Name: Venkata Puneet Ravuri
Andrew ID: vravuri

Design and Implementation Report

1. Abstract

This project deals with annotating gene names from an input dataset using named entity recognizers and rule based grammars. The input data will be a document containing multiple lines of biology specific data, each line identified by a unique formatted ID. The goal of the project is to identify these genes for every line and present an output of all the identified annotations.

2. Design and Implementation

The UIMA interface provides necessary framework for the design. There are three major components in the design – Input parsing, analyzing the input to create annotations and finally printing the annotations to an output source. These three tasks are to be performed serially in the correct order.

Input Parsing:

Input parsing has been implemented using the Collection Reader framework of UIMA. One collection reader has been implemented that takes input file as a parameter and parses the document. It then uploads the document as a string to the Common Analysis System (CAS). It does not perform any validations on the data. Its responsibility is restricted to loading the data into memory (CAS data structure).

Gene Name Annotations:

Creating annotations is a complex process that has to be done in multiple phases since the document has to be analyzed in multiple perspectives. UIMA supports an Aggregate Analysis Engine for this purpose which has been used for this project. The process of annotation has been divided into three phases:-

- Identifying annotations based on certain keywords which frequently occur in protein names.
- Identifying annotations based on abbreviation patterns occurring in sentences.
- Identifying annotations related to protein p53 specific gene names since they follow specific grammatical rules.

For each of these phases, a separate analysis engine has been implemented. These engines are independent of each other and add the respective annotations to the CAS. The aggregate analysis engine holds references to these three “primitive” analysis engines. To control the flow of calling the analysis engines, UIMA provided Flow Controller interface has been used. A flow controller

has been implemented that calls the analysis engines one after the other (since there is no specific dependency between each of the analysis engines). The aggregate analysis engine refers to the flow controller to call the individual analysis engines.

The individual analysis engines perform the annotations. The Stanford NLP has been used to identify named entities within the strings. The frequently occurring keywords like “protein”, “vasopressin”, etc will be used along with named entities to identify the gene names. Generally, gene names are camel-cased and some of them are alphanumeric words. These heuristics are taken into consideration when recognizing the gene tags.

Validation and Output:

The third and final component is the validation and output of the annotations. For this purpose, UIMA provided CAS Consumer interface has been used. The implemented CAS consumer in this project performs the below operations in the mentioned sequence:-

- Collects all the annotations from CAS object
- Sorts them based on the unique line code of the sentences
- Removes duplicate annotations if present
- Prints them into the file that has been taken as a parameter

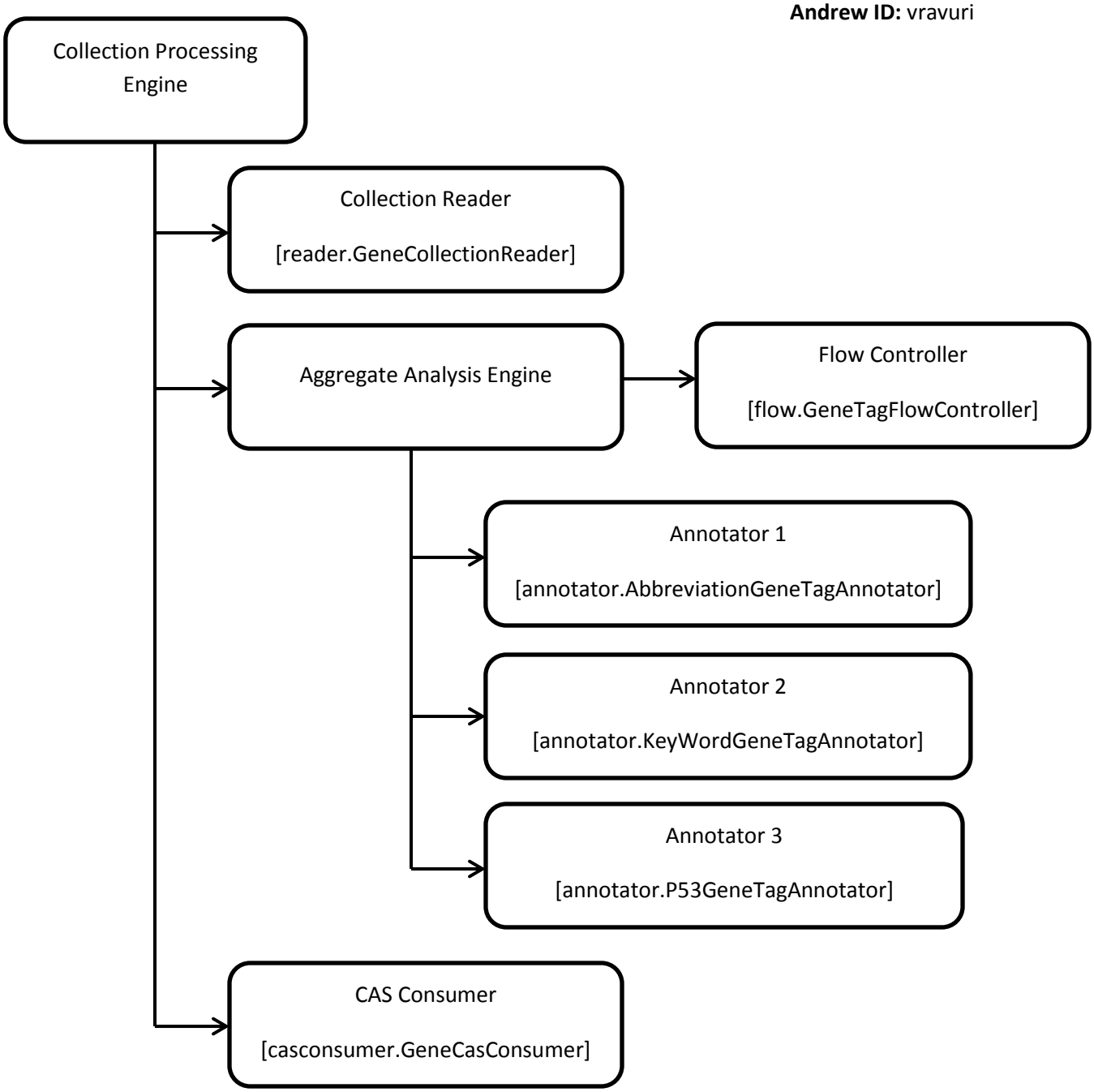
For the sorting functionality, a comparator for the defined annotator type has been implemented. The comparator compares two annotations based on the unique line code of the sentence. The sort method of Collections interface is used to sort using the custom comparator. Duplicate annotations can exist since multiple independent annotators act on the same input file and there are chances that the same annotation can be picked by more than one annotator (very less chance though). The CAS consumer initializer opens an output writer which will be used to write the annotation as an output.

Program Flow:

UIMA provides a collection processing engine (CPE) which is the main driver of a UIMA based application. This component takes the details of the collection reader, analysis engine and the CAS consumer for the application. The flow of control of the program has been designed below.

Each block of the below diagram refers to one main component of the program. The implemented Java class for each component has also been specified except for the collection processing engine and aggregate analysis engine since they aggregate the functionality rather than provide implementation.

Name: Venkata Puneet Ravuri
Andrew ID: vravuri



Data Types:

For every annotation, there are four parts:

- Unique code of the line from where the annotation is present
- The begin offset of the annotation
- The end offset of the annotation
- The annotation string

To store these artifacts for every annotation, one data type called "GeneTag" has been created in the type system. The same type system has been used by all the analysis engines since all the annotators deal with same type of annotations.

Utilities:

The project uses three analysis engines that deal with annotations and act on the same data structure. Hence, some of the functionality will be common. To avoid repetition of such code, a utilities package has been created. Some of the modules that have been added to the utilities classes are:-

- Stanford NLP to get the names entities of the strings.
- Metadata creation for each string. For every line of input that matches a particular annotator's criteria, certain metadata is calculated that will ease the computations. For instance, start index of each word and end index of each word. These calculations are done only for one string at a time so that memory size doesn't increase.
- Calculating the adjusted indexes for each word. This module calculates the index in a line without white spaces.
- Adding the annotation to CAS object. After calculating all the required information for an annotation, the information is passed to this module that initializes an object of Annotator type and adds it to the CAS.

Packaging:

Separate packages have been used for annotators, CAS consumer, flow controller, collection reader, type system and utilities since their functionality is unique from each other. The same package system has been reflected in the descriptors defined for each UIMA component for ease of correlation between a descriptor and its implementation.

3. Testing

For testing the output, the primary task is to compare it with the gold output (sample.out) file provided since it is mostly related to identifying string patterns. For this purpose, bash scripts were written to compare the gold output and program output. For instance, while testing p53 protein name annotator, the occurrence of the string "p53" in sample.out and program output have been retrieved using bash scripts and then compared to check for accuracy. Also, each annotator was tested separately since there is no dependency among annotators. After the output of each annotator was satisfactory, combined testing was done to check the running of the program.