COM-35702 Design and Engineering of Intelligent Information Systems
Homework 1

Logical Data Model and UIMA Type System Design & Implementation

Important dates

- **Hand out: January 31.**[a]
- **Turn in: February 14.** In this homework, you will create a logical data model for a sample information processing task. Based on your analysis of the requirements, you will need to submit a report to describe how you design an appropriate UIMA type system to model the required information types. Then, you will use the UIMA tooling in Eclipse to create a type system `.xml` file; use the `JCasGen` plugin to compile your type system into Java type classes.

  If you have ever looked into your `target/` directory or Maven course-release repository for Homework 0, then you will find that Maven will package the binary files, source files, and Javadocs into different jars, and deploy them on the server, which means all you need to do is to put all the source codes and descriptors, as well as the documents, in the right place of your hw1-ID project. You should organize your project in the same hierarchy as shown below[b]:

```
hw1-ID
|- pom.xml
`- src
   `- main
      |- java
      |  `- **/*.java      /* Java classes generated by JCasGen */
      `- resources
         |- hw1-ID-typesystem.xml /* your type system */
         |- **/*.*                /* (optional) other resources */
         `- docs
            `- hw1-ID-report.pdf  /* your report for design */
```

[a] This version was built on February 1, 2014

[b] To simplify your submission process and our evaluation process, we ask you to create `src/main/resources/docs` for your documents. But you should keep in mind it is NOT a good practice to have documentation within a `src` folder.

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

1. **Submission:** The same way as you did for Homework 0 (set up GitHub repo, create Maven project, write your code, submit to Maven repo), except that the name has changed to hw1-ID.

2. **Your source files and type system descriptor(s):** `**/*.java` and `**/*.*` tell you that you don't need to flatten your folder hierarchy, instead we encourage you to place your Java codes in the right package by specifying the right package name for each type you defined in the `hw1-ID-typesystem.xml` file. You can also create sub type systems and import them into your main type system. We will look into your jar packages.

3. **Your report for design:** We expect to see your requirement analysis 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. If you think it might be easier to describe your design with a UML diagram, you can optionally put one in your report.

4. **Javadocs:** Please remember to give an appropriate description for each type you create and feature you add, which will end up being the Java comments in the Java classes generated by the `JCasGen`. You might want to refer to any best practice (yes, there might be more than one) to write your Javadoc, e.g., `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`. We expect you to describe your major components at the class level of your Javadoc, and put additional comments on the most important methods if any.

Useful information

1. We encourage you to start Homework 1 as early as possible, especially if you haven't got a chance to browse the UIMA portal.

2. Please post your questions regarding Homework 1 on Piazza `https://piazza.com/itam.mx/spring2014/com35702`. For other issues or concenrs, you can also send us mails to, Elmer Garduno (`elmer.garduno@itam.mx`).

3. Again, both source files and pdf file of this assignment are publicly available on GitHub

   `http://github.com/ziy/software-engineering-preliminary`

   Please feel free to fork the project and send a pull request as some of you did for Homework 0 for any error. Or you can just report an issue at

   `http://github.com/ziy/software-engineering-preliminary/issues`

Task 1

Designing Logical Data Model for a Sample Information Processing Task

In this task, you will create a logical data model for a sample information processing task. Analyze the requirements described below and design an appropriate UIMA type system to model the required information types. In addition, all annotations must record the name of the component that produced the annotation, and the confidence score assigned to the annotation by the component.

Task 1.1 The Sample Information Processing Task

Given a question and a set of sentences, determine which sentences answer the question:

1. Assign a score to each sentence [0..1];
2. Rank the sentences according to score;
3. Select the top N sentences where N = the number of correct answers;
4. Measure performance by Precision@N (how many of the top N are correct).

Task 1.2 Input

The input to the pipeline is a test element (text file), e.g.:

```
Q John loves Mary?
A 1 John loves Mary with all his heart.
A 1 Mary is dearly loved by John.
A 0 Mary doesn't love John.
A 0 John doesn't love Mary.
A 1 John loves Mary.
```

Each line in the file is either the question or a candidate answer.

Questions are of the form: Q <question>.

Answers are of the form: A <isCorrect> <answer>. For <isCorrect>, '1' indicates the answer is correct, and '0' means not.

Task 1.3 The Processing Pipeline

The information processing pipeline will consist of the following steps:

1. **Test Element Annotation**: The system will read in the input file as a UIMA CAS and annotate the question and answer spans. Each answer annotation will also record whether or not the answer is correct.

2. **Token Annotation**: The system will annotate each token span in each question and answer (break on whitespace and punctuation).

3. **NGram Annotation**: The system will annotate 1-, 2- and 3-grams of consecutive tokens.

4. **Answer Scoring**: The system will incorporate a component that will assign an answer score annotation to each answer. The answer score annotation will record the score assigned to the answer.

5. **Evaluation**: The system will sort the answers according to their scores, and calculate precision at N (where N is the total number of correct answers).

Task 2

Implementing Logical Data Model with UIMA SDK

In this task, you need to create a type system based on UIMA SDK and use the `JCasGen` plugin to compile your type system into Java type classes. We assume you have finished all the reading assignments up to now, including the UIMA specification, and browsed the Apache UIMA project website. You will find *Apache UIMA Manuals and Guides* (`http://uima.apache.org/documentation.html#manuals_and_guides`) will be helpful for you to finish this task. The *Tutorials and Users' Guides* gives you a step-by-step reference to create a type system.

1. After you install the UIMA Eclipse plug-in, you will need to create a UIMA project with a type system from an archetype.
2. You might want to follow the illustrative instructions from *Tutorials and Users' Guides* to compose the type system descriptor and Java classes.

Task 2.1 Installing UIMA SDK

Similar to the installation processes you have gone through for Git and Maven, you will install the UIMA binaries and a UIMA Eclipse plug-in. Different from the installation instruction you have learned from class or the official tutorial, you **DO NOT** need to download the UIMA SDK, uncompress and copy the jars into the Eclipse installation path. In fact, the Maven archetype `hw1-archetype` that we built for this task makes UIMA SDK ready out of the box for you, and the Eclipse plug-in for UIMA can be installed with the Eclipse **Install New Software** function, which is the same as described in the official tutorial.

1. As mentioned in the class and the official UIMA tutorial, you should install Eclipse EMF plug-in, but if you strictly followed the instruction from last homework, then EMF should have been installed as it is a default component of Eclipse Kepler for Java Developers. Otherwise, you should install it by yourself.

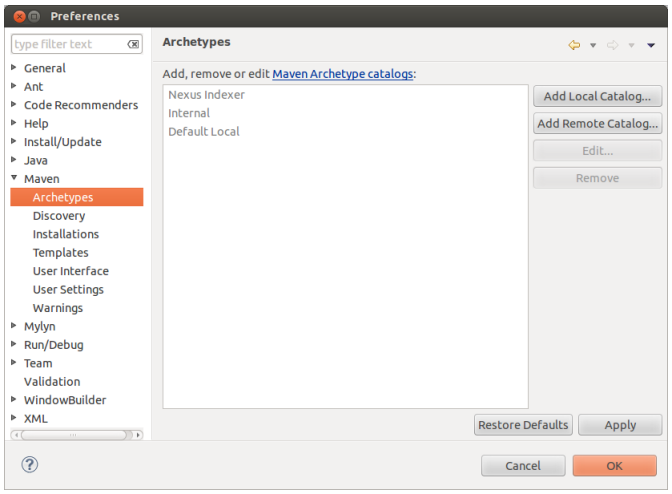

Figure 2.1: Configuring Maven catalog

2. Follow the instruction in subsection 3.1.2[1] of the *Overview & Setup* section of *UIMA Manuals and Guides* by adding the UIMA Eclipse Plug-in Update site (`http://www.apache.org/dist/uima/eclipse-update-site`)

3. After installation, you are able to create or edit UIMA descriptors with a nice GUI.

Task 2.2 Creating Maven project from the archetype

For this task, we have created an archetype for you to base on, which means you don't need to manually input shared information, e.g., configurations for the Maven repositories, compile plug-in, etc. The tutorial for Homework 0 might also be helpful for you when you are creating a Maven project.

1. The first thing you should do is to inform m2e where to find the archetypes. Open your Eclipse's **Preferences** window, and navigate to **Maven** → **Archetypes**, and click **Add Remote Catalog...**. (See Figure 2.1)

2. Type the following URL into the **Catalog File** field.

   `http://oaqa.github.io/DEIIS-hw1-archetype/repository/archetype-catalog.xml`

   Optionally, you can add a **Description** for this catalog, for example "Course Catalog". See Figure 2.2. Then click **OK** on the **Remote Archetype Catalog** window and another **OK** on the **Preferences** window.

[1] `https://uima.apache.org/d/uimaj-2.4.0/overview_and_setup.html#ugr.ovv.eclipse_setup.install_uima_eclipse_plugins`

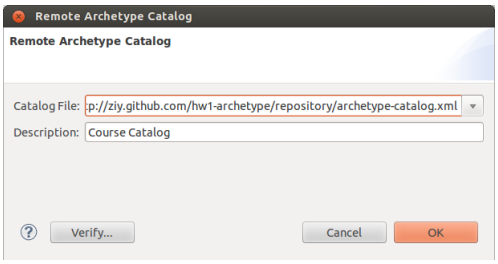

Figure 2.2: Adding a remote catalog

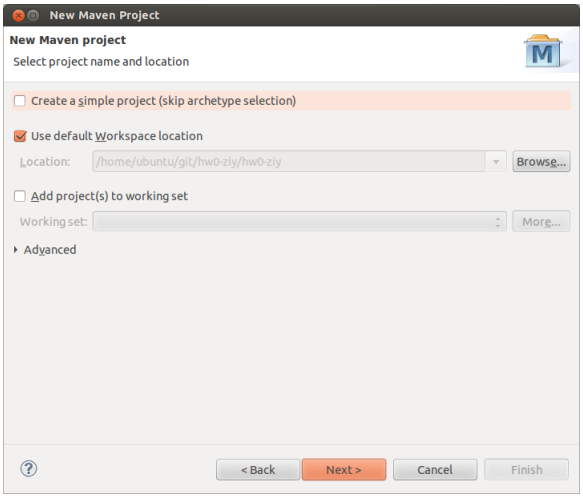

Figure 2.3: Unselecting “Create a simple project”

3. Now you can follow almost the same steps to create a Maven project hosted on GitHub, you can refer to Homework 0 to find out how to create an empty project on GitHub, and how the project can be imported to Eclipse. Since we have created the archetype for you, remember to unselect **Create a simple project (skip archetype selction)** (see Figure 2.3). Then click **Next**.

4. Here you can select “Course Catalog” (or other names you specified in the previous step) or “All Catalogs” in the drop-down menu for **Catalog**. Then, type in “hw1-archetype” (without quotes) in the **Filter** field. While you are typing, you will find the progress bar indicates you that it is busy “Retrieving archetypes”. Select the archetype listed below, and click next to continue. See Figure 2.4. Note that since we will continue improving the archetype or fixing bugs if any, the version number cannot different from what you see in the figure.

5. In the next window, you are asked to specify the **Group Id** and **Artifact Id**. Similar to Homework 0, the Group Id is

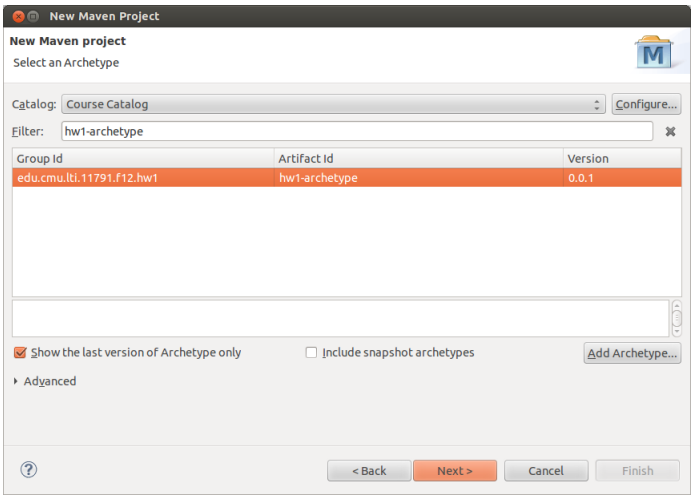

Figure 2.4: Filtering the archetype you want to base on

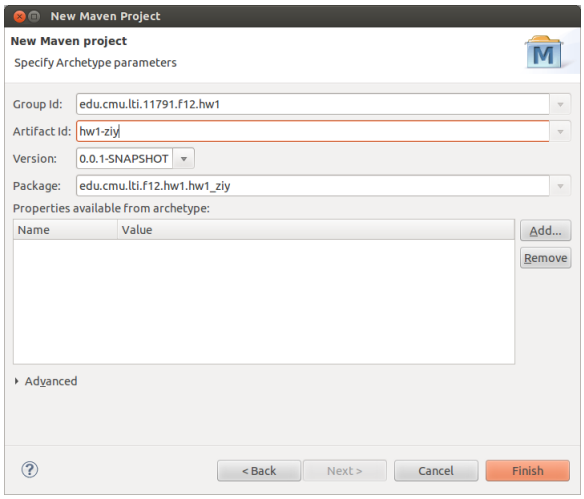

Figure 2.5: Specifying artifact parameters

edu.cmu.lti.11791.f13.hw1

and Artifact Id is

hw1-ID

with ID being your Andrew Id. Then click **Finish**. See Figure 2.5.

6. Now you have a new Maven project created from the archetype, which means there will be no extra steps to manually edit the `pom.xml` file on your own. Instead the archetype includes most information Maven needs to execute goals.

   The only information that Maven doesn't know about from archetype or the information we have typed to create the project is SCM. You need to edit the `pom.xml` file to type in the SCM information of your GitHub repository for Homework 1 as you did in Homework 0.

You can see that we have included

- and two files in the `src/main/resources/inputData` folder, `q001.txt` and `q002.txt`, which correspond to two sample input files.
- the `pom.xml`. You can go to the **Dependencies** tab after you double-click the pom file, and you will be able to see all the UIMA SDK packages have been added to your Maven project by the archetype, and they will be stored on your local Maven repository (e.g., `/.m2/repositories`). You will not need to follow the official instruction to uncompressed the SDK package and specify the `UIMA_HOME` as your environment parameter unless you want to execute a UIMA program outside your project.

Your implementation starts here.

Q1 I found a bug in the archetype and I know how to fix it. How should I proceed to patch the archetype to help many others who may suffer from this?

A1 The archetype project is also hosted on GitHub at `https://github.com/oaqa/software-engineering-hw1-archetype`, and you can send a pull request to us regarding any issue.

Task 2.3 Adding a type system descriptor and Java classes for your pipeline

If you have one or several idea(s) for the sample information processing task already, it's time to implement all of them in this task. You will find the official tutorial is helpful for you to accomplish this task. Except that you are required to build your component based on UIMA framework, there is little limitation on how you should design and implement your type system. Nevertheless, there are still some suggestions (or you can say tips or our expectations) to consider.

1. The minimal type system to accomplish this task should include types for input and output elements.

2. If you intend to design a pipeline of two or more phases (e.g., each of the first few phases extracts a certain feature, and the last phase predicts the gene mentions based on all the features), you should include all the intermediate types in your type system as well.

3. If your type system is complex, then you can try to refactor the type system by categorizing the types according to their functions (e.g., NLP related, biological related, etc.), and creating multiple type systems for each of the categories.

4. Another common practise in the type system design is to create a *base annotation* type which includes two features: a string feature *source* and a numeric feature *confidence* to help keep track of where an annotation was originally made by, and how confidence the annotation was. All other types then inherit from this base annotation type.

5. You could assign a (or several) name space(s) for your types, e.g., instead of `Sentence` (default name space), you could name it `model.Sentence` (a type with name space `model`). Once you click the **JCasGen** button, you will find a nice hierarchical folder structure created under `src/main/java` for the Java classes corresponding to the types. We also recommend to follow the same naming convention for your artifact groups, Java packages, and type paths.

Task 2.4 Writing up your report

Once you test that everything works smoothly as expected, you need to summarize how your design and implement the type system for the task in your report.

There is no template for the homework reports, but we expect you to mention the system design from several architectural aspects (UML, type system, engineering, design pattern, etc.) and several algorithmic aspects (knowledge sources, NLP tools, machine learning methods, etc.).

Finally, don't forget to put your name and Andrew ID at the top of the document, name the file as "hw1-ID-report.pdf" and put it under `src/main/resources/docs`.

Appendix

Grading Guidelines

Designing and Implementing Logical Data Model (60 pts)

- We will evaluate if the type system for the necessary IIS steps and its overall structure are designed and implemented in a proper way, by looking at your codes (Java classes and descriptors) and report. Specifically, each of following items has 5 pts for design and 5 pts for implementation: Overall Structure, Test Element Annotation, Token Annotation, NGram Annotation, Answer Scoring, and Evaluation.
- If the basic requirement (i.e., it works!) for each of the items is met, then the student can get 3 points, otherwise 0 to 2 points will be given.
- If the design of each component is considered more carefully (according to the suggestions given in the homework instruction or outside), then 4 or 5 pts points will be given. For example, the type system is implemented in an inherited way.

Documentations, comments, coding style (30 pts)

- We will first look into your report to see if it is complete, if you have given explanations (examples) and solutions to all our concerns (20 pts), and then we will take a look at the Javadoc package the Maven automatically generated for your project to see if it covers all the necessary information that potential users of your package need to know (10 pts).
- Basic points is 6 pts (12 pts for the report), if some important part is missing, the 0 to 5 points (0 to 11 pts for the report) might be given.
- A detailed document can be given 7 to 10 points (13 to 20 pts for the report), for example,

    1. good insights to the problem or architecture design covered in the report,
    2. include UML design or other forms of software design in the report or Javadoc,
    3. apply design pattern in an appropriate way,
    4. detailed comparison of more than one methods in the report,
    5. interesting discovery,
    6. etc.

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

- We will first check the correctness of your submission (5 pts), and we will look into your report, code and descriptor to see whether they follow the normal Java name convention [2] (5 pts).
- Specifically, we expect your homework is located at the right repository and your project folders/files are organized as required (5 pts). 0 to 4 points might be given if you submitted to wrong path and/or badly organized folder/files.
- The names of your types and files should follow the normal Java name convention (5 pts). Otherwise, 0 to 4 points will be given for bad naming.

[2] `http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html`