

Otto-von-Guericke-Universität Magdeburg



Fakultät für Informatik  
Institut für Wissens- und Sprachverarbeitung

# Labour Praktikum

## Coreference in UIMA

Author:

Sagar Sunkle

July 18, 2007

Advisor:

Prof. Dr. Dietmar Rösner,  
Dipl.-Inf. Manuela Kunze

Universität Magdeburg  
Fakultät für Informatik  
Postfach 4120, D-39016 Magdeburg  
Germany

# Contents

<b>Contents</b>	<b>2</b>
0.1 Coreference Task . . . . .	3
0.2 Coreference . . . . .	4
0.2.1 Coreference in OpenNLP . . . . .	5
0.2.2 English Coreference . . . . .	5
0.3 UIMA . . . . .	7
0.4 Challenges & Plan of implementation for the Coreference Task . . . . .	9
0.4.1 Challenges . . . . .	9
0.4.2 Implementation Details . . . . .	9
0.5 Sample sentences and coreferring entities . . . . .	12
<b>Bibliography</b>	<b>13</b>

## 0.1 Coreference Task

OpenNLP is an organizational center for open source projects related to natural language processing. OpenNLP also hosts a variety of java-based NLP tools which perform sentence detection, tokenization, pos-tagging, chunking and parsing, named-entity detection, and Coreference using the OpenNLP Maxent machine learning package[Rat98]. IBM's UIMA (Unstructured Information Management Architecture) is an open, industrial-strength, scalable and extensible platform for creating, integrating and deploying unstructured information management solutions from combinations of semantic analysis and search components. UIMA contains example wrappers for OpenNLP so that OpenNLP Tools can be run as UIMA annotators. The wrappers provide a thin layer over the OpenNLP classes and use the "outermost" APIs to those classes. All of the OpenNLP components were wrapped as annotators in UIMA except the Coreference component. The proposed Coreference task was to create such a wrapper for the Coreference component and integrate it with UIMA like the other OpenNLP components.

## 0.2 Coreference

A natural language expression used to perform reference is called a referring expression, and the entity that is referred to is called the referent. Two referring expressions that are used to refer to the same entity are said to corefer. Reference to an entity that has been previously introduced into the discourse is called anaphora, and the referring expression used is said to be anaphoric.

An important related term is Discourse. A Discourse is a collocated, related groups of sentences e.g. monologue, dialogue and HCI (Human Computer Interaction). A Discourse Context is some prior knowledge about entities involved. A Discourse Model is a mental (or computational) model of ongoing discourse that encodes a set of beliefs about referred entities and relationships between them.

A variety of referring expressions & referents have been recognized such as indefinite noun phrases, definite noun phrases, pronouns, demonstratives , one anaphora, inferrables, discontinuous sets, generics etc[JM00].For identifying referents of referring expressions many criteria can be used such as syntactic and semantic constraints, number agreement, person and case agreement, syntactic constraints, selection restrictions etc. Particularly for pronoun interpretation, recency, grammatical role, repeated mention,and verb semantics are used more often.

## 0.2.1 Coreference in OpenNLP

OpenNLP has various classes for various Natural Language Processing Tasks.

The following table lists these classes and the task they carry out.

<i>OpenNLP(API)Class</i>	<i>NLPTask</i>
opennlp.tools.lang.english.SentenceDetector	Sentence Detector
opennlp.tools.lang.english.Tokenizer	Tokenizer
opennlp.tools.lang.english.PosTagger	Pos-tagger
opennlp.tools.lang.english.TreebankChunker	Chunker
opennlp.tools.lang.english.NameFinder	Name Finder
opennlp.tools.lang.english.TreebankParser	Parse
opennlp.tools.lang.english.TreebankLinker	Coreference

Table 1: OpenNLP Classes and Tasks

The class `TreebankLinker` performs Coreference for Treebank style parses. It only performs Coreference over constituents defined in the trees and does not generate new constituents for pre-nominal entities or sub-entities in simple coordinated noun phrases. This linker requires that named-entity information also be provided. This information can be added to the parse using the `-parse` option with `EnglishNameFinder`.

In OpenNLP the coreferring entities are treated as Coreference relationships which specify when two separate spans of text refer to the same entity in the world. In OpenNLP co-referring entities are the Mentions of same entity grouped together. Reference resolution modules in OpenNLP attempt to identify the Coreference relationships using a data structure that creates a discourse model, and the entities are represented in terms of file cards where each mention of an entity is listed on the file card for that entity. File cards are created when entities are first mentioned and updated as additional mentions are encountered. The file card also typically contains information about the entity that is pertinent to how it is referred to. For English these properties include the entity's gender, number, semantic category, and syntactic position as all of these characteristics affect reference resolution. File cards can also be merged in cases where a new mention unifies two previous mentions that were incorrectly considered separate entities.[Mor05]

## 0.2.2 English Coreference

The process of finding Coreference parses is as follows : Perform sentence detection over the text using model for sentence detection (`EnglishSD.bin.gz`), parse the text using Penn TreeBank Parser models, embed named entity information in the parse thus obtained using `NameFinder` models, and using `Wordnet` and the `TreeBankLinker` class obtain a Coreference Parse. (OpenNLP term)

The factors syntax, locality, gender and accessibility are considered when determining which noun phrases a pronoun is coreferent with. Using these features and a collection MUC annotated data; OpenNLP has trained statistical models of Coreference resolution[Mor97].

<i>Features</i>	<i>S</i>	<i>G</i>	<i>L</i>	<i>A</i>
1.The distance in NPs between pronoun and antecedent			x	
2.Hobb's distance in NPs between pronoun and antecedent and pronoun's gender	x	x	x	x
3.Distance in sentences between pronoun and antecedent of pronoun	x	x	x	
4.NP's position in the sentence	x			x
5.The word and POS tag preceding and following the antecedent	x			x
6.Number of time antecedent has been mentioned				x
7.Headword of antecedent and gender of pronoun		x		
8.Head POS tag of antecedent and gender of pronoun		x		
9.Modifier words and POS tags of antecedent and gender of pronoun		x		
10.Modifier POS tags of antecedent and gender of pronoun		x		
11.The word and POS tag preceding and following the pronoun	x			
12.The pronoun	x			

Table 2: Features for Maximum Entropy model of Coreference Resolution in OpenNLP(S-Syntax,G-Gender,L-Locality,A-Accessibility)

## 0.3 UIMA

An Analysis Engine (AE) is a program that analyzes artifacts (e.g. documents) and infers information from them. A TAE is a specialization of an Analysis Engine that analyzes a particular artifact, which is often, for example, a text document (but could be, in general, audio streams, etc.). In the UIMA SDK, Analysis Engines are constructed from building blocks called Annotators. An annotator is a component that contains analysis logic. Annotators analyze an artifact (for example, a text document) and create additional data (metadata) about that artifact. It is a goal of UIMA that annotators need not be concerned with anything other than their analysis logic - for example the details of their deployment or their interaction with other annotators. An Analysis Engine (AE) may contain a single annotator (this is referred to as a Primitive AE), or it may be a composition of others and therefore contain multiple annotators (this is referred to as an Aggregate AE). Primitive and aggregate AEs implement the same interface and can be used interchangeably by applications. Annotators produce their analysis results in the form of typed Feature Structures, which are simply data structures that have a type and a set of (attribute, value) pairs. An annotation is a particular type of Feature Structure that is attached to a region of the artifact being analyzed (a span of text in a document, for example).

It is also possible for annotators to record information associated with the entire document rather than a particular span.. All feature structures, including annotations, are represented in the UIMA Common Analysis Structure (CAS). The CAS is the central data structure through which all UIMA components communicate. Included with the UIMA SDK is an easy-to-use, native Java interface to the CAS called the JCas.

Annotator implementations all implement a standard interface, having several methods, the most important of which are: `initialize()`, `process()`, and `destroy()`.

1. `initialize()` is called by the framework once when it first creates the annotator.
2. `process()` is called once per item being processed.
3. `destroy()` may be called by the application when it is done.

There is a default implementation of this interface for annotators using the JCas, called `JCasAnnotator_ImplBase` (since v2.0).

For every component specified in UIMA there are two parts required for its implementation :

- The declarative part
- The code part

The declarative part contains metadata describing the component, its identity, structure and behavior and is called the Component Descriptor. Component descriptors are represented in XML. The code part implements the algorithm. The code part is generally a

program in Java. Component descriptors aid in component discovery, reuse, composition and development tooling. Component descriptors contain standard metadata including the component's name, author, version, and a reference to the class that implements the component. In addition to these standard fields, a component descriptor identifies the type system the component uses and the types it requires in an input CAS and the types it plans to produce in an output CAS.

The following diagram shows how the UIMA Framework creates and interacts with an Analysis Engine.[IBM06]

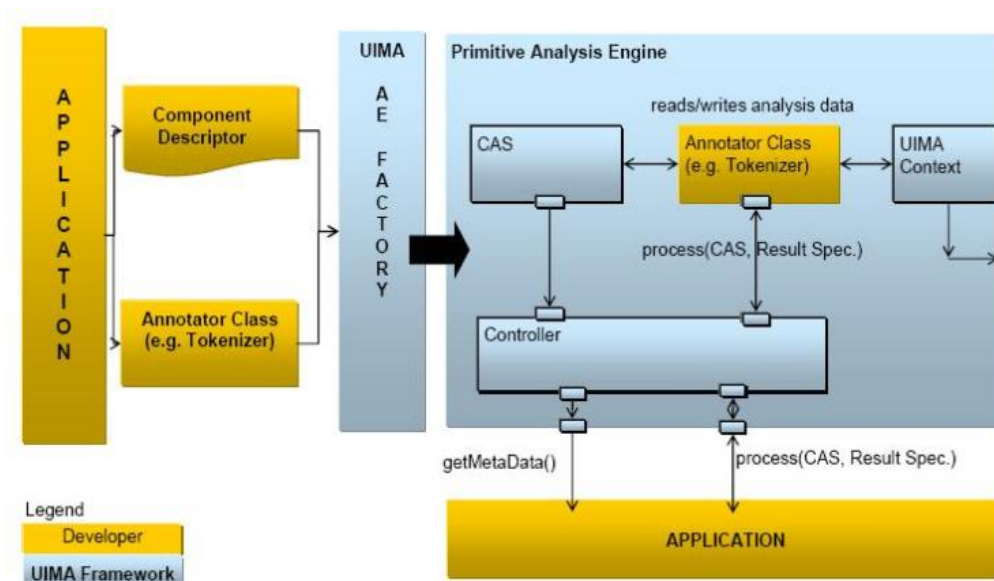


Figure-1 Creating an Analysis Engine with Annotators and descriptors.

## 0.4 Challenges & Plan of implementation for the Coreference Task

### 0.4.1 Challenges

1. There was no easy to use interface in OpenNLP for integration of Coreference in OpenNLP to UIMA.

Some classes in OpenNLP, specifically `TreeBankLinker` and `NameFinder`, that are very important to Coreference processing did not have methods that would enable me to show the coreferring entities the way I wanted them to. The representation of Coreference parse needs to be rewritten so that I can operate on it better.

2. UIMA requires that component to be integrated in it, should be an annotator. The UIMA way is to create a descriptive xml storing metadata about the annotator and provide a separate java code implementing the annotator. I would have to represent the interface to OpenNLP I created as an annotator and possibly create a specific type system for the same.

I had also thought of possibility of an aggregate AE out of pre-existing OpenNLP annotators, but they are not designed in such way that their output capabilities may be used to pipe into one another to obtain the Coreference functionality. It would have required changing the pre-existing OpenNLP annotators to a substantial degree.

The implementation of OpenNLP Coreference component for UIMA consists therefore, of two distinct phases. The first phase deals with the OpenNLP library obtaining the Coreference functionality and in the other phase, this functionality is wrapped so that a Coreference annotator can be created for use within UIMA.

### 0.4.2 Implementation Details

#### Phase 1: Creating interface to Coreference modules of OpenNLP

The command line executable provided in OpenNLP is not usable for our purpose; therefore we must create our own interface to OpenNLP.

Various OpenNLP classes can take the text as input and output the processed text which can be piped to another Class for further processing. To integrate the Coreference module in UIMA though, we require more fine grained control over how the Coreference is carried out. Within OpenNLP the classes responsible for Coreference are `TreeBankLinker` and `NameFinder`. The class `TreeBankLinker` creates an inner class called `CorefParse` which I had to augment with methods that output the parse of a sentence in various formats along with useful information. The `NameFinder` embeds

names for various entities within sentences, but again the class had no public interface so that it could not be used with finer control. I had to therefore refactor and rewrite both `TreeBankLinker` and `NameFinder` with the kind of utility methods I required to create a useful interface to Coreference in OpenNLP. I created classes `TreeBankLinkerSG` and `NameFinderSG` as modified versions of the old classes. A class called `Coreference` that hides the details of call structure and presents user with some simple methods that output coreferring entities in various ways was also created. Its important methods are listed in the table below.

<i>Method</i>	<i>Meaning</i>
Map CoreferenceMap- pings()	Returns the Map of coreferring Entities. This map is provided as input to the <code>CoreferenceText</code> and similar methods to obtain a stringified representation of coreferring entities.
String Coreference- Text(Map m)	Returns the coreference text with the format as displayed in the document analyzer.
void Init_Model_Paths(String Coref, String parse, String names, String sentdet, String token)	Sets various Directory Paths to models required for Coreference Analysis. Uses 'Preferences' API of Java to provide configuration file like functionality.
Other utility methods	Show Coreference output in different ways, Set the text to be analyzed etc.

Table 3: Coreference Methods and Meaning

As described earlier the Coreference in OpenNLP requires an input which is processed through sentence detection, parsing, name finding etc. The two classes `NameFinderSG` and `TreeBankLinkerSG` in the `Coreference.jar` file interact with these modules of OpenNLP to obtain an internal representation of parse suited for Coreference. The Coreference module requires various English language models and a WordNet interface apart from the functionality mentioned above.

## Phase 2-A: Creating the UIMA Analysis Engine for Coreference

As we have seen, UIMA requires that every component in it should have a declarative part and a code part. In order to process and display coreferring entities in the Document Analyzer, first of all I required a type system to present the coreferring

entities in annotated form. The descriptor for the same is `OpenNLPcusCoref.xml`. Every `CorefAnnotation` has two features called `CorefMap` which is a stringified version of the Coreference map we obtain via the `Coreference Class` and `ComponentID` that specifies the name of the annotation type. Once I had the type system defined and java files generated via `JCasGen`, the next task was to call methods of `Coreference` class from within UIMA and set various features of a `CorefAnnotation` instance. The descriptor for this purpose is `CoreferenceDescriptor.xml` which is also the main descriptor for `Coreference` component AE in UIMA. It points to `CoreferenceComp.java` for the java code. Its parameters are paths to the various OpenNLP model files. In parameter settings I have to specify the default location for each model, i.e. a value for each parameter. Individual user must change these values to point to the OpenNLP model paths on his/her system. The type system is imported 'By Location'. The capabilities include only the `CorefMap` feature of the `CorefAnnotation` as an output capability. No indexes or resources have been used.

### **Phase 2-B: Packaging the UIMA Coreference Component**

With the above two main subtasks completed, what remained was to create a PEAR package for the UIMA component. The procedure given in [IBM06, p. 14-247] was followed. The component of course requires paths to UIMA core and OpenNLP libraries to be added to the project. This can be achieved by editing the `setenv.txt` file in the metadata folder for the component.

## 0.5 Sample sentences and coreferring entities

Following are some texts against which the OpenNLP's coreference module obtains the coreferring entities as shown:

<i>Text</i>	<i>CoreferringEntities</i>
1. John likes to eat his burgers. He enjoys to eat them with his wife. Her name is Jane. She likes to eat burgers with sauce.	(He,his,John,his),(his wife.,Her,She ),(them,his burgers.)
2. French President Nicolas Sarkozy's UMP party says it will press ahead with wide-ranging reforms, after winning a majority in parliamentary elections.	(President Nicolas Sarkozy's UMP party,it)
3. Most computational linguists prefer their own parsers.	(Most computational linguists,their)
4. General Motors announced their third quarter profit of \$0.02.	(their,General Motors)
5. Producers don't like to see a hit wine increase in price. Producers have seen this market opening up and they're now creating wines that appeal to these people.	(Producers,Producers)
6. Ford announced a new product line yesterday. Ford spokesman John Smith said they will start manufacturing widgets.	(Ford spokesman John Smith,they)
7. Emily is a very good student. She studies philosophy at the University. John is her fellow student. The University offers many courses. Many of them belong to the humanities. Some are interesting. Others are boring.	(her fellow,a very good student.),(them,She,Emily,her )

Table 4: Sample texts and coreferring entities.

In the above table texts 1 to 4 are accurately resolved. In the 5th text, 'they' of 'they're' is not resolved correctly. In the 6th text, 'they' should refer to the Ford company, in the 7th 'her fellow' and 'a very good student' as well as 'them', 'she', 'Emily', 'her' are resolved incorrectly.

OpenNLP Coreference is specifically concerned about pronoun resolution. In this, it tries to resolve the following kinds of Coreference: common noun, definite noun, plural nouns, proper nouns, singular pronouns and speech pronouns[Mor05]. The accuracy of results of course depends on annotated data on which resolution models have been trained. All other kinds of Coreference such as demonstratives, one Anaphora, inferrables, discontinuous sets, generics etc are NOT handled by OpenNLP.

# Bibliography

- [All95] Allen, J.: *Natural Language Understanding*. Benjamin Cummnings, 1995.
- [IBM06] IBM: Unstructured information management architecture (uima)- sdk user's guide and reference (version 2.0), 2006.
- [JM00] Jurafsky, D.; Martin, J. H.: *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice-Hall, 2000.
- [Mor97] Morton, T.: Coreference for nlp applications. *In Proceedings ACL*, 1997.
- [Mor05] Morton, T.: *Using Semantic Relations To Improve Information Retrieval*. Dissertation, University of Pennsylvania, 2005.
- [Rat98] Ratnaparkhi, A.: *Maximum Entropy Models For Natural Language Ambiguity Resoultion*. Dissertation, University of Pennsylvania, 1998.