Execution Architecture with CPE and Deployment Architecture with UIMA-AS
HW3 - Report

Gabriela N. Góngora S. - 139547

Task 1.1 Learning CPE

The first task in this homework is to learn about the Collection Processing Engine (CPE) in UIMA, so that we can execute homework 2's pipeline with it. The CPE has three main parts to it:

1) A collection reader.
2) An analysis engine(s). This part can contain one or many analysis engines, or even an aggregate analysis engine.
3) CAS consumer(s).

A collection reader is an interface to raw input data, and knows how to iterate over a data collection.
A CAS Consumer extracts the analysis results from the CAS and performs collection level processing.

Task 1.2 Creating and Running CPE

After reading and learning about CPE through the manuals provided in Task 1.1 of the homework description we proceeded to create and run our own CPE.

Step 1: Create a Collection Reader

For this specific homework we only needed to take into account a folder of files located on the file system, i.e. the implementation of FileSystemCollectionReader suffices. This reader converts each file in a file system into a CAS. Additionally it needs a "SourceDocumentInformation" type to be added (already stablished in UIMA project files) to store the file's uri and size in case of future implementations. This implementation comes under: org.apache.uima.tools.components.FileSystemCollectionReader and was copied to our project in the following folder:
/hw3-139547/src/main/resource/hw3-139547-fileSystemCollectionReader_Descriptor.xml
With its corresponding java implementation:

`/hw3-139547/src/main/java/edu/cmu/deiis/cpe/FileSystemCollectionReader.java`

Step 2: Create a CAS Consumer

There are various implementations of CAS Consumers within the UIMA project, but we decided to implement our own. The CAS Consumer extends parent class to CasConsumer_ImplBase, and we need to implement the "processCas" method. The requirements for this homework specify our CAS Consumer to be based on the Evaluator from homework 2, therefore it prints out the answer scores and the precision@N for each of the two files we are using to test our system. For this we had to add another type (precision) to the types system, in other to annotate the last part of the pipeline defined in homework 2.
Descriptor:

`/hw3-139547/src/main/resources/hw3-139547-casConsumerDescriptor.xml`

Corresponding java file:

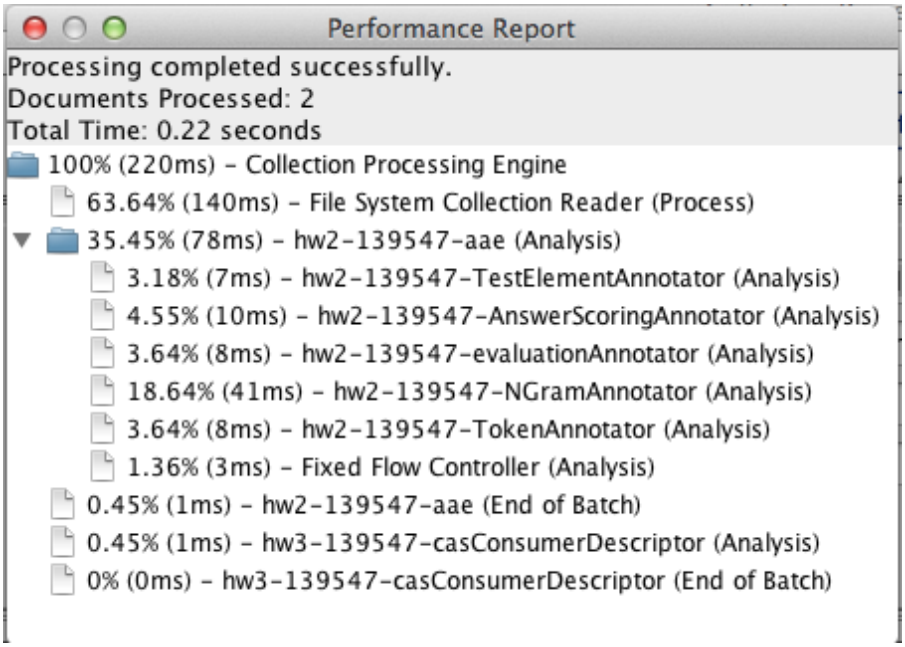
/hw3-139547/src/main/java/edu/cmu/deiis/cpe/CASconsumer.java

Step 3: Create a CAS Consumer

The CPE can be created in two manners: 1) With the CPE GUI provided by the UIMA project or 2) by having a java file that will help create the CPE descriptor. Actually, both approaches were tested on this homework and the resulting CPE descriptor is in:

/hw3-139547/src/main/resources/hw3-139547-cpe.xml

The CPE is constructed from the three previously explained sections. The CPE's Performance report is the following:



Task 2.2

The UIMA-AS is similar to the CPE concept, but in this case services are deployed for clients to call upon remotely. The UIMA-AS project can be found in UIMA's webpage and must be downloaded. Afterwards, in Eclipse, UIMA_HOME must be set to the UIMA-AS path. This project contains the same structure as the one we were previously using as our UIMA_HOME, but it has additional dependencies so that we can make use of the ActiveMQ broker in order to provide services.

Task 2.3

Step 1: Create a Deployment Descriptor

This descriptor was tricky, because if we created it with the wizard as a "Deployment Descriptor File" then we get an implementation with a timeout (that seems unremovable). This timeout will make our queue terminate if it is not being used in a certain amount of time, making it more difficult for some services testing to take place. Instead, the actual implementation of the deployment descriptor was created as a simple .xml file and

constructed upon the examples provided in the UIMA-AS project. The following xml code specifies everything we need to deploy:

```
<deployment protocol="jms" provider="activemq">
    <service>
      <inputQueue endpoint="hw3Queue" brokerURL="tcp://localhost:61616"/>
      <topDescriptor>
       <import location="hw2-139547-aae.xml"/>
      </topDescriptor>
    </service>
  </deployment>
```

The endpoint is our queue (which will be started in another terminal and explained in the following steps). The brokerURL needs to be specified, in this case we are running all services locally. Finally we provided the location for our homework 2 aggregate analysis engine.

Step 2: Start Broker Locally and Deploy Service

First of all, we need to start a broker (on localhost), inside hw3 project, through a terminal. For this part, in case you are running a MacOS system you need to add the required environment variables (UIMA_HOME and update PATH and CLASSPATH) in the .bash_profile, so that they are available at any time and the UIMA-AS services can run smoothly in terminal instances.

```
IBAGNOG:resources IBAGNOG$ startBroker.sh
INFO: Using default configuration
(you can configure options in one of these file: /etc/default/activemq /Users/IBAGNOG/.activemqrc)

INFO: Invoke the following command to create a configuration file
/Users/IBAGNOG/Desktop/apache-uima-as-2.4.0/apache-activemq-5.4.1/bin/activemq setup [ /etc/default/activemq
 | /Users/IBAGNOG/.activemqrc ]

INFO: Using java '/usr/bin/java'
Java Runtime: Apple Inc. 1.6.0_43 /Library/Java/JavaVirtualMachines/1.6.0_43-b01-447.jdk/Contents/Home
  Heap sizes: current=83008k  free=81611k  max=126912k
    JVM args: -Dactivemq.classpath=amq/conf;/Users/IBAGNOG/Desktop/apache-uima-as-2.4.0/apache-activemq-5.4.
1/conf; -Dactivemq.home=/Users/IBAGNOG/Desktop/apache-uima-as-2.4.0/apache-activemq-5.4.1 -Dactivemq.base=am
q
ACTIVEMQ_HOME: /Users/IBAGNOG/Desktop/apache-uima-as-2.4.0/apache-activemq-5.4.1
ACTIVEMQ_BASE: amq
Loading message broker from: xbean:file:amq/conf/activemq-nojournal.xml
INFO  BrokerService                 - Using Persistence Adapter: MemoryPersistenceAdapter
INFO  BrokerService                 - ActiveMQ 5.4.1 JMS Message Broker (localhost) is starting
INFO  BrokerService                 - For help or more information please see: http://activemq.apache.org/
INFO  ManagementContext             - JMX consoles can connect to service:jmx:rmi:///jndi/rmi://localhost:1
099/jmxrmi
INFO  TransportServerThreadSupport  - Listening for connections at: tcp://IBAGNOG.local:61616
INFO  TransportConnector            - Connector openwire Started
INFO  BrokerService                 - ActiveMQ JMS Message Broker (localhost, ID:IBAGNOG.local-57690-139605
4668434-0:0) started
```

Create Queue inside broker (previously running):
The script provided in hw3, called “run-as.sh” will provide the required commands to initialize a queue, given that our broker is already running.

```
export UIMA_CLASSPATH=./target/:./target/dependency/
deployAsyncService.sh ./src/main/resources/hw2-139547-aae-deploy.xml -brokerURL tcp://localhost:61616
>>> Setting defaultBrokerURL to:tcp://localhost:61616
Service:hw2-139547-aae Initialized. Ready To Process Messages From Queue:hw3Queue
Press 'q'+'Enter' to quiesce and stop the service or 's'+'Enter' to stop it now.
Note: selected option is not echoed on the console.
```

Run Service in Queue:
The script provided in hw3, called “run-client.sh” will provide the necessary commands to deploy the service to the queue and broker already running in different terminals.

```
export UIMA_CLASSPATH=./target/:./target/dependency/
runRemoteAsyncAE.sh tcp://localhost:61616 hw3Queue -c ./src/main/resources/hw3-139547-
fileSystemCollectionReader_Descriptor.xml
UIMA AS Service Initialization Complete
..Completed 2 documents; 459 characters
Time Elapsed : 2531 ms
```

Step 3: Create Client Descriptor
The Client Descriptor was created by reading the UIMA-AS manuals and by looking at the examples provided in its project folder. It looks like this:

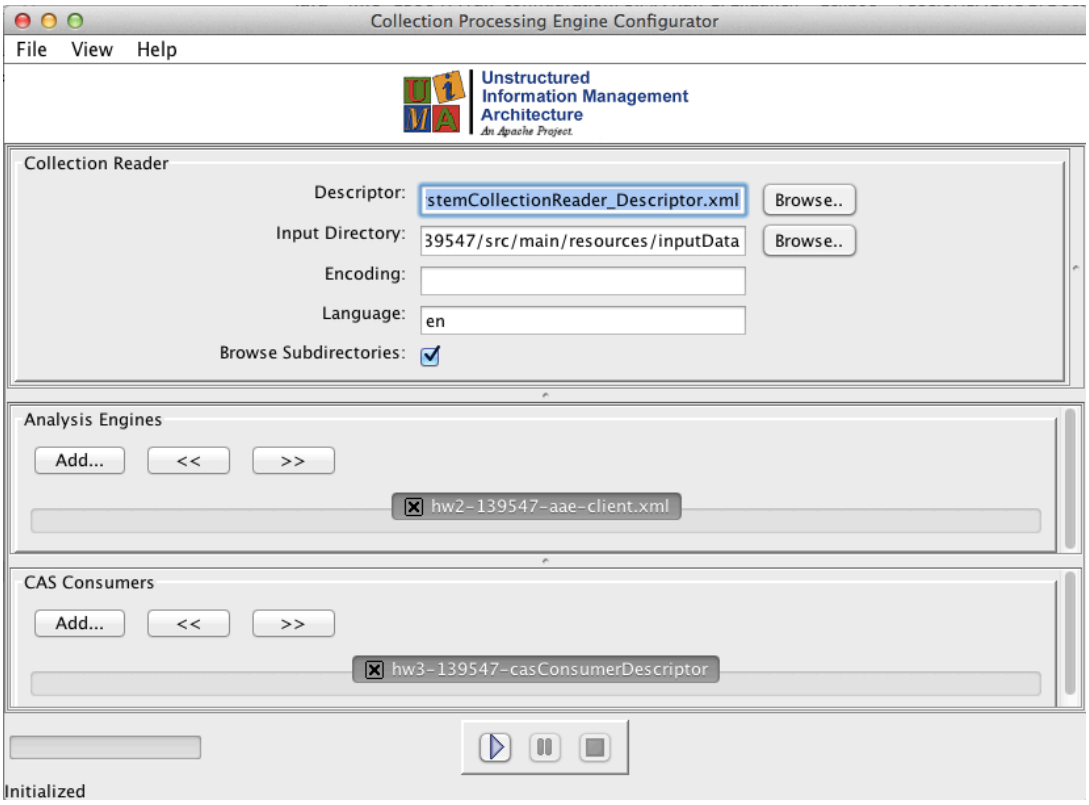
```
<customResourceSpecifier xmlns="http://uima.apache.org/resourceSpecifier">

<resourceClassName>org.apache.uima.aae.jms_adapter.JmsAnalysisEngineService
Adapter</resourceClassName>
   <parameters>
     <parameter name="brokerURL" value="tcp://localhost:61616"/>
     <parameter name="endpoint" value="hw3Queue"/>
     <parameter name="timeout" value="5000"/>
     <parameter name="getmetatimeout" value="5000"/>
     <parameter name="cpctimeout" value="5000"/>
   </parameters>
</customResourceSpecifier>
```

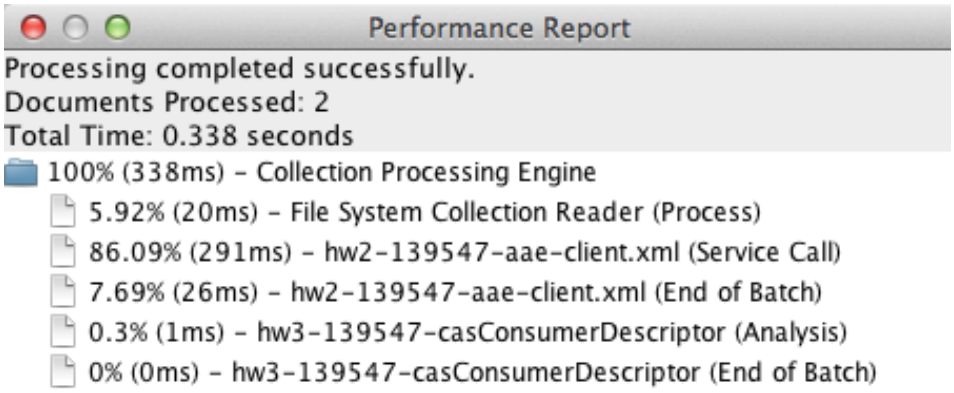
The broker will be, again, ran on our localhost, and our queue is named the same way as in the previous step. This descriptor will allow us to run everything we did in the previous step (step 2) but with the CPE GUI, in a single step (instead of three different terminal interfaces).

Step 4: Create CPE descriptor to test service client
We will use the CPE GUI int his step to deploy the client previously created. Our collection reader is the same one (fileSystemCollectionReader), our input directory takes in the two .txt files we are using to test our system. Our CAS consumer remains the same. The analysis engine will now be different, this time we will use the client descriptor created in step 3 to

deploy our aggregate analysis engine to the service provided by UIMA-AS. The following image shows the specifications and the CPE GUI running.



The results obtained from this task were the following:



Everything described earlier was delivered by friday 28th March 2014. The deployment of the StanfordCoreNLP was not added because the server was not working (it is still not working).

The following steps were followed for this part of the homework:

1) Corresponding dependencies were added to the project (cleartk).
2) Added corresponding typesystem to project (org.cleartk.TypeSystem).
3) Modified typesystem.
    1) Added features to type “Token” (namedEntity, POS and lemma, all Strings).
    2) Added type “NamedEntity”.
    3) Added type “Sentence”.
4) Created an annotator that implements the StanfordCoreNLP (StanfordCoreNLPAnnotator.java) with its corresponding descriptor.
5) Just as with UIMA-AS we created a client (scnlp-139547-client.xml) descriptor and a deployment (scnlp-139547-deploy.xml) descriptor.
6) In the same fashion as UIMA-AS two scripts were created to deploy and run the descriptors previously created (deploy-scnlp.sh and runscnlp.sh).