HW1

Gabriela N. Góngora Svartzman

The objective of this first homework is to create a logical data model for a sample information processing task. The task consists in retrieving a question (in the form of text) and classifying a series of answers according to this specific query. The answers can be correct or incorrect, and the system should be able to asses them in this manner. A Type System model in UIMA (Unstructured Information Management Architecture) is required (implemented and designed).

1. Analysis of Information Processing Pipeline

The information processing pipeline will consist of the following steps:

1. Test Element Annotation: The system will read in the input file as a UIMA CAS and annotate the question and answer spans. Each answer annotation will also record whether or not the answer is correct.

For this part of the pipeline we will need to annotate question and answers. If we go from a general view to a particular, then both answer and question would belong to sentence. Both answers and questions are specific types that inherit from sentence.

2. Token Annotation: The system will annotate each token span in each question and answer (break on whitespace and punctuation).

We need a “Token” Annotation in our system for this part of the pipeline.

3. NGram Annotation: The system will annotate 1-, 2- and 3-grams of consecutive tokens.

We require an array of type Token (previously defined) and the size of the N being used at that time.

4. Answer Scoring: The system will incorporate a component that will assign an answer score annotation to each answer. The answer score annotation will record the score assigned to the answer.

This part of the pipeline can be implemented in various ways. Adding a feature to the answer annotation or creating an annotation for the score. The easiest would be the first choice, otherwise the design could become excessive or redundant.

5. Evaluation: The system will sort the answers according to their scores, and calculate precision at N (where N is the total number of correct answers).

This will require an annotation itself and features for the size of the N, the actual calculation of the precision and the list of the answers in the order of the evaluation performed. Various information retrieval techniques can be used for this task. In order to identify questions from answer and be able to grade every answer (is correct or not) we need to incorporate methods into our pipeline, ranging from NLP to machine learning. In this case we are going to NLP, as the problem does not have further complexity or requirements.

2. Type System Design

Sentence Annotation
This annotation represents the general view of the problem, a sentence, from which a question or answer can derive.

Question
The question comes from an input text file, which will have the following format:
<Identifier> < QuestionText >
Identifier = String, length=1, (specifically the letter "Q").
QuestionText = String { A...Z | a...z | 0...9 | "," | "´", ... }
Therefore the following features are defined for this annotation:
sentencesText

Answer
The answers originate, as well, from an input text file and will have the following format:
<Identifier><isCorrect><AnswerText>
Identifier = String, length=1, (specifically the letter "A").
isCorrect = Int, either a 1 or a 0.
AnswerText = String { A...Z | a...z | 0...9 | "," | "´", ... }
Therefore the following features are defined for this annotation:
isCorrect
sentencesText

Token
Each sentence will be divided into several tokens. For this matter we need to save where the token starts and begins, plus the total number of tokens, in order to annotate the sentences correctly.

NGram
For this problem we will use three different NGram, 1-gram, 2-gram and 3-gram. Therefore we need a feature that contains the size of the N that is being used. NGram will take the tokens obtained from the sentences.

Base Annotation
It is suggested in the UIMA documentation to have a base annotator that will keep track of the annotations, where they were made and how confident the annotation was. All other types then inherit from this base type. This type is already incorporated into UIMA.

Results

The answers need to be evaluated in terms of precision. Precision @ N is measured. Different values of N will be calculated. The following features are required: sizeN (number of answers to evaluate), precision (according to the values of N) and ordListAnswers (answers ordered according to their precision).

The following UML diagrams exemplifies the overall design of the system.

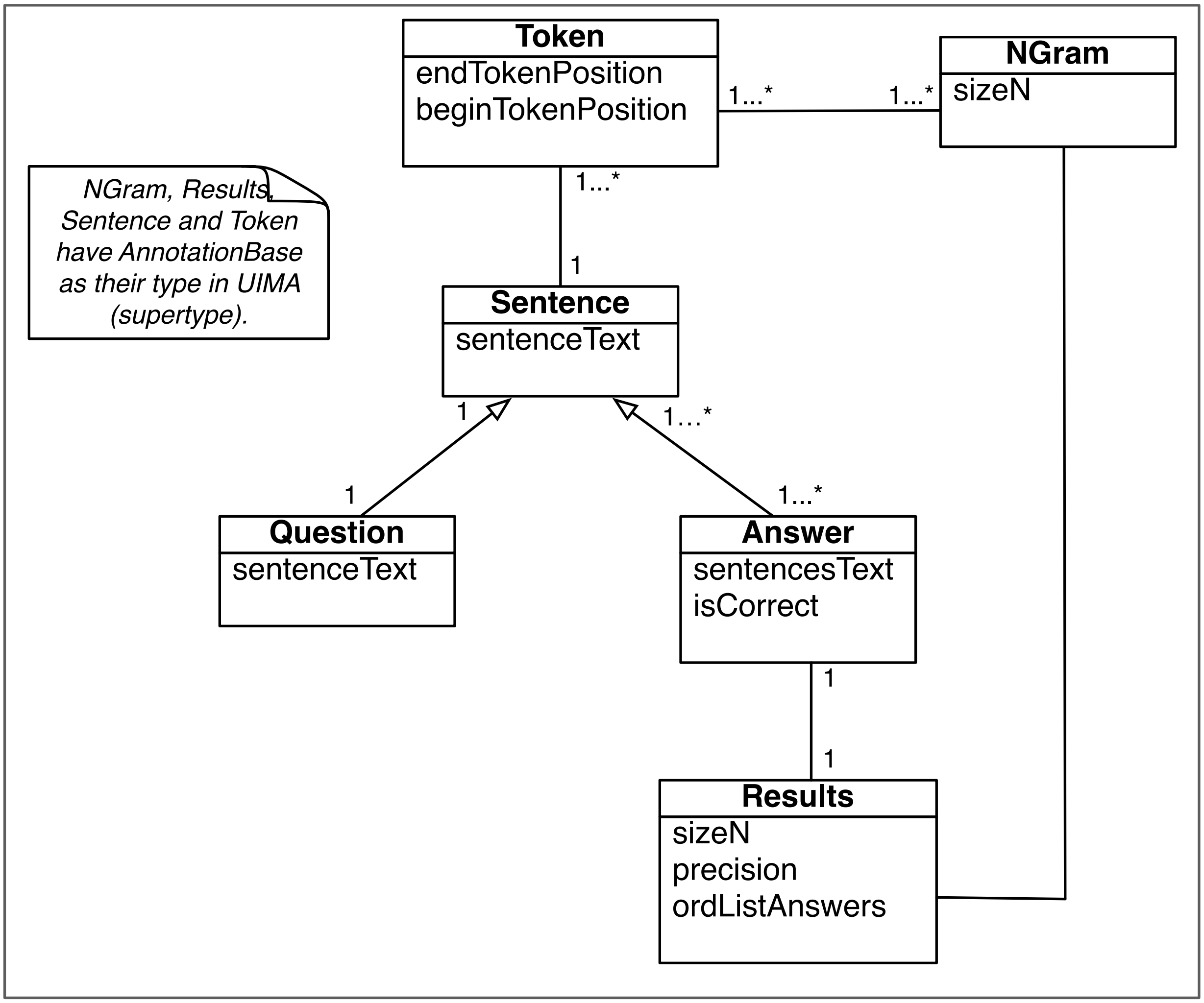
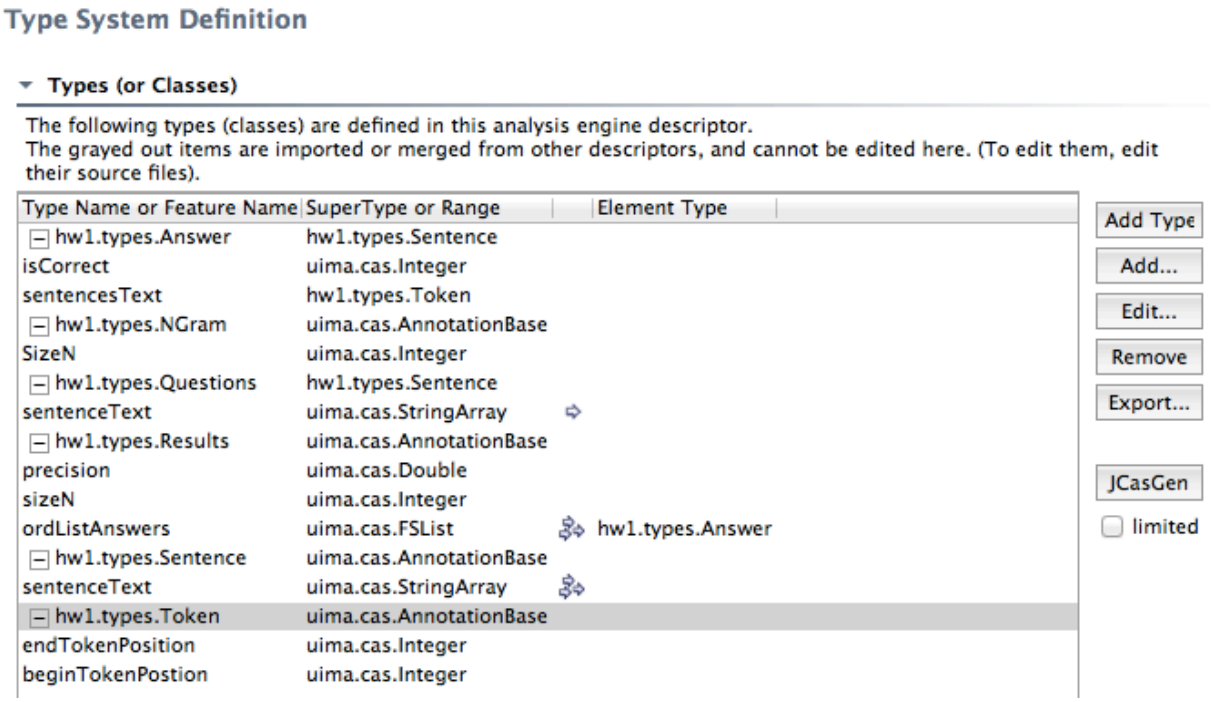


Figure 1: Description of the Type System designed.

3. Implementation

The design previously described was implemented in UIMA.



Type System Definition

Types (or Classes)

The following types (classes) are defined in this analysis engine descriptor.
The grayed out items are imported or merged from other descriptors, and cannot be edited here. (To edit them, edit their source files).

Type Name or Feature Name	SuperType or Range		Element Type
⊟ hw1.types.Answer	hw1.types.Sentence		
isCorrect	uima.cas.Integer		
sentencesText	hw1.types.Token		
⊟ hw1.types.NGram	uima.cas.AnnotationBase		
SizeN	uima.cas.Integer		
⊟ hw1.types.Questions	hw1.types.Sentence		
sentenceText	uima.cas.StringArray	⇨	
⊟ hw1.types.Results	uima.cas.AnnotationBase		
precision	uima.cas.Double		
sizeN	uima.cas.Integer		
ordListAnswers	uima.cas.FSList	⇨	hw1.types.Answer
⊟ hw1.types.Sentence	uima.cas.AnnotationBase		
sentenceText	uima.cas.StringArray	⇨	
⊟ hw1.types.Token	uima.cas.AnnotationBase		
endTokenPosition	uima.cas.Integer		
beginTokenPostion	uima.cas.Integer		

After defining the types and their features in the typeSystem xml file, the java files are created using the JCasGen.

The implementation of this design is uploaded in GitHub:
https://github.com/gabygs/DEIIS-139547-hw1.git

The paths, names and structure is as specified in the homework handout.