apache UIMA 설치하기

UIMA SDK는 Eclipse를 이용하는 것이 여러모로 편리하다. Eclipse Plugin을 이용하여 여러 편리한 tool을 사용할 수 있기 때문이다. 따라서 Eclipse를 이용하여 설치하는 방법을 설명할 것이다. Eclipse와 java JRE는 설치된 것으로 간주한다.

먼저 UIMA Eclipse Plugin을 설치해야 한다. Eclipse Update 메커니즘을 이용하면 쉽게 설치할 수 있다. 이를 위해 메뉴바에서 Help -> Install new software…를 선택한다. 다음 페이지에서 "Work with" 박스에 다음 URL을 입력하고 엔터를 누른다.

http://www.apache.org/dist/uima/eclipse-update-site/

이후에 설치하고자 하는 플러그인 툴을 선택하고 설치한다. 다음으로 UIMA SDK를 다운로드하여 설치해야 한다. 아래에 주소에 접속하여 binary UIMA SDK의 binary 파일을 다운로드 받는다

http://uima.apache.org/downloads.cgi

Apache UIMA 패키지를 .zip이나 tar.gz 형태로 받아서 원하는 위치에 풀고 난 후엔 환경변수를 설정해야 한다. "UIMA_HOME"이라는 환경변수를 만들어 apache UIMA를 압축해제한 디렉토리로 설정한다. 그리고 PATH에 UIMA_HOME/bin을 추가한다.(PATH 설정은 운영체제에 따라 다르므로 자세한 설명은 생략한다.) 그 후 UIMA_HOME/bin/adjustExamplePaths.bat(혹은 .sh)를 실행한다. 이는 examples에 있는 path를 실제 UIMA_HOME의 위치로 설정한다.

이제 Eclipse를 실행해야 한다. 만약 실행중이었다면 종료한다. 그리고 -clean 옵션을 추가하여 실행해야 한다.(윈도우의 경우: http://prolite.tistory.com/213)(맥의 경우: http://stackoverflow.com/questions/6853053/how-to-run-eclipse-clean-on-a-mac) 이를 하지 않으면 Eclipse는 변경사항을 적용하지 않는다. -clean 옵션을 추가하면 모든 플러그인들을 다시 확인하며 캐쉬를 초기화한다.

추가적으로 UIMA의 예제 코드를 Eclipse 프로젝트에 추가할 수 있다. 방법은 다음과 같다.

- Eclipse의 메뉴바에서 Window →Open Perspective → Java를 통해 Java Perspective를 연다.
- 메뉴바에서 Window -> Preferences -> java -> Build Path -> Classpath Variables를 선택한다.
- "New"를 클릭한다.
- "Name" 영역에 UIMA_HOME을 입력한다.
- "Path" 영역에 apache UIMA가 설치된 디렉토리를 입력한다.
- "New Variable Entry"에서 "OK"를 클릭한다.
- "Preferences"에서 "OK"를 클릭한다.
- full build를 원하는지 묻는다면 "Yes"를 클릭한다.
- File -> Import menu option을 선택한다.
- "General/Existing Project into Workspace"를 선택하고 "Next" 버튼을 클릭한다.
- "Browse"를 클릭하고 %UIMA_HOME%/ 디렉토리를 찾는다.
- "Finish"를 클릭한다. 이로써 "uimaj-examples"라는 이름의 새로운 프로젝트가 추가된다. 절차가 옳았으면 이 프로젝트는 컴파일 에러가 없어야 한다.
- 설정이 제대로 됐는지 확인하기 위해 "Problem"에 에러 메세지가 없는지 확인한다.

위의 과정을 통해 apache UIMA를 이용하기 위한 준비가 끝난다.

UIMA annotator 개발

간단한 UIMA annotator를 개발하기 위한 절차는 다음과 같다.

1. CAS(Common Analysis Structure) 타입 정의하기
2. 그 타입에 해당하는 Java 클래스 생성하기
3. 실제 annotator의 Java 코드 작성하기
4. Analysis Engine Descriptor 작성하기
5. annotator 테스트하기

이 절차에 따라 간단한 annotator를 만드는 예제를 설명할 것이다. (프로젝트명은 "RoomNumberAnnotator")

0. UIMA 프로젝트 만들기

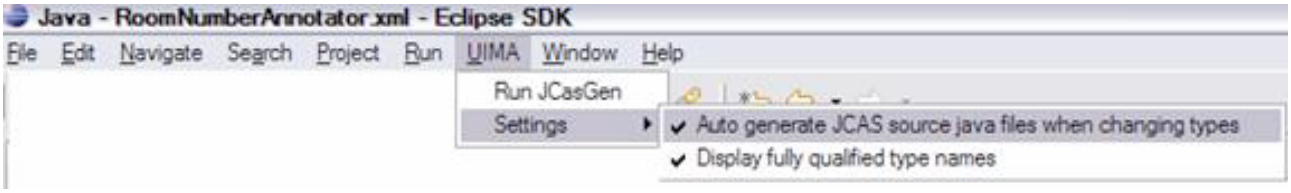
Eclipse를 이용해 Java 프로젝트를 만든다. 이때 "Create separate folders for sources and class files" 항목이 체크되도록 한다. 프로젝트를 만든 후 해당 프로젝트를 마우스 우클릭 하여 "Add UIMA Nature" 항목을 클릭한다. 이를 통해 필요한 폴더들이 자동으로 만들어진다. 그 후 UIMA 라이브러리를 Path에 추가해야 한다. 해당 프로젝트를 우클릭하여 Build Path -> Configure Build Path 항목을 선택한다. "Add Variable…" 버튼을 클릭 후 "UIMA_HOME" variable을 선택한다. Extend 버튼을 눌러 lib 디렉토리에 있는 uima-core.jar를 선택한다. 여기서 필요한 다른 jar 파일들도 추가할 수 있다.

1. CAS(Common Analysis Structure) 타입 정의하기

출력 결과를 저장할 클래스를 생성하기 위해서는 우선 Type System Descriptor를 작성해야 한다. desc 폴더에 마우스 우클릭 -> New -> Other… -> UIMA -> Type System Descriptor File을 선택하여 Type System Descriptor 파일을 생성한다. 이 파일은 XML 파일이지만 UIMA 플러그인의 Component Descriptor Editor를 이용하면 XML 문법을 알 필요 없이 수정할 수 있다. 만들어진 파일을 마우스 우클릭 -> Open With -> Component Descriptor Editor를 선택하여 파일을 연다. 아래에 Overview, Type System, Source 탭을 확인할 수 있으며 Overview와 Type System만 수정하면 된다. Source 탭에는 실제 XML 코드가 있으며 직접 수정할 수도 있으나 Overview와 Type System을 수정하면 자동으로 수정된다. Overview 탭에는 해당 타입의 메타 정보(이름, 버전, 설명 등)를 적을 수 있다. Type System 탭에서는 실제 타입을 정의한다. Add Type 버튼을 통해 타입명을 적고 Supertype을 정한다. 이 타입은 Java 클래스에 해당한다. 여기서는 Annotation를 사용한다. 그 후 만들어진 타입에 Add 버튼을 눌러 Feature들을 추가할 수 있다. Java가 제공하는 기본 타입(int, float, String 등)과 그것의 배열을 추가할 수 있다.

2. Java 클래스 생성하기

Type System Descriptor를 수정하고 저장하면 해당 클래스가 src에 자동으로 생성된다. 이 클래스에는 기본적인 get, set 메서드가 생성된다. 만약 생성되지 않을 경우 다음 탭이 체크되어 있는지 확인한다.



수동으로 생성할 수도 있으며, Type System Descriptor의 Type System 탭에서 JCasGen 버튼을 누르면 된다.

3. 실제 annotator의 Java 코드 작성하기

이제 실제로 annotator를 구현해야 한다. 새로운 Java 소스 파일을 생성하여 annotator를 구현한다. annotator를 구현하기 위해서는 규정된 인터페이스들을 구현해야 하며, 몇 가지 메서드들이 있지만 중요한 것은 다음의 세 가지 메서드이다.

- initialize
- process
- destroy

initialize는 해당 annotator 클래스가 처음 생성될 때 초기화에 사용되는 메서드이며, process는 실제로 분석을 수행하는 메서드이다. destroy 메서드는 annotator의 사용이 끝나면 호출된다. annotator 클래스는 반드시 public으로 선언되어야 하며, abstract으로 선언될 수 없다. 그리고 public으로 선언된 0개의 인자를 받는 생성자가 필요하다. 이 생성자가 있어야 프레임워크가 초기화를 할 수 있다.

JCas를 사용하는 JCasAnnotator_ImplBase를 상속하면 디폴트로 구현된 인터페이스를 사용할 수 있다. 이 클래스는 process 메서드를 제외한 다른 메서드들을 모두 구현하였다. 그러므로 이 메서드를 상속하면 process 메서드만을 구현하면 된다. 이 process 메서드는 JCas를 유일한 인자로 받으며 이 JCas는 분석할 문서와 모든 분석 결과를 갖고 있다.

분석결과를 annotation으로 저장하는 것은 간단하다. Java 객체를 생성하고 몇 가지 메서드를 호출하면 된다.

```
RoomNumber annotation = new RoomNumber(aJCas);
annotation.setBegin(matcher.start());
annotation.setEnd(matcher.end());
annotation.setBuilding("Yorktown");
annotation.addToIndexes();
```

분석된 문자열의 시작 위치와 끝위치, 그리고 추가로 정의한 Feature들을 set한다. 그리고 마지막으로 addToIndexes() 메서드를 호출하여 CAS에 해당 annotation의 인덱스를 추가한다. 이 인덱스는 CAS가 annotation들을 출력하거나 다음 분석기의 인풋으로 사용할 때 사용되며, 순서대로 인덱스가 부여된다.

4. Analysis Engine Descriptor 작성하기

UIMA 아키텍쳐는 XML파일 형식의 annotator에 대한 정보를 필요로 한다. 이 정보는 런타임에 UIMA 프레임워크에 제공되어야 한다. 따라서 이러한 XML 파일을 작성해야 하며 이를 Analysis Engine Descriptor라고 한다. Analysis Engine Descriptor와 Type System Descriptor의 차이는 전자는 annotator에 대한 Descriptor이고, 후자는 CAS 타입에 대한 Descriptor라는 것이다. Analysis Engine Descriptor 파일은 다음과 같은 정보들을 포함한다.

- Name, description, version, and vendor
- The annotator's inputs and outputs, defined in terms of the types in a Type System Descriptor
- Declaration of the configuration parameters that the annotator accepts

마지막 줄의 configuration parameter는 자주 쓰이는 파라미터들을 일일이 하드 코딩하지 않고 목록을 만들어 사용할 수 있도록 하는데, 프로젝트를 진행하는 데에 중요한 내용은 아니므로 생략한다. 이를 사용한다고 하여 성능상에 이점이 있는 것은 아니다. 자세한 내용은 UIMA Tutorial and Developer's Guide의 1.2.1 절 Configuration Parameter에서 찾아볼 수 있다.

이 Descriptor를 작성하기 위해, desc 폴더를 우클릭하여 New -> Other… -> UIMA -> Analysis Engine Descriptor File을 선택한다. 작성된 파일을 우클릭하여 Open With -> Component Descriptor Editor를 선택하여 해당 파일을 연다. 이 파일에서는 Overview, Type System, Capabilities를 수정하면 된다. Overview에서는 사용하는 언어를 선택할 수 있고, Engine Type으로 Primitive와 Aggregate 중에 선택할 수 있다. Primitive는 단일 분석 엔진을 의미하며, Aggregate는 이런 각각의 분석 엔진을 합쳐서 사용하는 경우 사용된다. 이는 후에 설명할 것이다. 그리고 밑의 Browse 버튼을 통해 해당 Descriptor의 대상 Java class 파일을 선택한다. 다음으로 Type System 탭에서는 이미 앞서 만든 Type System Descriptor를 가져오면 된다. 오른쪽 섹션(Imported Type Systems)에서 Add 버튼을 클릭하여, by location을 체크하고 앞서 만든 Type System Descriptor를 찾아 가져온다. Capabilities 탭에서는 Add Type 버튼을 통해 인풋 혹은 아웃풋을 정할 수 있다. 텍스트를 직접 인풋으로 사용하여 분석하는 경우 인풋을 따로 정의할 필요는 없다. 하지만 다른 분석기의 결과를 인풋으로 사용하는 경우는 인풋도 정해야 한다. 그리고 현재 분석기의 분석 결과(2에서 생성한 Java Class)를 아웃풋으로 설정한다. 이제 간단한 annotator를 만드는 것은 완료했다.

5. annotator 테스트하기

이제 annotator가 잘 돌아가는지 테스트할 차례이다. UIMA SDK에서 제공하는 툴을 사용하면 쉽고 간단하게 테스트해볼 수 있으며 결과를 시각화해준다. Eclipse의 메뉴 바에서 Run -> Run Configurations… -> 왼쪽 박스의 Java Applications 밑에서 "UIMA CAS Visual Debugger"를 선택한다. 이 창에서 Classpath 탭으로 이동하여 "User Entries"를 선택하고 "Add Projects…" 버튼을 누른다. 작업하던 프로젝트를 선택하여 추가한다. 그리고 Run 하면 된다.

CVD 창이 열리면 Run -> Load AE에서 앞서 만든 Analysis Engine Descriptor를 작업하던 프로젝트의 desc 폴더에서 찾아 로드한다. 그리고 Text 영역에 다음과 같은 분석할 텍스트를 쓴다.

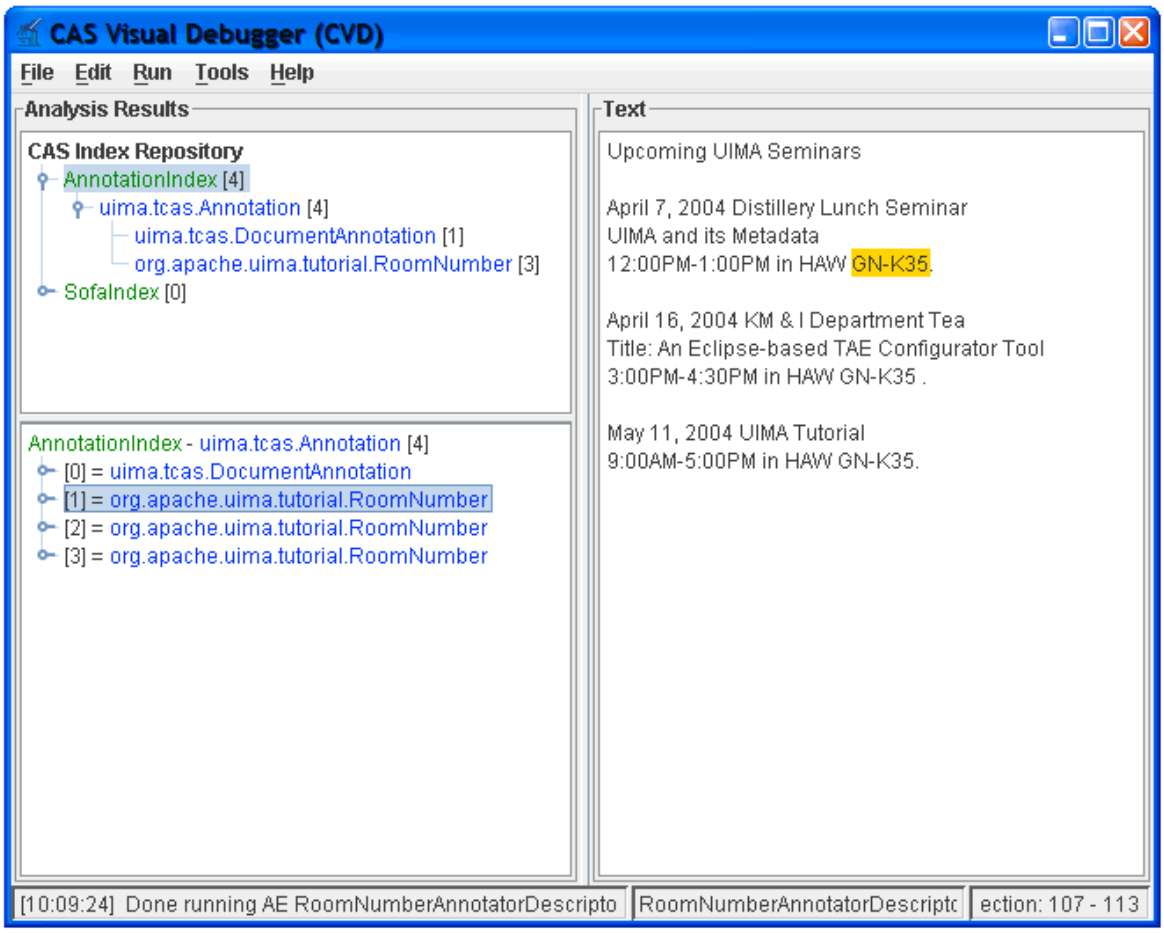
Upcoming UIMA Seminars

April 7, 2004 Distillery Lunch Seminar
UIMA and its Metadata
12:00PM-1:00PM in HAW GN-K35

April 16, 2004 KM & I Department Tea
Title: An Eclipse-based TAE Configurator Tool
3:00PM-4:30PM in HAW GN-K35

May 11, 2004 UIMA Tutorial
9:00AM-5:00PM in HAW GN-K35

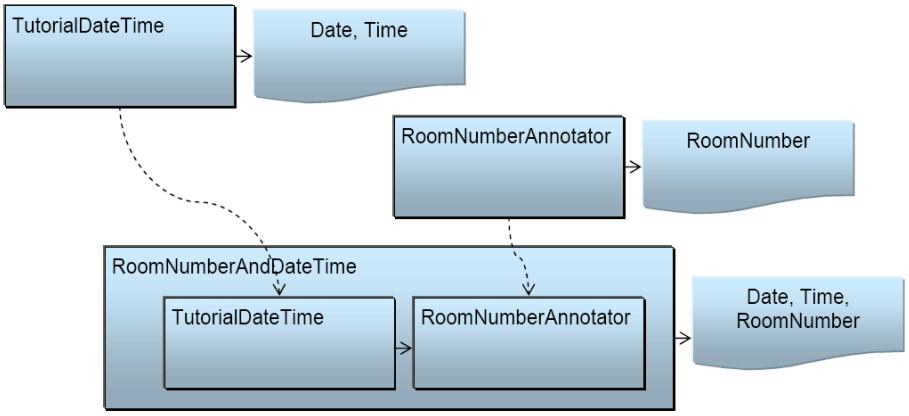
그리고 annotator를 실행하기 위해 "Run -> Run[앞서 로드한 Analysis Engine Descriptor 이름]"를 선택한다. 그러면 왼쪽 아래의 창처럼 분석 결과를 확인할 수 있다.



혼합 UIMA Annotator 개발

1. 병렬 혼합

UIMA SDK는 단일 annotator를 합쳐서 보다 큰 혼합 annotator를 쉽게 만들 수 있게 한다. 추가적인 자바 코드조차 필요가 없다. 여기서는 아래 그림과 같이 DateTime annotator와 RoomNumber annotator를 합쳐 RoomNumberAndDateTime annotator를 만드는 예제를 살펴볼 것이다.(프로젝트명은 "RoomNumberAndDateTimeAnnotator") DateTime annotator와 RoomNumber annotator 사이에는 의존 관계가 없으며 병렬적으로 각각 다른 결과를 출력할 뿐이다. 따라서 선후 관계는 의미가 없다. 의존 관계가 있는 혼합 annotator 개발은 이후에 다룬다.



DateTime annotator를 만드는 과정은 위에서 설명한 단일 annotator를 만드는 과정이므로 생략한다. 따라서 두 개의 annotator가 만들어져 있다고 가정하자. 추가적으로 할 일은 두 개의 annotator를 합친 분석기의 Analysis Engine Descriptor를 작성하는 것 뿐이다.

앞서 설명한 대로 desc 폴더에 Analysis Engine Descriptor를 새로 만든다. 역시 Component Descriptor Editor로 연다. Overview 탭에서 Engine Type을 Aggregate로 체크한다. 그리고 Aggregate 탭에서 Add 버튼을 눌러 만들어져 있는 DateTime annotator와 RoomNumber annotator의 Analysis Engine Descriptor를 추가한다. 오른쪽의 Component Engine Flow 섹션은 이 분석기가 실제로 작동하는 순서를 보여준다. 이 영역을 수정하여 annotator의 분석 순서를 정할 수 있다. 여기서는 분석 순서가 무의미하므로 넘어가도 좋다. Type System 탭을 확인해보면 자동으로 설정되어 있으며 수정이 불가능하다는 것을 확인할 수 있다. 하지만 input과 output을 설정하기 위해 Capabilities 탭은 수정해야 한다. Add Type 버튼을 눌러 아웃풋으로 내놓을 타입을 추가한다. 여기까지 완료하면 필요한 일은 모두 끝난다. 이 annotator를 테스트 하기 위해서는 앞의 절과 똑같은 방식으로 방금 새로 만든 Analysis Engine Descriptor를 추가하여 테스트하면 된다.

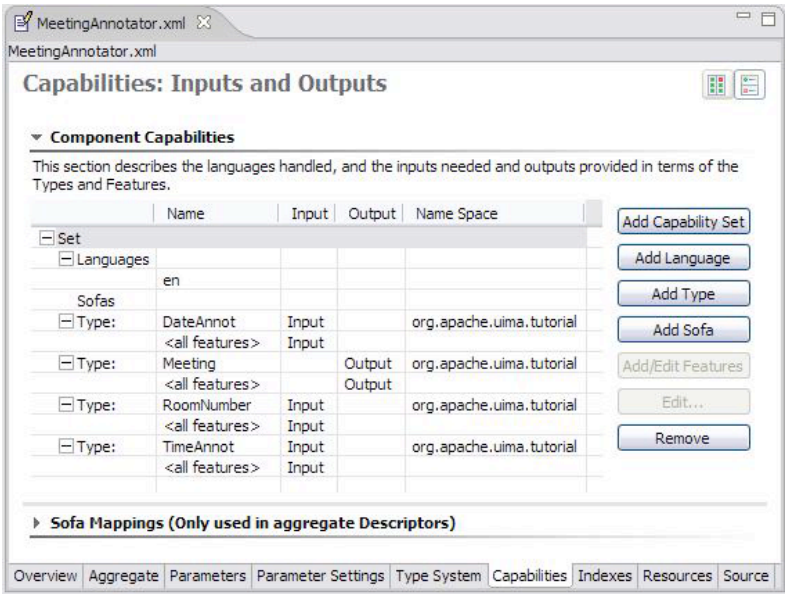
2. 직렬 혼합

하나의 annotator는 당연히 다른 annotator의 결과를 인풋으로 사용할 수도 있다. 이러한 annotator는 앞서 설명한 annotator와는 다른 새로운 종류의 annotator이므로 추가 설명이 필요하다. 프로젝트 “MeetingAnnotator”를 참고하면 된다.

src 디렉토리의 ids.intern.uima.tutorial.ex3.MeetingAnnotator 코드를 보면, 이 annotator는 앞서 살펴본 RoomNumber와 DateTime의 결과를 인풋으로 받아서 분석을 수행하고 있다. 앞서 언급했듯이 JCas는 분석해야 할 텍스트 원본 데이터뿐만 아니라 annotator들의 분석 결과 또한 모두 가지고 있다. 그리고 그 분석 결과들은 인덱스가 순서대로 부여되어 관리된다. 따라서 이 인덱스를 이용할 수 있는 Iterator를 통해 선행하는 분석결과들을 이용할 수 있다. 방법은 다음과 같다.

```
FSIndex timeIndex = aJCas.getAnnotationIndex(TimeAnnot.type);
Iterator timeIter = timeIndex.iterator();
while (timeIter.hasNext()) {
TimeAnnot time = (TimeAnnot)timeIter.next();
//do something
}
```

이를 통해 앞서 분석된 TimeAnnot을 순서대로 가져와서 다음 분석에 활용할 수 있다. 이런식으로 만들어진 MeetingAnnotator의 Analysis Engine Descriptor를 작성하는 것도 유의할 사항이 있다. 다른 점은 똑같으나 Capabilities 탭에서 input을 설정해줘야 한다는 것이다. 앞서 설명한 annotator들은 분석할 텍스트를 직접 인풋으로 활용하여 input을 따로 설정할 필요가 없었으나 MeetingAnnotator는 앞의 annotator의 분석 결과를 인풋으로 받아야 하므로 인풋으로 이를 설정해줘야 한다. 결과는 다음과 같다.

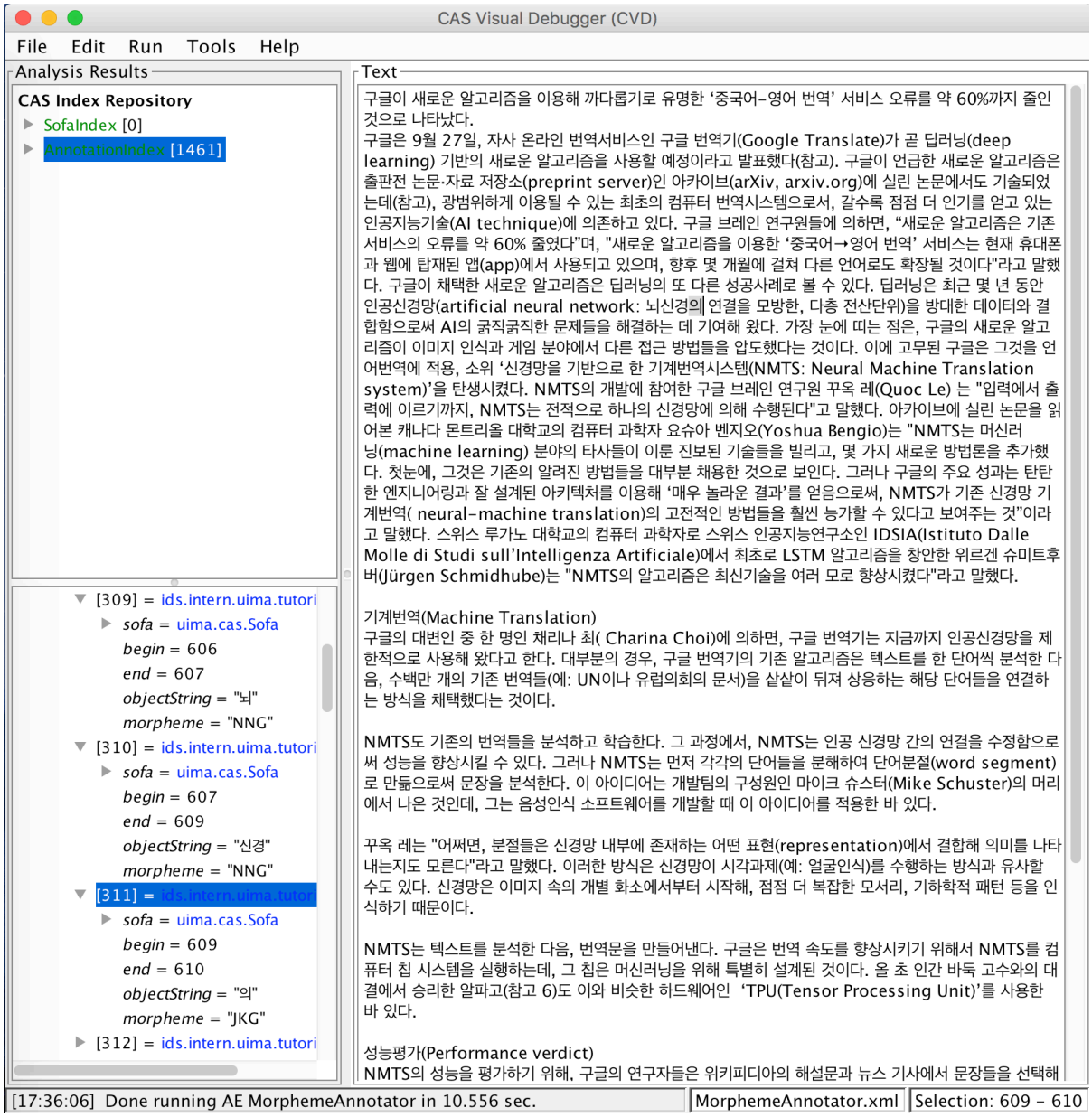


이 annotator는 단독으로 작동할 수 없다. 앞의 annotator의 분석결과를 인풋으로 활용하기 때문이다. 이렇게 앞선 분석 결과를 활용하는 annotator를 만들었다면 직렬 혼합을 위한 준비는 끝난다.

직렬 혼합을 하기 위해서는 앞서 병렬 혼합의 경우와 마찬가지로 Analysis Engine Descriptor만 만들면 된다. 병렬 혼합과 다르게 유의해야 할 것은 Aggregate 탭에서 AE(Analysis Engine)을 추가할 때 오른쪽 Component Engine Flow 섹션에서 순서를 정해줘야 한다는 것이다. 작동 순서상 MeetingAnnotator는 RoomNumber annotator와 DateTime annotator의 뒤에 위치해야 하는 것이다. 나머지 과정은 위와 동일하다. 테스트 과정도 동일하다.

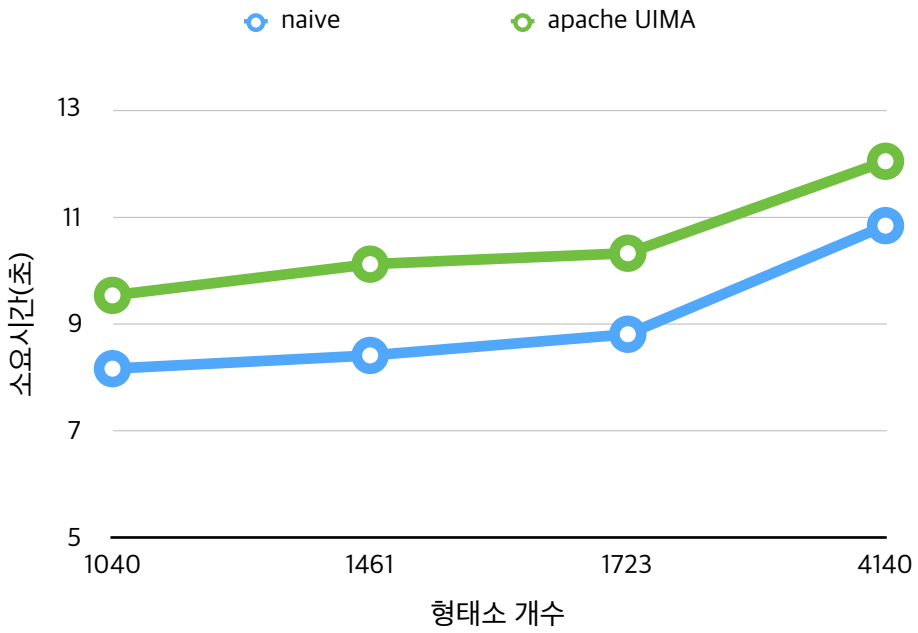
한글 텍스트를 이용한 UIMA Annotator 사용 예

apache UIMA를 한글 텍스트 분석에 사용하기 위해서 꼬꼬마 형태소분석기를 UIMA로 감싸보았다. 꼬꼬마 형태소분석기의 버전은 2.0이다. 꼬꼬마 형태소 분석기를 단일 annotator로 사용하였으며 임의로 찾은 한글 기사 텍스트에 대해서 분석을 시행한 결과는 다음과 같다. 오른쪽 아래의 섹션에서 분석 결과를 차례로 확인할 수 있다.(프로젝트명: MorphemeAnalyzerUIMA)



성능비교

apache UIMA를 사용하여 기존의 형태소 분석기를 감쌀 경우 성능의 감소가 기대되었다. 이는 기존의 형태소 분석기의 결과를 저장하고 관리하기 위한 보다 많은 과정이 추가되기 때문이다. 따라서 apache UIMA를 활용했을 경우 어느정도 성능이 감소하는지 확인해보았다. 아주 기본적인 형태소 분석기 코드를 사용했을 경우와 apache UIMA를 사용했을 경우를 비교하였다. 텍스트는 서로 다른 크기의 4개의 한글 텍스트를 사용했으며 각각 기사 또는 서평이다. 이러한 4개의 텍스트에 대해서 각각 25번씩 분석을 수행하여 시간을 측정하여 평균냈다. 결과는 다음의 그래프와 같다.



예상한 바와 같이 apache UIMA를 사용할 경우 평균 1.44초만큼 느려졌다. 형태소 개수가 늘어남에 따라 시간 차가 커지거나 줄어드는 것은 확인되지 않았다. 따라서 대체로 시간차는 UIMA를 사용하는 경우 전처리와 후처리 과정이 늘어남에 따라 발생하는 것으로 여겨진다.

결론

apache UIMA를 이용하면 분석기를 합치는 것이 매우 쉬워진다. Analysis Engine Descriptor를 이용해 몇 가지만 설정하면 되기 때문이다. apache UIMA를 이용하기 위해서는 Eclipse를 사용하는 것이 확실히 유리하며, 이를 이용하지 않을 경우는 XML 코드를 직접 작성해야 하는데, 이보다는 차라리 하드 코딩으로 분석기를 합치는 게 나을 것이다. 따라서 UIMA를 이용할 경우 Eclipse를 사용해야 한다. 또한 apache UIMA는 언어와 무관하게 사용할 수 있다고는 하나 JCas와 같은 클래스를 이용하기 위해서는 Java 코드를 사용하는 것이 불가피하다. 이렇게 annotator들을 만드는 데에만 추가적인 노력을 들이면 UIMA를 이용해 분서기를 혼합하는 것은 간단해진다. 다만 분석기를 그대로 사용하는 것에 비해 UIMA를 사용할 경우 전처리와 후처리 과정이 추가되어 실행 시간이 좀 더 늘어나는 것으로 확인되었다.