Mexico Autonomous Institute of Technology
Design and Engineering of Intelligent Information Systems
COM-35702

Homework 3
Execution Architecture with CPE and Deployment Architecture with UIMA-AS
Report

Fernando Aguilar – 114297

Abstract

In this homework we deployed our aggregated analysis engine in a UIMA Asynchronous Scaleout (AS) Service. The UIMA AS is a framework for deploying analysis engines as a service in order to handle requests in a scalable manner. Also, in this homework, we created a Collection Processing Engine (CPE) that helps to create analysis engine workflows that can process artifacts from various sources, with help of a Collection Reader. Also, using a CAS consumer in a CPE it's possible to consume the final results of a analysis engine.

Introduction

Once we get ready an analysis engine, it is desirable that other applications could send input and consume the results that produces the analysis. Also, the operations that an analysis engine perform are computationally intensive. Hence, a mechanism for managing distributed execution of a analysis engine was created in order to cope with these challenges. In this homework, we explore a local execution solution, CPE, and a distributed execution environment, called UIMA AS. The main difference between CPE and UIMA AS is that the former executes the aggregate analysis engine in a linear, single-threaded way, while the later can execute the aggregate analysis engine in a pipelined manner.
In the following sections we describe the tasks in we deployed our Homework 2 aggregate analysis engine, first in a CPE and later with UIMA AS.

Task 1: Creating and Running CPE

We created a CPE for running the aggregate analysis engine we designed and implemented in the previous homework. The CPE is mainly composed of three elements:

- **A Collection Reader**: this is an artifact that creates several CAS from several information sources. For example, it's possible to build a Collection Reader that generates CAS from records in a database. In this homework, we use the FileSystemCollectionReader class provided in the UIMA AS SDK examples. We just created a Collection Reader Descriptor file (descriptors/FileSystemCollectionReader.xml file under resources folder) that fits our needs.
- **An (aggregate) analysis engine**: We used the aggregate analysis engine we implemented in Homework 2 (see hw2-114297-aae.xml). Note that there's no need for modifying the code or the analysis engine descriptor file.
- **A CAS consumer**: Once the aggregate analysis engine finishes the processing of a CAS, a CAS consumer can read the results of the analysis and it can use those results. For this homework, a CAS consumer for printing the precision@n metric was created (See descriptors/casConsumer.xml and edu.cmu.deiis.hw3.consumers.EvaluatorX class).

For this task, we used the UIMA CPE GUI tool included in the SDK for generating the CPE descriptor. The file that contains this descriptor is hw3-114297-CPE.xml. It's very important to note that, in order to successfully execute the CPE with the CPE GUI, the environment variable UIMA_CLASSPATH must be defined with the path that contains jars with our code and their dependencies.

Task 2. Deployment with UIMA AS

For this task, two subtasks were developed. The first one was for creating a UIMA AS client (see scnlp-114297-client.xml) for calling a remote analysis engine that uses the Stanford Core NLP library for producing various text annotations, like Part Of Speech Tagging (POS) and Named Entity Recognition (NER). The client descriptor that invokes the UIMA AS service was created but, unfortunately, we were unable to use it for getting the POS and NER annotations. This file was created by figuring out the structure of the examples/descriptors/as/MeetingDetectorAsyncAE.xml file included in the SDK examples.

The second task was for deploy our Homework 2 analysis engine as a UIMA AS service. The process for deploying the engine as an UIMA AS service has three main steps:

1. Create a message broker: UIMA AS uses Java Message Service for managing incoming and outgoing CASes. The UIMA AS SDK provides the Apache ActiveMQ message broker as a default message service. So, this message broker was used alongside the UIMA AS SDK. For starting the broker, we use the `startBroker.sh` script included in the SDK.

2. Deploy the analysis engine: For this step, a UIMA AS deployment descriptor (hw2-114297-aae-deploy.xml) was created based on the examples provided in the SDK. This descriptor is an XML file that tells UIMA AS where is the descriptor for the analysis engine and the name of the queue where the analysis engine will receive requests. The command for deploy the analysis engine is:

```
deployAsyncService.sh ./src/main/resources/hw2-114297-aae-deploy.xml -brokerUrl tcp://localhost:61616
```

This command is executed at the root folder of the project source. Note that the broker URL listens to requests in the TCP port 61616. Also, it's crucial to define the UIMA_CLASSPATH with the path of the jar artifacts (code and dependencies) of our project. For reference, a script called `deploy-as-aae.sh` was created for ease the deploying process of our aggregate analysis engine.

3. Consume the service: Once we have deployed our UIMA AS service, we can request analysis in two ways:

- We can use a script included in the SDK to create a client request. The following line shows the runRemoteAsyincAE.sh script usage for invoking our UIMA AS service with the agreggate analysis engine we created in Homework 2:

  ```
  runRemoteAsyncAE.sh tcp://localhost:61616 hw2-114297-aaeQueue -c \
  ./src/main/resources/descriptors/FileSystemCollectionReader.xml
  ```

  A helper script called `run-as-client.sh` was created for creating a client for our UIMA AS service.

- We created a client descriptor file (see hw2-114297-aae.xml) in order to invoke the aggregate analysis engine service. Therefore, we can use this client descriptor file in a CPE descriptor file (see hw3-114297-aae-as-CPE.xml) or with the Document Analyzer in the fact that this file act as a “definition” of an aggregate analysis engine.

Technical considerations

We used Apache UIMA AS 2.4.0 framework in a computer with Ubuntu 13.10 operating system. For development, we used Eclipse Kepler with the Eclipse UIMA plug-ins for developing Java code and for calling the UIMA tools required for creating descriptor files. Finally, for script development we used nano and Sublime Text.
Also, a little more deep knowledge of Maven was required for executing some goals and adding new dependencies. Several issues concerning environment variable configuration were arose for deploying the analysis engines but, with enough research we resolved this issue. The environment variables that have to be defined are:

UIMA_HOME = <<Path of the UIMA AS SDK>>
JAVA_HOME = <<Path of the Java SDK>>
CLASSPATH = .:$JAVA_HOME/lib
ACTIVEMQ_BASE = $UIMA_HOME/apache-activemq
PATH = $PATH:$UIMA_HOME/bin/

There's an additional environment variable called UIMA_CLASSPATH that have to point to the Java classes or jars that contains the implementation of our analysis engines and related stuff.
A very important lesson learned in this homework is to take into account the version of the SDK we are using. In previous experiments we were using UIMA AS 2.4.2 but, an error with maven regarding to dependency apache-activemq appeared with this version of the SDK. Later using the 2.4.0 version of the SDK, the error disappeared. Also, the examples project included in the UIMA AS SDK was very helpful for completing the tasks.

Conclusions

In this homework we deployed our Homework 2 aggregate analysis engine first in a CPE, later in UIMA AS. For both tasks, various client and deployment descriptors were created. Several issues regarding configuration aroused and were resolved. Finally, as a future work, we are planning to integrate to our pipeline the ClearTK annotator that is based upon Stanford Core NLP library.