11-791 Homework 3 Report

Chen Sun

chens1@andrew.cmu.edu

10/05/2013

Homework 3 is largely based on Homework 2. Thus this report focuses mainly on the CPE part and UIMA-AS service implementation. Also, an integrated service using Standford NLP package is also involved as part of the homework report.
***Please Read the README in the package to get more info about different CPEs.**

1. Running CPE

1.1 CPE is composed of three parts: **the Collection Reader, the Analysis Engine and the CAS Consumer**. For this homework, the collection is only based on local files, thus it is quite useful to import FileSystemCollectionReader.xml from /examples/descriptors/collection_reader/ of UIMA binary sources.

1.2 As for the consumer, the default is to use xmiwriter provide by the UIMA example. If this is used by default, then the whole CPE runs just as a Document Analyzer. UIMA CAS Consumer typically takes the annotated CASes and does the work locally, including but not limited to persisting the files, batching level processing and collection level processing like overall output. For example, our annotator could use the consumer to output the overall average precision when all the files are read and processed.

1.3 As for the Analysis Engine part, we can simply use the service provided by our own AAE annotator or remote service through a **UIMA-AS** client to run the service.

1.4 By default, there can be multiple AAEs and multiple CAS consumers to be used in the pipeline. The AAEs and CAS added are used **sequentially**, thus the order of the AAEs and CAS Consumers are quite important to load and use.

1.5 In my homework, as has been mentioned above, the AAE part needs to be adjusted to fit the needs of the CPE. In my previous homework 2, for the purpose to output the results at

last, I used a **Score Printer Annotator** to do the output. However, in this homework, it is more suitable to put it as a CAS Consumer, since the output of our Analysis Engine is required to output the scores and ranked answers locally. However, if the AAE is deployed remotely and called through a client, it is impossible to see our desired output locally. Thus, to ensure the output locally, we need to "consume" the CAS at our local side and output the average results when all the processing of the batch is properly done.

2. UIMA-AS Client

2.1 UIMA-AS client descriptor represents a remote service that has been deployed and provided by others. Through a connection protocol and service endpoint, one can simply find the service on certain domain through certain port (usually 61616) and a service endpoint queue.

By doing this, the local client could call the service without knowing the low level implementation or knowing what exactly the execution environment or the programing language. This is very good for the service caller since it simplifies the running of the service and put more concern on the output, namely the annotation output. With the original file and the annotation, the local service now can go on with the processing provided by the the remote service. This implementation, enables the engagement of separate services to one service.

With the combination of different remote services, we now can develop our own based on this.

The nlp client provided by the class TA serves as an apt example. With service enabling the name entity annotation run on the server, I used it in my scoring mechanism and to run it with identifying the name elements order and passive state. Short introduction will be covered in the later chapters.

3. Deploy My UIMA Service

With the UIMA AS binary source available at hand, I can finish this part of homework with continuing trying .

3.1 Create the deploy descriptor. With written AAE to available, simply put in the aae deployment descriptor and specify the name of the endpoint, which is not necessarily the same as the AAE name.

3.2 With the descriptors, enter the binary folder and start the broker.sh to start the broker to listen on the port on which the

service will be deployed.
Secondly, use deployment script to deploy the service to register the service to the broker so that the broker now knows where to process the incoming CAS files. The default port goes to 61616, which can also be specified by the deployer.
Thirdly, to test the correctness of the code, we now need to call the service as if it were a remote service. Just start the CPE and write a new client to represent the remote service so that we now can read in the files, pass the CAS to the service (although they run on the same machine) and get the returning annotated service with the annotation already in them.
The service runs smoothly and the consumer successfully runs as if the AAE is run locally. Thie proves my client works normally.

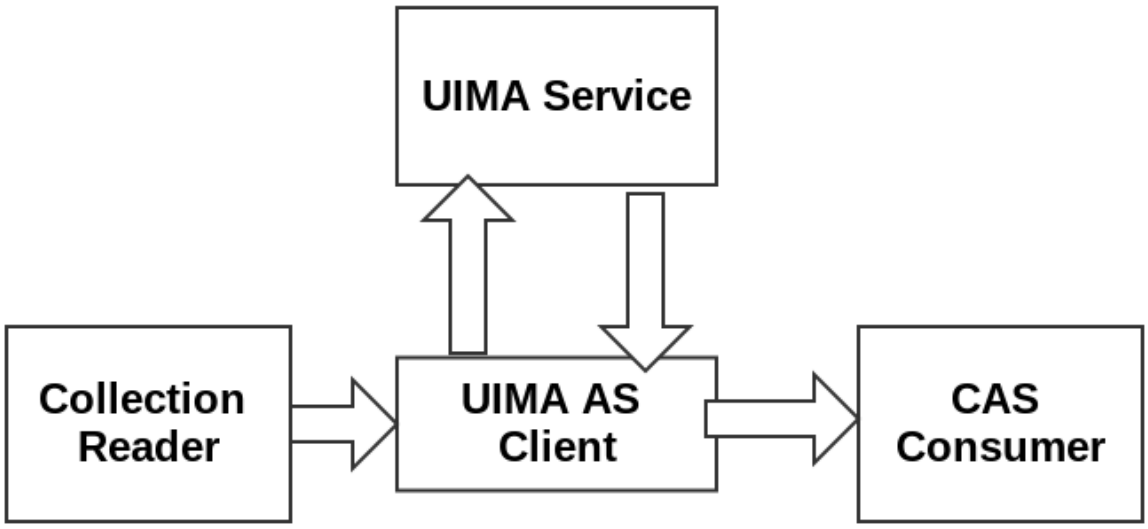
3.3 Implementation notes.

Although the whole process is simple, the general development and configuration is quiet difficult.
First of all, we need to configure the environment variable UIMA_HOME correctly.
Secondly with a MAVEN build, we can put all the necessary .jar files into the the dependency class folder, and thus set the UIMA_CLASSPATH here so that the service would know where to run the dependency.
Thirdly, we need to change all the XML files where "import by name" to "import by location" and the location path has to be relative path to the XML files so that we won't miss the fies if it is run on the server side or another machine as long as the structure remains the same.
Last but not least, we need to pack the the source files including the raw type files into a .jar file with the structure from src/main/java since the annotator descriptor only contains the file path starting there and put it in the UIMA_CLASS path. The deploy service strangely can only find the class in .jar files. If I put the nested files there, nothing will be recognized by it.

3.4 Monitor the service

To monitor the service, we can use jconsole to connect the given address and monitor the running of the service. When creating the service, we specify the instance numbers. By default it was set to 1. However, we can add it to more than 3 so that our service could handle multiple requests at one time.

4. UIMA AS Service Structure



I used a strcuture diagram to discuss it with my classmates how UIMA CPE and UIMA AS works. And I think it is quite right job. This shows how CPE works and of course the middle part can be replaced by other local service.

5. Incorporate Stanford Name Entity

To incorporate Stanford Name Entity, I created a local annotator named **edu.cmu.deiis.types.NameEntityAnnotator** to deal with the annotated JCas with NameEntityMention Annotation. My annotator treats the apperance of people's name as a key to detect the answer. When the same name occurs in the answer as they did in the question, record it temporarily. If the name entities have a "by" between them, I will treat them as a passive state and swap the position with its previous Name Entitiy. Finally compare them with original question Name Entities with both relative position and string content and give each match a score of 1/total Name Entities of quesion.
In this way, I got a 0.5 precision of Booth and Lincon sentence and 0.33 of the second. The average precision was 0.43, much lower than the N-Gram method. I checked it carefully and find out that it was due to the Standord NLP could not split the

short answer sentence such as “A 0 Booth shot Lincon”, which resulting the most obvioud answers ignored by the evalution system.

6. Comparing[bonus part]

I wrote a annotator descriptor named hw3-chens1-scnlp-local.xml to run the org.cleartk.* staff locally. The class automatically imports lots of classes and annotators to ensure it running smoothly.

The running time of this component of 9098ms while the client runs in 280ms. It is obvious that the running time has a close relationship with the JVM of the running machine. As I can see and infer, my laptop is much worse than the remote server, which resulting the much longer running time of the same componet. This, as a result, tells us one benifit of the UIMA SERVICE that you don’t need to have every key compent, you only need to know how to deal with annotated part and leave the complex annotation to the server. Intelligient machines never need to have all the part installed on it. Only network would help it a lot.

7. Discussion

However, there are still many concerns about UIMA-AS service. Firstly, how do we trust the UIMA AS service. We don’t see other behaviours UIMA AS does at the server side in addition to the annotation part. Or perhaps the UIMA AS does not include sensible information.

Secondly, We still need to add the dependency to our project if we would like to continue work on the UIMA AS client, this gives increases the coupliing of the project which goes against our original expectation.