
format	valid formats are “ <i>MST</i> ” and “ <i>CONLL</i> ”	CONLL
posTagSetPref	set to a valid type, full class name required	de.julielab.jules.types.PennBioIEPOSTag
secondOrder	valid values are “ <i>true</i> ” and “ <i>false</i> ”	false

Table 3: Parameter settings of this AE.

4 Training of the MST Parser

The MST Parser must be trained in the version we provide in this package. Use the class `DependencyParser.class` from this package to train a new model with the following parameters `train train-file:'trainfile' format:'format'`. For more information about the training of MST parser please consult the README file in the original MST parser distribution.

Note that the model contained in this package was trained on the material provided in the original MST parser distribution (CoNLL data). When applying this component in practice you should consider to retrain the parser on a large set of training material (e.g., WSJ).

5 Copyright and License

This software is Copyright (C) 2008 Jena University Language & Information Engineering Lab (Friedrich-Schiller University Jena, Germany), and is licensed under the terms of the Common Public License, Version 1.0 or (at your option) any subsequent version.

The license is approved by the Open Source Initiative, and is available from their website at <http://www.opensource.org>.

References

- [MPRH05] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. Non-projective dependency parsing using spanning tree algorithms. In *Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 523–530, Vancouver, B.C., Canada, 2005. Association for Computational Linguistics.