Mexico Autonomous Institute of Technology
Design and Engineering of Intelligent Information Systems
COM-35702

Homework 1
Logical Data Model and UIMA Type System Design & Implementation
Report

Fernando Aguilar – 114297

Abstract

In this homework, a UIMA type system was designed in order to support a Question Answering information processing pipeline. The main design goal involved in this type system is to model a semantic model that represents all the necessary information required to accomplish the Question Answering task, and facilitating information query retrieval. However, this design involves some issues that are discussed in the following sections. Finally, the type system was implemented in a UIMA XML type descriptor file, also a Java implementation of the type system was generated using the JCasGen plug-in for Eclipse and the UIMA Java SDK (uimaj-2.5.0).

1. Introduction

UIMA stands for Unstructured Information Management Architecture, a framework for standardizing, managing, and discover information in unstructured documents. According to the UIMA specification[1], unstructured information may be defined as the direct product of human communication. Examples of such unstructured information include natural language documents, email, speech, images and video.

The Apache Software Foundation has an implementation of the UIMA specification, called Apache UIMA[2], which it consists of: (1) frameworks such as the UIMA SDK for Java (uimaj), (2) infrastructure tools like UIMA tools for Eclipse and (3) components like annotators and type systems.

The components part of Apache UIMA are the most important concern for the user, because type systems define the information domain that an UIMA system can process and the annotators establish the way the system extracts information from unstructured documents.

In UIMA, all the information regarded a unstructured document is stored in a Common Analysis Structure (CAS) object. The CAS is mainly composed by two parts: the Subject of Analysis (SofA) and standoff annotations. The SofA is the representation of the unstructured document; for example a string containing the text of a email could be a SofA. The standoff annotations are the information extracted from the SofA; for example, all the names found in the email string could be annotations, hence, these annotations points to the string indexes that contains these names.

The *annotators* are mechanisms that extracts annotations from the SofA or even from another annotations. In the case of the UIMA SDK for Java (uimaj), an annotator is a class that extends `org.apache.uima.analysis_component.JcasAnnotator_ImplBase` class and implements a method called `process(JCas aJCas)` that receives a CAS as a parameter, which is represented by

1 Standard, O. A. S. I. S. Unstructured Information Management Architecture (UIMA) Version 1.0. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.192.5351&rep=rep1&type=pdf
2 Apache UIMA. Available at: http://uima.apache.org/

`org.apache.uima.jcas.JCas` class. It's important to note that it's only permissible to add information (annotations) to the CAS.

Because annotators can be of multiple types, a Type System needs to be defined. It's more easy to understand the necessity of a type system if we make an analogy between annotators and objects and between types an classes. Hence, it's possible to use standard object-oriented tools and frameworks to design a type system, like UML.

In this Homework, a type system is proposed and implemented for a Question Answering information system. Section 2 analyzes the requirements and needs stated in the information system description. Section 3 discuss the types and their relationships that forms the type system. Section 4 provides implementation details of the type system. Section 5 discuss the design issues found in the type system and in section 6 there are some conclusions.

2. Requirements analysis

The main goal of the information processing system is to recognize and provide an answer to a question given a set of predefined answers. To accomplish this goal, the system has a processing pipeline that realizes these steps in the following order:

1. **Test Elements**: The system will recognize lines of text that represent either a question or an answer. Each line forms a sentence.
2. **Tokens**: For each sentence, a set of tokens will be extracted. A token is a substring of a sentence, which are formed by breaking the sentence on whitespaces and punctuation signs.
3. **N-grams**: Sets of 1-grams, 2-grams and 3-grams will be formed with the tokens created in the previous step.
4. **Answer Scoring**: For each answer, a score will be calculated using the annotations generated in the previous steps. Note that each answer belongs to only one question.
5. **Evaluation**: The system will sort the answers according to their scores, and it will calculate the precision@N score, where N is the total number of correct answers.

Finally, all annotations must record the annotator that created the annotation and a confidence score that the annotator gives to the annotation. Thus, a type system was designed to fulfill these requirements. This type system is presented in the following section.

3. The Type System

In figure 1, there is an UML diagram showing the classes (types) that conforms the type system. We'll discuss each type (class), as follows:

- **PipelineAnnotation**: This type was created in order to fulfill the design requirements related to storing the name and the confidence of the annotator that creates and evaluates the annotation, respectively. All of the following classes of this type system inherit this type.
- **Sentence**: This type represents a sentence, that will be, either a question or an answer. Thus, a sentence can't be an answer and a question at the same time. A sentence can contain tokens generated by a pipeline step.

- **Question**: Represents a question found in the input data. A question can contain multiple answers. Also, there is a slot (field or attribute) used by an annotator for storing the precision@N score. A question is composed by exactly one sentence.
- **Answer**: Represents an answer that was found in the input data. An answer only belongs to one question. Similar to a question, an answer is composed by exactly one sentence. This type has slots (attributes) for storing whether this answer is correct and a score attributed used by an annotator to evaluate the system.
- **Token**: All the tokens created in the second step of the pipeline will be instances of this class. This token type has two set of indexes: the indexes inherited from uima.cas.Annotation class, which they point to the absolute reference of the SofA, and the local indexes, that points to the indexes that a sentence belongs to.
- **Ngram**: It consists of an array of consecutive tokens. An Ngram instance can contain up to three tokens.

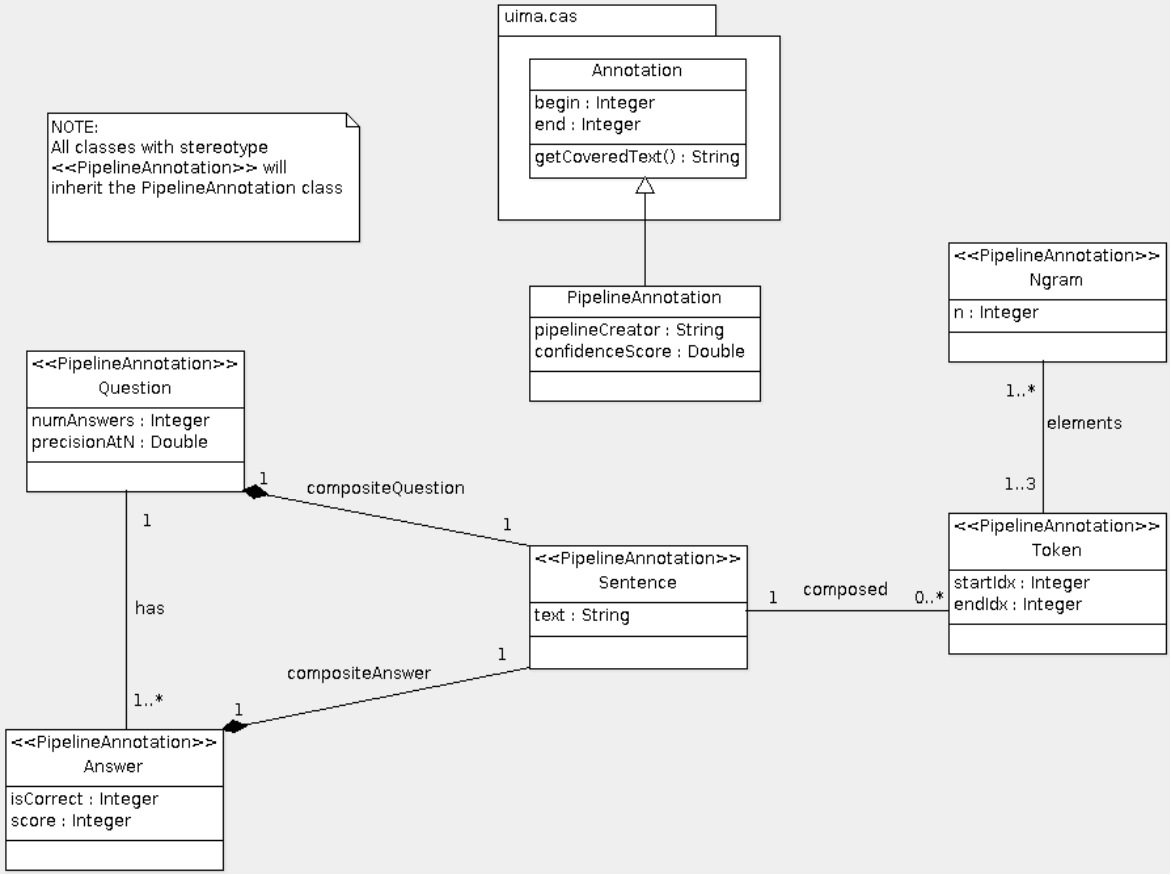

Figure 1: An UML class diagram showing the design of the UIMA Type System

4. Implementation

Once the UML design was completed, the type system was implemented in a UMA XML type descrip-

tor file. Here, `FSArray` and `FSList` types were used in order to model the relationships between types in the type system. Some requirements like constraining whether a sentence is either a question or an answer and limiting the number of tokens in a Ngram are not taken into account in the type system implementation. These requirements must be contemplated in the design of annotators. In figure 2, we show the types implemented in the descriptor file. This file was created using the Eclipse plug-ins for UIMA.

Later, the JCasGen plug-in was used for generating the Java code that represents the UIMA types as Java classes.

Also, Maven is used to create the initial project via an archetype and a `pom.xml` file is used in order to manage the project dependencies and for generating the Javadocs.

Type Name or Feature Name	SuperType or Range		Element Type
mx.itam.deiis.hw1.Answer	mx.itam.deiis.hw1.PipelineAnnotation		
isCorrect	uima.cas.Boolean		
score	uima.cas.Double		
sentence	mx.itam.deiis.hw1.Sentence		
question	mx.itam.deiis.hw1.Question		
mx.itam.deiis.hw1.Ngram	mx.itam.deiis.hw1.PipelineAnnotation		
n	uima.cas.Integer		
tokens	uima.cas.FSArray	→	mx.itam.deiis.hw1.Token
mx.itam.deiis.hw1.PipelineAnnotation	uima.tcas.Annotation		
pipelineCreator	uima.cas.String		
confidenceScore	uima.cas.Double		
mx.itam.deiis.hw1.Question	mx.itam.deiis.hw1.PipelineAnnotation		
numAnswers	uima.cas.Integer		
precisionAtN	uima.cas.Double		
sentence	mx.itam.deiis.hw1.Sentence		
answers	uima.cas.FSList	→	mx.itam.deiis.hw1.Answer
mx.itam.deiis.hw1.Sentence	mx.itam.deiis.hw1.PipelineAnnotation		
text	uima.cas.String		
mx.itam.deiis.hw1.Token	mx.itam.deiis.hw1.PipelineAnnotation		
localStartIdx	uima.cas.Integer		
localStartEnd	uima.cas.Integer		
sentence	mx.itam.deiis.hw1.Sentence		
ngrams	uima.cas.FSList	→	mx.itam.deiis.hw1.Ngram

Figure 2: The XML Type System designed with the UIMA Eclipse Plug-in

5. Design Issues

Although much effort was put in the design of the type system for fulfill the design goals of supporting all the information required in the processing pipeline and facilitate future queries to the model, there are some issues that were discovered in the type system, as follows:

- First, it adds some memory overhead due to the associations between UIMA types (classes).
- Second, for the Question and Answer types, the annotation that stores the name of the pipeline

annotator that created the Question or Answer Type annotation is not referred to the creator. Instead, this annotations is used to store the name of the annotator that scores the Question of the Answer type.

- Third, there is a Maven convention that are not followed: the groupId (`edu.cmu.lti.11791.f13.hw1`) and artifactId (`hw1-114297`) does not coincide with the main package name for all the types (`mx.itam.deiis.hw1`).

6. Conclusions

In this homework, a type system for a Question Answering system was created. It fulfills the requirements established by the Question Answering system. Also, an UIMA type system descriptor was implemented and Java classes were generated from the descriptor. As future work, we could modify the original proposed design in order to eliminate the issues found in the design process.