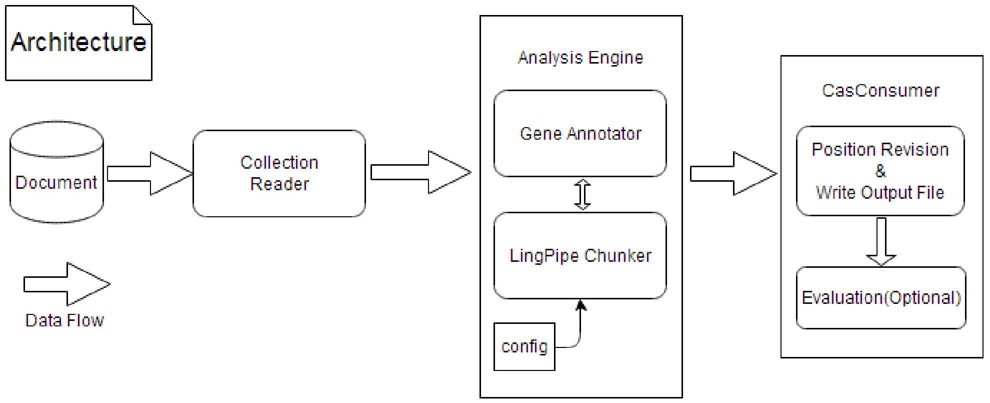
Name: Zhoucheng Li

Andrew ID: zhouchel

Homework 1

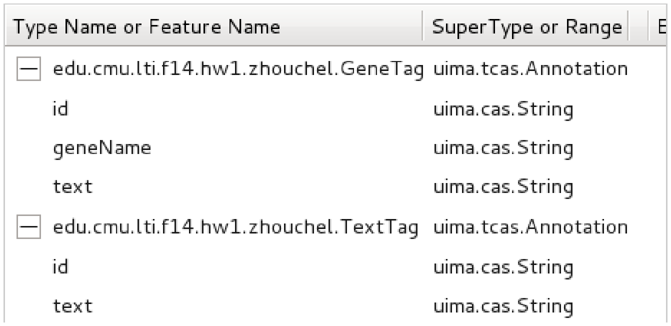
1 Architecture Overview



This architecture mainly contains three parts (excluding the input documents). The Collection Reader would read data **line-by-line** from the document, separate the each line into ID and text, and wrap them into CAS indices. Here, the Analysis Engine is a primitive analysis engine. Gene Annotator would fetch the feature structures from the results given by Collection Reader, and read configuration file to decide use which model and parameters to annotate gene names. Then, the extracted gene mentions and positions would be wrapped as GeneType, then added into CAS indices. Finally, CasConsumer would iterate all GeneType Feature Structures, revise the positions by calculating whitespace-excluded offsets, and write all results into a file. The CasConsumer can also do an evaluation process if evaluation option and golden standard output file are both specified in a configuration file.

2 Detailed Implementation

2.1 Type System



Type Name or Feature Name	SuperType or Range	E
edu.cmu.lti.f14.hw1.zhouchel.GeneTag	uima.tcas.Annotation	
id	uima.cas.String	
geneName	uima.cas.String	
text	uima.cas.String	
edu.cmu.lti.f14.hw1.zhouchel.TextTag	uima.tcas.Annotation	
id	uima.cas.String	
text	uima.cas.String	

There are two types (***GeneTag*** and ***TextTag***) in this system. They are both inherited from ***uima.tcas.Annotation*** which defines two integer features indicating begin and end of the span being annotated. ***TextTag*** is used for input which annotates sentences and corresponding IDs, while ***GeneTag*** is used for output which annotates gene name' positions and original sentences and corresponding IDs.

2.2 Collection Reader

Several functions (arguments are omitted here) like initialize(), next(), hasNext() and close() are rewritten in Collection Reader.

In Initialize(), document would be opened. hasNext() is used to check if there's next line in the document. If there exists next line, we can use next() to process this line, separating it into ID and text, wrapping ID and text into a TextTag, adding the annotation into CAS index. When this document is processed over, close() would close the document.

2.3 Analysis Engine

2.3.1 Configuration File

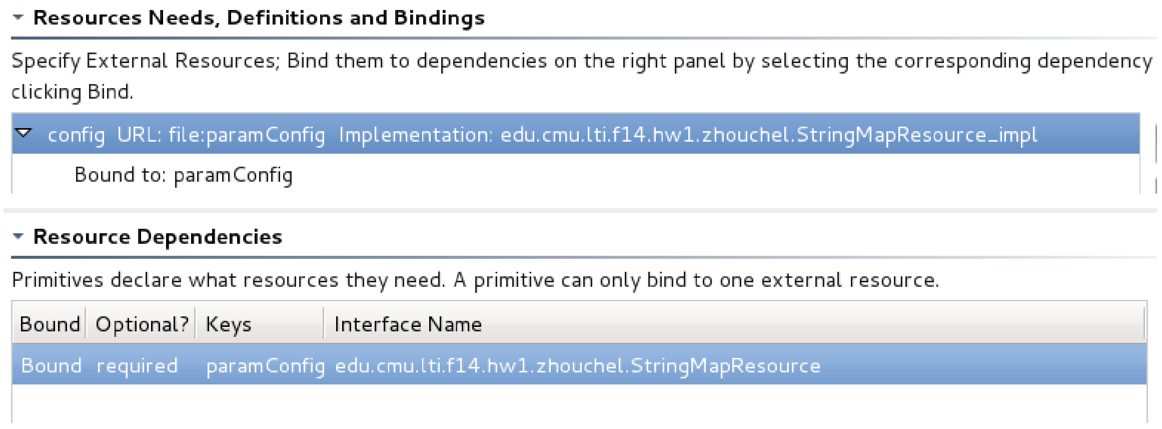
I would like to introduce this component here because every component in analysis engine use parameters or model paths in the configuration file. I'm not sure if there's any better way to achieve this (maybe configured all parameters in annotator description file?).

The parameters and model paths are specified in *paramConfig* using following format:

```
model = ne-en-bio-genetag.HmmChunker
MAX_N_BEST_CHUNKS = 20
threshold = 0.65
N-Best_NER = true
Evaluation = true
Gold_Standard = data/sample.out
```

The left part of "=" servers as key and the right part of "=" is treated as associated value to that key.

UIMA provides an easy way to access external resources from UimaContext. In GeneAnnotator.xml, add external dependencies, specify external resource (i.e. the configuration file) and bind it to the dependency. In this way, annotator code does not need to know the location of external data that may be used.


Resources Needs, Definitions and Bindings

Specify External Resources; Bind them to dependencies on the right panel by selecting the corresponding dependency clicking Bind.

config URL: file:paramConfig Implementation: edu.cmu.lti.f14.hw1.zhouchel.StringMapResource_impl

Bound to: paramConfig

Resource Dependencies

Primitives declare what resources they need. A primitive can only bind to one external resource.

Bound	Optional?	Keys	Interface Name
Bound	required	paramConfig	edu.cmu.lti.f14.hw1.zhouchel.StringMapResource

Here, StringMapResource is an interface which separates the resource file from annotators (i.e. StringMapResource_impl would directly access the configuration file while annotator cannot).

In class StringMapResource_impl, configurations are read and separated into "key-value" pairs and stored in a public *static* HashMap. So other components like CasConsumer could also access to the configurations latter.

2.3.2 Gene Annotator

Gene Annotator is the core component of this analysis engine.

I implemented two Gene Annotators. One uses PosTagNamedEntityRecognizer (provided along with HW1, uses Stanford coreNLP tookit). Another annotator uses LingPipe toolkit to annotate all gene mentions. Now the source code is hard coded which can only uses annotator based on LingPipe. But you can choose use ***First-Best*** or ***N-Best*** (including ***Confidence N-Best***) algorithm via configuration file (more details will be illustrated in section 2.3.3).

In class GeneAnnotatorWithLingPipe, I rewrote the initialize() function so I could **load the configuration file before process phrase**. Also I could load model file and **initialize the NER before process phrase**. Whether the NER use First-Best or N-Best algorithm depends on the parameters in configuration file.

In process() function, the annotator would retrieve the sentences from CAS and feed them to NER. Results would be returned to annotator, wrapped to GeneType and added to CAS index.

2.3.3 LingPipe Chunker

I use statistical NER in this project. The model has been trained from GENETAG corpus and could be downloaded via LingPipe's website. With this model, we can use LingPipe's API to do First-Best Named Entity Chunking or N-Best Named Entity Chunking.

First-Best NER only returns the first answer it recognizes while N-Best NER would return N best answers in order of their estimated likelihoods (descending order).

If I set the confidence threshold in configuration file to 0, the NER would return all N answers. Else, the NER would only return the answers whose confidence is larger than the given threshold (i.e. ***Confidence N-Best***).

2.4 CasConsumer

2.4.1 Position Revision and Write Output File

In CasConsumer, Gene annotations would be iterated using Feature Structure iterator. Begin and end positions were **updated with calculated whitespace-excluded offsets**. Sentence ID, gene name, begin and end positions were written into output file. The file path is specified in casConsumer.xml.

2.4.2 Evaluation Process (optional)

In case there is no specified golden standard output file. I use an option "Evaluation = true/false" in configuration file to decide whether perform evaluation or not. Also, if evaluation option is set to "true", file path (relative to src/main/resources) of the standard output must be specified.

The evaluation function is really simple. First, set a variable *TruePositive* to 0, read each line in golden standard output file, add it a HashSet. Then read each line in predicted output, check if this line is contained in previously generated HashSet. If it's in there, add one to *TruePositive*, else continue. Then the precision and recall could be calculated as:

$$\begin{aligned} \text{Precision} &= \frac{TP}{N_{\text{Predicted}}} \\ \text{Recall} &= \frac{TP}{N_{\text{GoldStandard}}} \end{aligned} \tag{1}$$

3 Algorithms and Model Resources

3.1 Baseline

The baseline algorithm uses Stanford coreNLP library. It will extract all nuns and thus the precision of base line is very low (refer to section 4).

3.2 LingPipe NER with Statistical Model

The model has been trained from GENETAG corpus. Download:HmmChunker Model

The selection of algorithm (First-Best, N-Best or Confidence N-Best) depends of the configuration file.

4 Experiments

	MAX_N_BEST	Threshold	Precision	Recall	F1 Score
Baseline	N/A	N/A	0.1025	0.5467	0.1727
First-Best	N/A	N/A	0.7685	0.8488	0.8067
N-Best	1	0	0.4595	0.3774	0.4144
	2	0	0.3097	0.6174	0.4125
	5	0	0.1853	0.8922	0.3069
	10	0	0.1086	0.9722	0.1954
	15	0	0.0761	0.9880	0.1414
	30	0	0.0401	0.9970	0.0771
Confidence N-Best	5	0.55	0.8067	0.7884	0.7975
	5	0.6	0.8204	0.7785	0.7989
	5	0.65	0.8338	0.7659	0.7984
	10	0.55	0.8002	0.8260	0.8129
	10	0.6	0.8149	0.8152	0.8151
	10	0.65	0.8295	0.8010	0.8150
	15	0.55	0.7996	0.8292	0.8141
	15	0.6	0.8144	0.8182	0.8163
	15	0.65	0.8293	0.8039	0.8164
	20	0.55	0.7996	0.8299	0.8145
	20	0.6	0.8144	0.8189	0.8166
	20	0.65	0.8294	0.8045	0.8168
	30	0.65	0.8295	0.8052	0.8172

Best F1 Score

For the above table, we can see that overall score for First-Best and Confidence N-Best outperform N-Best. But N-Best tests give us an intuition that when N >= 30, N-Best would retrieve almost all gene names in golden standard output. From the tests of Confidence N-Best, we could also see that larger N and higher Confidence threshold could have better F1 Score.

However, the choice of parameters really depends on the system requirement. If you want higher precision, then set a higher threshold. If you want higher recall, lower the threshold and make MAX_N_BEST a slightly larger (bigger than 30 rarely affects the final results).

For an unknown data source, I would like to choose larger MAX_N_BEST, then use a confidence threshold around 0.65.

If we want achieve a better F1 Score, a possible way is to train Model by ourselves. Here is overview of all team's system description: Overview of BioCreative II gene mention recognition. The firs team could get a 0.8721 F1 Score. That' a long way to go.