11-791 Design and Engineering of Intelligence Information System Homework 2

Logical Architecture and UIMA Analysis Engines Design & Implementation

Lab Report

Andrew id: xchu
email: xchu@cs.cmu.edu

1 General Description

In this lab, I created analysis engines based on UIMA SDK and wrote Java annotators based on the annotation which the maven project have provided us. First of all, I wrote the Java annotators and then created analysis engines for each annotator. In order to achieve the whole pipeline, I finally set up an aggregate analysis annotator which include other descriptors and worked in a fix flow. For this homework, I used two methods to compute the answer score. First is token overlap which will calculate how many question tokens are in answer tokens. The second one is Ngram overlap which concerns about how many Ngrams(1,2,3-grams) are in answer Ngrams. Finally, the system will output the score for each answer and the precision score for the documents.

2 Design Pattern

Based on the first laboratory, in this homework, we need to design the whole pipeline in order to get the annotation and answer score for each answer. Thus for the whole system, it includes 3 parts.

(1) Lexical Analysis

This part of system I mainly do the lexical analysis for the input document. Based on the annotation type which the originally project has provided, the lexical analysis mainly include these elements: question, answer, token, Ngrams(1,2,3-grams).

(2) Score

The function of this part of system lies in giving each answer a score based on some language model strategy. For instance, we can use Ngrams overlap algorithm to annotate the score for each answer. For homework 2, I use both token overlap and Ngrams overlap to calculate the answer score.

(3) Evaluation

After we get the score for each input answer, then this final part of system will use these scores to sort answers and get calculate the final precision score.

3 Java Annotators

In this chapter, I will describe the annotators in details.

(1) TestElement Annotator

input: raw document
output: question annotation, answer annotation

In this annotator, we use regular expression to parse the question and answer. The regular expression to parse question is “Q\s(.*\r)\n” and the expression for

answer is “A\s([01])\s(.*\r)\n”. These parameters are set in the Analysis Engine Descriptors which is designed for test element annotators. During the initialization of the annotator, it will get the value of these parameters from the uima context. After the regular pattern find the question and answers, it will add them into question index and answer index individually. Besides, the answer annotation will have one more feature, that is iscorrect which indicates whether the answer is right or wrong.

(2) Token Annotator

input: question and answer annotation
output: token annotation

This annotator is similar to test element annotation which will parse each token in the document. The reguar expression to parse each token will be “\w+” and the annotators will get the value of these parameters from the umia context. Different from the testelement annotators, we don't need to get the raw document in order to acquire the tokens. As we have created the question and answer annotation index, we can extract the token based on them.

(3) NGram Annotator

input : question, answer and token annotation
output : NGram annotation

This annotator will generate the Ngrams(1,2,3-grams) for the document. In this annotator, we no longer need regular expression in order to get Ngrams. Based on the token annotation, we can gather the whole tokens for each single question or answer. After we get the tokens, we can assemble the Ngrams based on the position of each token. The parameter n will be set in the analysis engine descriptor.

(4) Token Overlap Score Annotator

input : question, answer and token annotation
output : answer score annotation

In this annotator, we will assign score for each answer based on the token overlap algorithms. We first get the tokens for the question and each answer. After we get this, we then calculate how many question tokens are contained in each answer and divided it by the total number of tokens for each answer.

(5) NGram Overlap Score Annotator

input : question, answer and ngram annotation
output : answer score annotation

This annotators are somewhat similar to the token overlap annotators but will use

the ngrams to get the score for each answer. Also, we should first get the ngrams for question and each answer. Here the ngrams contains 1,2,3 grams. Then we count the number of question ngrams in each answer and divided by the total number of ngrams in each answer.

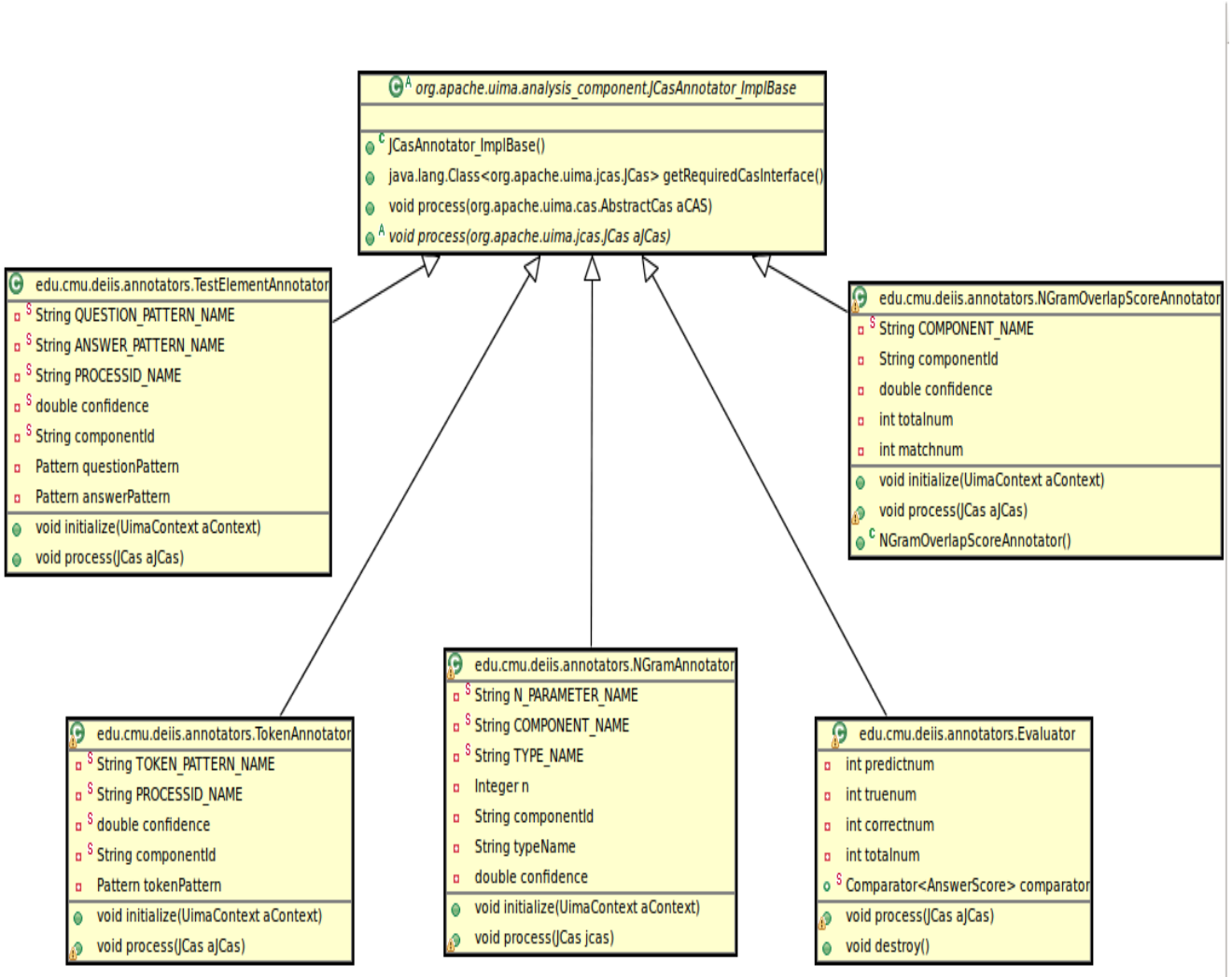
(6) Evaluator

input : question annotation and answer score annotation
output : precision score for each document and average precision score

This is the final part of the whole pipleline. First it will sort the answers based on the score of each answser. After that, it will count the number of correctly predicted answers and the total number of correct answers. Finally, it will ouput the precision score and average precision score. Besides, the method for calculating the average precision score will be set in the analysis engine destroy method.

4 UML

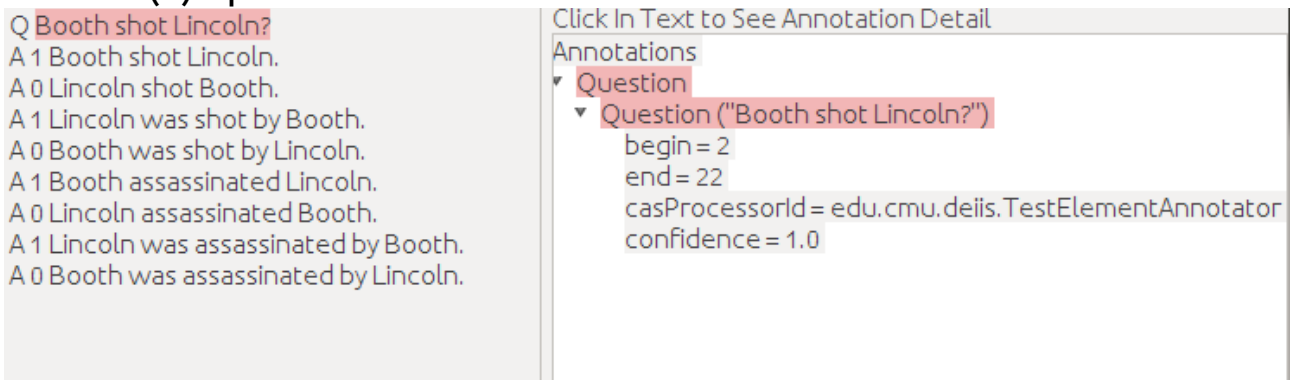
The UML for the whole system is suggested as below:



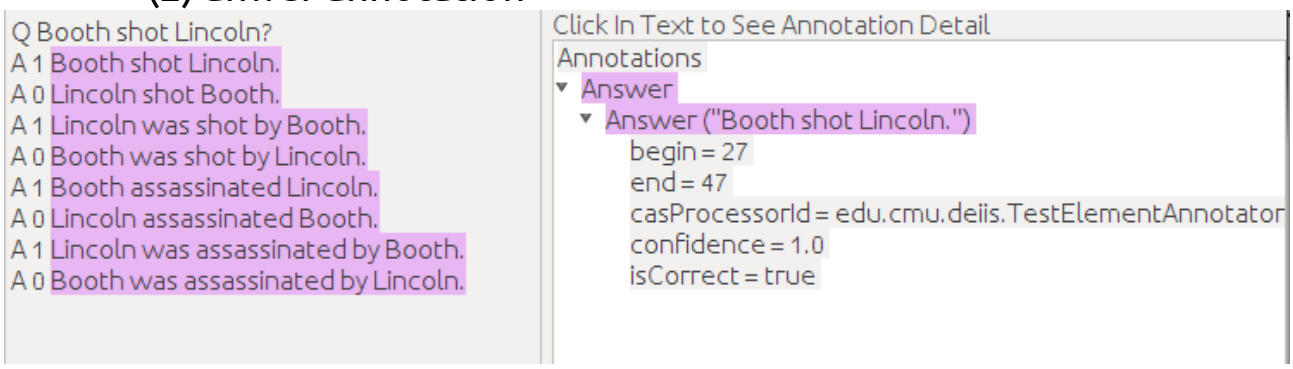
5 Result

First we will see the annotation results

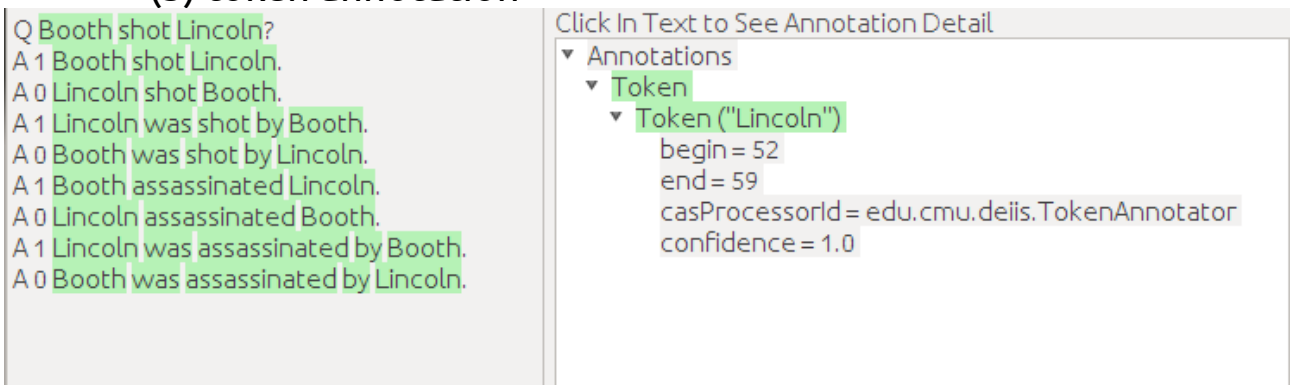
(1) question annotation



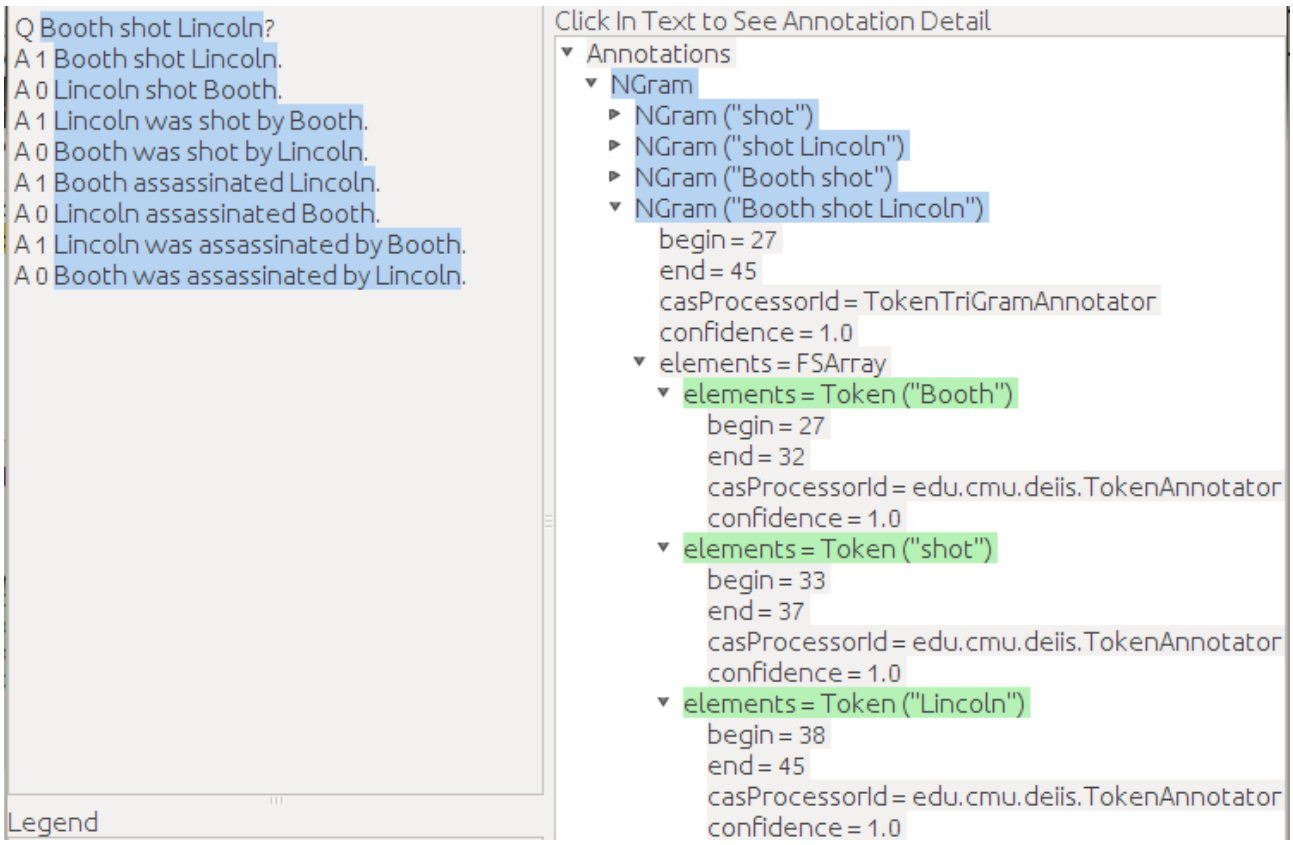
(2) anwer annotation



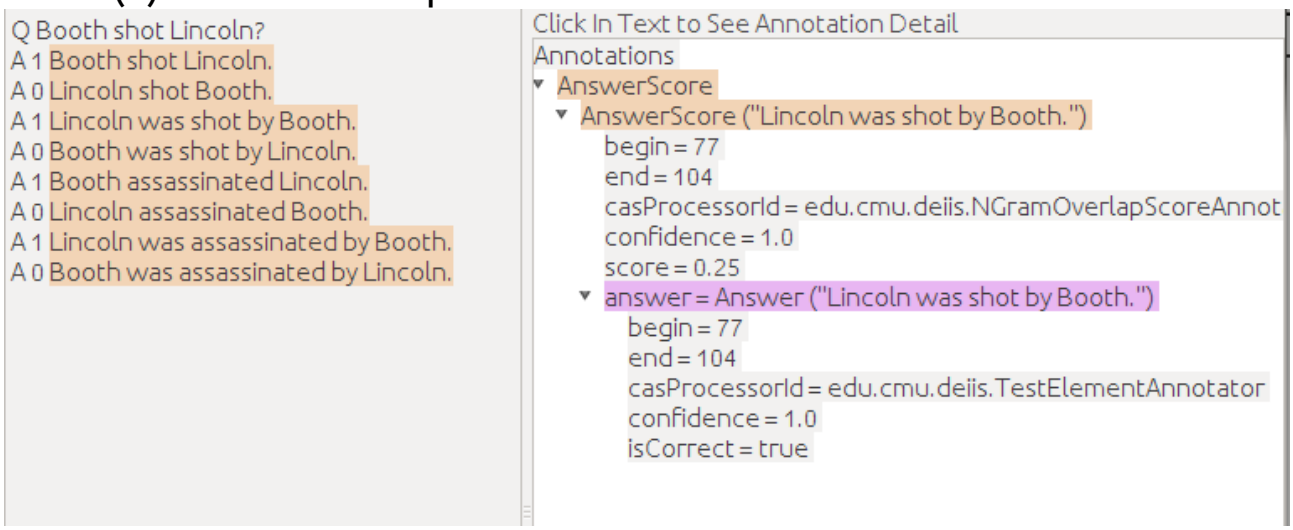
(3) token annotation



(4) NGram annotation



(5) NGram Overlap Score Annotation



Then we can get the precision score result:

Precision score for ngram overlap score annotation:

```
Question: Booth shot Lincoln?
+ 1.0 Booth shot Lincoln.
- 0.5 Lincoln shot Booth.
+ 0.3333333333333333 Booth assassinated Lincoln.
- 0.3333333333333333 Lincoln assassinated Booth.
+ 0.25 Lincoln was shot by Booth.
- 0.25 Booth was shot by Lincoln.
+ 0.16666666666666666 Lincoln was assassinated by Booth.
- 0.16666666666666666 Booth was assassinated by Lincoln.
Precision at 4: 0.5
Question: John loves Mary?
+ 1.0 John loves Mary.
+ 0.3333333333333333 John loves Mary with all his heart.
- 0.16666666666666666 Mary doesn't love John.
- 0.16666666666666666 John doesn't love Mary.
+ 0.13333333333333333 Mary is dearly loved by John.
Precision at 3: 0.6666666666666666
Average Precision: 0.5714285714285714
```

6 conclusion

From this homework, we have build up a small question answering pipeline. It is very convenient for us to design analysis engine descriptor for each annotator and combine them cohesively in an aggregate descriptor. Also, in this homework, I have tried two score algorithms: token overlap score and ngram overlap score. Actually we can think of token overlap as 1-gram overlap algorithm. By contrast, the ngram overlap algorithm have better precision score for the document sets.