

# 11-791 HOMEWORK 2 REPORT

Siping Ji  
09/22/2013

## Tasks

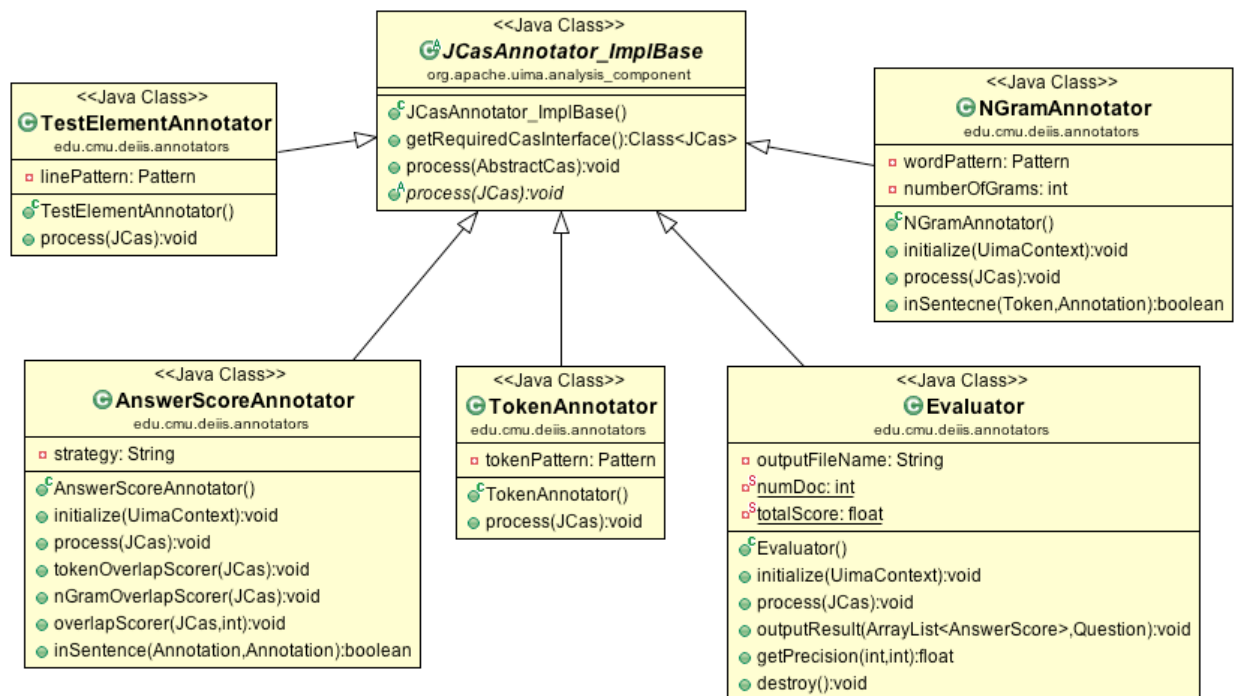
The tasks in this homework is to implement the logical architecture for the sample information processing task based on the type system we've designed in the last homework. The key to this homework is to learn how UIMA decouples different parts of an information system and how these different parts aggregates into a whole functioning part.

## Logic

We can divide the task to build an information system which can do basic answer scoring according to some strategy, to three major steps.

1. Lexical analysis. In this step, we need to do some lexical analysis to get lexical elements we need for further processing. More specifically for this homework, we need to annotates lexical elements like questions, answers, tokens, n-grams. The techniques involved in this step are mainly regular expression.
2. Scoring. In this step, we need to score answer using some strategies/algorithms. The lexical elements generated from last step will be the input for the scoring algorithms. In my implementation, I use the token overlap strategy and n-gram overlap strategy.
3. Evaluation. In this step, we need to rank answers according to scores assigned in last step. We further use some metrics like precision to evaluate the ranked answers.

## Analysis Engine classe

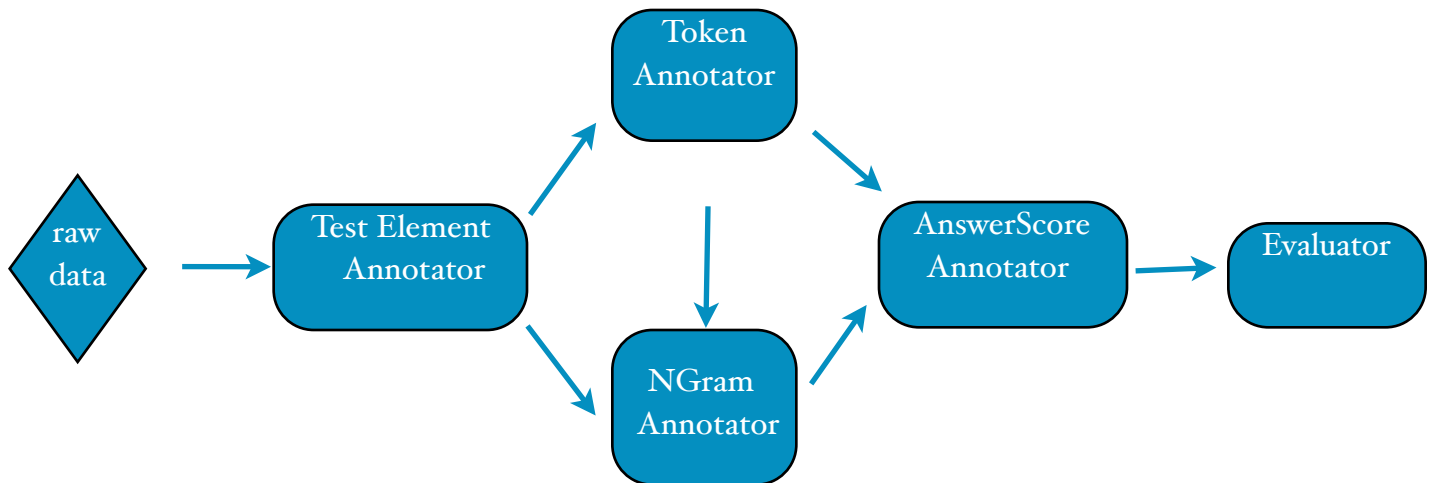


The figure above is the uml diagram for annotator classes i implemented for this assignment.

We can note that all these classes implements their own logic and are loosely coupled from each other (high cohesion, low coupling). This is one benefit that we got from using the UIMA framework.

The TestElementAnnotator, TokenAnnotator and NGramAnnotator classes are the building blocks for step 1 - lexical analysis. AnswerScoreAnnotator and Evaluator are for step 2 and 3 respectively. In the next section, I'll describe the implementation of these classes in further detail and how these separate parts aggregates into a whole information system.

## Analysis Engine Pipeline



The picture above shows the annotators and its how it is aggregated to form the whole system. The way to form this pipeline is to use an aggregate analysis engine under UIMA framework.

### **(1)Test Element Annotator**

Use regular expression to parse the question and answers, and then generate Question, Answer and Sentence annotations. Note that I add a sentence type to the type system, this is for the simplicity of the implementation of other analysis engine - to treat question and answer uniformly.

Input: raw document

output: question, answer, sentence annotations.

### **(2)Token Annotator**

Use regular expression to parse the sentences into tokens, and generate Token annotations. The input sentences are generated from TestElement Annotator.

Input: sentence

output: token

### **(3)NGram Annotator**

Generate n-grams( $n = 1, 2, 3$ ) given the tokens and sentences. Here  $n$  is a parameter set in the annotator descriptor, so we just need one N-Gram Annotator class, and multiple descriptor files to generate n-grams in different  $N$ s.

input: sentence, token

output: n-gram

#### (4) AnswerScore Annotator

Generate AnswerScore Annotations. The score is assigned using token overlap and n-gram overlap strategy described in class. In the implementation of this strategy, I generalize the implementation of the two strategies into one overlap scorer method, i.e. user can decide which strategy to use by passing different value of the parameter 'strategy' in the descriptor file - 'answerScoreAnnotator'.

Input: NGram, Token, Question, Answer

Output: AnswerScore

#### (5) Evaluator

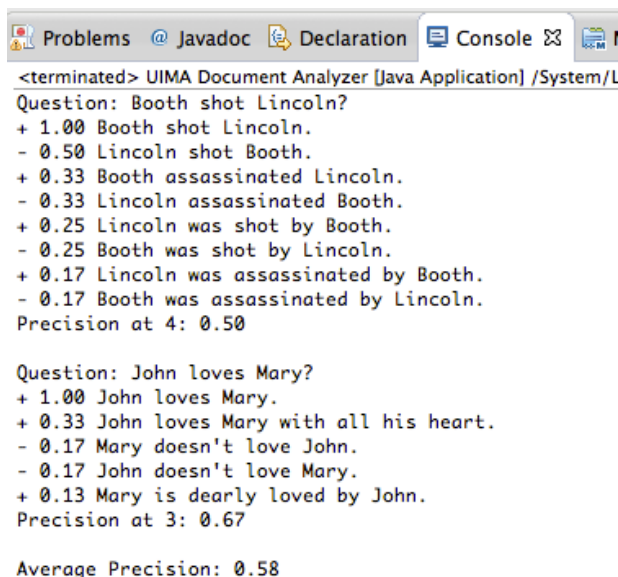
Rank the answers according to score assigned by the AnswerScore Annotator. Then evaluate precision of each strategy, output the result and to file or to the command line.

Input: AnswerScore

Output: file or command-line

## Output

Following is the output of the information system using N-gram overlap scorer.



```
<terminated> UIMA Document Analyzer [Java Application] /System/L
Question: Booth shot Lincoln?
+ 1.00 Booth shot Lincoln.
- 0.50 Lincoln shot Booth.
+ 0.33 Booth assassinated Lincoln.
- 0.33 Lincoln assassinated Booth.
+ 0.25 Lincoln was shot by Booth.
- 0.25 Booth was shot by Lincoln.
+ 0.17 Lincoln was assassinated by Booth.
- 0.17 Booth was assassinated by Lincoln.
Precision at 4: 0.50

Question: John loves Mary?
+ 1.00 John loves Mary.
+ 0.33 John loves Mary with all his heart.
- 0.17 Mary doesn't love John.
- 0.17 John doesn't love Mary.
+ 0.13 Mary is dearly loved by John.
Precision at 3: 0.67

Average Precision: 0.58
```

## Conclusion

Different annotators (analysis engine) are highly cohesive and are nicely decoupled in UIMA framework, we can use a simple descriptor file (aggregate analysis engine) to combine these small building blocks to realize complicated IIS logic. Further, these small building blocks can be reused in different scenarios by using different aggregate analysis engine without re-compilation. Thus it is easy for us to use aggregate analysis engine to try out different algorithms and ideas to form a robust information system.