Software Engineering for Intelligent Information System Homework #1

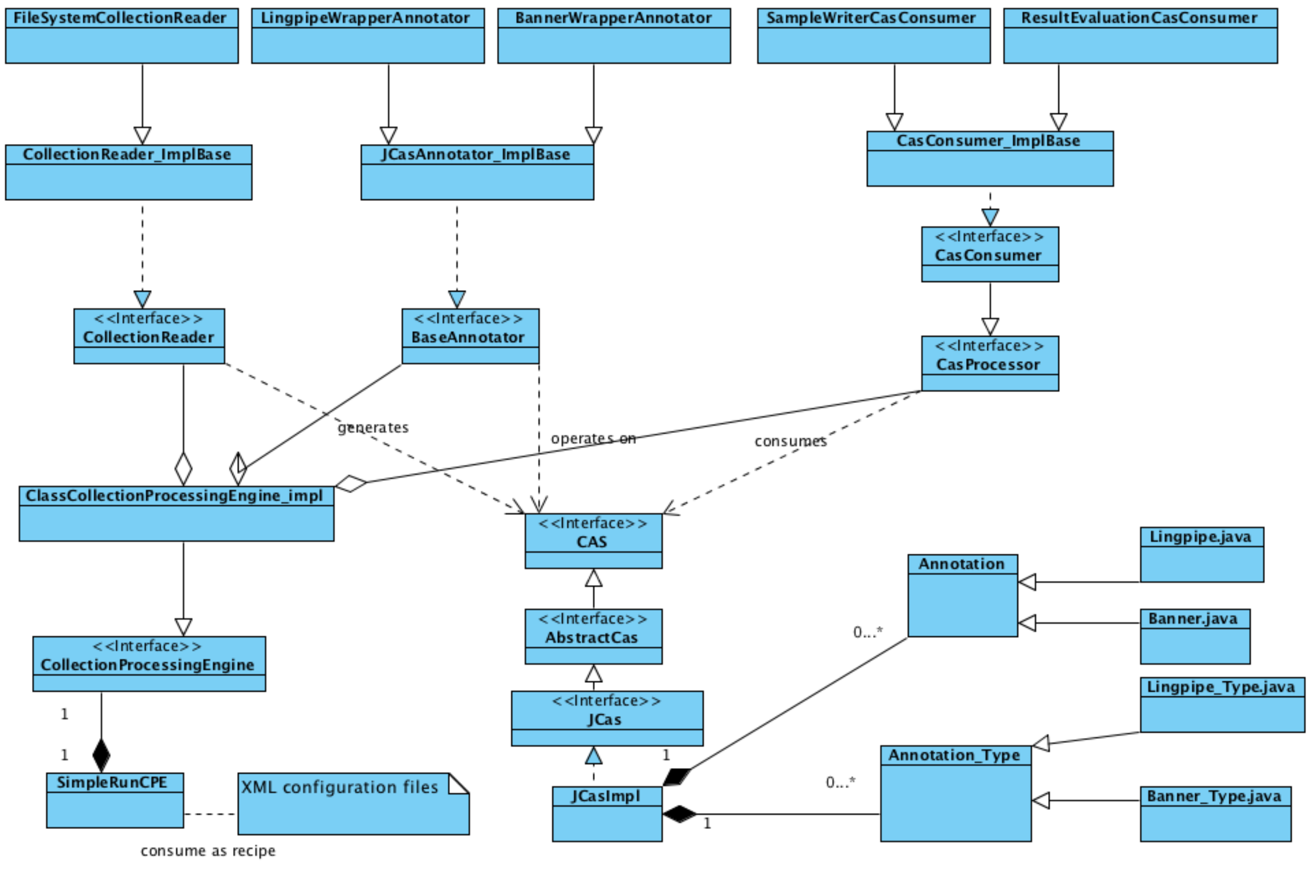
Building Gene Named Entity Recognition Component under UIMA Framework

Andrew ID: mingtaoz
Mingtao (Craig) Zhang
Global MISM
Heinz College

1 System Design

1.1 UML Class Diagram

The class diagram describes how my classes interact with UIMA Framework roughly. I tried my best to draw it in detail, while looked at bother User-Level API and full API.



1.2 Type System

1.2.1 Initial Type System

a. POS-NN provided by stanford CoreNLP, which provided about 50% recall and 10% precision.

b. Regular Expression based on "GENETAG Annotation Guidelines":

1. Mutants/ Example: p53 mutant

```
"[A-Za-z0-9]* [Mm]utan(t|ts)"
```

2. Parens at start or end when embedded in the name/ Example: (IGG) receptor

```
"([A-Za-z0-9]+\\([A-Za-z0-9]*\\))|(\\([A-Za-z0-9]*\\)[A-Za-z0-9]+)")
```

3. Fusion proteins

```
"[A-Za-z0-9]* [A-Za-z0-9]* fusion"
```

But it really improves too little.

c. A self-trained CRF based on ⅘ of the sample.in/out and testing against the lest ⅕. I was using a CRF (Conditional Random Fields) package in Stanford's CoreNLP package, and read through the CRF introduction paper roughly. This turned out to be a disaster, the training generally took me 30 min each and I tried several times because the input should be this format: <word, tag> for each line, and any space will be seen as separation between word and tag by the system. So the CRF can only help me deal with one word gene.

1.2.2 An acceptable Type System

Later on, I wrapped Lingpipe as a Annotator. And I deleted all my previous Types because they are not competitive with Lingpipe. Here is the Lingpipe result on sample.in/out:

```
-------- PRECISION REPORT --------------------------
        Precision: 0.7685
        Recall: 0.8488
-------- Performance REPORT --------------------------
        Completed 15000 lines; 2500605 characters
        Total Time Elapsed: 6460 ms
        Initialization Time: 1002 ms
        Processing Time: 5458 ms
```

1.2.3 Final Type System

At the end, I wrapped Banner as another annotator. Here is Banner's result:

```
-------- PRECISION REPORT --------------------------
```

```
            total result matching: 16525
            total result lines: 17527
            sample result lines: 18265
            precision: 0.942831060649284
            recall: 0.9047358335614564
        -------- Performance REPORT --------------------
            Completed 15000 documents; 2500605 characters
            Total Time Elapsed: 360335 ms
            Initialization Time: 2918 ms
            Processing Time: 357417 ms
```

The precision is great, but it is a little bit slow. Based on the output and their different core algorithm: HMM and CRF, I made Banner to deal with long sentences while Lingpipe deal with short sentences and vise versa to make a combined Type System.

1.3 Learned Design Patterns

I don't have any special design pattern in my program because UIMA Framework has done that for me. I am writing program against interfaces instead of concrete classes. Here is two patterns I learned from UIMA Framework.

1.3.1 Module Pattern (Structural patterns)

In UIMA Framework, the CAS groups several related elements, such as Features, Types, Annotations into a single conceptual entity. It made different Analysis Engines and Cas Consumers can operate on CAS in different processes.

1.3.2 Adapter/Wrapper/Translator Pattern

In my program, LingpipeWrapperAnnotator is a Wrapper for Lingpipe's chunking process. It trims Lingpipe's into a UIMA annotator, which separate concerns. In Lingpipe system, LingpipeWrapperAnnotator is a client. In UIMA Framework, LingpipeWrapperAnnotator is an "annotator", we no longer need to worry about Lingpipe's detail when we use it in UIMA context.

2 Algorithmic

2.1 Tools

2.1.1 Lingpipe

I use Lingpipe in a shallow way by wrapping the Chunking Process and the existed Gene Tagging feature file (ne-en-bio-genetag.hmmchunker) into a UIMA annotator.

2.1.2 Banner

I use Banner in a shallow way by wrapping the Tagging Process and the existed Gene Tagging feature file (gene_model_v02.bin) into a UIMA annotator.

2.2 Evaluation

I coded a CasConsumer called ResultEvaluationCasConsumer to deal with evaluation. Here is my evaluation record when I tried to merge Banner and Lingpipe into fast/precise system.

Condition/Character	Total Time	Precision	Recall
Single Lingpipe	6460 ms	76.85%	84.88%
Single Banner	360335 ms	94.28%	90.47%
if sentence.length <= 200 characters Lingpipe else Banner	90536 ms	82.11%	85.37%
if sentence.length <= 220 characters Lingpipe else Banner	73263 ms	80.76%	85.02%
if sentence.length <= 200 characters Banner else Lingpipe	181929 ms	87.84%	89.99%
if sentence.length <= 180 characters Banner else Lingpipe	136854 ms	86.02%	89.48%
if sentence.length <= 150 characters Banner else Lingpipe	86479 ms	82.89%	88.46%

I made the decision to use Banner to do <= 200 characters sentences while Lingpipe to deal with >200 characters sentences.

2.3 Knowledge Sources

a. introduction to CRF acdemic paper:

http://people.cs.umass.edu/~mccallum/papers/crf-tutorial.pdf

b. Linpipe's competitor list

http://alias-i.com/lingpipe/web/competition.html

c. Abner

http://pages.cs.wisc.edu/~bsettles/abner

d. BioTagger

http://www.seas.upenn.edu/~strctlrn/BioTagger/BioTagger.html

e. Banner

http://cbioc.eas.asu.edu/banner

f. UIMA full API

http://javasourcecode.org/html/open-source/uima-as/uima-as-2.4/overview-summary.html

3 Other Information of Interest

3.1 CSE Question

I think from language model view, whether to remove stop words is an open topic because the writer's idea is changed by stop words removing. For example, questions like: "Where can I see <The Music Man>?" is difficult to be answered if we remove stop words.
So we will have different versions of "question analyser" as well as "searcher based on different index", while in the middle, the components may be the same as they were.
In this scenario, I feel there should be a "slice" concept, vertical with "phase". And it is not a fully combination of all possibilities. Do we support it with CSE?

3.2 Unexpected Benefit from learning UIMA Framework

Before seeing UIMA, I have never thought carefully about how XML document can guide the whole program execution (I was just the user of that).
Now, I am implementing similar idea on my own in Search Engine (11-641) course. Solid design make my program easy to extend while my classmates are tackling with complicated int arrays.
Here is a brief:

3.2.1 OperationUnit as Wrapper (After CAS introduction)

Given I need to deal with both Inverted List and Score List and the query operators they can support are different with each other, I came up with a class called “OperationUnit”, which has the following characters:

a. It is composed of an InvertedList and a ScoreList.
b. It will maintain only one reference while another is null.
c. It will support all the query operators and the transition from InvertedList to ScoreList.

3.2.2 ScoreConverter as Strategy (After Design Pattern Introduction)

I need to deal with Scoring, which transit InvertedList to ScoreList, and test through Boolean Rank/Unranked Retrieval/Indri/Okapi BM25. So I used a strategy pattern here.

3.2.3 XML configuration (After UIMA configuration Introduction)

I have several strategy patterns and template method patterns with a recipe configuration file (partially displayed) to support testing:

```
<constant>
        <preprocessedDocumentDirectory>...</preprocessedDocumentDirectory>
        <stopWordList>...</stopWordList>
        <averageDocumentSize>...</averageDocumentSize><vocabularySize>...</vocabularySize>
        <corpusSizeNumberOfDocuments>...</corpusSizeNumberOfDocuments>
        <corpusSizeTotalWordOccurrences>...</corpusSizeTotalWordOccurrences>
</constant>
<processes>
        <process>
                <engineType>indri</engineType><defaultOperator>AND</defaultOperator>
                <lambda>0.5</lambda><mu>1500</mu>
                <converter>indri</converter>
                <resultFileName>testIndri.txt</resultFileName><queryFileName>queries.txt</queryFileName>
        </process>
        <process>
                <engineType>BM25</engineType><defaultOperator>SUM</defaultOperator>
                <k1>1.2</k1><b>0.75</b><k3>500</k3><converter>BM25</converter>
                <resultFileName>testBM25.txt</resultFileName><queryFileName>queries.txt</queryFileName>
        </process>
</processes>
```