# 11-791 Design and Engineering of Intelligent Information System

Homework 2 Report

*Logical Data Model and UIMA Type System Design & Implementation*

Name : Da Teng

AndrewID: dateng

Date: 09/23/2013

# Part 1. Type System Design

## 1.1 Logical Data Model

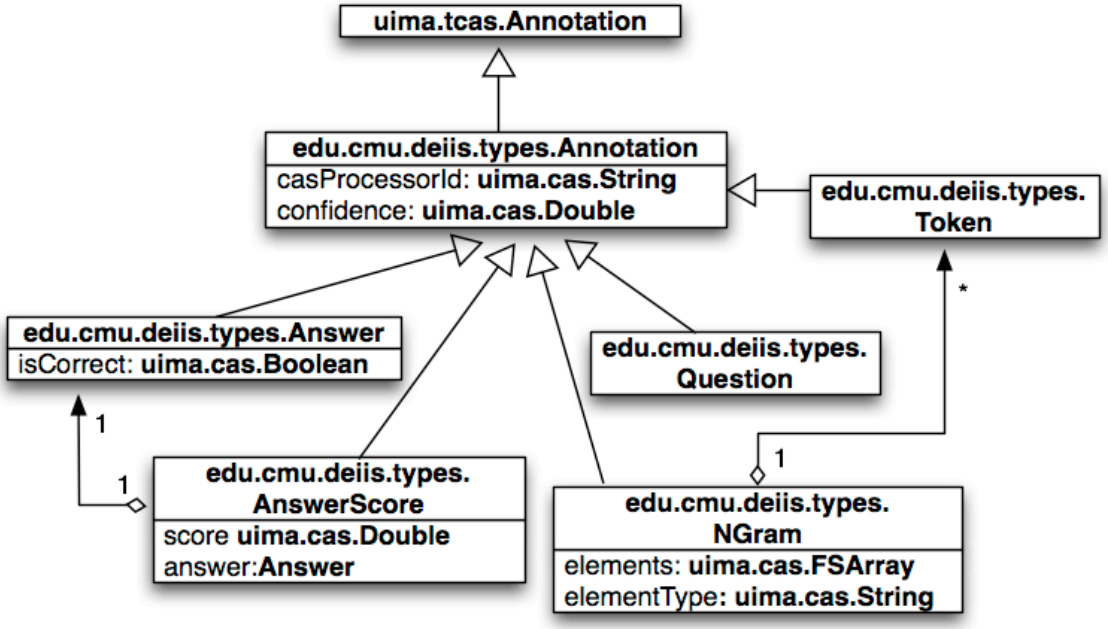



Figure 1.1: System logical data model

According to the requirement of the sample information processing task, five types is designed to complete the task, namely ***Annotation***, ***Answer***, ***Question***, ***Token***, ***AnswerScore*** and ***NGram***. The logical data model of these types are shown in Figure 1.1.

**Annotation:** is a base annotation type for the system. It inherit from the built-in uima.tcas.Annotation type to keep track of the span of text to which an annotation refers. It declares two features. The producedBy feature records the name of the component that produced the annotation. The confidence feature records the confidence score assigned to the annotation by the component. All other types in the system inherit from this base annotation type, so that they all have the features stated above.

- **Question**: is a type for recording the input question sentences.

- **Answer**: is a type for recording the input answer sentences. The isCorrect feature of it is used to record whether the sentence is a correct answer for the question. Since an answer can only have values 0 and 1(either correct or incorrect), the isCorrect feature is defined as a Boolean type.

- **Token**: is a type for representing a token in a sentence. When an answer

sentence is entered, it will be parsed into tokens. Then these tokens will be used in NGram analyzing to get an estimated score for whether or not this sentence is a correct answer.

- **NGram**: is a type for representing N contiguous words in a sentence, including unigram, bigram and trigram. It has a feature tokens of FSArray type, which is used to store the N contiguous words. N-grams found in an answer sentence will be used to match with the N-Grams in the Question and generate a score for the answer.

- **AnswerScore**: is a type for recording the score for a certain input answer sentence. The score feature is a score from the N-Gram matching method, ranging from 0 to 1. The answer feature is the corresponding answer sentence of the score.

## 1.2 Sentence scoring method

Given an answer sentence, system is going to use following N-Gram matching method in generating a score for it.

$$\text{answer score} = \frac{\text{number of N-Grams found in question sentence}}{\text{total number of N-Grams in the answer sentence}}$$

## 1.3 Information Processing Pipeline

The system will go through following steps for an information processing task:

1. **Test Element Annotation**: System reads question and answer sentences from an input file and generate corresponding Question and Answer annotation. The isCorrect feature of an Answer annotation records whether or not the answer is correct.

2. **Token Annotation**: System creates a Token annotation for each token in question and answer sentences.

3. **NGram Annotation**: System creates N-Gram annotations using contiguous tokens.

4. **Answer Scoring**: System uses N-Gram matching method to generate a score for each answer sentence and creates an AnswerScore annotation for the answer-score pair.

5. **Evaluation**: System sorts answer sentences according to their scores and calculate how many of the top N sentences are correct.

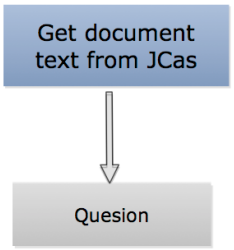
# Part 2. System Implement

## 2.1 System Annotator

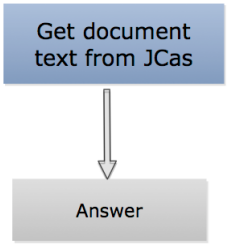
For those annotations that we want UIMA to generate when analyzing the documents, we need to design annotators to tell UIMA framework how to generate them. In this homework, we will need to create objects of Answer class, Question class, AnswerScore class, NGram class and Token class. Therefore, I designed a annotator for each of theses classes, shown as follows:

| Annotator | Output Annotation | Input Annotation |
|---|---|---|
| QuestionAnnotator | Question | - |
| AnswerAnnotator | Answer | - |
| TokenAnnotator | Token | - |
| NGramAnnotator | NGram | Answer, Qustion, Token |
| AnswerScoreAnnotator | AnswerScore | Answer, Qustion, Token |

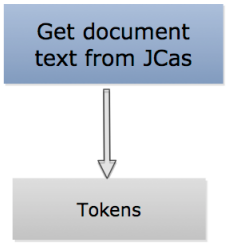
- **QuestionAnnotator**: takes only the JCas class as input. It creates a Question annotation that annotates the question sentence in the document.



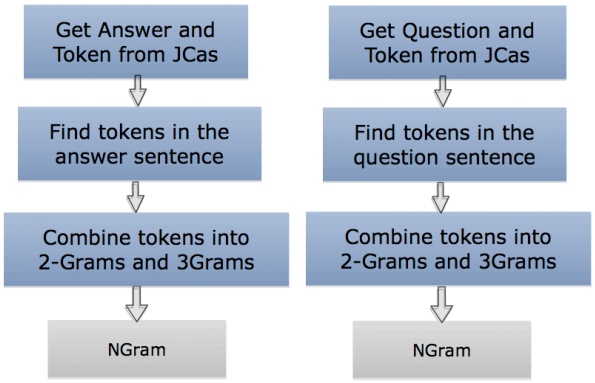

- **AnswerAnnotator**: takes only the JCas class as input. It creates an Answer annotation for each of the answer sentence in the document.



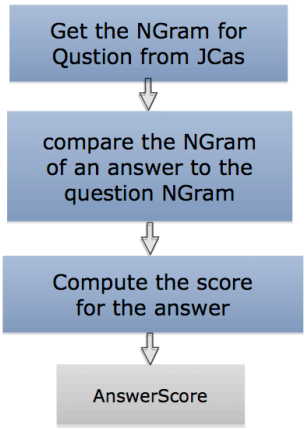

- **TokenAnnotator**: takes only the JCas class as input. It creates a Token annotation for each word in the document.



- **NGramAnnotator**: takes Answer, Question, Token as input. It creates an NGram annotation that containing the 1-Grams, 2-Grams and 3-Grams for each of the answer and question sentence in the document. It gets Answer annotations from the JCas. For each of the answer annotation, it finds all the tokens in the answer sentence, then expands these tokens into 2-Grams and 3-Grams, and creates an NGram annotation using the 1-Gram, 2-Gram and 3-Gram it found. This annotator does the same thing to generate an NGram annotation for the Question annotation in JCas. Since this annotator require Answer, Question, Token as input, it must be process after Answer, Question, Token annotations are generated, we can specify this processing order in the aggregate analysis engine descriptor (which will be discussed later).




- AnswerScoreAnnotator: takes Answer, Question, Token as input. It creates an AnswerScore annotation containing an Answer annotation and its corresponding score for each of the answer sentence in the document. This annotator will first get the NGram of the question from JCas. Then compare each answer NGram with the question NGram and generate a score for the answer using NGram overlap method. The score and its corresponding answer is store in an AnswerScore annotation.



## 2.2 System Descriptor

To run the annotator described above, we need to provide a descriptor file written in xml file to the UIMA analysis engine as a guideline for its execution. This file is called analysis engine descriptor in the UIMA system. If we only want to run a single annotator, we just need to define a primitive analysis engine descriptor. If we want to run multiple annotators at a time, we will need to define an aggregate analysis engine descriptor, which aggregates all the descriptors we assign to it and run multiple annotators in a specified order. I defined following descriptors for my sytem:

| Descriptor Name | Engine Type | Function |
|---|---|---|
| AnswerAnnotator | Primitive | Run Answer annotator |
| QuestionAnnotator | Primitive | Run Question annotator |
| TokenAnnotator | Primitive | Run Token annotator |
| NGramAnnotator | Primitive | Run NGram annotator |
| AnswerScoreAnnotator | Primitive | Run AnswerScore annotator |
| hw2-dateng-aae | Aggregate | Run all the annotators above |

Note that:

1. NGramAnnotator and AnswerScoreAnnotator require input from Answer, Question and Token annotators, so this two descriptor cannot be run individually. Otherwise, they will annotate nothing.
2. hw2-dateng-aae is an aggregate analysis engine descriptor that runs all the annotators together, so I specified the processing order as following, so that NGramAnnotator and AnswerScoreAnnotator can be fed with the input annotators properly.
   AnswerAnnotator-> QuestionAnnotator-> TokenAnnotator-> NGramAnnotator-> AnswerScoreAnnotator