**Napat Luevisadpaibul ID:nluevisa**

# Homework 2

## 1 Type System Design

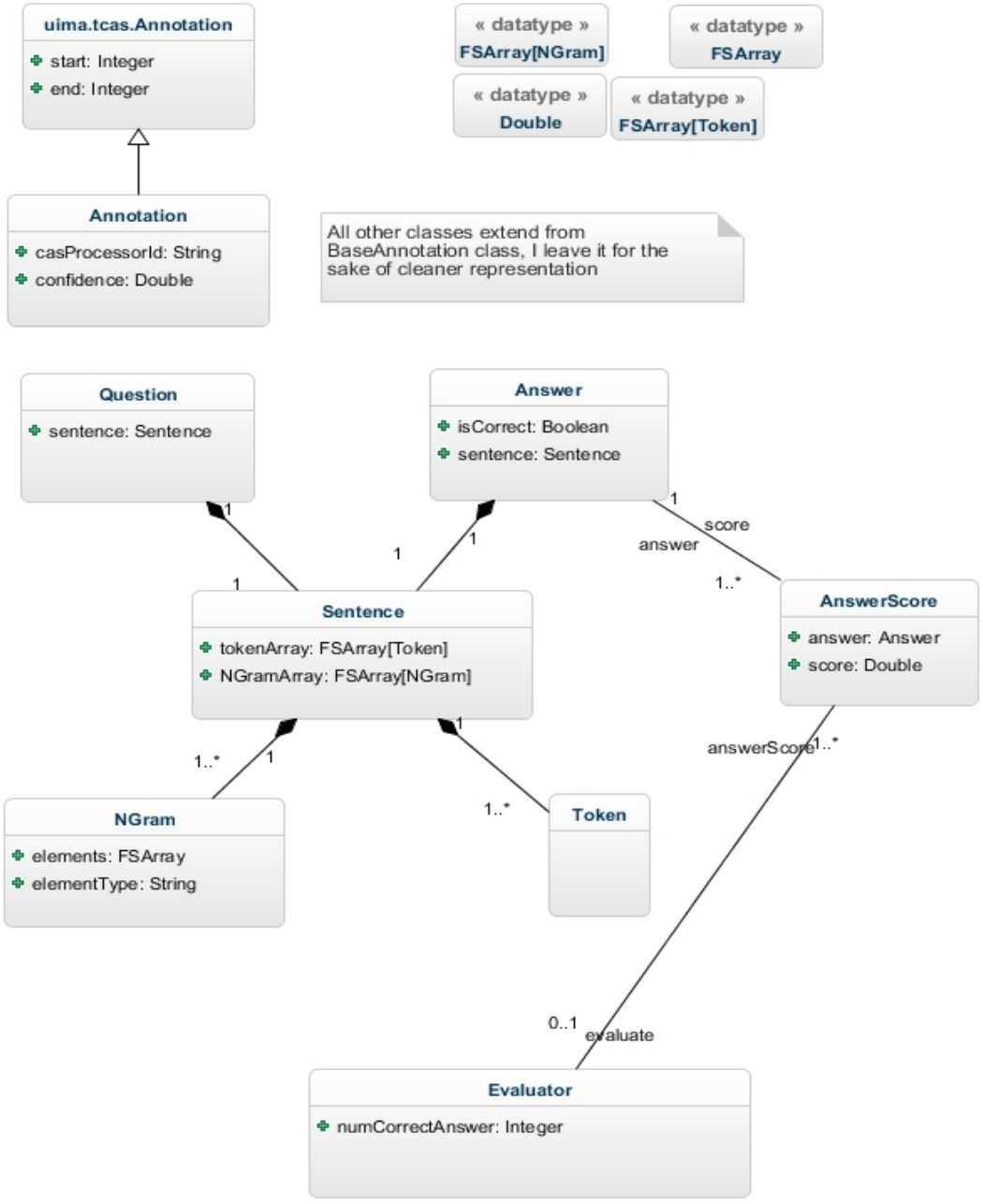



Figure 1 Type System UML Class Diagram

All of my defined types extend from base type “Annotation” which extends from “uima.tcas.Annotation”. This base type “Annotation” contains properties (start, end, confidence casProcessorId ) which are common to all annotations in our Question and Answering system. Start and end need to keep track of position of annotated text in document. Confidence is used to indicate the confidence score produced from annotator and casProcessorId is needed to keep track of the class that produce annotation.

“Question” and “Answer” types both contain “Sentence”. “Sentence” consist of array of “Token” and array of “NGram”. I design it this way because we can pass “Question” and “Answer” to our scoring module, then we can easily get either NGram or Tokens that constitute Question/Answer, necessary to compute score, by just call question/answer.getSentence().getNGramArray/getTokenArray. Moreover “Answer” has feature ‘isCorrect’ which indicate if it is a correct or wrong answer.

“AnswerScore” holds score value of an answer and the reference to that answer. Additionally each “AnswerScore” may have different casProcessorId, which indicates the method used to calculate this score. “Evaluator” is used to keep track of number of correct answer in gold standard answer and may hold additional information after evaluation if we want to extend our system such as write the evaluation result to a file or database

## 2 Annotator Class Design

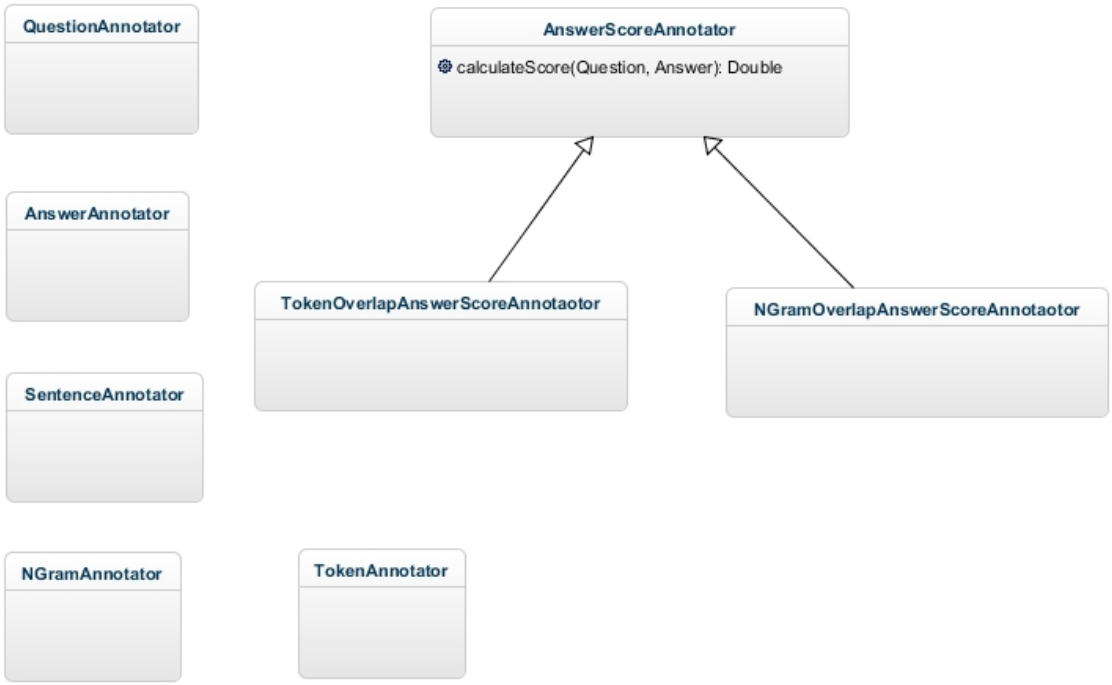



Figure 2 Annotator UML Class Diagram

Basically each annotator will annotate the span of text of its output types and set some features, which I will elaborate more in the next section. The most notable thing of this diagram is to show that "AnswerScoreAnnotator" is an **abstract class**. The only abstract method it has is calculateScore. "AnswerScoreAnnotator" has implementation that annotated the answer associated with it, set casProcessorId and set the score to value return by calculateScore method. This means we can have as many methods to calculate score as we want by just simply create a new subclass that extends "AnswerScoreAnnotator", then implement only one method "calculateScore". This uses concept of **program to interface** and **loosely coupling.** The class we newly add will not affect the code in other classes at all so we can provide more capability to our system really easily. In current system, I create two scoring methods, NGram overlapping and Token overlapping to illustrate this point.

## 3 Aggregate Analysis Engine Design

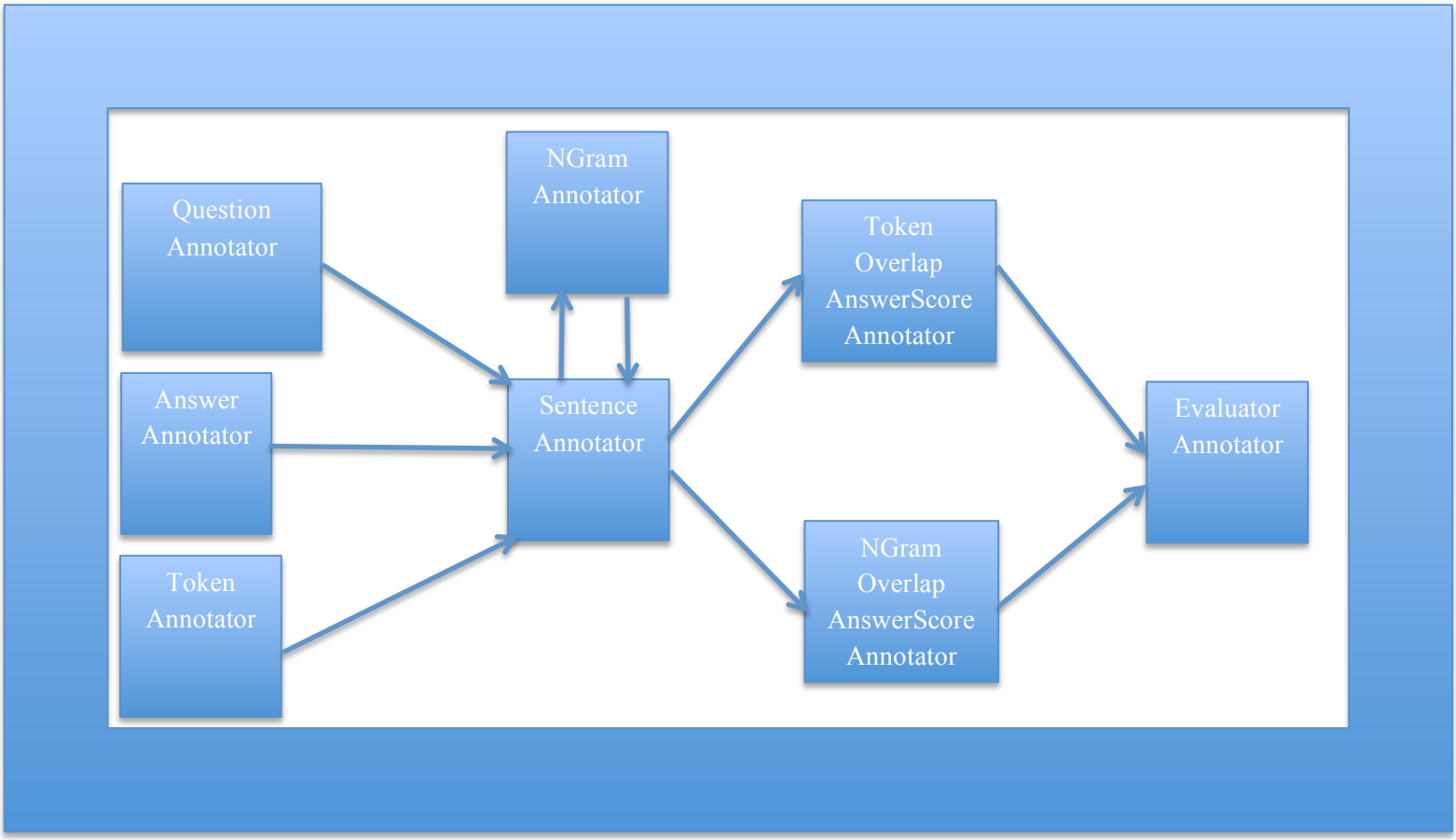



**Figure 3 hw2-nluevisa-aae Aggregate analysis engine work flow**

First QuestionAnnotator will annotate the question in the input document text and produces output “Question”. AnswerAnnotator annotates answer section, set feature isCorrect (this is done by using regular expression and little bit of string operation) and produces output “Answer”. TokenAnnotator tokenizes all the text in the document and produce output “Token”. Sentence Annotator will get the information of “Question”, “Answer” and “Token” to create TokenArray (also discard ‘.’ And ‘?’), put it in its output “Sentence” and associate it with Question and Answer (we can do this just by looking at the span position). NGram will get the sentences in CAS and produce Unigram, Bigram and Trigram(we determine which one is Unigram,Bigram or Trigram by elements.size()) then form NGramArray and add it to Sentence that it produce NGram from.

From the pipeline I describe above, we have enough information to calculate score for the answer. Both TokenOverlapAnswerScoreAnnotator and NGramOverlapAnswerScoreAnnotator get Question and Answer as input (to be more precise what we want is Sentence in Question and Answer) and produce output “AnswerScore”. TokenOverlapAnswerScoreAnnotator calculate score from

Number of token overlap in question and answer/ number of token in the answer

NGramOverlapAnswerScoreAnnotator calculate score from

(Number of NGram overlap in question and answer * weight ) / (number of NGram in the answer * weight)

Weight for Unigram is 1, Bigram is 2 and Trigram is 3. That means this method prefer the answer that have more Trigram or Bigram matching over just Token(Unigram) match.

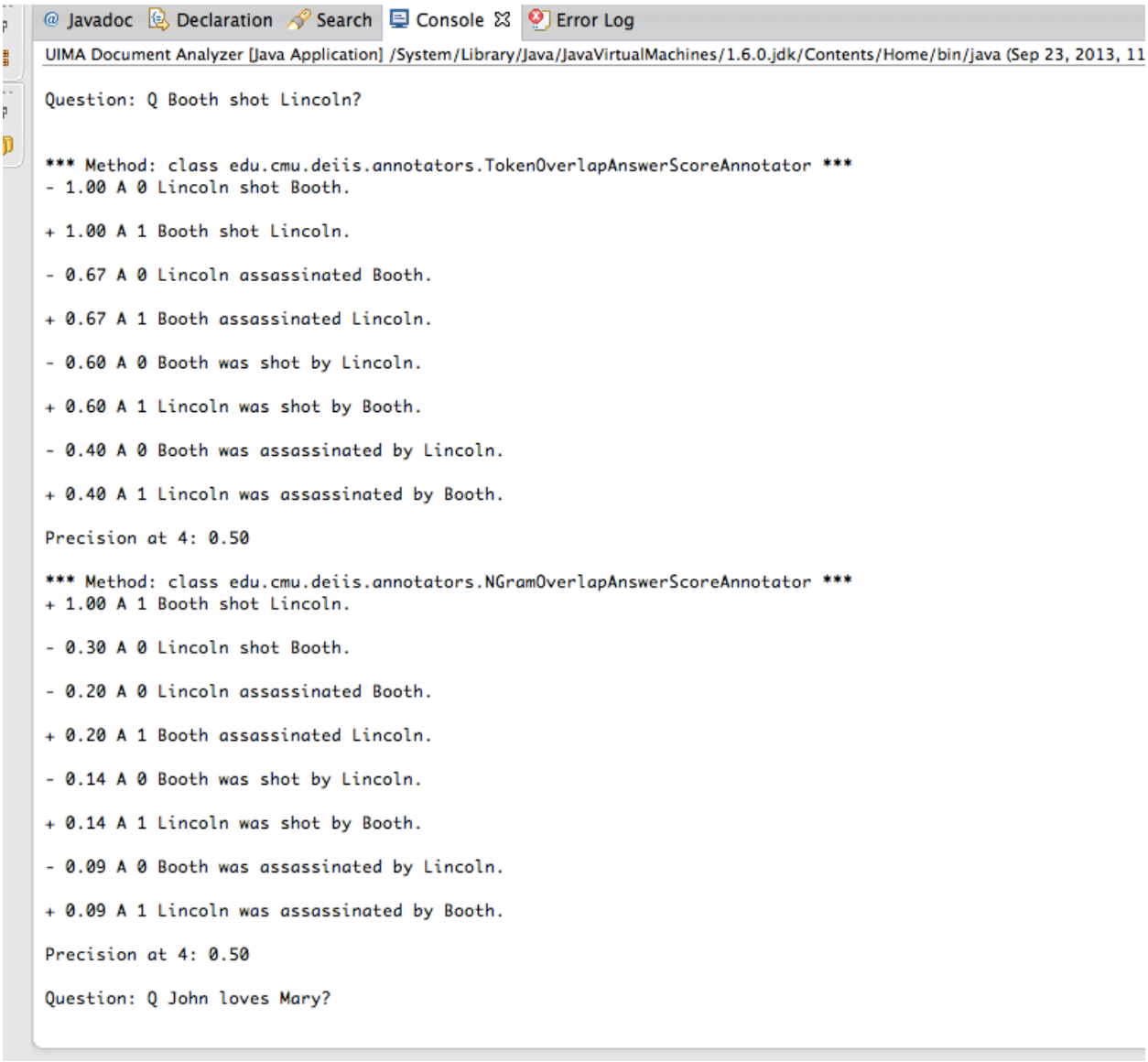
Finally EvaluatorAnnotator will grab all AnswerScore produces by all methods and put them in its internal hashmap. Then EvaluatorAnnotator will separate AnswerScore by the method producing it(ie look at answerScore.getCasprocessorId()). Then arrange score with in the same method to select top N answers where N is number of correct answers for the question. Finally it will print result of on the screen with show the ranked score and precision of each methods (You will see the example result in the next section).

The important thing is I don’t want to change my EvaluatorAnnotator code after I add some more calculating class so I use HashMap, which map the class name to AnswerScore produce by that class. This way I can always print and compare results of every score calculating methods in our pipeline without affecting the code of this class (You can see the code).

## 4 Running Program Result

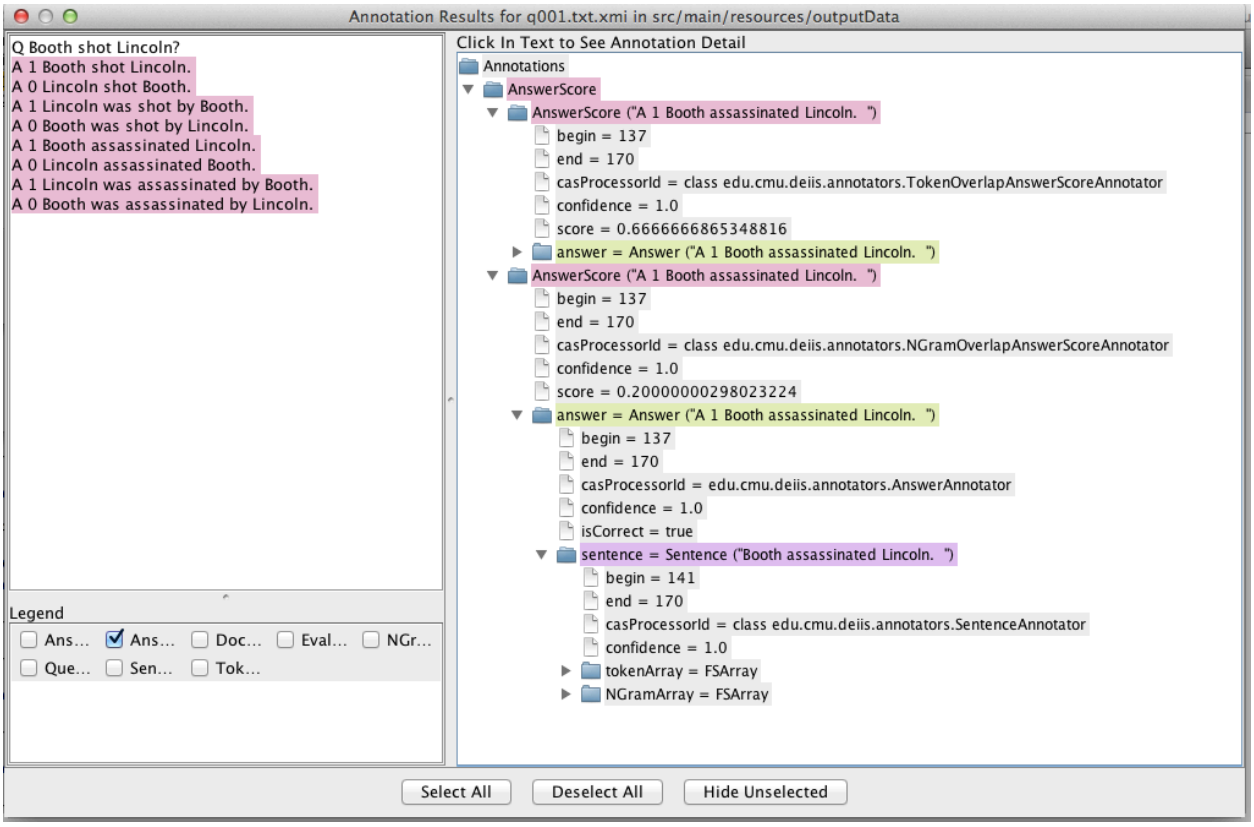
```
Question: Q John loves Mary?


*** Method: class edu.cmu.deiis.annotators.TokenOverlapAnswerScoreAnnotator ***
+ 1.00 A 1 John loves Mary.

- 0.50 A 0 John doesn't love Mary.

- 0.50 A 0 Mary doesn't love John.

+ 0.43 A 1 John loves Mary with all his heart.

+ 0.33 A 1 Mary is dearly loved by John.

Precision at 3: 0.33

*** Method: class edu.cmu.deiis.annotators.NGramOverlapAnswerScoreAnnotator ***
+ 1.00 A 1 John loves Mary.

+ 0.29 A 1 John loves Mary with all his heart.

- 0.13 A 0 John doesn't love Mary.

- 0.13 A 0 Mary doesn't love John.

+ 0.07 A 1 Mary is dearly loved by John.

Precision at 3: 0.67
```

This result on screen will show the question, class to compute score, answer with score ant + to indicate that it's correct or – if wrong and precision at top N answer. From this example NGram method is more precise

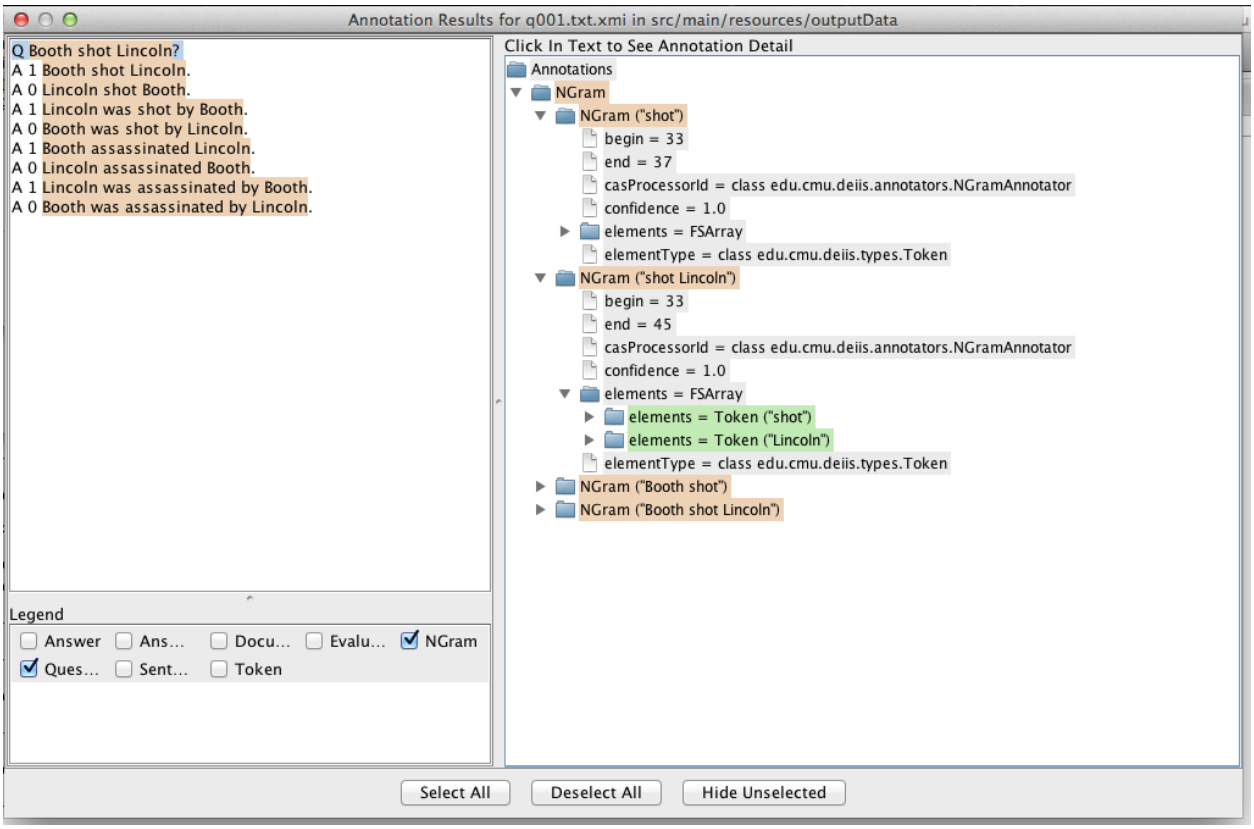



This example show that the precision of both method are about the same; however NGram tend to give lower score to incorrect answer (no tie score in the first rank as in Token overlap method)

From the visual debugger we can see this result




This is an example of AnswerScore annotation. You can see that in the same sentence “Booth assassinated Lincoln”, there are two AnswerScore which have different casProcessorId and get different score.




This is an example of NGram Annotation

## 5 Possible Improvement

I want to add more scoring method to the system but don't have enough time and I also plan to improve the capability of evaluator to show result in different ways than on the screen (ie xmi or JFrame).