

HW #3 Report: CPE and UIMA-AS

Version: 0.1

Fall 2013 11-791

Jeffrey Gee

Contents

1	Introduction	3
1.1	Purpose	3
1.2	Terminology	3
1.3	Document References.....	3
1.4	Tasks.....	3
2	Collection Processing Engine	4
2.1	CPE Deployments.....	4
2.2	Collection Readers	4
2.2.1	FileSystemCollectionReader	4
2.3	Analysis Engines	4
2.3.1	HW2 Aggregate Analysis Engine (hw2-jgee1-aae.xml)	4
2.3.2	HW3 Aggregate Analysis Engine (hw3-jgee1-aae.xml)	5
2.4	CAS Consumers	5
2.4.1	TopNEvaluator (CAS Consumer)	5
2.5	HW3 vs. HW2 Aggregate Analysis Engine	6
3	UIMA-AS vs. Local Annotator	7
3.1	Performance	7
3.1.1	Initialization Cost.....	7
3.1.2	Input File Size	7
3.1.3	Simple Comparison	7

1 Introduction

1.1 Purpose

The purpose of this document is to describe the contents of Assignment #3.

1.2 Terminology

The following is a list of unobvious terms/acronyms that may be used throughout this document.

Word/Acronym	Definition
AAE	Aggregate Analysis Engine
CPE	Collection Processing Engine
JMS	Java Message Service
UIMA-AS	UIMA Asynchronous Scaleout

1.3 Document References

The following documents may include supplemental information useful for understanding this document.

Document	Comment
Homework #2 Report	Outlines the design of a simple Q & A AAE.

1.4 Tasks

The following tasks were accomplished in Assignment #3.

1. Create a Collection Processing Engine (CPE) based on our Assignment #2 Aggregate Analysis Engine (AAE).
2. Develop a JMS client descriptor for a remotely deployed instance of the ClearTK's Stanford Core NLP Annotator.
3. Develop remote deployment descriptors for an Apache UIMA-AS deployment of our Assignment #2 AAE.
4. Develop a JMS client descriptor for the Apache UIMA-AS service described in task #3.
5. Using the JMS client descriptor developed in task #2, integrate the NamedEntityMention annotations into an answer scoring mechanism. Compare this answer scoring mechanism to the answer scoring mechanism in the HW#2 AAE (N-Gram Overlap Answer Scorer).
6. Compare performance between a locally deployed and remotely deployed instances of Stanford Core NLP Annotator.

2 Collection Processing Engine

2.1 CPE Deployments

The CPE deployments for this assignment are described in the table below.

Name	Collection Reader	Analysis Engine	CAS Consumer
hw3-jgee1-CPE.xml	File System Collection Reader	HW#2 AAE	Top N Evaluator
hw3-jgee1-as-aae.xml		HW#3 AAE (Stanford Core NLP + modified HW#2 AAE)	

2.2 Collection Readers

This section will describe the Collection Reader components used in homework assignment #3.

2.2.1 FileSystemCollectionReader

The FileSystemCollectionReader is a CPE component that reads files from a local file system and generates CAS objects. The FileSystemCollectionReader requires an input parameter specifying the location of the input files. The FileSystemCollectionReader is part of the UIMA binaries and therefore the only piece contributed as part of this assignment was its Collection Reader Descriptor file.

2.3 Analysis Engines

This section will describe the Analysis Engine components used in homework assignment #3.

2.3.1 HW2 Aggregate Analysis Engine (hw2-jgee1-aae.xml)

As the Analysis Engine for the CPE in this assignment, we deploy the aggregate analysis engine from previous assignment. Below is a summary of the HW#2 AAE.

(*Note: The TopNEvaluator component was removed from the original HW#2 AAE, as evaluation is now accomplished via an identical CAS Consumer.)

	Annotator	Comment
1	Test Element Annotator	Read in the simple input file and annotate question and answer spans.
2	Token Annotator	Annotate each token span in each question and answer.
3	Unigram Annotator	Annotate each unigram in each question and answer.
4	Bigram Annotator	Annotate each bigram in each question and answer.
5	Trigram Annotator	Annotate each trigram in each question and answer.
6	N-Gram Overlap Scorer	Scores each candidate answer based on the % of its N-grams (unigrams, bigrams, and trigrams) overlap with those of the question.

For more details regarding the HW#2 AAE, please see the report accompanying HW#2.

2.3.2 HW3 Aggregate Analysis Engine (hw3-jgee1-aae.xml)

In the HW#3 we incorporated a remote deployment of a Stanford Core NLP Annotator into our Analysis Engine. The Stanford Core NLP Annotator reads in the document text and produces various annotation types (such as its own Token and Sentence types, and Dependency Node types). The annotations that we incorporated into the new analysis engine were the NamedEntityMention types.

In order to incorporate the annotations into our answer scores, we replaced the typical N-Gram Overlap Scorer from HW#2 with a new Named Entity Scorer, which scores answers based on a named entity overlap. In reality, as we are no longer using N-Grams, a majority of the annotators could be removed from the AAE (only Test Element Annotator is still required).

The following table shows the HW#3 AAE flow:

	Annotator	Comment
1	StanfordCoreNLP (scnlp-client.xml)	Reads in the document text and outputs a variety of Annotations including NamedEntityMention annotations.
2	Test Element Annotator	Read in the simple input file and annotate question and answer spans.
3	Token Annotator (not required)	Annotate each token span in each question and answer.
4	Unigram Annotator (not required)	Annotate each unigram in each question and answer.
5	Bigram Annotator (not required)	Annotate each bigram in each question and answer.
6	Trigram Annotator (not required)	Annotate each trigram in each question and answer.
7	Named Entity Scorer	Scores each candidate answer based on the % of its named entity mentions that overlap with those of the question.
8	Top N Evaluator	Evaluate the precision for a set of AnswerScores by seeing how many of the top N answer scores correspond to answers that are factually correct. The value N is dependent on how many of the answers are indeed correct.

2.4 CAS Consumers

This section will describe the CAS Consumer components used in homework assignment #3.

2.4.1 TopNEvaluator (CAS Consumer)

The TopNEvaluator CAS Consumer acts identically to the TopNEvaluator Annotator. However, it extends CasConsumer_ImplBase, and is called by the CPE engine rather than the Document Analyzer. The TopNEvaluator utilizes the TopNEvaluator (Annotator) class' *evaluate()* method in order to calculate the average precision of the top N documents. N is determined by the number of true answers for each question.

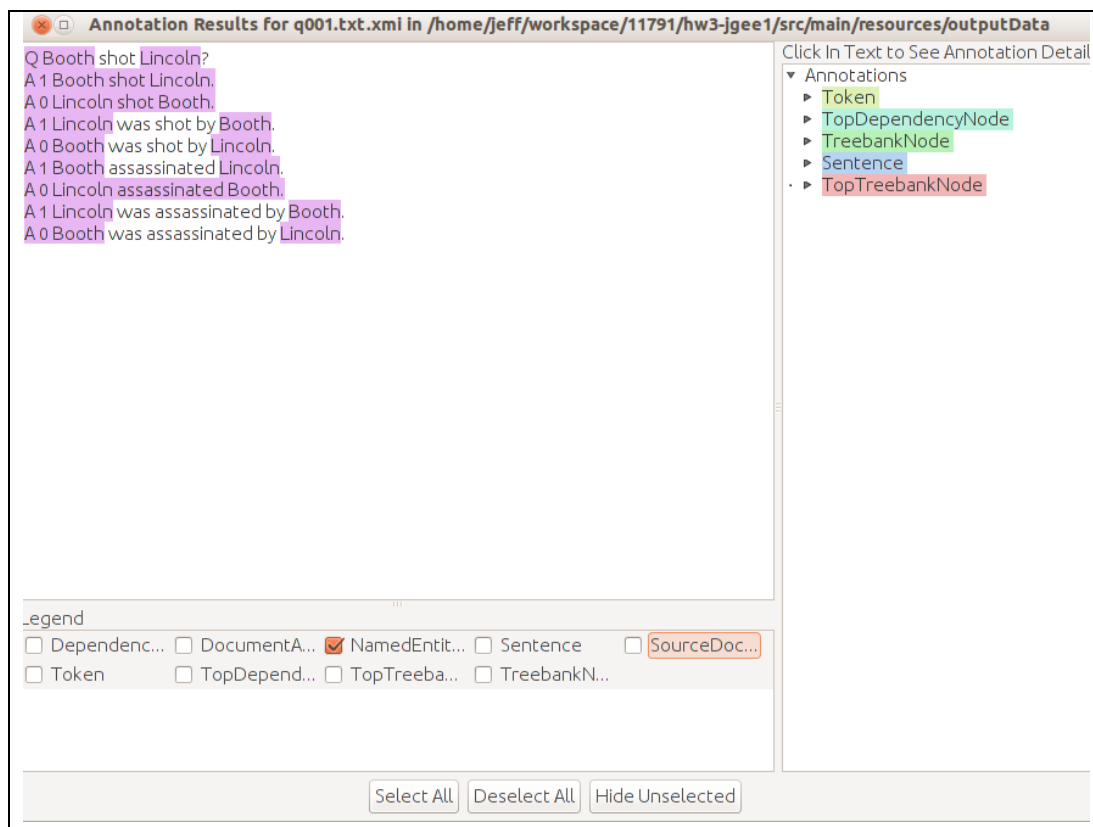
Implementing an evaluator as a CAS Consumer makes much more sense than implementing it as an Annotator due to the fact that it may output files or print to console. This does not always work when implemented as an Annotator as part of an AAE, as AAE's can be deployed remotely, preventing user access to the contents of the console or the remote file system.

2.5 HW3 vs. HW2 Aggregate Analysis Engine

We compared the Named Entity Scorer (HW3 AAE) vs. N-Gram Overlap Scorer (HW2 AAE). The Top N precision values are shown in the table below.

	q001.txt (Lincoln)	q002.txt (John & Mary)	Average
HW#2 AAE	0.5	0.67	0.58
HW#3 AAE	0.75	0.33	0.54

As can be seen from the results above, the Named Entity (Overlap) Scorer in HW#3 performed poorly in comparison to HW#2's N-Gram Overlap Scorer. This is primarily due to how the Stanford NLP Annotator recognizes the input files. As the Stanford NLP Annotator reads in the raw document text, it tokenizes characters that are not part of actual question and answer spans (Q/A, 1/0). These are then included when the annotator decides on which token collections are named entities. For example, in the sample below "Q Booth" is recognized as a named entity (as if the Q was an initial).



Annotation Results for q001.txt.xml in /home/jeff/workspace/11791/hw3-jgee1/src/main/resources/outputData

Click In Text to See Annotation Detail

Annotations

- Token
- TopDependencyNode
- TreebankNode
- Sentence
- TopTreebankNode

Legend

☐ Dependenc... ☐ DocumentA... ☒ NamedEntit... ☐ Sentence ☐ SourceDoc...

☐ Token ☐ TopDepend... ☐ TopTreeba... ☐ TreebankN...

Select All Deselect All Hide Unselected

This made it difficult for the Named Entity Scorer, as "Q Booth" does not actually lie within the boundaries of "Booth shot Lincoln" (the "Q"). Therefore the only named entity detected in within the question span is "Lincoln", which leads to a poor overlap between question and answers.

3 UIMA-AS vs. Local Annotator

3.1 Performance

The performance comparison between a remotely deployed UIMA Annotator and a locally deployed Annotator will be highly dependent on a variety of factors, each of which should be considered. In this section we will describe some of these factors and compare the performance of a remotely deployed UIMA-AS Stanford Core NLP Annotator (remote server) with a locally CPE deployed version.

3.1.1 Initialization Cost

The first factor that plays a role between the performance of an UIMA-AS analysis engine and a locally deployed analysis engine is the cost of initialization. Many analysis engines require loading dictionaries and various resources into memory. This overhead should be considered when running a local CPE deployment. UIMA-AS analysis engines are initialized once when they are deployed, whereas CPE's tend to be initialized each time they are run. For analysis engines that have significant overhead this can be costly, often outweighing the I/O or bandwidth costs of transferring data over a network.

3.1.2 Input File Size

The size of your input files should always be considered when utilizing UIMA-AS. If input documents are very large, then transferring the data to remote servers may cost time. For example, if input files amount to GB's of data, then the transfer cost of that data over the wire is likely to outweigh the initialization time.

3.1.3 Simple Comparison

In an experiment, the execution time of a CPE utilizing a remote deployment of a Stanford Core NLP Annotator was compared against that of a CPE utilizing a local deployment. The results are shown below.

	Stanford Core NLP (Remote)	Stanford Core NLP (Local)
Execution Times	1.074s	4.454s

As you can see from the times above, the remote execution was significantly faster. This is due to a combination of better hardware and the small size of our input files (negligible transfer cost). These times ignore the initialization cost of Stanford Core NLP, which runs about 15.5 seconds locally. If we were to add that to the time cost of the local deployment of Stanford Core NLP, then the time difference would be quite drastic (local deployment would be 20 times slower).