**Design and Engineering of Intelligent Information Systems**

**Homework 1**

**Gene Name Tagging System**

**Bharat Dutt**

[bdutt@andrew.cmu.edu](mailto:bdutt@andrew.cmu.edu)

* **Overview**

The Gene Name Tagging Systems processes a simple input file and returns all the gene name annotations it can find. These annotations along with the spans are written into an output file. This system is based on the Apache UIMA framework.

* **UIMA Type System**

Every object in the CAS must be associated with a Type in the Type System . I have defined my type system in the **GeneTaggingTypeSystem.xml** . My type system consists of 2 types defined as follows –

1. **org.apache.uima.GeneTaggingSystem.Gene**

This consists of 4 feature – ID (Sentence ID where the gene mention was found) and Name (Name of the Gene), StartSpan, EndSpan.

1. **org.apache.uima.GeneTaggingSystem.SourceText**

This consists of 2 features – ID (Sentence ID of the sentence to be annotated), SourceText (Sentence Text to be passed on to the Analysis Engine)

* **Collection Processing Element (CPE) 1**

A CPE includes an Analysis Engine and adds a Collection Reader, a CAS Initializer (deprecated as of version 2), and CAS Consumers. The part of the UIMA Framework that supports the execution of CPEs is called the Collection Processing Manager, or CPM.

A CPE is executed by a UIMA infrastructure component called the *Collection Processing Manager* (CPM). The CPM provides a number of services and deployment options that cover instantiation and execution of CPEs, error recovery, and local and distributed deployment of the CPE components.

The components of a CPE are:

1. **Collection Reader** – interfaces to a collection of data items (e.g., documents) to be analyzed. Collection Readers return CASes that contain the documents to analyze, possibly along with additional metadata.
2. **Analysis Engine** – takes a CAS, analyzes its contents, and produces an enriched CAS. Analysis Engines can be recursively composed of other Analysis Engines (called an Aggregate Analysis Engine). Aggregates may also contain CAS Consumers.
3. **CAS Consumer** – consume the enriched CAS that was produced by the sequence of Analysis Engines before it, and produce an application-specific data structure, such as a search engine index or database.

**CPE**

Gene Annotations

SourceText annotations

Sample.out

Sample.in

CAS Consumer

Abner NER Analysis Engine

Collection Reader

Collection Reader

Abner NER Analysis Engine

CAS Consumer

Basic Overview of the Gene Tagging System

Sample.in

Sample.out

SourceText annotations

Gene Annotations

Basic Overview of the CPE

(The arrows in the above diagram only represent the flow in the system. This is not a Class diagram. And these classes donot directly interact with each other)

* **Components of CPE**

1. **Gene Tagging System Collection Reader**

This reads the input file and converts into a form the Annotator can understand.

This outputs annotations of the type – SourceText.type (GeneTaggingTypeSystem)

1. Type System – Gene Tagging Type System.xml
2. Descriptor File - GeneTaggingCollectionReader.xml
3. Class File – GeneTaggingCollectionReader.java
4. **Gene Tagging System Abner NER Analysis Engine**

This annotator inputs the SourceText Annotation Types which consists of the Sentence ID and the sentence to be annotated. It outputs Gene Type annotations which are then fed to the CAS Consumer.

1. Type System – Gene Tagging Type System.xml
2. Descriptor File - GeneTaggingAbnerNER.xml
3. Class File – GeneTaggingAbnerNER.java
4. **Gene Tagging System CAS Consumer**

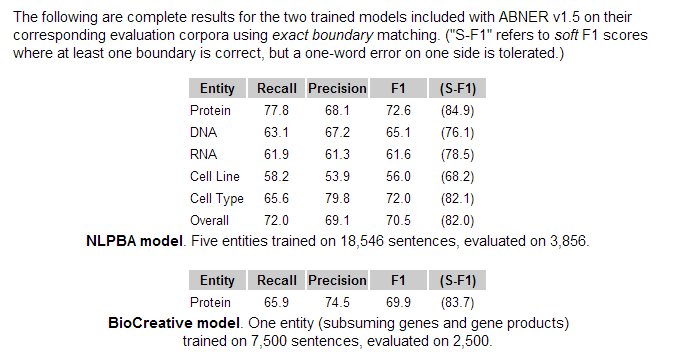
This consumes the CAS and persists it. In our case it writes out all the annotations on to an output file.

1. Type System – Gene Tagging Type System.xml
2. Descriptor File - GeneTaggingCasConsumer.xml
3. Class File – GeneTaggingCasConsumer.java

* **Abner – A Biomedical Named Entity Recognizer ()2**

ABNER is a software tool for molecular biology text analysis. At ABNER's core is a statistical machine learning system using linear-chain conditional random fields (CRFs) with a variety of orthographic and contextual features.

**Performance of Abner**



Abner has been created by [Burr Settles](http://www.cs.wisc.edu/~bsettles/), Department of [Computer Sciences](http://www.cs.wisc.edu/), University of Wisconsin-Madison.

For the implementation of this project I have used the NLPBA model of Abner to annotate and tag gene names from the source text.

We used the following API from the Abner jar file to annotate our document –

|  |  |
| --- | --- |
| API | Description |
| Tagger() | Basic Constructor. Loads the NLPBA method by default |
| public java.lang.String[] getEntities(java.lang.String text, java.lang.String tag) | Returns only segments corresponding to the entity provided in the tag argument (do not us "B-" or "I-" prefixes). |

The following tags have been used to annotate through the document –

1. Protein
2. DNA
3. RNA
4. CELL-LINE
5. CELL-TYPE

We did no train the model ourselves. We used it as it is to obtain our annotations.

* **Data Flow of the System**

Phase 1 – Collection Reader

The collections reader inputs the file and reads it line by line and breaks it into sentence ID and source text (to be passed on to the AE). This then stores in SourceText type annotation objects which are then indexed by the JCas.

Phase 2 – Analysis Engine (AE)

The analysis engine gets the indexed SourceText type annotations from the JCas and annotates over it trying to find the gene mentions. It then stores all the gene mentions it can in the Gene type annotation objects which are again indexed by the JCas. We use External Abner jar to annotate over the SourceText type annotation obejects/

Phase 3 – CAS Consumer

It inputs the gene type annotation objects indexed in the JCas object and simply writes it to the output file.

* **Design Patterns**

Some design patterns I have made of use while developing the system are as follows –

* 1. Controller Pattern

The Collection Processing Manager is responsible for firing up the system, reading the CPE descriptor file given to it, constructing the flow of the system by calling the various components of the CPE which will annotate over documents and perform the clean up when the system shuts down. We as developers only worry about the logic to annotate over the documents. Rest if the system events are all handled by the CPM controller.

* 1. Creator Pattern

I create SourceText annotation objects in the CollectionReader because it has access to the input file and thus have the information which is to be passed on to the annotations. Meanwhile the Gene Type annotation objects are created in the Analysis engine which is responsible for communicating with the external Abner jar to give gene annotations.

* 1. Information Expert Pattern

Responsibilities in the system have been divided in such a way that only classes which have the necessary information to perform the work are given responsibility of actually doing it. For example, the CollectionReader has access to the input file, so it is given the responsibility to read the input file and break it into sentence and sentence ID types and pass it on to the JCas. On the other hand the responsibility of annotating the sentences to identify gene name mentions is being passed on to the Analysis engine which has access to the External Abner jar. Finally the responsibity of the storing my result is given to the CAS Consumer which writes the data on to the output file.

* 1. High Cohesion
  2. Low Coupling

The components do their work and save their outputs in the global JCas object which can then be read by other components to do their jobs. So changing implementation of one component will not affect the other components. If you see my change history in the github you will see that I changed the implementation of my analysis engine multiple times, but every time I could simply plug in my new analysis engine and run the framework as before.

* 1. Polymorphism

We are making extensive use of Polymorphism in this project. One example would be when we get the annotation indexes from the JCas in the Analysis Engine. We then iterate over these using the hasNext() and getNext() methods which we have overridden in out annotation class. So simply by changing the hasNext(), getNext(), getProgress() methods we can iterate over different annotation types in our project.

* 1. Indirection

We can think of Cas Consumer and Collection reader as sort of adaptors which help us avoid direct coupling between the AE and the input/output implementation of the system.

* **Limitations of the system**

1. My system is constrained by the efficiency of the Abner NER and the StanfordCoreNLP. While performing my preliminary tests I found that I was getting lesser gene mentions than the sample.out file provided in homework 1. Had I used a more efficient system I could probably get more gene mentions in my system.
2. My system is constrained to take in inputs only through the file system and output also on the filesystem . Ideally we should be able to handle different input and output options.

* **Citations**

1. UIMA overview and SDK setup - <http://uima.apache.org/downloads/releaseDocs/2.2.2-incubating/docs/html/overview_and_setup/overview_and_setup.html#ugr.ovv.conceptual.applicaiton_building_and_collection_processing>
2. Abner – A biomedical Named Entity Recognizer - <http://pages.cs.wisc.edu/~bsettles/abner/>