

1. Learning Outcomes

You will by the end of this tutorial:

- have basic knowledge of the GATE Development Environment .
- know how to create a simple NLP application of your own in the GATE Development Environment.
- know how to create the same GATE application programatically.
- learn how to exploit IE in GATE
- have been exposed to JAPE for NE Extraction and Relation Extraction

Glossary

GATE – General Architecture for Text Engineering

NLP – Natural Language Processing

IE – Information Extraction

JAPE – Java Annotation Patterns Engine

NE – Named Entities

Materials:

This document and source code to be provided in the beginning of the Tutorial or the tutorial folder accompanying this document.

2. About GATE

“GATE is an infrastructure for developing and deploying software components that process human language. GATE helps scientists and developers in three ways”:

1. by specifying an architecture, or organisational structure, for language processing software;
2. by providing a framework, or class library, that implements the architecture and can be used to embed language processing capabilities in diverse applications;
3. by providing a development environment built on top of the framework made up of convenient graphical tools for developing components.”

[1]

Recommended Reading:

[1]GATE User Guide See <http://gate.ac.uk/sale/tao/>

[2] Software Architecture for Language Engineering
<http://gate.ac.uk/sale/thesis/>

**If you have not already, you should go through the following material:
of the GATE User Guide**

Chapter 1, 7, 8

3. Practical

Task 1

Start Gate either using:

Use your platform specific installer
See Gate User Guide Section 3.2.1

<http://gate.ac.uk/sale/tao/index.html#sec:howto:install>

[1]

OR

```
cd YourGateHomeDir
```

```
bin/ant run #(Linux or Unix)
```

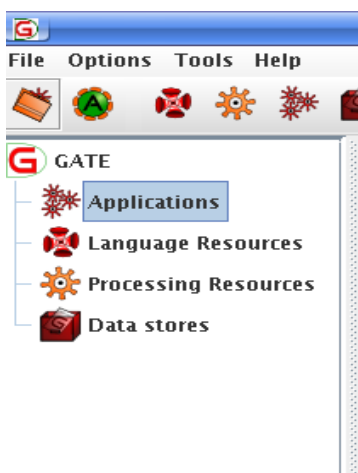
```
bin/ant.bat run #Windows
```


In the USB stick

Task 2

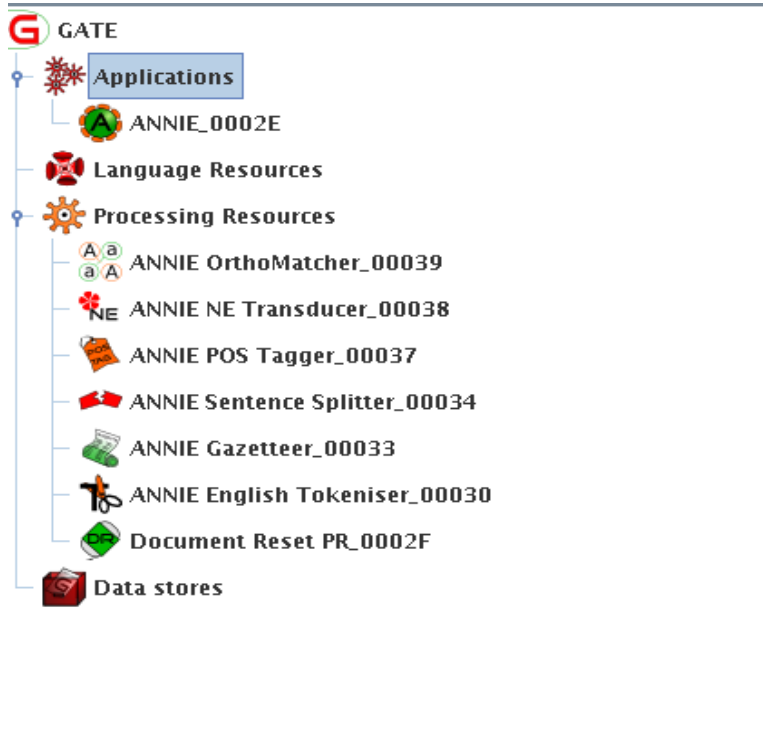
Load ANNIE (A Nearly New Information Extraction System) with defaults.

See Gate USER Guide Chapter 8 for further details,
<http://gate.ac.uk/sale/tao/index.html#chap:annie>



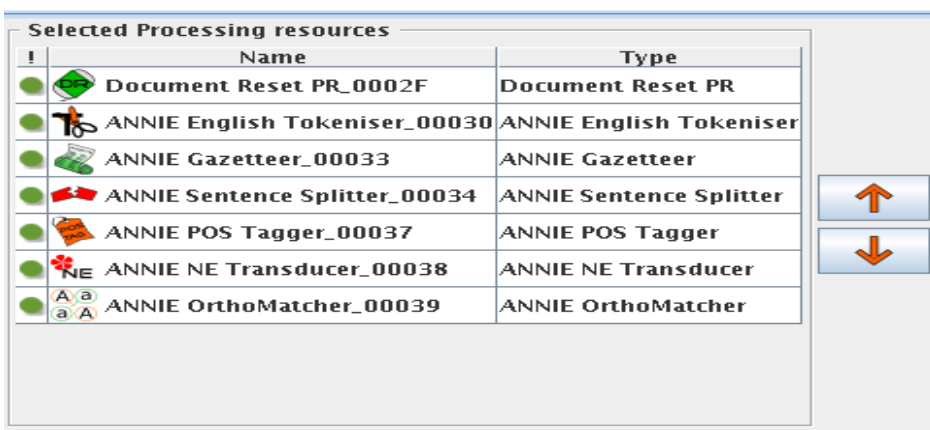
Click on  and choose "Load ANNIE with defaults"

The following will be loaded:



An instance of a Serial Analyser Controller will appear under "Applications" with the name "Annie_####". This type of Controller is used to manage a pipeline of NLP processing resources which will be run over a Corpus. A Corpus is a collection of documents. GATE consists of Processing Resources (PRs) – Programs or Algorithms which will do some sort processing on text i.e. Tokenising or dictionary lookup, parsing etc and Language Resources (LRs) – documents, Corpora, Ontologies – data essentially. The third component is Visual Resources (Vrs) but you do not need to worry about them here. All GATE Resources are realised as Java Beans. See Chapter 1 in the Gate User Guide for more details.

If you double click on the the "ANNIE" instance, the following will appear:



Notice how the PRs you initially loaded when you double clicked on the "ANNIE with defaults" icon, were also automatically added to the pipeline in the appropriate order. Note that the order of your PRs is very important, for instance you cannot really split sentences until you have tokenised the document, this is much more subtle with respect to JAPE processing later. Briefly the default PRs loaded with ANNIE are:

Document Reset PR

GATE keeps annotations and text separately (StandOff Markup), in fact any GATE document will pass through a pipeline unaltered – in other words no annotation or parsing information is embedded in a document. For design reasons which are beyond the scope of this tutorial, GATE tends to employ weak referencing in the JVM, thus overriding garbage collection. Consequently you will need to explicitly delete Resources – mostly documents when using GATE programmatically. GATE will also keep annotation sets derived from previous processing in memory. This has two consequences:

- 1) Old annotations residing in memory from a previous document may be taken into account when processing the next document.

- 2) The JVM heap will begin to fill up with old annotation sets generated from previous documents which can severely impact performance.

Hence the purpose of the Document Reset PR is to clean out any previous annotations, essentially releasing the annotation set objects for Garbage Collection.

ANNIE English Tokeniser

This PR splits a text into annotations of type Token, taking into account spaces etc. In addition a JAPE transducer is used to conduct post processing in order to handle contractions in English i.e. Don't, Can't. **Only use the ANNIE English tokeniser if you are certain your text contains only English.** In addition, if your document contains junk "markup" attempt one way or another to clean it out of the text or ensure that it is well formed. Failing this, will cause the English tokeniser to choke on certain input, particularly at the post processing phase. If you cannot ensure this use the GATE Unicode Tokeniser. Always use the GATE Unicode Tokeniser for languages other than English.

ANNIE Gazetteer

This PR compiles wordlists into Finite State Machines in order to conduct NE (Named Entity) or Key Phrase look up. Output is annotations of type *LookUp*. Used for NE Recognition. An NE is a: Person, Location, Date, Address
See http://en.wikipedia.org/wiki/Named_entity_recognition

ANNIE Sentence Splitter

Assigns annotations of type *Split* to sentence boundaries in text based on punctuation and \t \r keys. Is implemented as cascade of JAPE transducers. Creates *Sentence* annotations.

ANNIE POS Tagger

The POS (Part of Speech) tagger assigns POS tags to Token annotations. The (Hepple) tagger is a modified version of the Brill tagger. The list of tags used is given in Appendix D of the GATE User Guide[1]. The tagger uses a default lexicon and ruleset (the result of training on a large corpus taken from the Wall Street Journal). A POS tag is the assignment of linguistic category to a Token i.e. NNP or NNPS for Proper Nouns in singular or plural forms respectively.

ANNIE NE transducers

This PR will

- (1) attempt to find unknown Named Entities based on extraction templates written in the JAPE language i.e. A Token with an NNP (proper Noun) POS tag followed by a Lookup annotation for Company suffixes like "Ltd or "GmbH" should be annotated as an NE of type Company or Organisation.
- (2) act on LookUp annotations and additional local contextual linguistic information to output a final Semantic Type or NE. i.e. A Lookups of type "title" followed by a "firstname" and "lastname" should be annotated entirely as an NE of type Person - "Mr John Smith"

A Brief Note on JAPE

JAPE – Java Annotation Patterns Engine

JAPE provides finite state transduction over annotations based on regular expressions. JAPE is based CPSL – Common Pattern Specification Language – See[1] Chapter 7

A JAPE grammar consists of a set of phases, each of which consists of a set of pattern/action rules. The phases run sequentially and constitute a cascade of finite state transducers over annotations. The left-hand-side (LHS) of the rules consist of an annotation pattern that may contain regular expression operators (e.g. *, ?, +). The right-hand-side (RHS) consists of annotation manipulation statements in the Java code. Annotations matched on the LHS of a rule may be referred to on the RHS by means of labels that are attached to pattern elements. Cascaded FSTs are shallow or partial parsers, examples include FASTUS[3] and Jfrost within IBM LanguageWare[4]. They are often employed in IE because they are robust and efficient. One could achieve the same result using a deep parser, however this would impact significantly on performance

and robustness, also the majority of parsing information would be discarded in order to complete the same task in IE.

We will look at a JAPE in detail later. If you double click on the NE Transducer in the Resources Tree you can view a JAPE file which calls a collection of phases, similar to a `main` method in C or Java

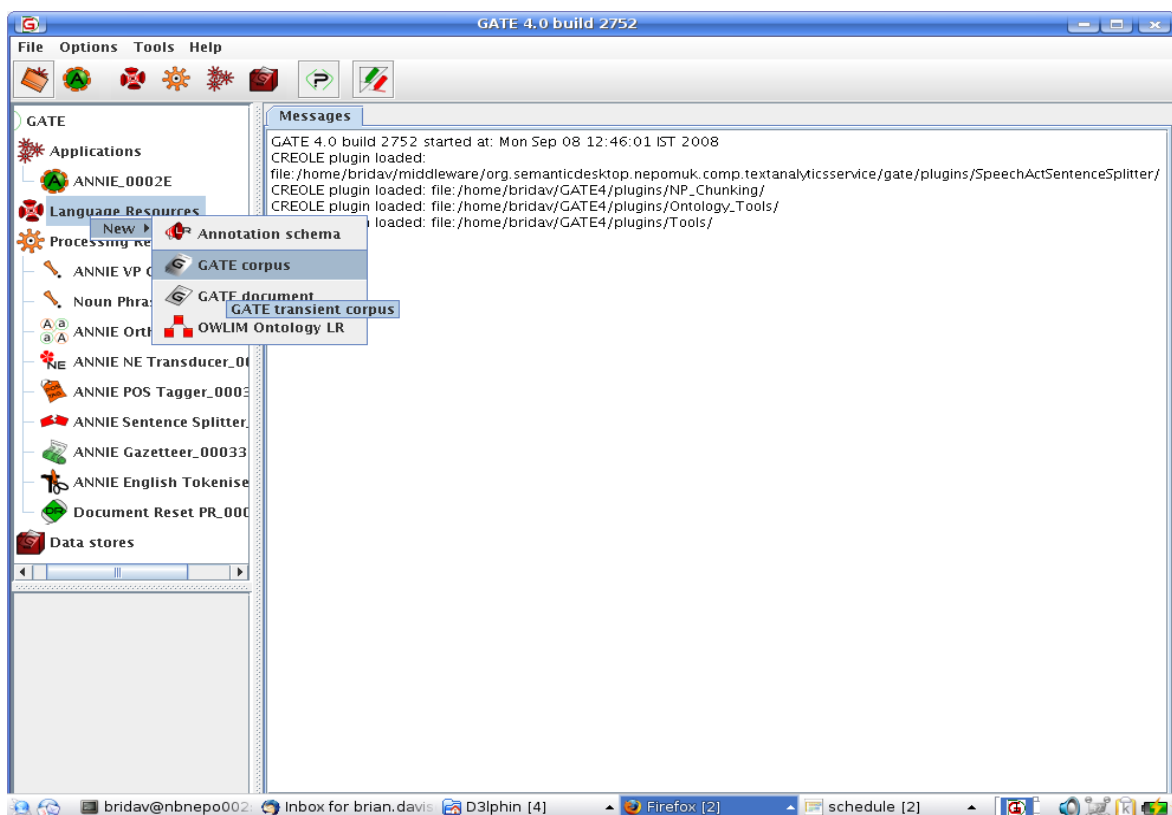
Orthomatcher PR

The Orthomatcher module adds identity relations between named entities found by the NE Transducer, in order to perform coreference. It does not find new named entities as such, but it may assign a type to an unclassified proper name, using the type of a matching name.

The matching rules are only invoked if the names being compared are both of the same type, i.e. both already tagged as (say) organisations, or if one of them is classified as 'unknown'. This prevents a previously classified name from being recategorised. An alias list exists also for Orthomatcher i.e. The PR would create an identity relation between the "Big Blue" and "IBM". The alias list however must be manually constructed.

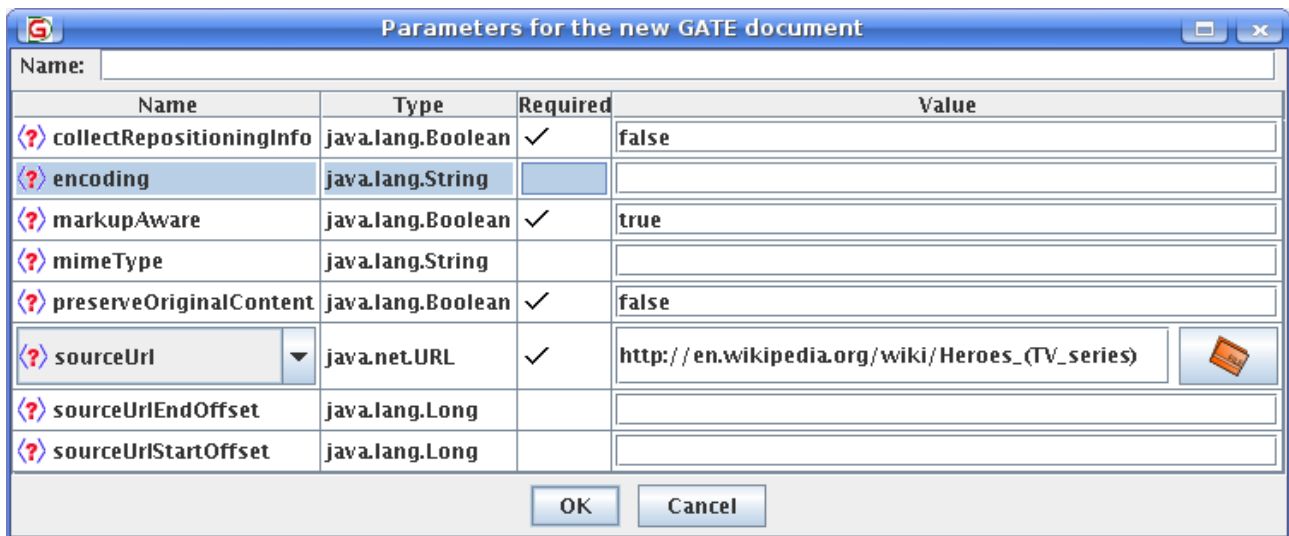
Task 3

Run ANNIE over a document.



Right Click on Language Resources and click on GATE corpus and click OK

Repeat the same for GATE document however, when prompted for the source URL, either provide a URL to for example your favourite TV Program on the Web (if internet access is working or alternatively choose a file from your desktop or from the `corpora` folder in the `tutorial` folder.

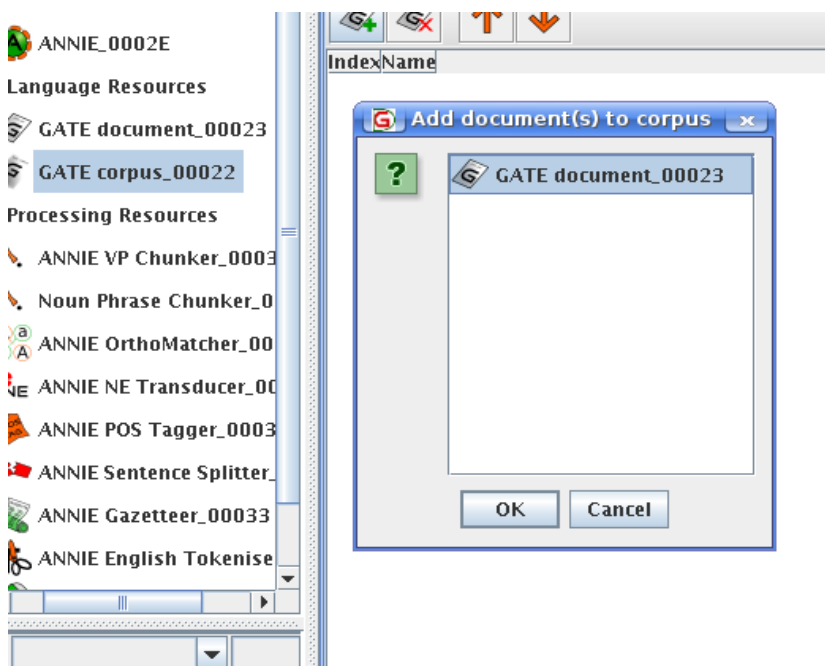


GATE will handle most file formats:

Plain Text

- HTML
- SGML
- XML
- RTF
- Email
- PDF (some documents)
- Microsoft Word (some documents)

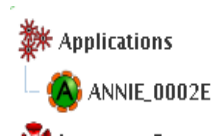
Recall the the application we are using is a Corpus pipeline. We are not processing a single document. If that were the case, we would have created a serial controller instead.



Hence we will need to add our document to our Corpus. Right Click on the Corpus Icon in the Resources and click on the following icon to add your document.



Double Click on the ANNIE application icon again .



Loaded Processing resources

Name	Type

Selected Processing resources

!	Name	Type
	Document Reset_PR_0002F	Document Reset PR
	ANNIE English Tokeniser_00030	ANNIE English Tokeniser
	ANNIE Gazetteer_00033	ANNIE Gazetteer
	ANNIE Sentence Splitter_00034	ANNIE Sentence Splitter
	ANNIE POS Tagger_00037	ANNIE POS Tagger
	ANNIE NE Transducer_00038	ANNIE NE Transducer
	ANNIE OrthoMatcher_00039	ANNIE OrthoMatcher

Corpus: GATE corpus_00022

The corpus and document parameters are not available as they are automatically set by the controller!

No selected processing resource

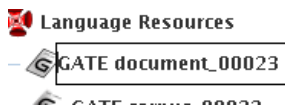
Name	Type	Required	Value

The ANNIE pipeline shown in the main viewing pane (see above) . If the corpus is not already loaded use the pulldown selector to find it. (To the right of **Corpus:** in the view pane)

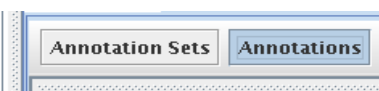
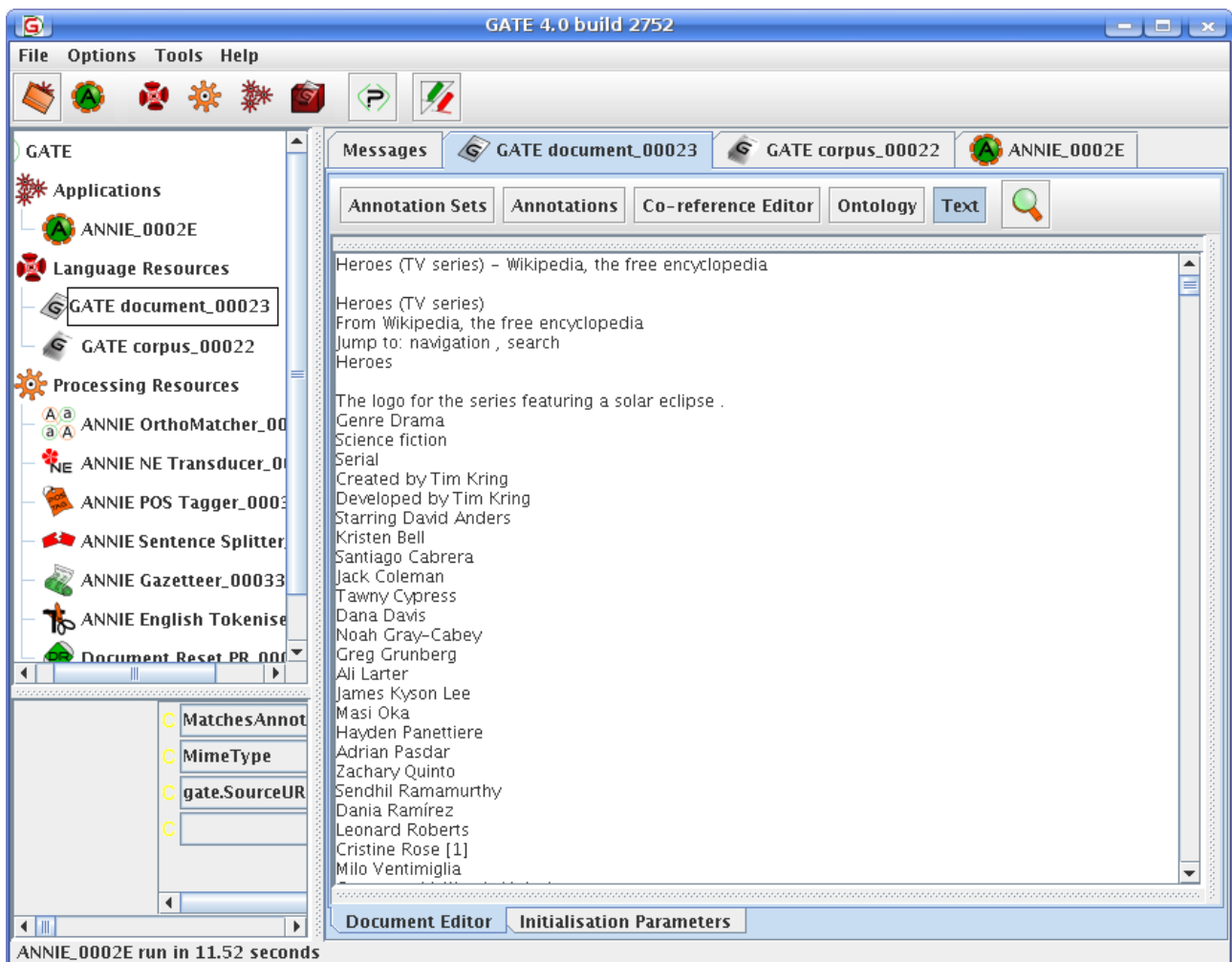
Run ANNIE by clicking the Run Button

Wait a few seconds :-)

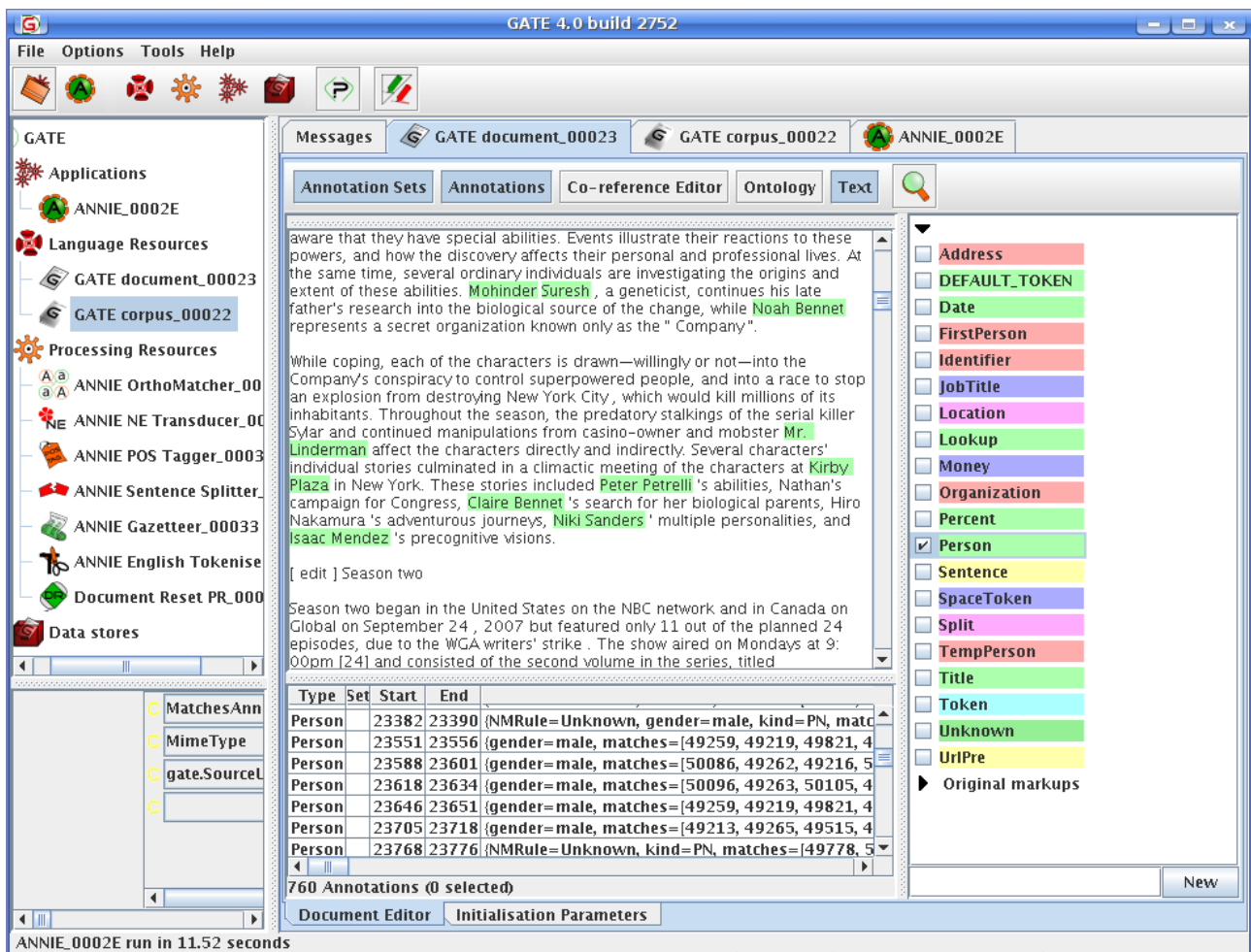
Now click on your GATE document in Language Resources tree. See icon below.



The document viewer should open as shown below.



Double Click on both **Annotations** and **Annotation Sets** buttons in the documentviewer.



Annotation Sets will appear on the Right hand side and Annotations (and their features) will appear on the left hand side.

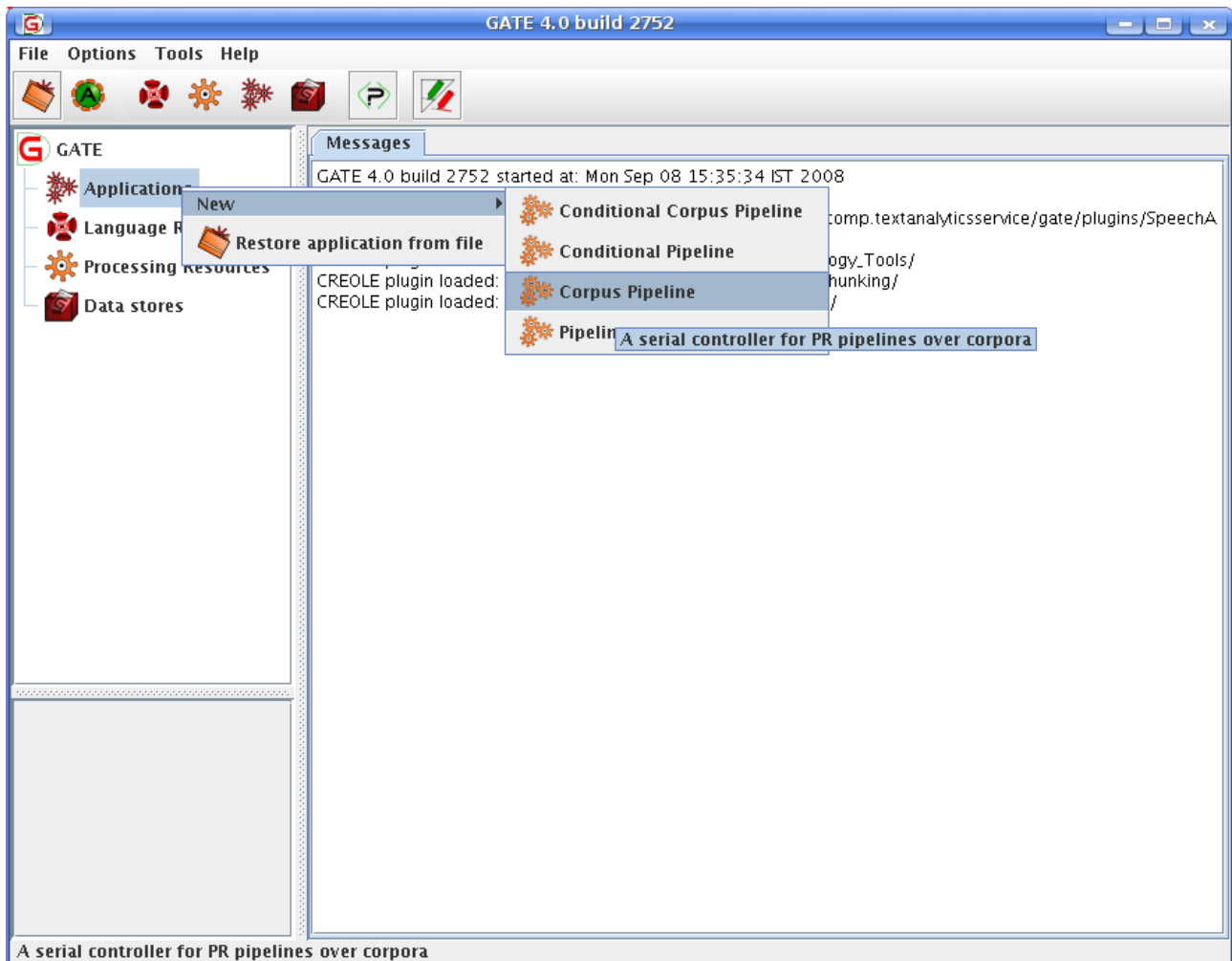
Take some time to explore the results.

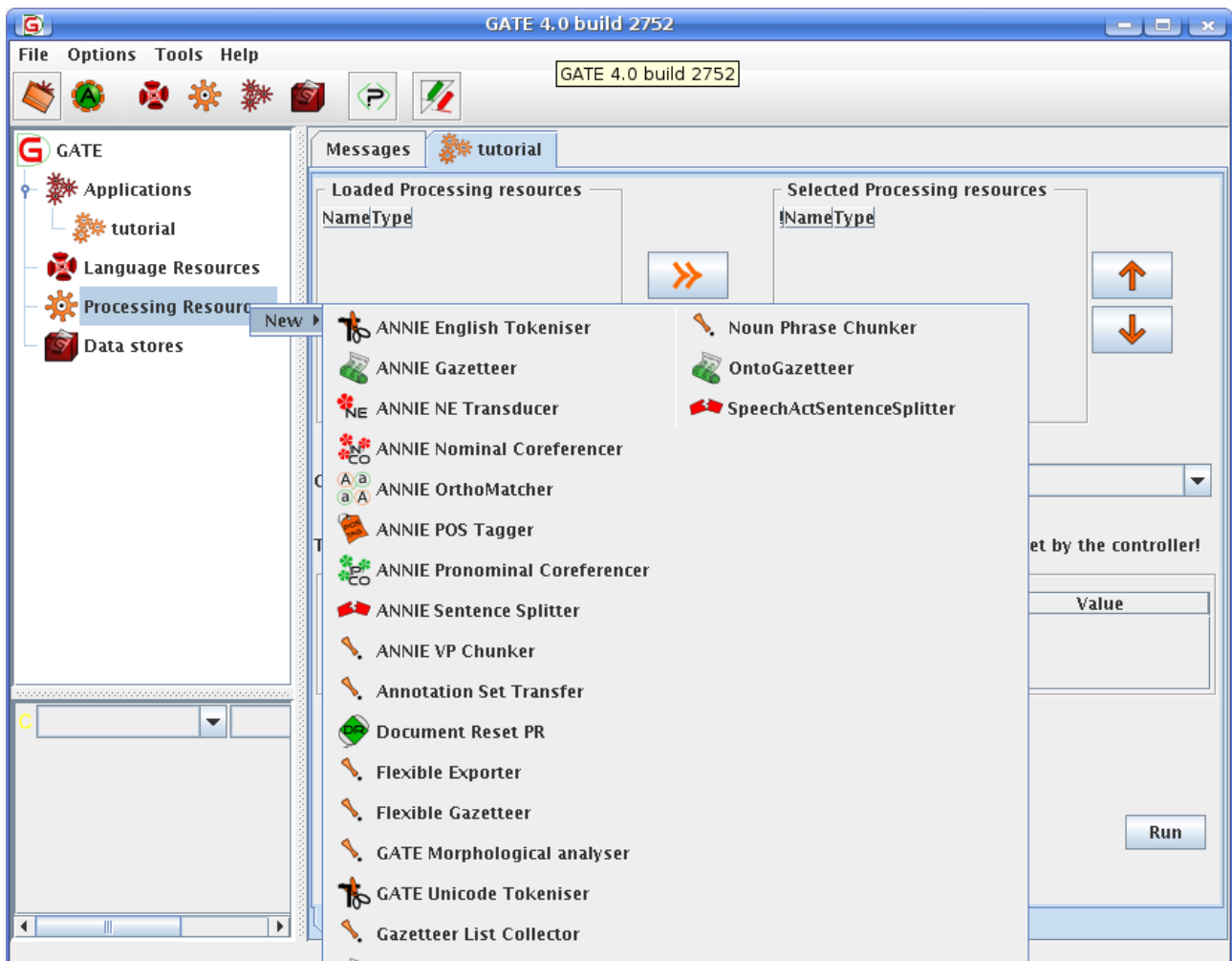
Task 4: Domain specific IE

Now that we are off to a start, lets create our own GATE application which can extract simple relations from a business text.

Step 1

Close your documents and the existing ANNIE application.
Create a Corpus pipeline in GATE.

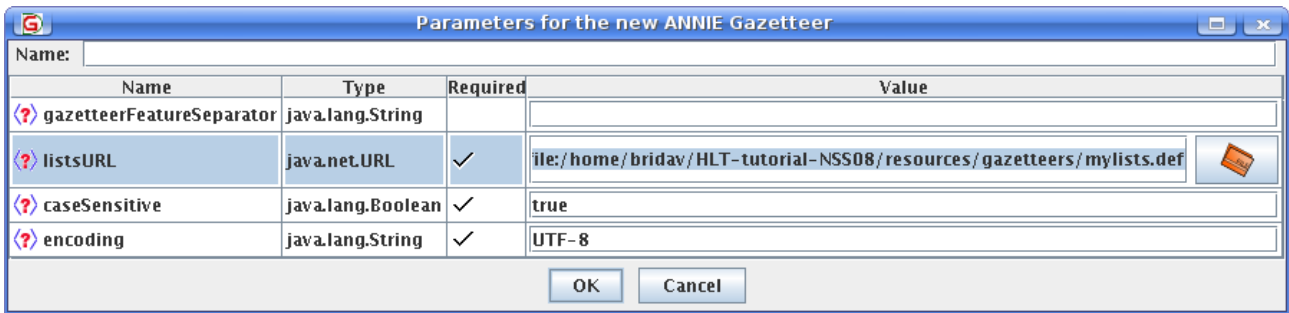




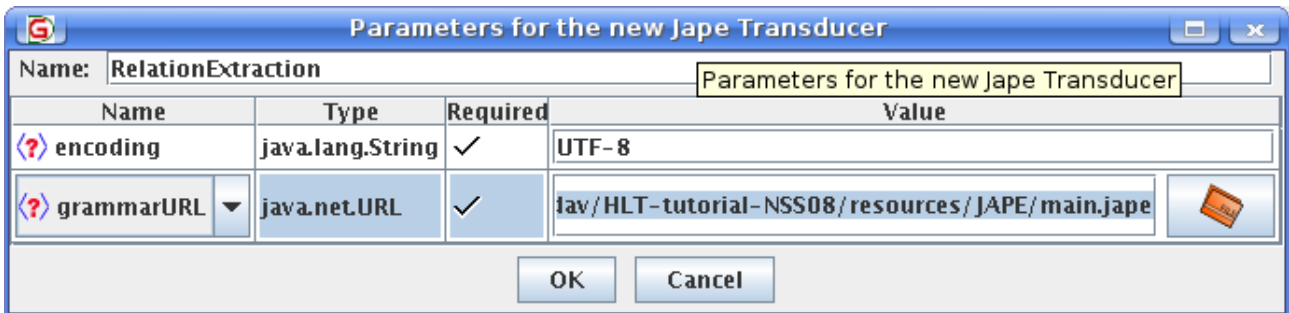
Load the following Processing Resources (PRs) by right clicking on the Processing Resource Tree:

- ANNIE English Tokeniser
- ANNIE Gazetteer
- ANNIE POS Tagger
- ANNIE SentenceSplitter

Click Ok for all of the above PRs to accept the default values **EXCEPT the ANNIE Gazetteer**. See the ANNIE gazetteer settings below. Point the **listsURL attribute** to following path:
file: <YOURHOMEDIR><MYTUTORIALDIR>/resources/gazetteers/mylists.def



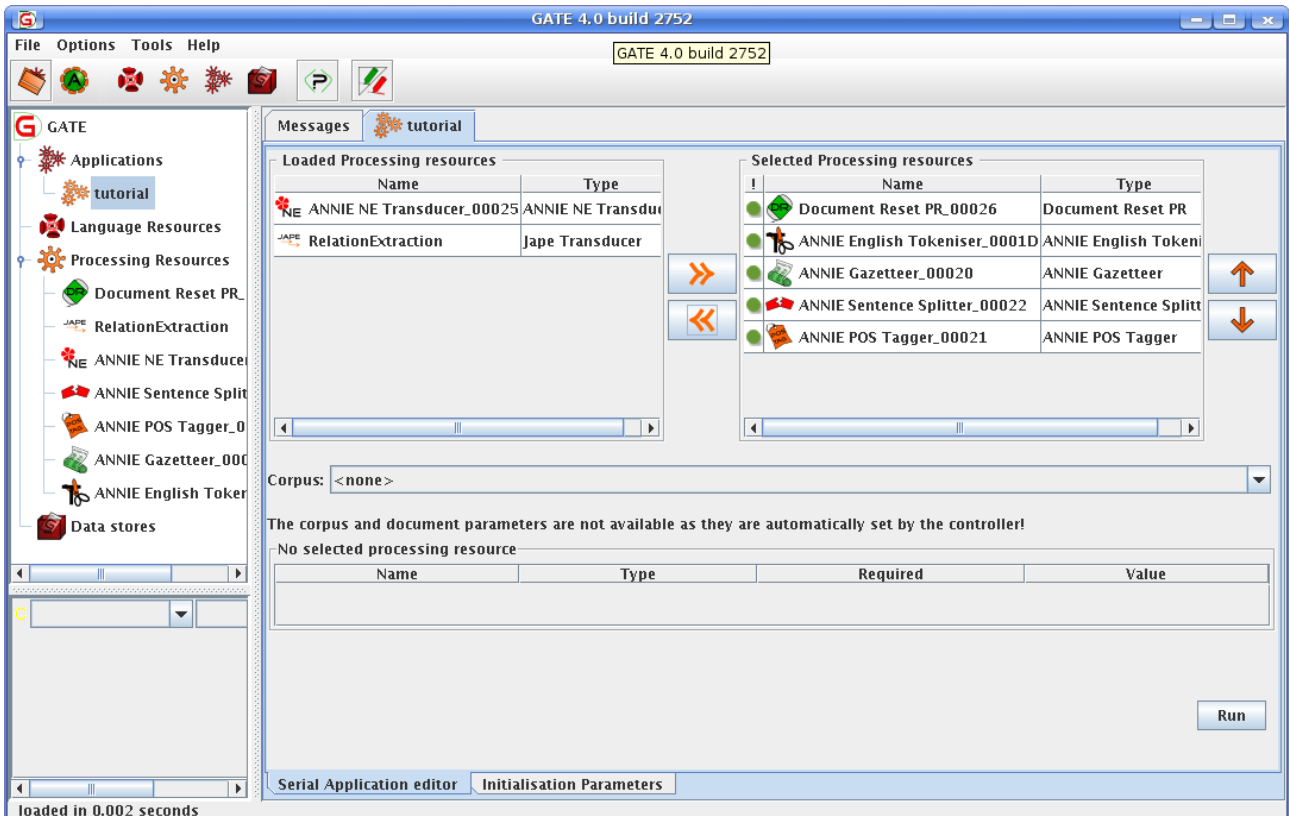
Now, create a new JAPE transducer PR.



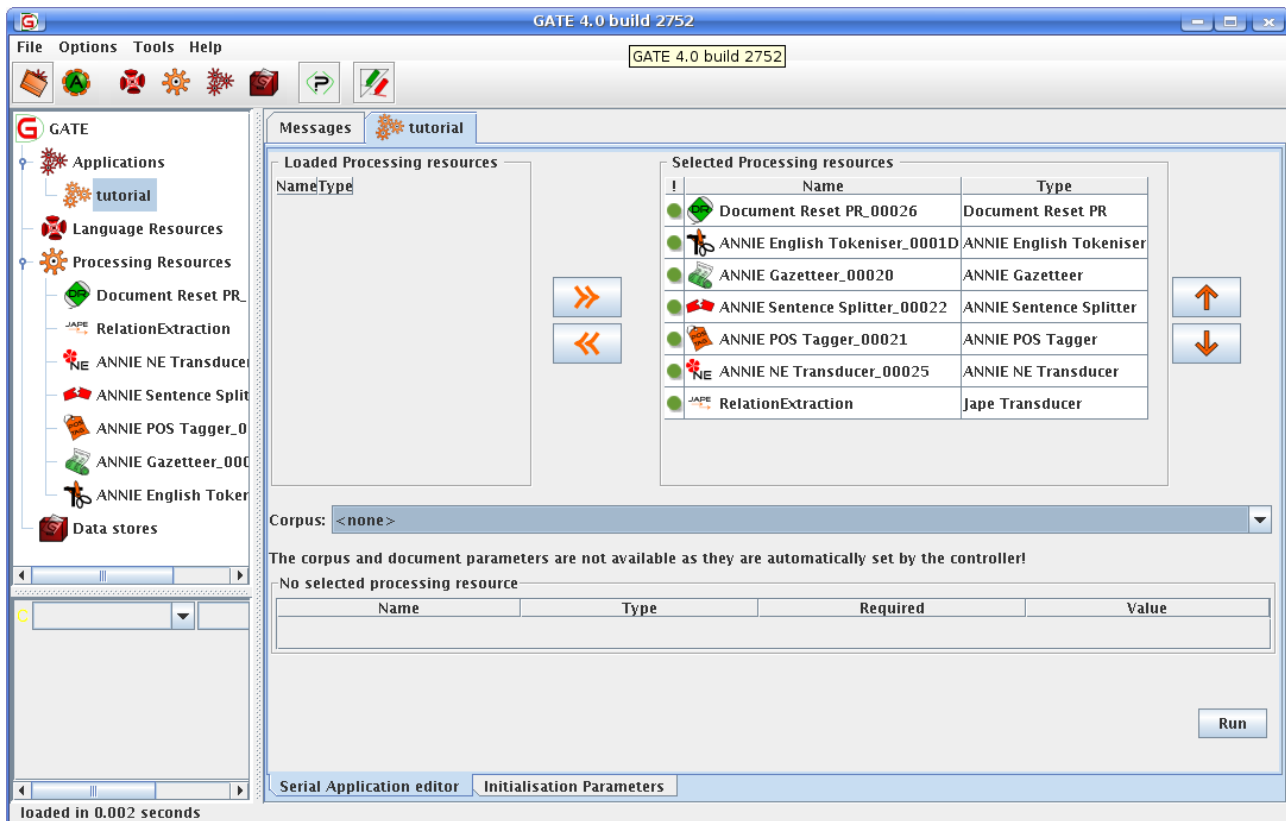
In the grammarURL attribute value, browse the material copied from within the Tutorial folder to this location:

<file:./<YOURHOMEDIR><MYTUTORIALDIR>/resources/JAPE/main.jape>

Using the green arrows in the Application Viewing pane for the "tutorial" pipeline. Move the PRs accross and "drop" them in the Corpus Pipeline.



until your pipeline looks like the following below in the **correct order**:

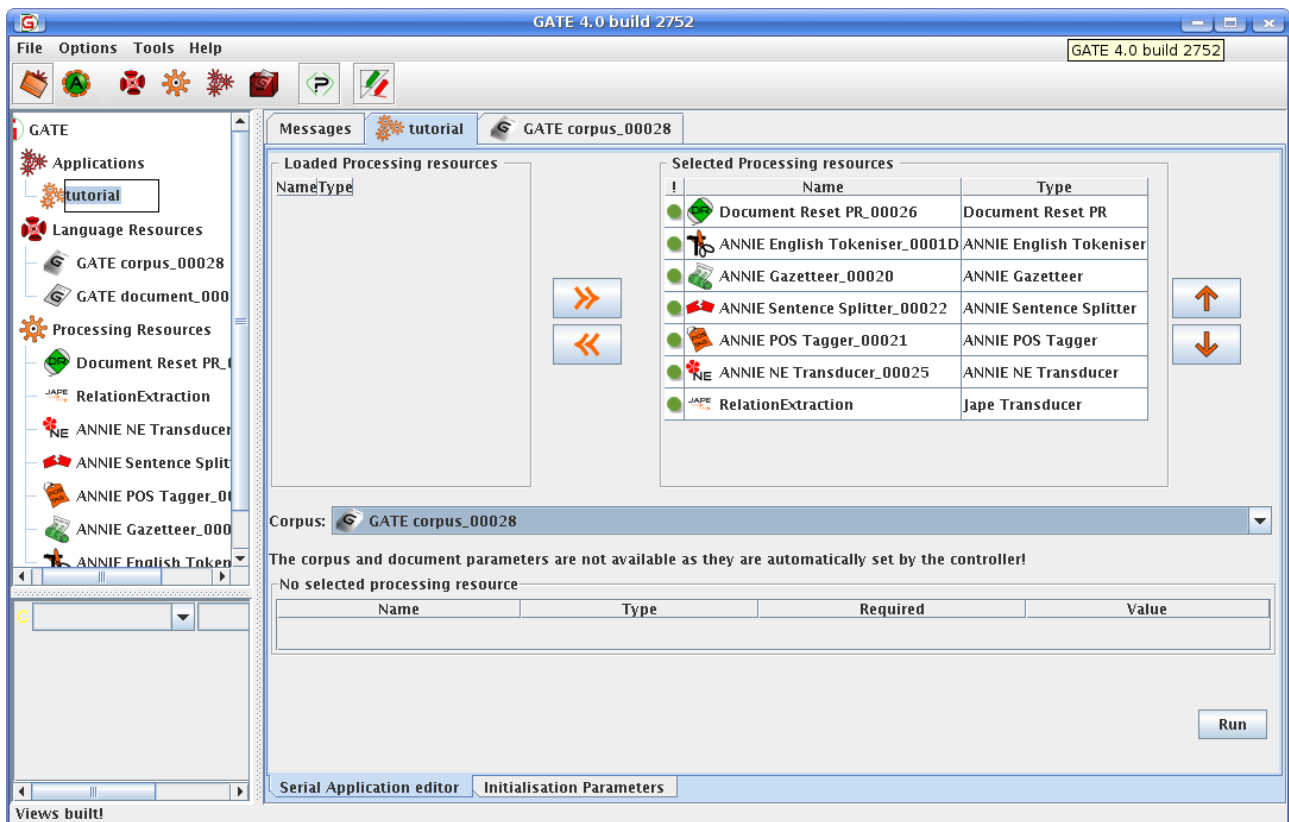


Follow the same steps used in Task 3 to create a corpus.

In addition create a document, but this time use the following file from the tutorial folder:

file/<YOURHOMEDIR><MYTUTORIALDIR>/documents/BusinessText.txt

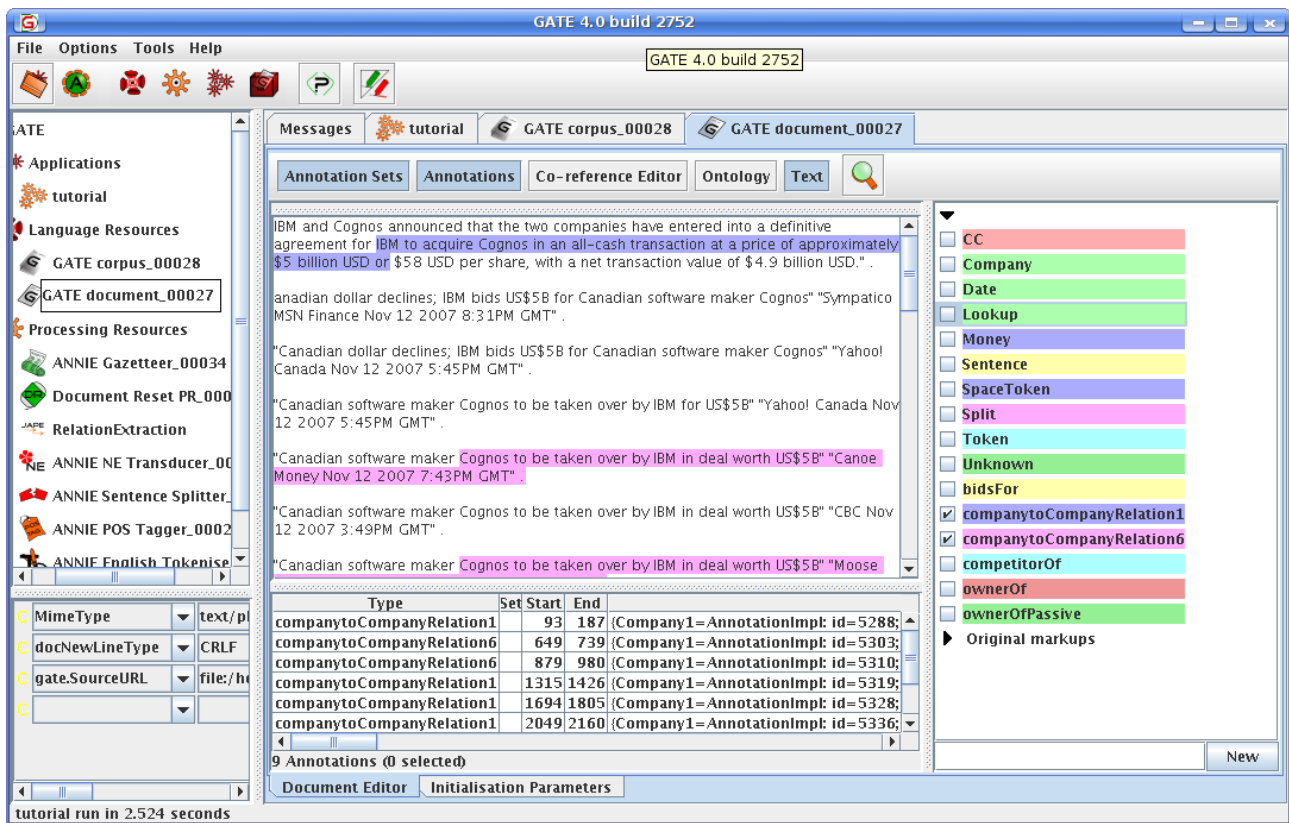
Add the GATE document to the GATE Corpus and set the Corpus in the "tutorial" pipeline. (Repeat of steps in Task 3)



Your Corpus pipeline should look like this.

Run the pipeline and wait until processing is completed. After this has occurred, double click on the GATE document (same steps as in Task 3).

Lets take a look at the annotations outputted:



Notice there are two annotation sets "clicked on" in the GUI – **companytoCompanyRelation1** and **companytoCompanyRelation6**

Both relations are concerned with relations of type acquisition between **Company** annotations. **companytoCompany1** is concerned with **Company1** **acquiring Company2** and **companytoCompany6** concerns acquiring relations where the trigger phrases are in the passive.

Lets have a little more look in detail at the JAPE grammars which do all the work.

Open the file: <YOURHOMEDIR><MYTUTORIALDIR>/resources/JAPE/main.jape

```
MultiPhase: RelationExtractionGrammars
```

```
Phases:
```

```
preprocessing1
```

```
preprocessing2
```

```
acquireRelation
```

Now Lets take a closer look at file

<YOURHOMEDIR><MYTUTORIALDIR>/resources/JAPE/preprocessing

2.jape

```
/*
 * preprocessing2.jape
 * Copyright (c) 2007- , DERI, National University of Ireland,Galway.
 * Brian Davis, 12 December 2007
 * Project: Training - Prototype
 * $Id: preprocessing2.jape,v 1.0 2007/12/12 19:24:00 GMT
 *
 * The Phase separates Annotations of Company relations from Lookup Annotations
 */

Phase: preprocessing2
Input: Lookup

Rule: Preprocess1
(
{Lookup.majorType==acquires_verb}
):acquiresVerb

-->
:acquiresVerb
{
    gate.AnnotationSet matchedAnns=
    (gate.AnnotationSet)bindings.get("acquiresVerb");
    gate.FeatureMap newFeatures= Factory.newFeatureMap();
    newFeatures.put("rule","Preprocess1");

    annotations.add(matchedAnns.firstNode(),matchedAnns.lastNode(),"ownerOf", newFeatures);
}

/* - acquires, took over */
```

As mentioned earlier JAPE consists of collection of rules each one consisting of a set of LHS --> RHS rules or <Pattern> --> {action} rules. The LHS of a JAPE rule uses Regular expressions (|, ?, *, +) over annotations while the RHS consists of a block of JAVA code for annotation manipulation. In the above rule

acquiresVerb is a binding variable for the Lookup annotation (Gazetteer Lookup). The binding variable is cast into an `AnnotationSet` object (`matchedAnns`). A `FeatureMap` object is created to store additional linguistic information, other annotations and/or in case this debugging information (i.e. If I want to know the name of the Jape rule that fired to create this annotation.) `FeatureMaps` store arbitrary attribute/value pairs as type `String/Object Java Objects` respectively. Hence the value of a `Feature` can be anything. Finally, in the last line we create and add a new annotation to the default annotationSet `annotations`.

```
annotations.add(matchedAnns.firstNode(),matchedAnns.lastNode(),"ownerOf", newFeatures);
```

The four parameters of the `add` method are:

The (1) start and the end (2) positions of the `matchedAnns` annotation, (3) the name of the new annotation as a `String` literal, in this case "ownerOf" and (4) the `FeatureMap` object, `newFeatures`.

Now a new annotation "OwnerOf" has been created, which we will manipulate at a later stage.

Now Lets take a closer look at file

<YourHomeDir><MYTUTORIALDIR>/resources/JAPE/acquireRelation.jape

```
/*
 * acquireRelation.jape
 *
 * Copyright (c) 2007- , DERI, National University of Ireland, Galway.
 * Brian Davis, 12 December 2007
 * Project: Lion - Prototype
 * $Id: acquireRelation.jape,v 1.0 2007/12/12 16:47:00 GMT
 */

Phase: CompanyRelations

Input: Company ownerOf Split CC

Rule: CompanytoCompanyRelation1

(
  ({Company}):c1 ({ownerOf}):v ({Company}):c2 ({Split}|{CC})
):companytoCompanyRelation1

-->

:companytoCompanyRelation1

{
  gate.AnnotationSet matchedCompanies=(gate.AnnotationSet) bindings.get("c1");
  Annotation company1=matchedCompanies.iterator().next();
  gate.AnnotationSet matchedCompanies2=(gate.AnnotationSet) bindings.get("c2");
  Annotation company2=matchedCompanies2.iterator().next();
  gate.AnnotationSet matchedVerb=(gate.AnnotationSet) bindings.get("v");
  Annotation verb=matchedVerb.iterator().next();
  gate.AnnotationSet matchedAnns= (gate.AnnotationSet)
  bindings.get("companytoCompanyRelation1");
  gate.FeatureMap newFeatures= Factory.newFeatureMap();
  newFeatures.put("Company1",company1);
  newFeatures.put("ownerOf",verb);
  newFeatures.put("Company2",company2);
  newFeatures.put("rule","CompanytoCompanyRelation1");
  annotations.add(matchedAnns.firstNode(),matchedAnns.lastNode(),"companytoCompanyRelation1", newFeatures);
}

}
```

This Jape file is responsible for extracting relations of type `companytoCompanyRelation1` and `companytoCompanyRelation6` - relations of type "ownership" between two `Company` annotations. The rules are similar to the above file with some notable exceptions:

In a Jape file you can specify the input annotations to a transducer much like you would pass parameters to a procedure or a function in any other programming languages.

Observe The line:

Input: Company ownerOf Split CC

states that the JAPE transducer should only consider the annotations of type Company #Companies already identify by gazetteer lookup and split into Company Annotations i.e "IBM", "Microsoft"

ownerOf #trigger phrases of type "ownerOf" already identified by gazetteer lookup and split into separate "ownerOf" Annotations

Split #annotations created by the Sentence Splitter - ?,.,\t,\r

CC #certain types of Coordinating conjunctions between clauses – or, and, ",,"

Note that due to specifying the Input, all other annotations will be ignored i.e. Lookup, Token etc. This also justifies separating Lookup annotations into Company etc – as JAPE patterns for relation extraction based on LookUp annotations only would be overgeneralising and the rules would never fire.

Now lets look at the LHS JAPE pattern

```
(  
{Company}:c1 {ownerOf}:v {Company}:c2 ({Split}|{CC})  
):companytoCompanyRelation1  
  
#Annotations are surrounded by {...}  
#Patterns or Annotations to be bound are to labels using (...):bindVar
```

With

```
{Split}|{CC})
```

You have a simple RE pattern which implies match either a Split or a CC annotation. The entire pattern is bound by the following label/binding variable companytoCompanyRelation1 in the following manner:

```
(<PATTERN>):bindVar  
-->  
:bindVar { JAVA BLOCK }
```

Consequently this rule will match:

"IBM to acquire Cognos in an all-cash transaction at a price of approximately \$5 billion USD or"

Now we will move The RHS action rule of this JAPE example:

Some explanations:

```
gate.AnnotationSet matchedCompanies=(gate.AnnotationSet) bindings.get("c1");
    Annotation company1=matchedCompanies.iterator().next();
```

#This piece of code converts the binding variable for the Company annotation into a an AnnotationSet called matchedCompanies. Since I am only interested in one Company, I used the AnnotationSet's iterator method to give me the first Annotation within the AnnoationSet. This process is repeated from the v and c2 labels bound to the JAPE LHS.

```
gate.AnnotationSet matchedAnns= (gate.AnnotationSet)
    bindings.get("companytoCompanyRelation1");
```

#We apply the same process again, but this time we create a new annotation set that spans the entire relation containing the Company ownerOf Company annotations.

```
gate.FeatureMap newFeatures= Factory.newFeatureMap();
newFeatures.put("Company1",company1);
newFeatures.put("ownerOf",verb);
newFeatures.put("Company2",company2);
newFeatures.put("rule","CompanytoCompanyRelation1");
```

Now we create a new FeatureMap and attribute\value pairs. Recall that value of attribute can be any Java Object. In the above example the new annoations for ownerOf and the companies involved in the relation are "chained" or embedded within the companytoCompanyRelation1 using the FeatureMaps.

```
    annotations.add(matchedAnns.firstNode(),matchedAnns.lastNode(),"companytoCompanyRelatio
n1", newFeatures);
```

Finally just as before we add our new companytoCompanyRelation1 to the default annotationSet annotations. Later we I will show how we will access these annoations programatically in GATE and map them into RDF triples.

BUT First lets play a little with JAPE -

Task 5: X marks the spot !

Create a new JAPE rule to extract isCustomerOf relations between companies.

Use the following file: **<YourHomeDir>HLT-tutorial /NSS08/resources/JAPE/myjape.jape**

```
/*
```

```
* myjape.jape
```

```
*
* Copyright (c) 2008- , Some University .
* Somebody, 09 September 2008
* Project XXXXXXXX
*/
```

Phase: CompanyRelations2

Input: Company customerOf Split

Options: control=all

Rule: CompanytoCompanyRelation2

```
(
({Company}):c1({customerOf}):v({Company}):c2 {Split}
):companytoCompanyRelation2
```

-->

:XXXXXXXXXXXX

```
{
```

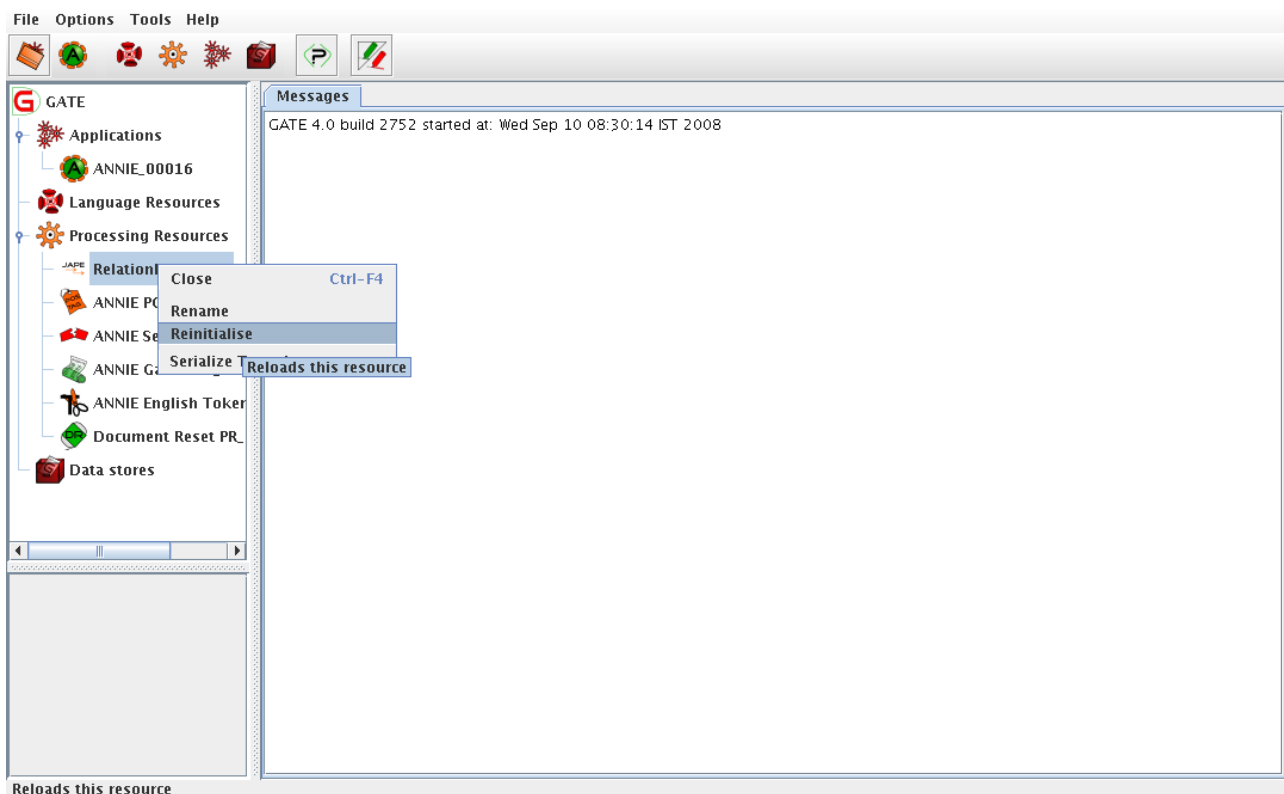
```
    gate.AnnotationSet matchedCompanies=(gate.AnnotationSet)
bindings.get("c1");
    Annotation company1=matchedCompanies.iterator().next();
    gate.AnnotationSet matchedCompanies2=(gate.AnnotationSet)
bindings.get("c2");
    Annotation company2=matchedCompanies2.iterator().next();
    gate.AnnotationSet matchedVerb=(gate.AnnotationSet) bindings.get("v");
    Annotation verb=matchedVerb.iterator().next();
    gate.AnnotationSet matchedAnns= (gate.AnnotationSet)
bindings.get("XXXXXXXXXXXX");
    gate.FeatureMap newFeatures= Factory.newFeatureMap();
    newFeatures.put("Company1", company1);
    newFeatures.put("customerOf", verb);
    newFeatures.put("Company2", company2);
```

```
newFeatures.put("rule", "CompanytoCompanyRelation2");  
annotations.add(XXXXXXXXXX);  
}
```

```
/* Company 2 Company relation - buys, purchases, is a customer of */
```

Once you have finished – reinitialize your JAPE transducer – but dont forget to amend your main.jape file to include your new Grammar.

Reinitialize your JAPE transducers like so:



More Advanced Problems:

What happens if we have false positives i.e.

Oracle's Larry Ellison has long threatened to acquire BEA Systems, finally making an offer on October 12, 2007, **however** this was rejected by BEAs Board of Directors.

SAP **no** longer plans to buy Business Objects.

Write a JAPE file to handle the following exceptions: contradiction and negation.

Tips

You will need to:
add new gazeteer entries.
Modify preprocessing2.jape
Modify acquireRelation.jape

Hint one way to do it is to create a *Contrst* annotation (for contrastive conjunction). A contrastive conjunction implies the preceding clause will be contradicted by the preceding subclause
We could create a new rule as such:

Phase: CompanyRelations2

Input: Company customerOf Split Contrst

Options: control=appelt

Priority:25

Rule: CompanytoCompanyRelationNeg

```
(  
  
  ({{Company}}):c1 ({{ownerOf}}):v ({{Company}}):c2 ({{Contrst}})  
  
):companytoCompanyRelationNeg  
  
--> {}
```

and add it into

<YourHomeDir><MYTUTORIALDIR>/resources/JAPE/acquireRelation.jape

Make sure it precedes the old rule. Why?

Notice in the file we currently have set the control of this jape grammar to *all*. This means all rules will fire! We can change it to:

Options: control=appelt

This means the rule with the longest match will fire first! In addition you can specify the priority of rules to fire (just to be on the safe side) i.e.

Priority:25

You do not need to change the priority of the other rules in this case since unless specified, the default priority for all rules is -1 which is the lowest

priority. Hence priorities are positive integers.

Finally you'll notice `--> {}`. This is an empty Java block, hence when it is fired it will do nothing. More importantly JAPE will consume the annotations and continue matching from the next offset. If the rule does not match then the next set of **positive** rules can proceed in priority to match.

You could use the negative operators in JAPE to collapse the rule `CompanytoCompanyRelationNeg` into `CompanytoCompanyRelation2`

```
Rule: CompanytoCompanyRelation2
```

```
(  
{Company}:c1({customerOf}):v({Company}):c2 !({Contrst}){Split}  
)
```

Be warned though that negating `Contrst` tells JAPE to match any other annotation except `Contrst`, hence in this case you could match `Company`, `customerOf` or `Split` which could lead to overgeneration of your rule or unexpected results. If you are unsure which approach to use, pick the first one (`<PATTERN> --> {}`).

For Testing use:

file/<YOURHOMEDIR><MYTUTORIALDIR>/**documents/BusinessTextDecoy.txt**

Finally, those of you with a theoretical computational linguistics background will not doubt express concerns over the oversimplicity of the rules to handle negation above. In particular, it would be difficult to account for all syntactic variations of negation before or after the verb trigger phrase `customerOf`. Furthermore JAPE patterns have the equivalent recognition power of a regular language and one might presume this would be insufficient for your needs. One would assume JAPE cannot backtrack, but in fact some degree of backtracking is permitted since the annotation sets you have created are always accessible via the right hand side Java block, hence you can always access the memory of your parse/annotation information and delete, amend, update annotation sets or specific annotations. Furthermore you can check for intersections with your matched annotation span and other annotations which is useful if you cannot account for the syntactic ordering of some linguistic phenomenon (HINT for negation:-) . Furthermore, there is no reason not to use other available parsers within GATE such as, NP and VP chunkers, Minipar parser, Stanford parser, SUPPLE parser, RASP parser, depending on your NLP task.

One useful advantage is that the annotations created by such parsers as Minipar, NP and VP chunkers can be later manipulated in JAPE.

A the JAVA code is available to build in the
/<YOURHOMEDIR><MYTUTORIALDIR>/**sources directory please inspect**

the README file for instructions to compile and run the program.

Thank You for your time and enjoy playing with GATE!

References

- [1]GATE User Guide See <http://gate.ac.uk/sale/tao/>**
- [2]Software Architecture for Language Engineering
<http://gate.ac.uk/sale/thesis/>**
- [3]<http://www.alphaworks.ibm.com/tech/lrw>**
- [4][http://citeseerx.ist.psu.edu/viewdoc/summary?
doi=10.1.1.36.7858](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.7858)**