11-791 Design and Engineering of Intelligent Information System & 11-693 Software Methods for Biotechnology Homework 2

Logical Architecture and UIMA Analysis Engines Design & Implementation

Important dates

- **Hand out: September 24.**[a]
- **Turn in: October 8.** In this homework, you will implement the logical architecture for the sample information processing task from the previous homework: *Gene Mention Tagging*. Based on a standard type system we give you in the new archetype, you will need to create analysis engines and annotators to annotate the sample input files.

  Note that, unlike the previous homework, we ask you to implement more than one UIMA annotator. These annotators will function as a part of an Aggregate Analysis Engine (AAE). Another important difference is that performance of your AAE is a substantial part of your homework. To produce a Collection Processing Engine (CPE) from your AAE, you also need to add a collection reader and a CAS consumer (as well as to write a top-level CPE descriptor).

  You should organize your project as shown below:

```
hw2-ID
|- pom.xml
`- src
   `- main
      |- java
      |  `- **/*.java      /* Java classes generated by JCasGen
      |                        and your UIMA annotators         */
      `- resources
         |- CpeDescriptor.xml     /* the main CPE  descriptor (the entry point) */
         |- **/*.*                /* analysis engine and other resources */
         `- docs
            `- hw2-ID-report.pdf  /* your report for design */
```

[a]This version was built on October 7, 2014

Several notes about organizing your Maven project and other additional information:

1. **Submission:** The same way as you did for Homework 0 and 1 (set up GitHub repo, create Maven project, write your code, submit to Maven repo), except that the name has changed to hw2-ID.

2. **Your report for design:** We expect to see how you design the logical architecture for the sample information system in your report. We will pull out your documents from your jar files. Remember to include your ID as part of file names, and put your name and ID in your document. Please submit in PDF format only.

3. **Javadocs:** Please remember to give an appropriate description for each annotator you create.

4. We expect that most of the general communication between the instructor team and students will take place on Piazza `https://piazza.com/class/hyvsubeilei6dd`. For private questions, e.g., regarding grades, you may contact instructors by e-mail. Your friendly TAs are: Avner Maiberg (`amaiberg@andrew.cmu.edu`), Parag Argawal (`paraga@andrew.cmu.edu`), Leonid (Leo) Boytsov (`srchvrs@cmu.edu`), or Xuezi (Manfred) Zhang (`xueziz@andrew.cmu.edu`),

Task 1

Implementing A Simple Information Processing Task with UIMA SDK

In this task, you need to create several analysis engines (as well as respective implementations of annotators) and combine them in a single Aggregate Analysis Engine (AAE) (in turn, you bundle up your AAE with a collection reader and a CAS consumer to obtain a full-fledged CPE). You may find Apache UIMA Manuals and Guides helpful as they provide step-by-step guidelines to create AEs and CPEs.

Task 1.1 Creating Maven project from the archetype

For this task, we have prepared another archetype to help you quickly build your project. The tutorial for Homework 1 might help you create a Maven project from an archetype.

For Homework 2, you need to add the following Catalog URL

https://raw.githubusercontent.com/amaiberg/DEIIS-hw2-archetype/master/archetype-catalog.xml

The archetype for Homework is

hw2-archetype

Also remember that the **Group Id** and **Artifact Id** for Homework 2 are

edu.cmu.lti.11791.f14.hw2

and

hw2-ID

with ID being your Andrew Id.

Similarly, you need to edit the `pom.xml` file to provide the SCM information related to your GitHub repository (it should be different from the previous homework).

Unlike Homework 1, we provide a standard type system that you should use. This type system is represented by an XML file (`deiis_types.xml`). In addition, we provide Java classes generated from this file (using `JCasGen`).

Task 1.2 Adding analysis engines for your pipeline

You are required to use the type system we included in the archetype to accomplish this homework. However, if your implementation needs additional types (e.g., to store intermediate data), you are welcome to extend the type system (don't forget to regenerate Java classes in this case you need to use `JCasGen`).

The ultimate goal is (1) implement a multi-annotator collection process engine (CpeDescriptor.xml) that annotate the inputs; (2) evaluate the performance of the aggregate analysis engine by comparing an output of your system against the gold standard. Multiple annotators should be bundled up into an aggregate analysis engine, which functions as a part of a collection processing engine. Remember that scoring in this assignment puts more weight on performance!

We will evaluate your implementation using held-out data.

- The input file should be `hw2.in` ;
- The output file should `hw2-ID.out`, where `ID` is **your Andrew ID**;
- The multi-annotator engine should be defined by the CPE descriptor `CpeDescriptor.xml`. Please, place it in the folder `src/main/resources/`.

You can test your multi-annotator CPE/AAE by running the UIMA Document Analyzer. One option is to use the training data from Homework 1, but you are also encouraged to obtain other test sets.

We provide an Eclipse configuration to run the document analyzer, which is `run_configuration/UIMA Document Analyzer.launch`. To use this configuration in Eclipse, navigate to the project sub-folder `run_configuration`, and right click on `UIMA Document Analyzer.launch`. Then, select `Run As`.

Here are some suggestions to consider.

1. Remember to add comments and Javadocs to your annotators, we will also evaluate the quality of your code.
2. If you want to employ a resource (e.g., a model you trained offline) in your annotator, you could consider UIMA's *resource manager* (refer to the official tutorial for details about this). Be sure to put your resource in `src/main/resources` so that your this resource will be properly bundled up with your code during the submission/release process.

3. If you want to incorporate other NLP or machine learning tools, **do not use non-Java** packages, as we will not be able to use them on an evaluating server. However, we might be able to post jar packages to our 3rd party repository (and make them available to you via maven).

4. The most creative part is to implement specific annotators. We outline some possible solutions (in Homework 1). Note that an aggregate analysis engine may contain only a single primitive analysis engine (and a single annotator). Yet, it is **best to implement multiple solutions**.

5. All the annotations should be kept in the CAS until a final "merging" component reads all the annotations and selects only the best ones (or all of them as one option). You should mark gene mentions using the type *edu.cmu.deiis.types.Annotation*. The *casProcessorId* feature can be used to indicate a type of the annotator that have tagged a gene name, and the *confidence* feature can be used to indicate an estimated annotation quality (the higher is the confidence value the better is the annotation).

   Note that many statistical NERs produce such confidence values. For rule-based annotators, you can use some ad hoc fixed value, e.g., one. You can use these confidence values to aggregate results from several annotators. For example, you can employ a voting procedure in which confidence values are used as voting weights. If a single annotation is produced by multiple annotators it will have a larger weight (a sum of individual weights from all contributing annotators) and a better chance to be selected by the aggregating algorithm (which, e.g., can use a threshold to reject low-quality annotations). This approach was used in IBM Watson system.

Task 1.3 Writing up your report

We expect you to describe the features of your system (and its design) in good detail. We also remind you about printing your name and Andrew ID in the beginning of the document. Please, name the report file as "hw2-ID-report.pdf" and put it under `src/main/resources/docs`.

Grading Guidelines

Designing and Implementing UIMA Analysis Engine (60 pts)

- Basic requirement: the system should work (20pts). If the system does not work or misconfigured you may lose points. For example, you will lose a few points if we need to edit your descriptors **to specify the correct input file**. If the system does not work and we cannot make it running (**we do not have to try very hard!**), you will get **zero** points.

- Additional 10 pts will be given for interesting design solutions. For example, you may get extra credit for an interesting type system, which, e.g., employs inheritance. Beware not to over-engineer! You can be penalized for introducing unnecessary complexity.

- Additional 30 pts can be earned for performance of your system. The f-score of 0.7 will earn you 30pts. If you get a lower or higher score, the nominal score of 30 pts will be scaled linearly (e.g., 15 for the f-score of 0.35). It is possible to get more than 30 pts here, but we do not many students to perform this well.

Documentations, comments, coding style (30 pts)

We will check a completeness of your report (you should provide all necessary examples/explanations and address all our questions) as well as the quality of your Javadocs (need to provide sufficient documentation for users).

- A basic documentation and report get maximum 20 pts. Additional 10 pts will be given for extra quality (surprise us). Some points will be subtracted if, e.g., Java docs are missing.
- To get the maximum score of 30 pts one needs to excel at the following:
  1. discussing of design aspects of the problem;
  2. discussing of algorithmic aspects of the problem, in particular, at comparing several alternative solutions;
  3. applying interesting yet simple design patterns (over-engineering is an evil!);
  4. providing concise yet complete documentation;
  5. providing other insights not covered by the previous items.

Submission, folder structure, and name convention (10 pts)

We will check the correctness of your submission, folder structure, and the code.

- We expect your homework to be located at the right repository and your project folders/files be organized as outlined in the beginning of the assignment (5 pts).
- The names of your types and files should follow Java naming convention (5 pts). See, e.g., this document for details on naming conventions.