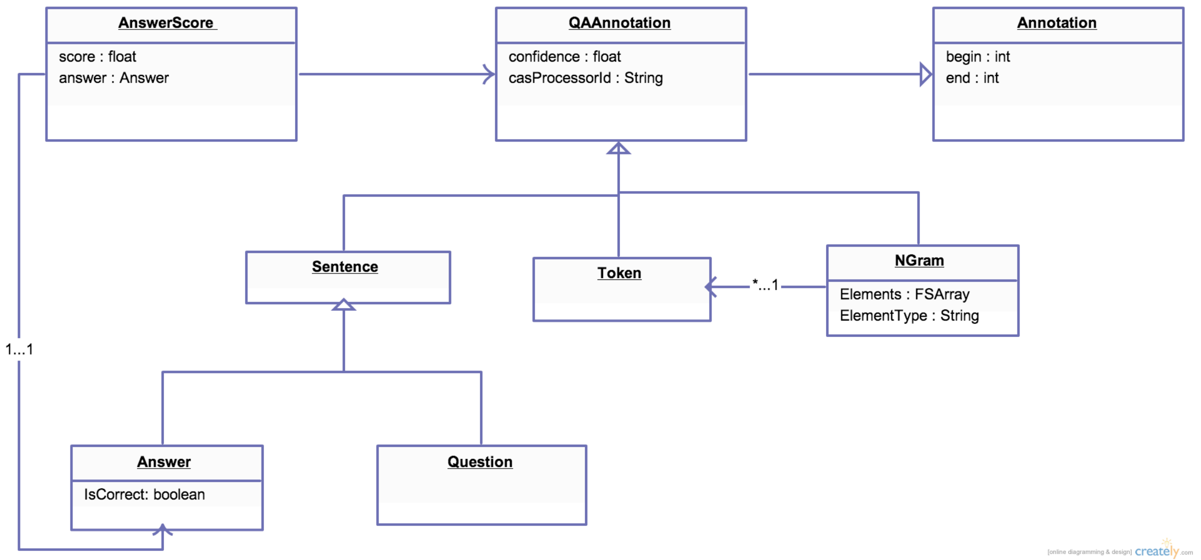
11-791 HOMEWORK 1 REPORT

Siping Ji
09/10/2013

Type System Design

The following is the UML Diagram of the designed type system.



(1) QAAnnotation: Base class for all other annotation classes. It inherits the Annotation class provided in UIMA. As all the annotations requires the a field that describes the confidence that the annotation is generated, and a field that refers to the component that generates the annotation, I put these two fields in this base class, so that all the inherited classes share these two features.

(2) Sentence: Sentence class directly inherits the QAAnnotation class and contains no other field. It represents the span in the user input that is a sentence. For example, "A 1 John loves Mary." The highlighted span is a sentence. This class is defined so that to simplify the processing of token annotator and n-gram annotator, as they can treat Answer and Question in a uniformed way.

(3) Question: Question class directly inherits Sentence class and contains no additional features. It represents a question in the user input.

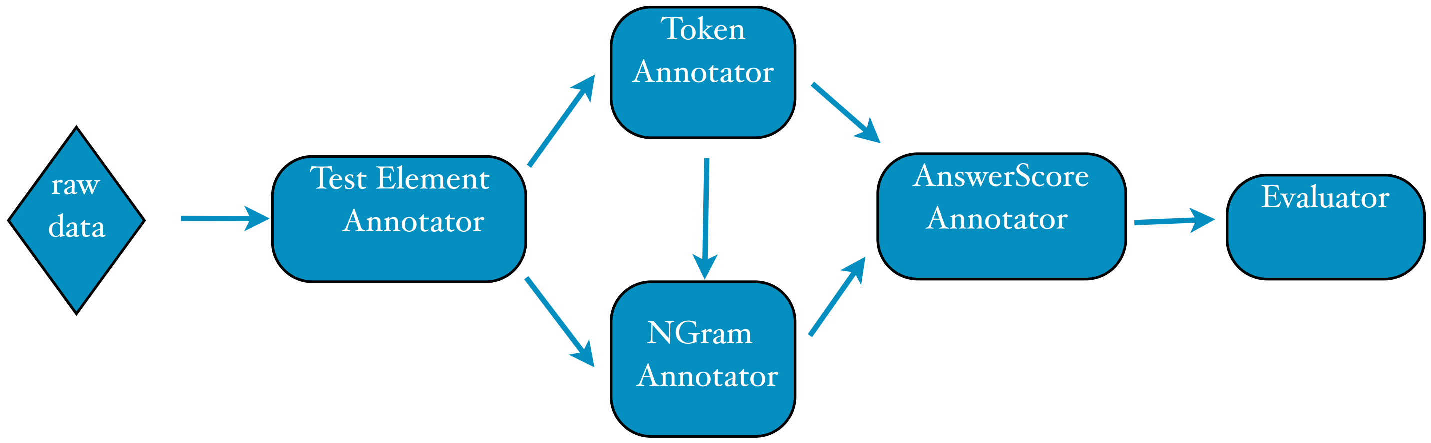
(4) Answer: Answer class directly inherits Sentence class and contains one additional feature - IsCorrect to represent whether the answer is correct to the question provided in the user input.

(5) AnswerScore: This class represents the AnswerScore annotation, it contains two features in addition to the QAAnnotation Class. Score feature refers to the score assigned by the system, the higher the score the more probable the system believes it is a correct answer. Answer field is a reference to the one instance of the answer class, which represents the answer being scored.

(6) Token: This class represents the token annotation, it only needs the begin and end feature so it has no additional feature to the QAAnnotation class.

(7) NGram: This class refers to the n-gram annotation. It consists two features, one is the elements feature, which is a FSArray of n tokens within this n-gram. Second feature is elementType, which is simply a string that represents the element type of elements array.

Analysis Engine Pipeline



The picture above shows the annotators and its how it is aggregated to form the whole system.

(1) Test Element Annotator: Use regular expression to parse the question and answers, and then generate Question, Answer and Sentence annotations.

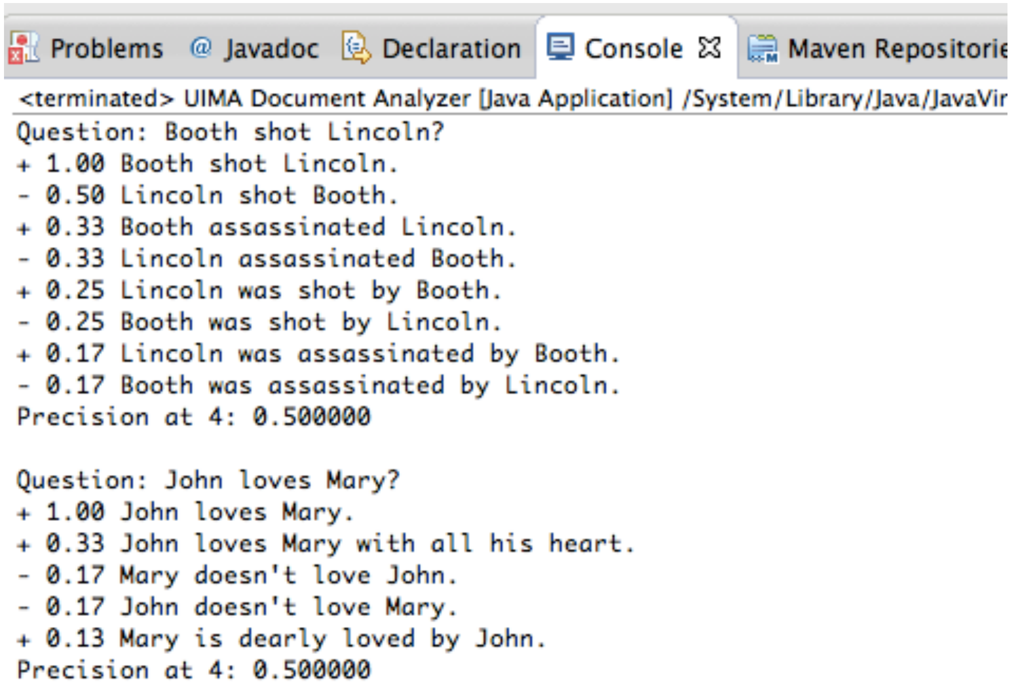
(2) Token Annotator: Use regular expression to parse the sentences into tokens, and generate Token annotations. The input sentences are provided from TestElement Annotator.

(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 Ns.

(4) AnswerScore Annotator: Generate AnswerScore Annotations. The score is assigned using token overlap and n-gram overlap strategy described in class.

(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.

Output



```
<terminated> UIMA Document Analyzer [Java Application] /System/Library/Java/JavaVir
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: 0.500000

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 4: 0.500000
```

Conclusions

1. It is easy to define type system using UIMA framework as it generates all the code for us.
2. Different annotator(analysis engine) are nicely decoupled in UIMA framework, we can use aggregate analysis engine to combine these small building blocks to realize complicated IIS logic. Further, these small building blocks can be further reused in different scenarios. We can also use aggregate analysis engine to form different pipelines to try out different ideas.