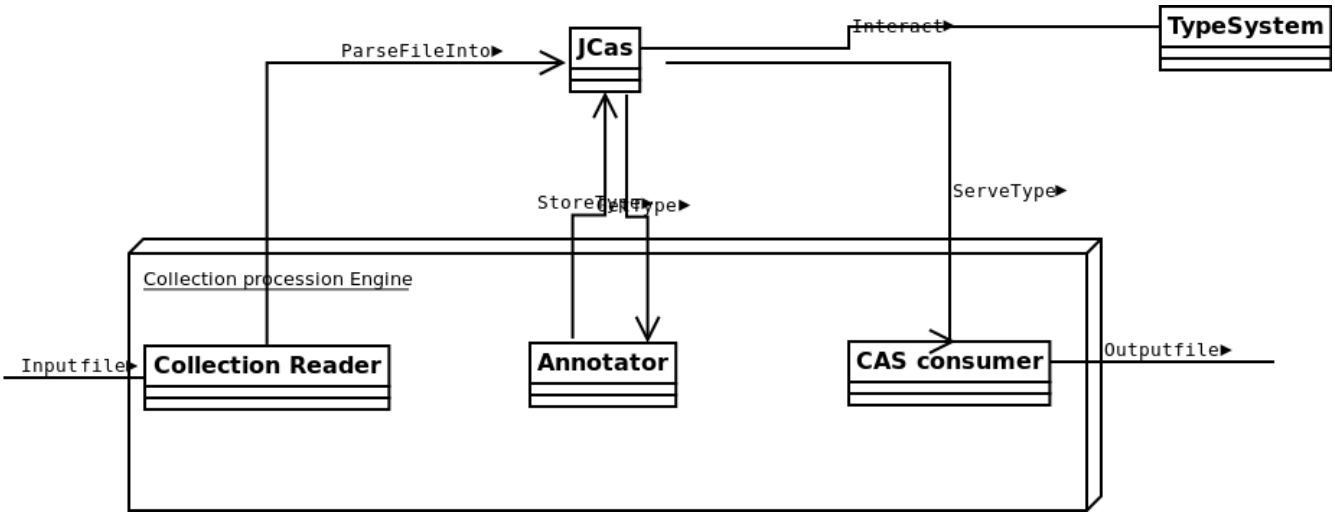
Name Entity Recognizer with UIMA SDK

Shu-Hao Yu
shuhaoy@andrew.cmu.edu

1. Introduction

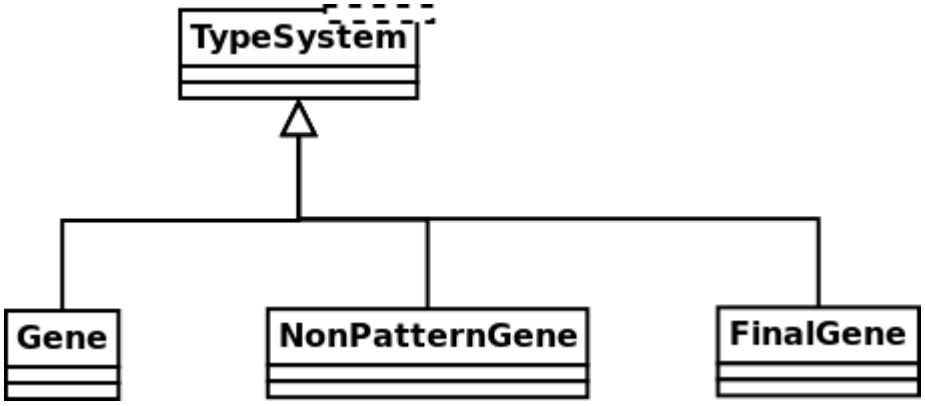
In this task, I went through the tutorial of UIMA, learning this framework comprehensively. After that, I built up my name entity recognizer(NER) system based on this framework. In each component, I carefully designed them to fit the specific requirement and tried to make performance strong while fully exploit the functions that UIMA provided. After this homework, I learn UIMA is a well-defined and organized tool. Also, I had a first look at the name entity recognition problem and algorithms dealt with it.

2. Method



When deal with this task, my concept is like this picture. We have to build up the whole collection processing Engine, which has three parts, Collection Reader, annotator, and CAS consumer. Though these three components form a pipeline process, which means collection reader's output feed as the input of annotator, they actually all interact with JCas system. Jcas interact with them with types we had pre-defined. For collection reader, it read the input file and them parse into strings into Jcas system. Annotator, which then retrieves the string from Jcas system and recognize the specific types and stores into Jcas system. Finally, the CAS consumer retrieves the type from Jcas system and print the result and format we specified.
For the following, I will explain each of my component and how I implemented them.

I. Type System

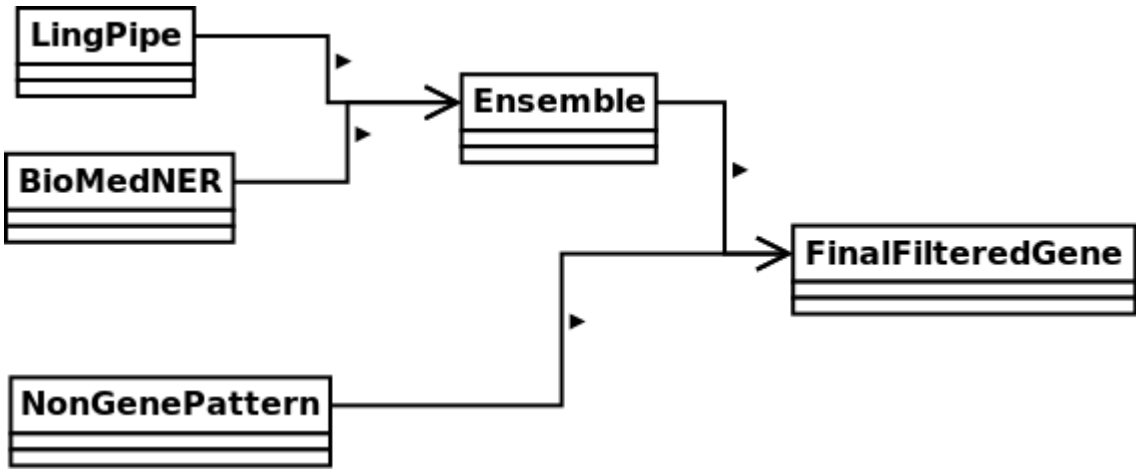


I created three types. I used LingPipe algorithm to detect geneTag and stored them as Gene type. Second, I searched the patterns that is not GeneTag and stored them in NonPatternGene type. Finally, I used NonPatternGene as filter to extract geneTag from Gene type and get the FinalGene type. One thing worth to note is that, we have to store sentence ID and the begin and end locations in the type since the task requires them to be in result.

II. Collection Reader

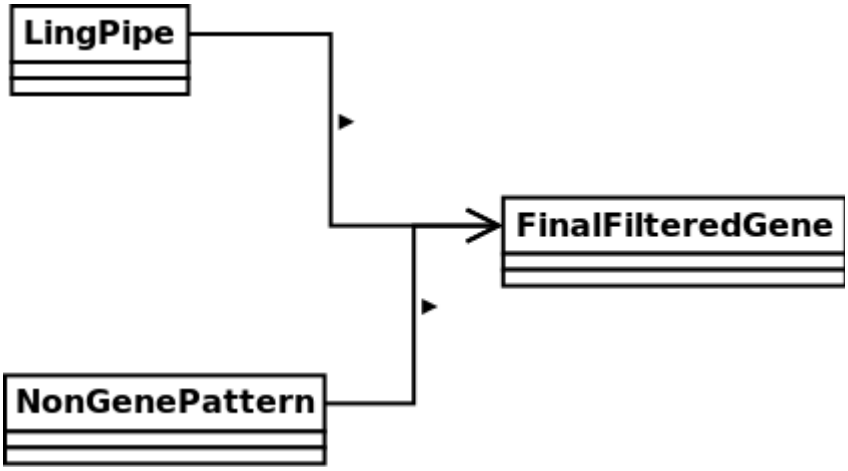
I mainly use the original collection reader in UIMA example. The only change I made is that I make input path possible as single file and directory instead of only directory in the example. The collection reader read in whole document and store into Jcas system.

III. Annotator



For the original thought, my implementation should be like this. LingPipe and BioMedNer are both algorithms for GeneTag recognition. Orginally I want to implement some algorithms and then ensemble them based on weight derived by training from sample file. After that, I will do post-processing by filtering out those are not gene patterns in the ensemble result.
However, BioMedNER didn't have set-up jar file. I tried to compile whole package by myself but in

vain. (The source code link: http://biomedner.googlecode.com/svn/!svn/bc/321/trunk/biomedner/src/bioner/)



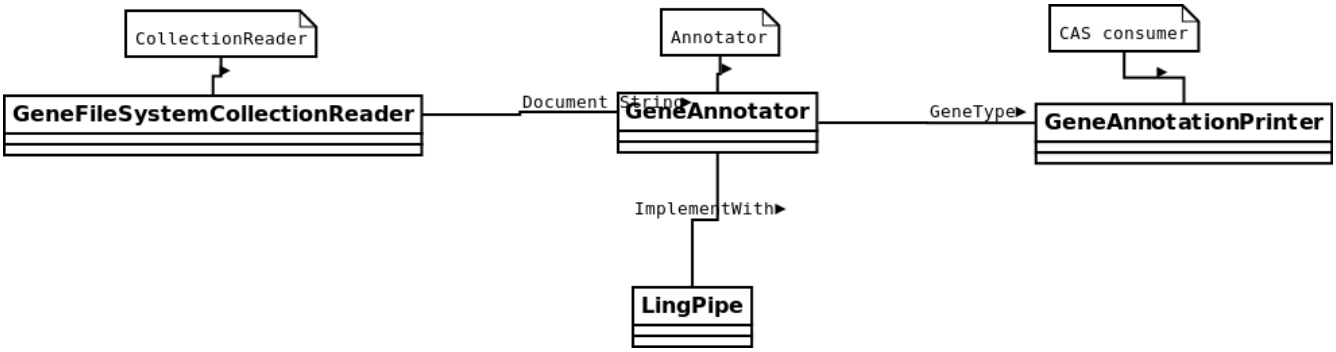
By giving up the adoption of BioMedNER, in the end I don't use ensemble technique. Instead I use one recognition annotator and one pattern annotator and followed by an aggregate annotator. LingPipe is a strong geneTag recognition that produce much geneTag, and we already knew some specific pattern should not be recognized as geneTag; therefore, I create an aggregate analysis engine to deal with the results I got from first two annotator.

I implemented this idea but it didn't go well. The main reason is that I am not familiar with the bio-patterns so I could not exploit them in my NonGenePattern annotator. Otherwise I believe it will boost the performance. In the end, I only included LingPipe in my CPE descriptor, but you can find the implementation in my package.

IV. CAS consumer

Which is not a complex cas consumer. Since I have already recorded the sentence ID and begin/end locations, I just read the specific type from Jcas system and then printed into file with the specified format.

V. Collection Processing Engine



I use UIMA SDK to produce my CPE descriptor, the architecture is like this image, where it includes collection reader, annotator and cas consumer.

3. Conclusion

This is a great homework to practice with UIMA. It helps me to see the whole picture with this framework. I would say this homework is somewhat like project-level, I will do better if I have much more time to do it. One suggestion is that the homework may be better to be segmented into different parts. For example, a homework to implement a simple annotator is a good idea. It is much pressure when we have to deal with the whole framework in one homework.