Tractor Manual*

Stuart C. Shapiro, Daniel R. Schlegel, and Michael Prentice
Department of Computer Science and Engineering
and Center for Multisource Information Fusion
and Center for Cognitive Science
The State University of New York at Buffalo
Buffalo, NY 14260-2000
shapiro@buffalo.edu

January 10, 2015

Contents

*The intended readership of this Manual includes users, maintainers, and developers. The development of Tractor has been supported by a Multidisciplinary University Research Initiative (MURI) grant (Number W911NF-09-1-0392) for "Unified Research on Network-based Hard/Soft Information Fusion", issued by the US Army Research Office (ARO) under the program management of Dr. John Lavery.

1 What is Tractor?

Tractor is a system for understanding English messages within the context of hard and soft information fusion for situation assessment. Tractor processes a message through text processors, and stores the result in a syntactic propositional graph. This graph is then enhanced with ontological and geographic information. Finally, Tractor applies hand-crafted syntax-semantics mapping rules to convert the enhanced syntactic graph into a semantic propositional graph containing the information from the message.

Tractor consists of four main components: GATE, the Propositionalizer, CBIR, and the Syntax-Semantics Mapper.

1.1 GATE

GATE is the General Architecture for Text Engineering developed at the University at Sheffield. It is a framework for building text processing applications which consist of pipelined sets of Processing Resource (PR) plug-ins that operate on corpora of documents, referred to as Language Resources. Most of the PRs being used are part of the ANNIE (a Nearly-New Information Extraction) System. The GATE Developer application is used to set up GATE applications and corpora of language resources (documents). Consult the GATE manual[1] for more information.

As used in Tractor, the output of GATE is a set of "annotations", including dependency relations that result from a dependency parse of the text.

1.2 Propositionalizer

The Propositionalizer translates the set of annotations output from GATE into a propositional graph containing mostly syntactic information, and therefore referred to as a "syntactic propositional graph." This component performs several tasks in preparing the GATE output to be sent to the Syntax-Semantics mapper, including annotation merging, correction of minor errors in syntactic categories, canonicalization of dates and times, and processing the structured portion of messages with structured components.

1.3 CBIR

CBIR is a component that adds ontological/taxonomic information about the nouns and verbs in the message, along with additional information about locations, to the propositional graph. The result is referred to as an "enhanced syntactic propositional graph."

1.4 Syntax-Semantics Mapper

The Syntax-Semantics Mapper converts the enhanced syntactic propositional graphs to semantic propositional graphs using a set of syntax-semantics mapping rules.

[1] Cunningham, et al. Developing Language Processing Components with GATE Version 8 (a User Guide). The University of Sheffield, Department of Computer Science, 2014. Available at http://gate.ac.uk/userguide/

2 Software Dependencies

Tractor is dependent on GATE, SNePS 3, Leiningen, VerbNet, WordNet, data from the NGA GEOnet Names Server, and NASA's World Wind SDK.

2.1 GATE

GATE, discussed in Sections 1.1 and 3.4, may be downloaded from http://gate.ac.uk.

On the CSE Linux servers, launch GATE with the command,

```
$ /projects/snwiz/bin/gate
```

2.2 SNePS and Allegro Common Lisp

SNePS is a logic-, frame- and network-based knowledge representation, reasoning and acting system developed by members of the SNePS Research Group at the University at Buffalo. Tractor uses SNePS 3[2], one of the latest members of the SNePS family.

On the CSE Linux servers, load SNePS 3 by running Allegro Common Lisp[3], and then evaluating

```
: (load "/projects/snwiz/Sneps3/sneps3")
```

2.3 Leiningen

Leiningen[4] is a project manager and build tool for the Clojure language on the JVM. The Leiningen installer script handles all setup and installation tasks for creating and running Clojure projects. The Leiningen project page includes instructions for setting up, which consists entirely of downloading the script and running the specified command.

Leiningen is installed on the CSE Linux machines at /util/bin/lein

[2]For information about SNePS 3, see http://www.cse.buffalo.edu/sneps/Projects/sneps3.html

[3]A free limited version of Allegro Common Lisp can be obtained from http://franz.com/products/allegrocl/, and trials of other versions can be obtained by emailing sales@franz.com.

[4]Leiningen may be downloaded from https://github.com/technomancy/leiningen.

2.4 VerbNet

VerbNet is a hierarchically organized verb lexicon for English, with mappings to WordNet. Verbs are organized into an extension of Levin classes, ensuring that each class contains verbs which are syntactically and semantically related. VerbNet can be installed by downloading the latest version from http://verbs.colorado.edu/~mpalmer/projects/verbnet.html, extracting the files to a convenient place, and updating the CBIR configuration file to point to that location.

2.5 WordNet

WordNet is a lexical database of English words, which groups nouns, verbs, adverbs, and adjectives into sets of synonyms (synsets), and relates synsets to each other through various means, such as hypernomies, and mereologies. WordNet can be installed according to the directions on the WordNet website – http://wordnet.princeton.edu/, or, for our purposes, simply by putting the `/dict/` folder included in WordNet in a convenient spot, and modifying CBIR's configuration file[5] to point to it.

2.6 NGA GEOnet Names Server Data and MySQL

The NGA GEOnet Names Server is a database of place names mapped to their coordinates. Variant spellings of names are included, as well as information about the type of the location.

Data from GEOnet is used for converting location names to coordinates and coordinates to names in CBIR. You will need a MySQL[6] server with a table created using the GEOnet schema[7], and populated with one or more of the country data files[8]. To make this easier, two SQL files are provided with Tractor in `$TractorHome/CBIR/Geonet-DB`. The first of these, `initialize-tables.sql`, will create a MySQL database named `geonet` and a table named `LocInfo` with the current GEOnet schema. The second SQL file, `load-countryfile.sql` may be customized with the name of a country file downloaded from the GEOnet website to load that data into the created table. Detailed instructions are below:

1. Be sure MySQL is installed and configured on your machine.

2. Download the desired country file(s) from http://earth-info.nga.mil/gns/html/namefiles.htm.

[5] `$TractorHome/CBIR/ClojureCBIR/src/CBIR/configuration.clj`
[6] The free MySQL Community Edition is available at http://www.mysql.com/products/community/.
[7] The GEOnet schema is described at http://earth-info.nga.mil/gns/html/gis_countryfiles.html.
[8] GEOnet country data files can be downloaded from http://earth-info.nga.mil/gns/html/namefiles.htm.

3. Each country file is a zip file. Extract from each the text file named with the country code for the country you downloaded.

4. In a terminal, run `mysql --local-infile -u root -p` and enter the MySQL root password when prompted.

5. At the `mysql>` prompt, type `source $TractorHome/CBIR/Geonet-DB/initialize-tables.sql`.

6. In a text editor, edit the `load-countryfile.sql` file to contain the path to your first country file.

7. At the `mysql>` prompt, type `source $TractorHome/CBIR/Geonet-DB/load-countryfile.sql`. You should see that many rows were affected. Warnings may be ignored.

8. Repeat steps 6 and 7 for each country file.

If you only wish to add an additional country's data to the database without removing the current contents, you may skip step 5.

2.7 NASA World Wind SDK

The NASA World Wind SDK[9] is a Java SDK for using features from World Wind (a tool for viewing satellite imagery) in other applications. We are using it for the conversion of MGRS coordinates to latitude and longitude. It is already included in the Tractor distribution.

3 Using Tractor

3.1 Summary

The input to Tractor is a corpus of documents (a set of messages, one message per file). The final output from Tractor is a set of files, each containing a propositional graph representing the information in and about one message, plus some files containing summaries and statistics. We will assume that there is a directory for each data set (set of messages), such as `BioWeaponsThread` or `BombBusterScene`. Within this directory, there should be a subdirectory containing a master set of the messages of the data set, and one subdirectory for each run of Tractor against these messages. These directories will be named for the date of the run, in the format `Mmmdd`. Each run of Tractor is comprised of the following steps:

[9]For more information about the World Wind SDK, see http://worldwind.arc.nasa.gov/java/.

1. Use a script to set the environment variable `TractorHome` to the home directory of the Tractor program suite to be used.

2. Use a script to create the directory `Mmmdd`, and a set of subdirectories.

3. Populate the subdirectory `Mmmdd/Messages` with files containing the message set.

4. Text Processing

   (a) Use GATE to perform text analysis on the message set, stored in the directory named `Mmmdd/Messages`, and produce a directory, named `Mmmdd/XMLParses`, of XML files, each containing the annotations of one message, including the annotations that represent the dependency parse of the message.

   (b) Use the Propositionalizer to convert the `Mmmdd/XMLParses` files of annotations into SNePS 3 files of syntactic propositional graphs stored in a directory named `Mmmdd/SNEPSParses`.

5. Syntax-Semantics Mapping

   (a) Use CBIR to add ontological and geographic information to the syntactic propositional graphs. Files of these enhanced syntactic propositional graphs are stored in a directory named `Mmmdd/EnhancedSyntacticGraphs`. Statistics about what CBIR accomplished are stored in a file named `Mmmdd/cbirStats.txt`.

   (b) Use Lisp and SNePS 3 to perform syntax-semantics mapping and produce files of semantic propositional graphs stored in a directory named `Mmmdd/SNEPSPropGraphs`. The same graphs, but in XML format are stored in a directory named `Mmmdd/SNEPSXML`. In addition, the following files are written.

      i. Counts of the number of words looked up by CBIR, and the number of them found by CBIR are written into the file named `cbirStats.txt`.

      ii. Counts of the number of times each syntax-semantics mapping rule fired on each message graph are written into the file named `Mmmdd/ruleUsage.csv`.

      iii. A file named `Mmmdd/synsemNumbers.csv` is written containing one line for each message. On that line is:

         A. the name of the message file;

         B. the number of tokens in the message;

         C. the number of syntactic assertions in the semantic graph;

D. the number of semantic assertions in the semantic graph;

E. the percent of syntactic and semantic assertions that are semantic.

Also, depending on the value of the environment variable, `CloudStorageDirectory`:

i. semantic propositional graphs represented in XML will either be in a directory named `Mmmdd/SNEPSXML`, or in some other directory in the Cloud;

ii. text files of descriptions of all the entities and events represented in each semantic propositional graph will either be in a directory named `Mmmdd/Descriptions`, or in some other directory in the Cloud.

iii. comma-separated values files of descriptions of all the entities and events represented in each semantic propositional graph will either be in a directory named `Mmmdd/Answers`, or in some other directory in the Cloud.

The following sections provide more detail, using, as an example, the Bioweapons subset of the SynCOIN dataset.

3.2 Setting `TractorHome`

There are multiple copies of the Tractor software, including: a copy in a CVS repository; an "official" copy in `/projects/snwiz/Tractor`; one development copy in the directory tree of each of the developers. The various scripts in the Tractor suite use the environment variable `TractorHome` to load and run the correct version of the programs. So the first thing to do is to set the desired value of `TractorHome` by doing the following:

1. Change the working directory to the home directory of the Tractor software you wish to use.

2. Execute the Unix command,

```
$ source anchorTractor
```

In the rest of this manual, we will use `$TractorHome` to mean the home directory of the Tractor suite being used.

3.3 Preparing the Directory

To create a directory for the run of Tractor, set your working directory to the data set directory, such as `BioWeaponsThread` or `BombBusterScene`. Then execute the Unix commands,

```
$ "$TractorHome/makedirectories"
$ cd Mmmdd
```

This will create a directory named `Mmmdd`, where `Mmm` is an abbreviation of the current month, and `dd` is the current date, and will make that your current working directory. It will also create the following subdirectories.

Messages for the messages in plain text.

XMLParses for GATE output containing annotations, including the dependency parses.

SNEPSParses for syntactic propositional graphs.

EnhancedSyntacticGraphs for syntactic propositional graphs enhanced by CBIR.

SNEPSPropGraphs for SNePS 3 representations of the semantic propositional graphs.

SNEPSXML for the semantic propositional graphs in XML format.

GMLPropGraphs for GraphML representations of the semantic propositional graphs.

CSVPropGraphs for comma-separated values representations of the semantic propositional graphs.

Descriptions for text files containing descriptions of entities and events.

Answers for comma-separated values files containing lists of the entities and events, and their attributes and relations for use in grading the performance of Tractor.[10]

Then populate the `Mmmdd/Messages` subdirectory with the messages to be processed, one message per file.

In the rest of this Manual, we will use `Mmmdd` to refer to the main directory created in this step.

3.4 Performing Text Analysis

If you have run GATE Developer previously, your previous session will be restored when you launch GATE. If you instead want to start with a fresh session, delete the directory `.gate` and the files `.gate.xml` and `.gate.session` from your home directory before launching.

On the CSE Linux servers, launch GATE with the Unix command:

```
$ /projects/snwiz/bin/gate &
```

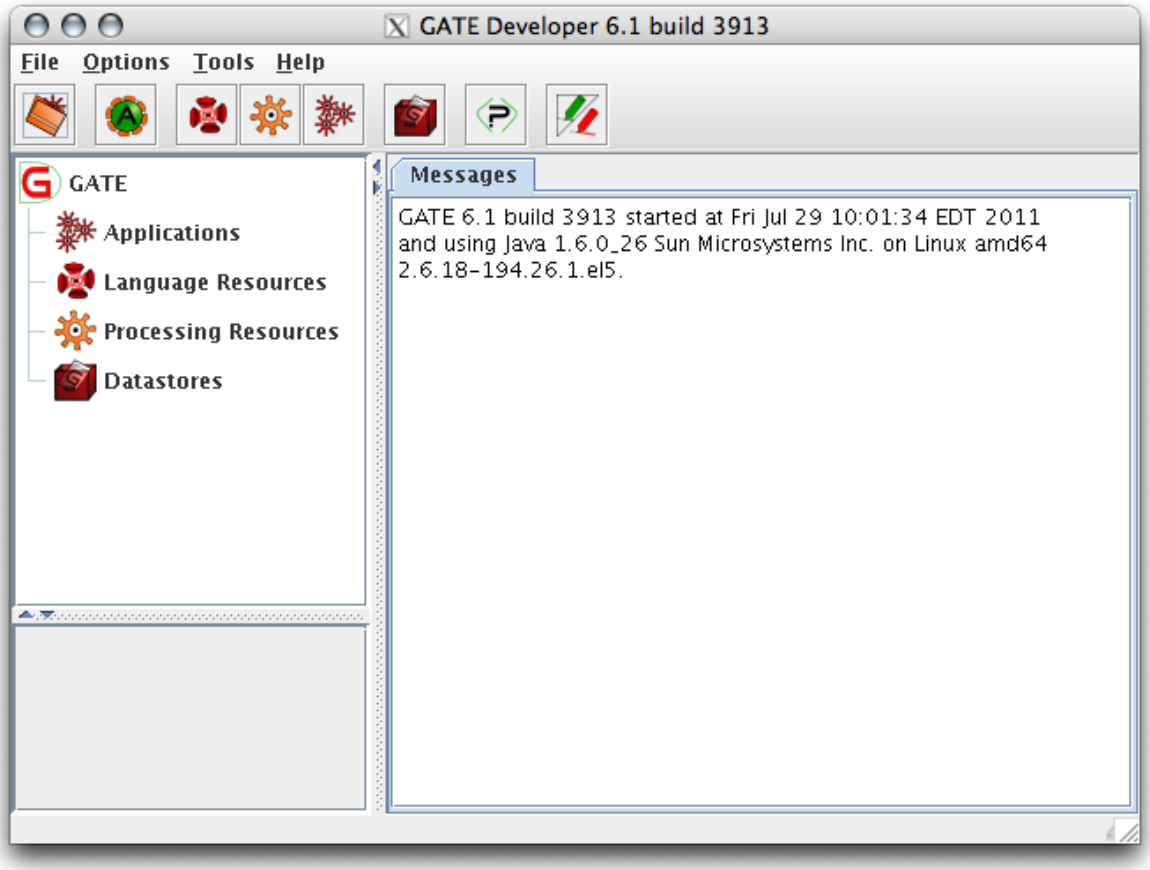

Figure 1: The GATE Developer application on startup

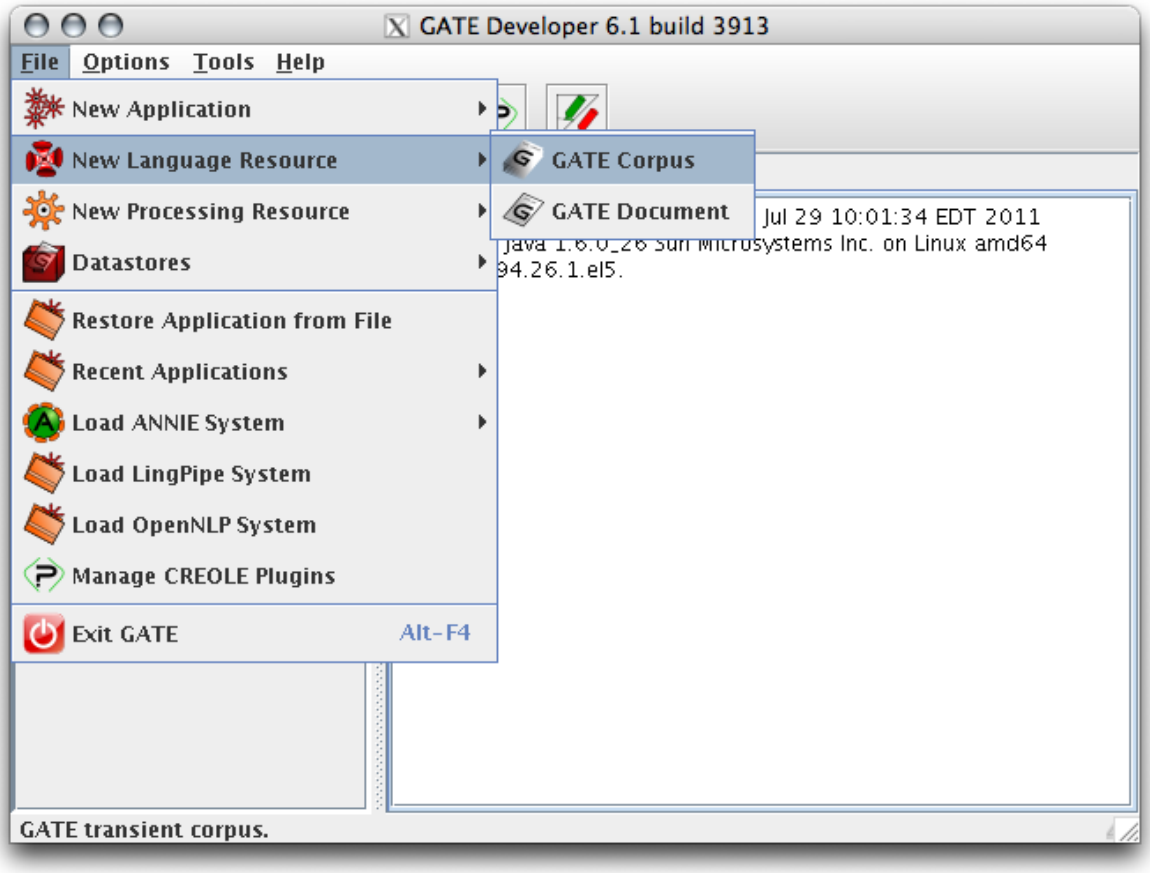

Figure 2: Create a new corpus

You should see the initial GATE Developer window as in Figure 1 Across the top is the **menus bar**, then the **tools bar**. Below the tools bar, on the left is the **resources pane,** containing the **resources tree**, and below the resources pane is the **small resource viewer.** The large pane to the right of the resources pane is the **main resource viewer**. A **messages bar** is at the bottom.

3.4.1 Set Up a Corpus

A GATE corpus is a container for the documents we want to analyze. Our files must be loaded into a GATE corpus.

From the File menu, choose New Language Resource, then GATE Corpus, as shown in Figure 2.

You should see a dialog window titled "Parameters for the new GATE Corpus", as in Figure 3. Type in a name for your corpus (named "biothread" in this example) and click the OK button.

[10] *See* Stuart C. Shapiro, A Gradng Rubric for Soft Information Understanding, University at Buffalo, 2013.

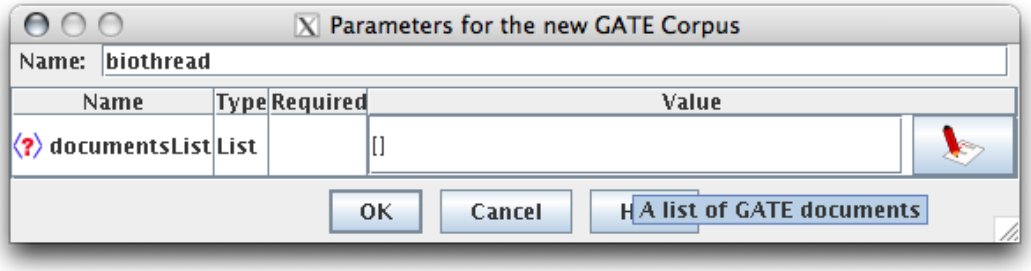

Figure 3: Name the corpus ("biothread" here)

Your new corpus shows up under Language Resources in the resources tree. Right click on the corpus name and choose `Populate` from the menu, as shown in Figure 4.

Click on the folder icon to the right of the Directory URL text field, as shown in Figure 5. Choose the directory containing your documents from the file chooser dialog (not pictured here). We will assume that the directory of documents is named `Mmmdd/Messages`.

Click the OK button.

The document names will appear under Language Resources with a unique suffix appended to the file names, as shown in Figure 6.

At this point, the documents are imported into GATE Developer and any changes made to the documents will not affect the original files.

If you want to delete any document from the corpus, right click on it, and select `Close`.

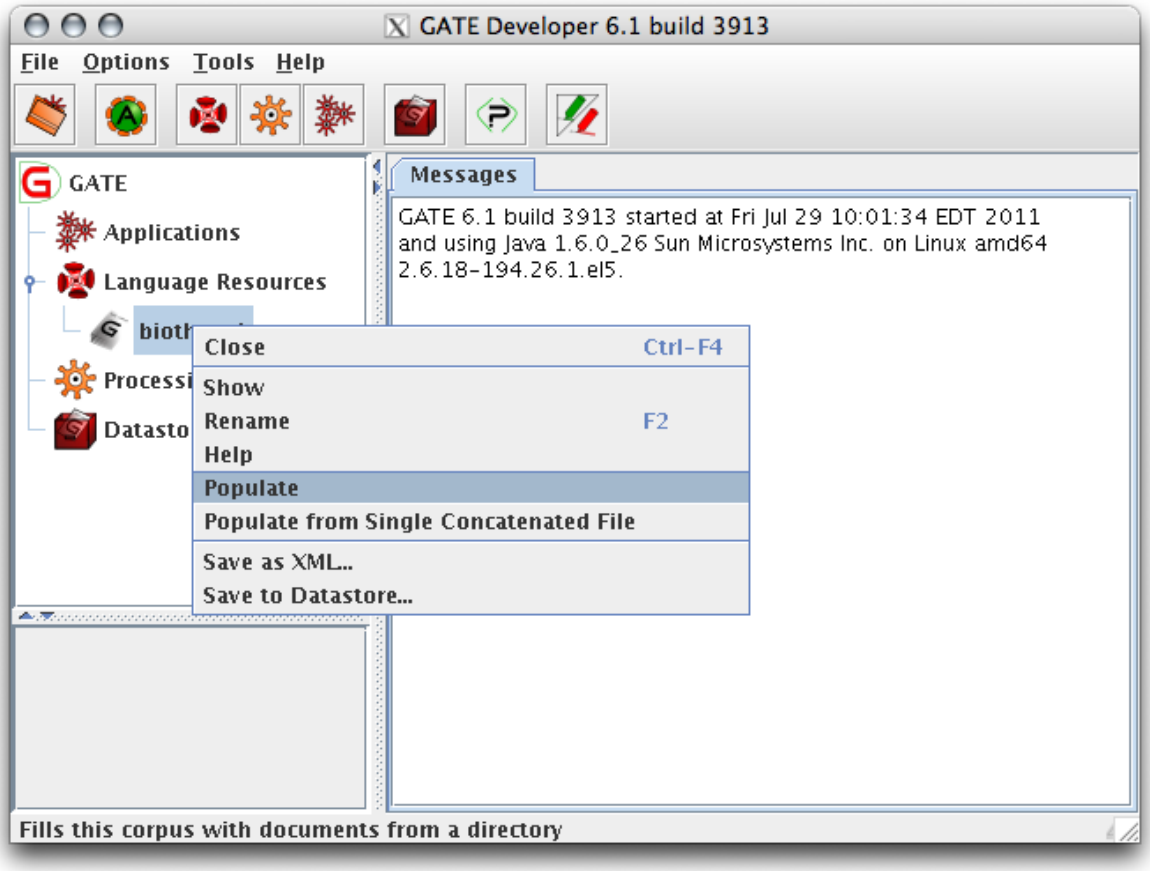

Figure 4: Populate the corpus with documents

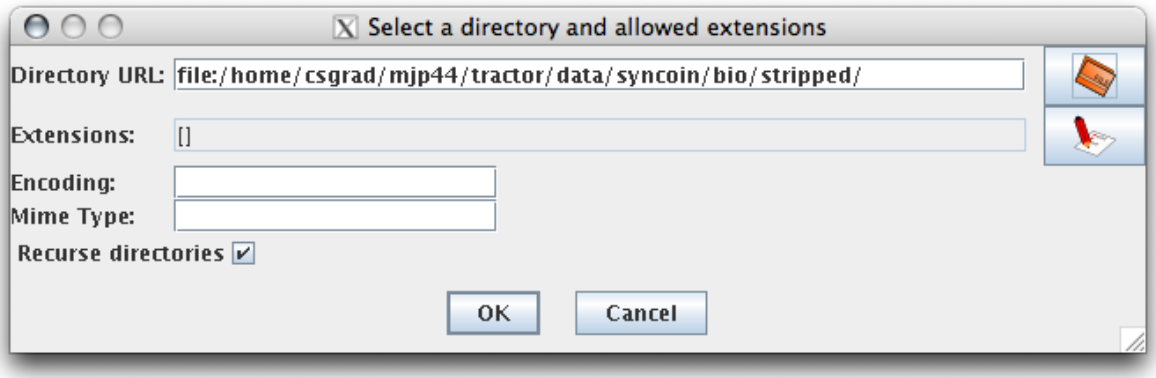

Figure 5: Click on the folder icon and select a directory

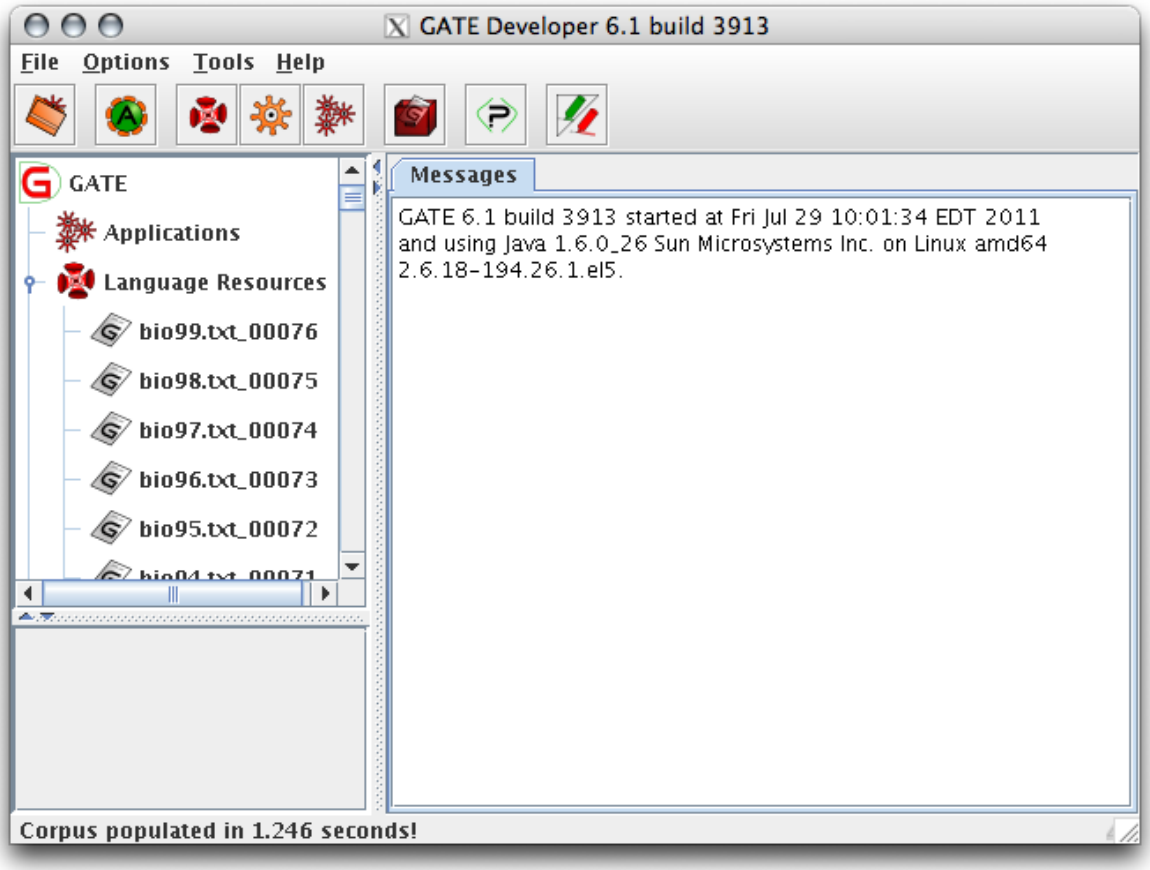

Figure 6: The corpus is populated

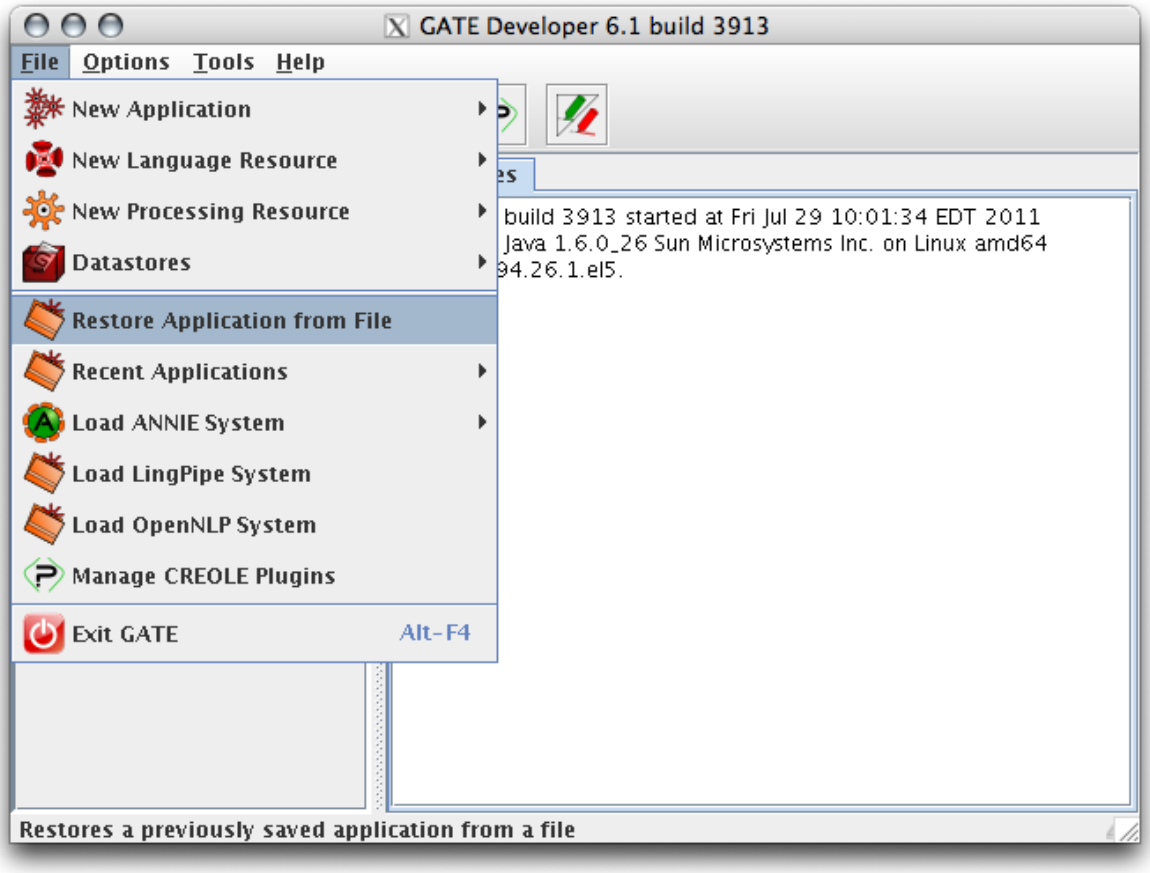

Figure 7: Restore Application from File

3.4.2 Set Up the GATE Application

Now we are ready to load the Tractor application into GATE Developer.

From the File menu, choose `Restore Application from File`, as shown in Figure 7.

The recommended default application is `$TractorHome/tractor.gapp`. So in the file chooser dialog window, choose the GATE application file `tractor.gapp`, as shown in Figure 8.

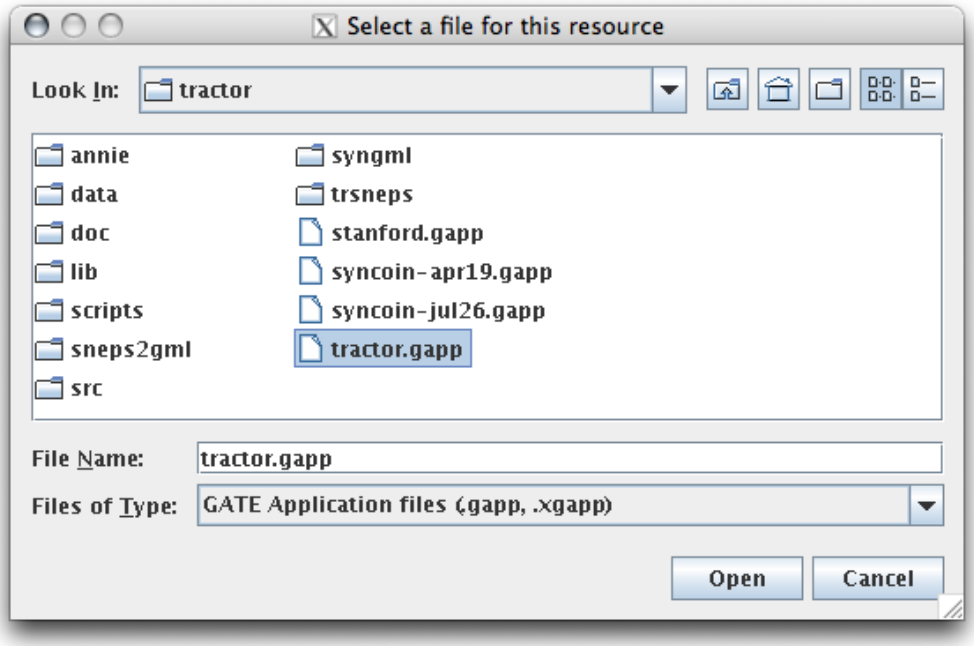

Figure 8: Choose "tractor.gapp"

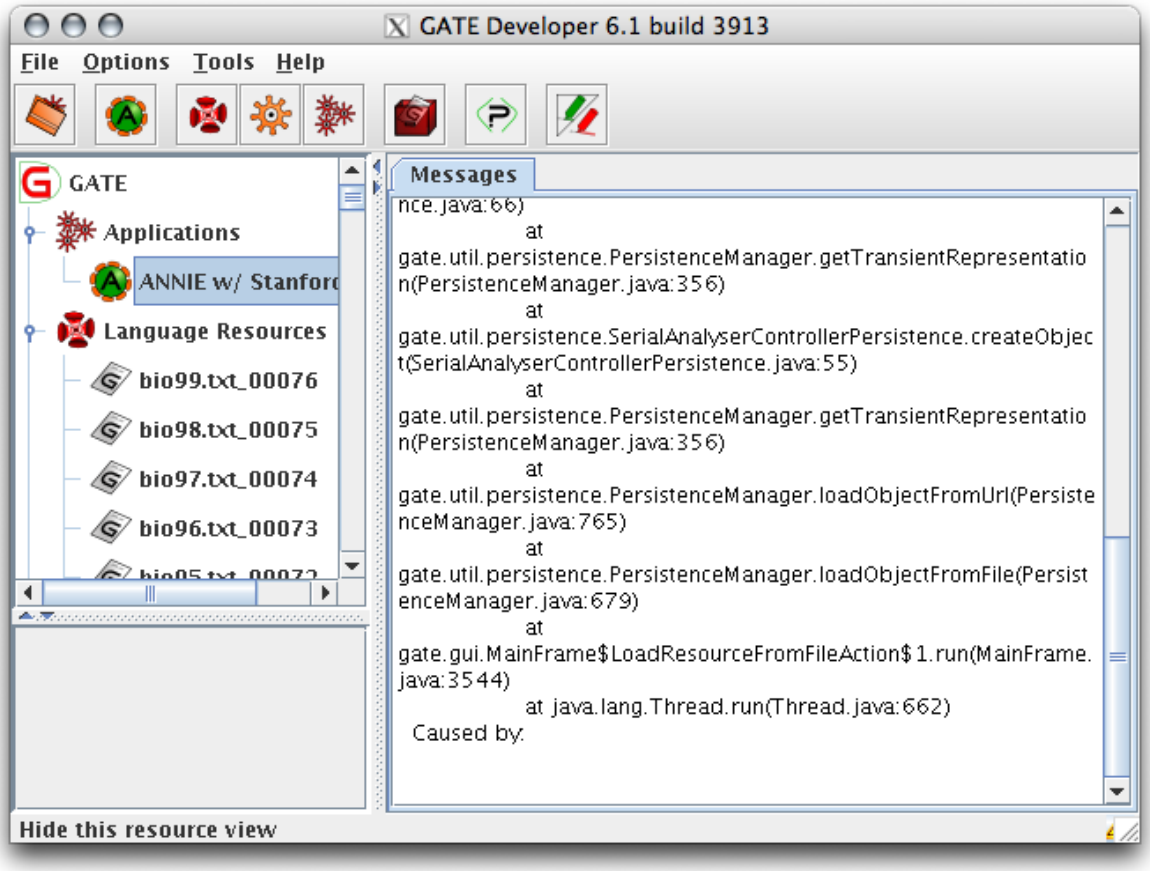

Figure 9: The application has been loaded, double click it to open

The application appears under Applications in the resources tree, as shown in Figure 9. However, the application will be named `Tractor App`, instead of `ANNIE w/ Stanford`.

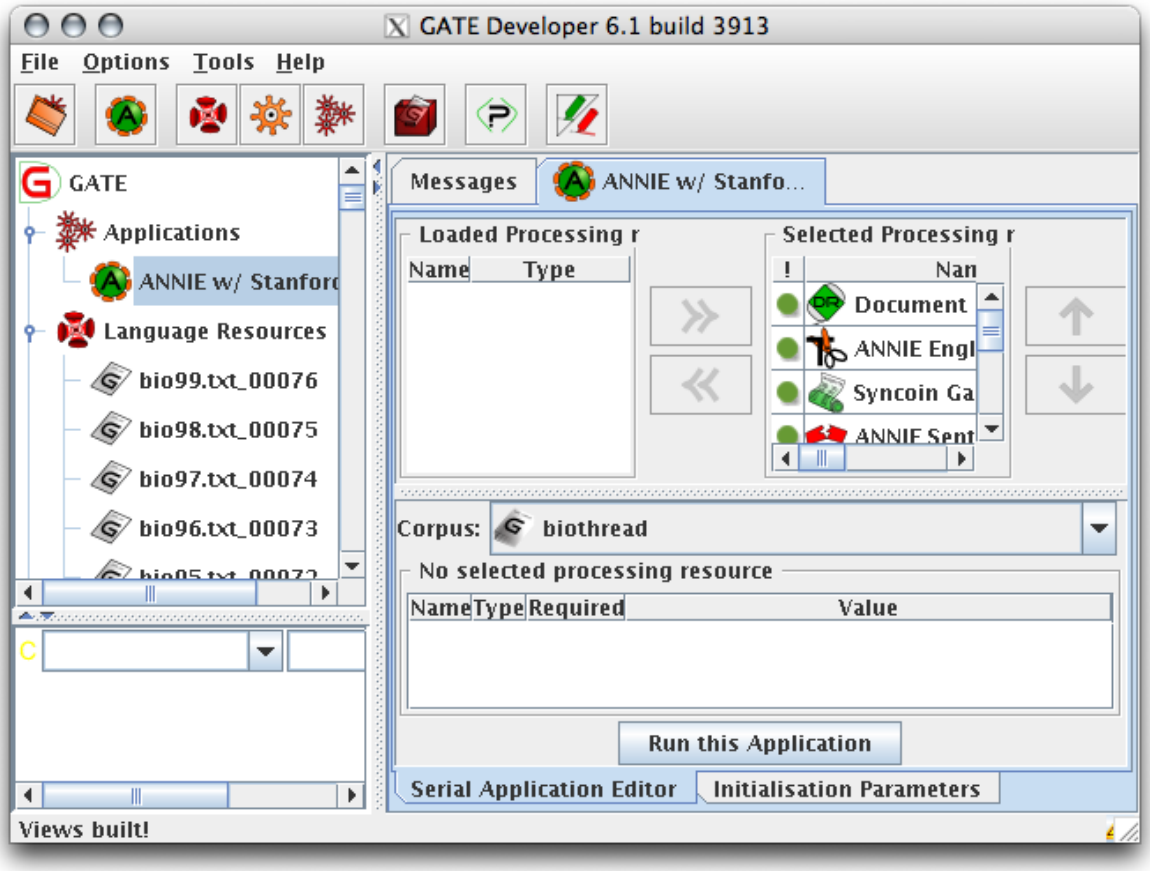

Figure 10: The application details appear in the main resource viewer.

Either double-left-click on the application in the resources tree, or click right on it and select `Show`. The application details will appear in the main resource viewer, as shown in Figure 10. Note, especially, the list of the Processing Resources (PRs) used by the application in the order shown.

Each PR has two sets of parameters: init-time parameters; and run-time parameters. To examine and modify the init-time parameters: expand the Processing Resources node of the resources tree; right-click on the resource; select `Show`. To examine and modify the run-time parameters: expand the Applications node of the resource tree; click right on the application (`Tractor App`); select `Show`; click left on the PR in the main resource viewer. All parameters of all the PRs `Tractor App` uses are stored in the file `$TractorHome/tractor.gapp`, which may be edited manually, and shown in the following tables. In those tables, there are two keywords in the paths for URL parameters: `$gatehome$` refers to the root directory of the GATE software, assumed to be `/projects/snwiz/GATE-7.0`; `$relpath$` refers to the directory from which the GATE application has been loaded, assumed to be `$TractorHome`

`Tractor App` currently uses the following Processing Resources (PRs), all of which are applied to each message in the corpus in order:

Document Reset PR Reinitializes message-processing by deleting all annotations so that the application can re-process the message.

`Tractor App` uses the following parameter settings for the Document Reset resource.

Init-time parameters			
none			
Run-time parameters			
Name	**Type**	**Required**	**Value**
annotationTypes	`ArrayList`		`null`
keepOriginalMarkupsAS	`Boolean`		`true`
setsToKeep	`ArrayList`		`null`
setsToRemove	`List`		`null`

ANNIE English Tokenizer divides the text into words, numbers, and punctuation marks. The default rules used by the tokeniser are given in the file

`$gatehome$/plugins/ANNIE/resources/tokeniser/DefaultTokeniser.rules`.

The results of the basic tokenizer are further processed by the rules in the file

`$relpath$/annie/tokeniser/postprocess.jape`

to combine tokens into common English constructs such as "U.S." and "'30s", and to make recombinations such as turning the sequence "... don ' t ..." into "... do n't ..." (*see* §6.2.3 of the GATE manual).

`Tractor App` uses the following parameter settings for the ANNIE English Tokeniser.

Init-time parameters			
encoding	`UTF-8`		
tokeniserRulesURL	`$gatehome$plugins/ANNIE/resources/tokeniser/DefaultTokeniser.rules`		
transducerGrammarURL	`$relpath$annie/tokeniser/postprocess.jape`		
Run-time parameters			
Name	**Type**	**Required**	**Value**
annotationSetName	`String`		

ANNIE Sentence Splitter segments the tokens produced by the Tokenizer into sentences. It uses the list of abbreviations stored in `$gatehome$/plugins/ANNIE/resources/sentenceSplitter/gazetteer/` to

distinguish a period at the end of an abbreviation from a period at the end of a sentence.

`Tractor App` uses the following parameter settings for the ANNIE Sentence Splitter.

Init-time parameters	
encoding	`UTF-8`
gazetteerListsURL	`$gatehome$plugins/ANNIE/resources/sentenceSplitter/gazetteer/lists.def`
transducerURL	`$gatehome$plugins/ANNIE/resources/sentenceSplitter/grammar/main.jape`

Run-time parameters			
Name	**Type**	**Required**	**Value**
inputASName	`String`		
outputASName	`String`		

Syntactic Pre-Processor uses a set of JAPE rules to provide part-of-speech tags for words that the Stanford Parser would otherwise tag incorrectly. The JAPE rules are stored in

`$relpath$/annie/pre-tagger/pre-tagger.jape`.

`Tractor App` uses the following parameter settings for the Syntactic Pre-Processor.

Init-time parameters			
Name	**Type**	**Required**	**Value**
annotationAccessors	**List**		`[]`
encoding	**String**	√	**UTF-8**
grammarURL	**URL**	√	`file:$relpath$/annie/pre-tagger/pre-tagger.jape`
operators	**List**		`[]`

Run-time parameters			
Name	**Type**	**Required**	**Value**
inputAsName	**String**		
ontology	**Ontology**		⟨**none**⟩
outputAsName	**String**		

Stanford Parser is a wrapper around version 1.6.5 of the Stanford Dependency Parser. The dependency relations that are used are given in [Marie-Catherine de Marneffe and Christopher D. Manning. 2008. Stanford Dependencies manual, http://nlp.stanford.edu/software/dependencies_manual.pdf]. It requires that the message has already been tokenized and tagged with `Sentence` annotations, as created by the Tokenizer and the ANNIE Sentence Splitter. It performs its own part-of-speech tagging using the tags shown

at http://gate.ac.uk/sale/tao/splitap7.html#x37-729000G.

`Tractor App` uses the following parameter settings for the Stanford Parser.

Init-time parameters			
mappingFile			
parserFile	`$gatehome$/plugins/Parser_Stanford/resources/englishPCFG.ser.gz`		
tippClass	`edu.stanford.nlp.parser.lexparser.EnglishTreebankParserParams`		
Run-time parameters			
Name	**Type**	**Required**	**Value**
addConstituentAnnotations	`Boolean`	√	`true`
addDependencyAnnotations	`Boolean`	√	`true`
addDependencyFeatures	`Boolean`	√	`true`
addPosTags	`Boolean`	√	`true`
annotationSetName	`String`		
debug	`Boolean`	√	`false`
reusePosTags	`Boolean`	√	`true`
useMapping	`Boolean`	√	`false`

Syncoin Gazetteer (unfortunately misnamed) is a named entity recognizer that operates on the output of the Tokenizer. It tags a token or a sequence of tokens with the type of the entity it names based on lists stored in the files in `$TractorHome/annie/gazetteer/`.

An index to these files is in `$TractorHome/annie/gazetteer/lists.def`. Each line in the index file is of the form

list file : *major type* [: *minor type*]

These files can be edited, but to take effect, the GATE Application (*e.g.,* `Tractor App`) must be reloaded.

`Tractor App` uses the following parameter settings for the Syncoin Gazetteer.

Init-time parameters			
listsURL	`$relpath$annie/gazetteer/lists.def`		
encoding	`UTF-8`		
gazetteerFeatureSeparator	`null`		
caseSensitive	`true`		
Run-time parameters			
Name	**Type**	**Required**	**Value**
annotationSetName	`String`		
longestMatchOnly	`Boolean`	√	`true`
wholeWordsOnly	`Boolean`	√	`true`

ANNIE NE Transducer, also referred to as the Semantic Tagger, operates on the output of the Syncoin Gazetteer and supplements it with a set of JAPE rules stored in `$relpath$annie/NE/main.jape` and sibling files. It also uses the part-of-speech tags produced by the Stanford Parser.

`Tractor App` uses the following parameter settings for the NE Transducer.

Init-time parameters			
none			
Run-time parameters			
Name	**Type**	**Required**	**Value**
inputASName	`String`		
outputASName	`String`		

ANNIE OrthoMatcher "adds identity relations between named entities found by the semantic tagger [also referred to as the NE Transducer], in order to perform coreference ... The matching rules are only invoked if the names being compared are both of the same type ... or if one of them is classified as 'unknown'." [GATE manual §6.8]

`Tractor App` uses the following parameter settings for the ANNIE OrthoMatcher.

Init-time parameters

Name	Type	Required	Value
caseSensitive	Boolean		false
definitionFileURL	URL	√	$gatehome$plugins/ANNIE/resources/othomatcher/listsNM.def
encoding	String	√	UTF-8
exLists	Boolean		true
highPrecisionOrgs	Boolean		false
minimumNicknameLikelihood	Double	√	0.5
organizationType	String		Organization
PersonType	String		Person
processUnknown	Boolean		true

Run-time parameters

Name	Type	Required	Value
annotationSetName	String		
annotationTypes	ArrayList		[Organization, Person, Location, Date]

ANNIE Nominal Coreferencer recognizes when several noun phrases refer to the same entity. It uses the output of the OrthoMatcher.

`Tractor App` uses the following parameter settings for the Nominal Coreferencer:

Init-time parameters

none

Run-time parameters

Name	Type	Required	Value
annotationSetName	`String`		

ANNIE Pronominal Coreferencer performs anaphora resolution using a set of JAPE rules and the output of the OrthoMatcher.

`Tractor App` uses the following parameter settings for the Pronominal Coreferencer:

Init-time parameters

none

Run-time parameters

Name	Type	Required	Value
annotationSetName	`String`		
inanimatedEntityTypes	`String`		`Organization;Location`
resolveIt	`Boolean`		`false`

GATE Morphological Analyser is a rule-based morphological analyser. It operates on the tokens produced by the Tokeniser, using the part-of-speech tags produced by the Stanford Parser. It adds the root of the token and its affix to the token annotation.

`Tractor App` uses the following parameter settings for the Morphological Analyzer.

Init-time parameters			
Name	**Type**	**Required**	**Value**
caseSensitive	`Boolean`	√	`false`
rulesFile	`URL`	√	`$relpath$annie/morph/default.rul`
Run-time parameters			
Name	**Type**	**Required**	**Value**
affixFeatureName	`String`	√	`affix`
annotationSetName	`String`		
considerfPOSTag	`Boolean`	√	`true`
failOnMissingInputAnnotations	`Boolean`		`false`
rootFeatureName	`String`	√	`root`

3.4.3 Run the GATE application

Ensure that the correct Corpus is selected ("biothread" shown in Figure 11), and click on the "Run this Application" button to process the corpus. A progress bar will appear while the application is running.

When the application is finished, you can examine the processed messages by doing the following.

1. Right click on a message in the resources tree.
2. Choose "Show" from the pop-up menu.
3. Make sure the "Text" button is selected. This will show the text of the message in the main resource viewer.
4. Select the "Annotations Sets" button. This will show a list of annotations to the right of the message text.
5. Click on any annotation button to highlight the tokens that have been tagged with that annotation.

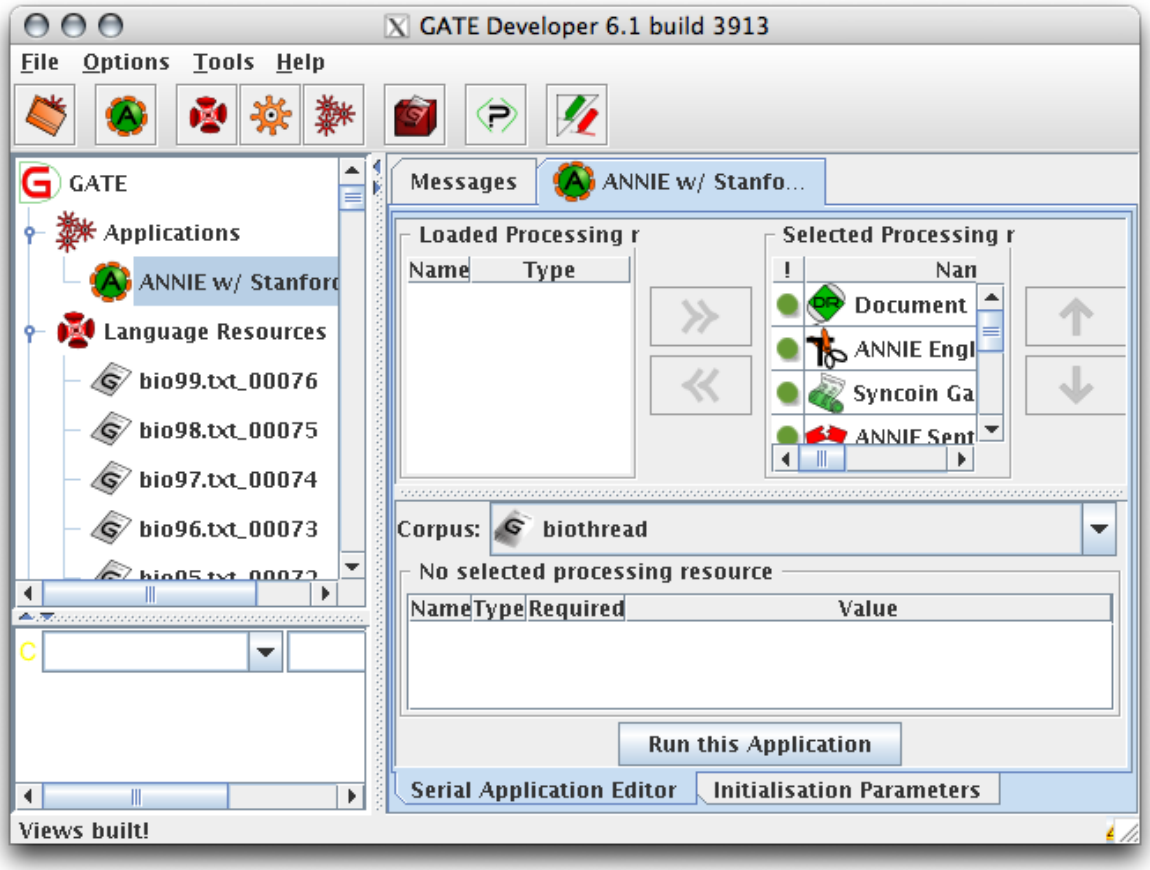

Figure 11: Click Run this Application button

3.4.4 User Coreference Editing

To make additions and corrections to the coreference "chains" identified by the Orthomatcher, Nominal Coreferencer, and Pronominal Coreferencer use the Co-reference Editor. First, you should make sure that at least one token of every eventual coreference chain has the correct Type. With a message in the main resource viewer, click on the "Annotation Sets" tab, and, one by one, click on each of the major Types on the right. If there is a word or word sequence that is not of the correct Type, and neither is any other token that is actually co-referential with it, then select the word or word sequence, and right-click. Then, in the dialog box, select the correct Type in the Type drop-down menu. For example "owner" or "resident" might not be recognized as of Type Person, so you should do this with one occurrence of that word.

When the Types are correct, you are ready to make sure that the coreference chains are correct. Click on the "Co-reference Editor" tab. Expand the "Default" Co-reference Data tree to expose any coreference chains that already exist. Click on a chain, and click one of the tokens in the message to remove it from that chain. Choose a Type from the drop-down menu above the Co-reference Data, click "Show", and click a token to add the token to a chain or to establish a new chain containing that token.

Advice:

- Look at the Types in order from the most specific to the most general, ending with Lookup, and then finally Token.
- Remove any mistaken coreferences from their chains.
- Add any unassigned pronoun to its correct coreference chain. Don't skip pronouns like "who", "which", and "that".
- In a noun-noun string, such as "known bomber Sayed Azhour", leave the modifying noun a modifier; don't make it co-referential with the head noun.
- When two multi-word entity names end at the same character, such as "Second District Courthouse" and "Courthouse", it doesn't matter which is included in a coreference chain. The two will be declared co-referential by the syntax-semantics mapping rules.
- Nouns in apposition, such as "a youth, Kalid Sattar" needn't be made co-referential. That will be done by the syntax-semantics mapping rules.
- Entities linked by a copula, such as "the rented vehicle is a white van" needn't be made co-referential. That will be done by the syntax-semantics mapping rules.

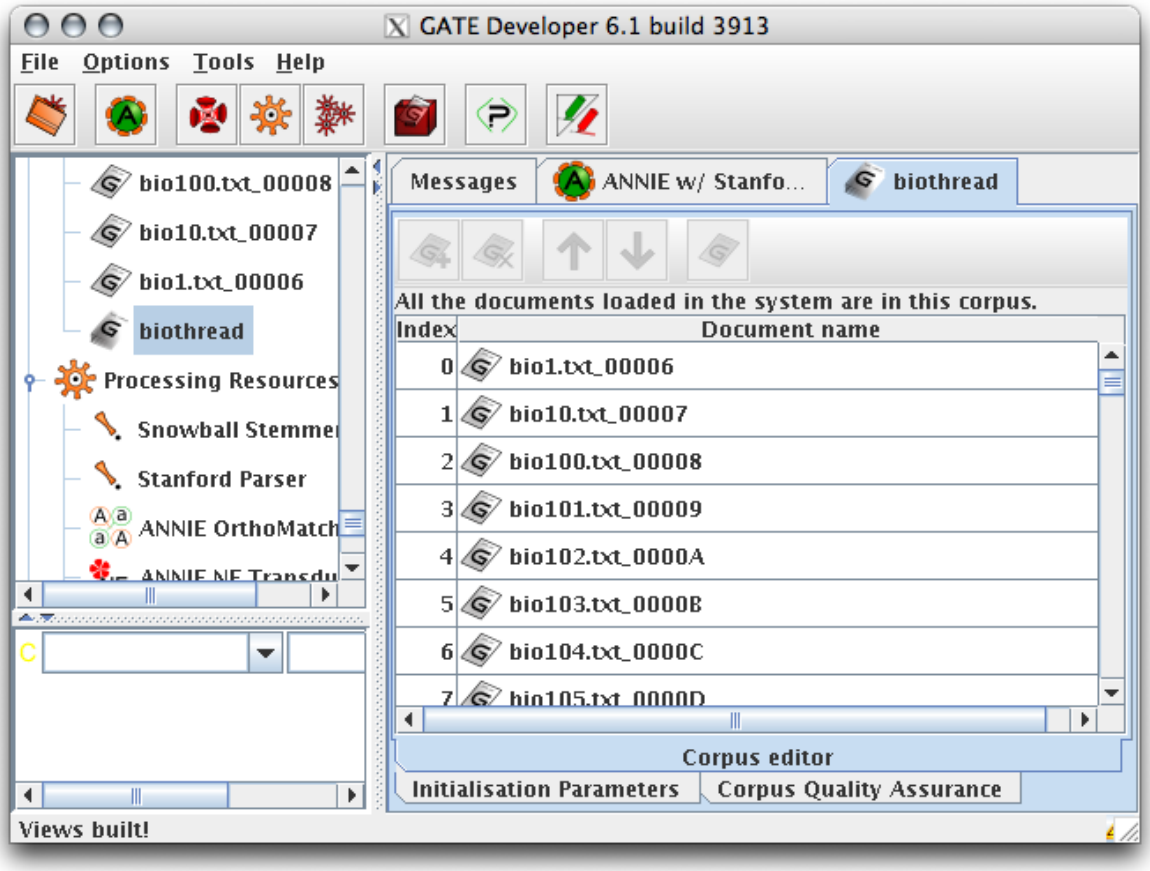

Figure 12: Select the corpus from Language Resources

- Be sure not to leave any coreference chain with only one token in it. If there is one, delete the one token from the chain.

3.4.5 Save to XML

Select the corpus from Language Resources in the resources tree. You may have to scroll down to see it. In Figure 12 we have selected the "biothread" corpus, and double-left-clicked on it.

Right click on the corpus and choose "Save as XML..." as shown in Figure 13. Save the GATE XML files to the directory `Mmmdd/XMLParses`.

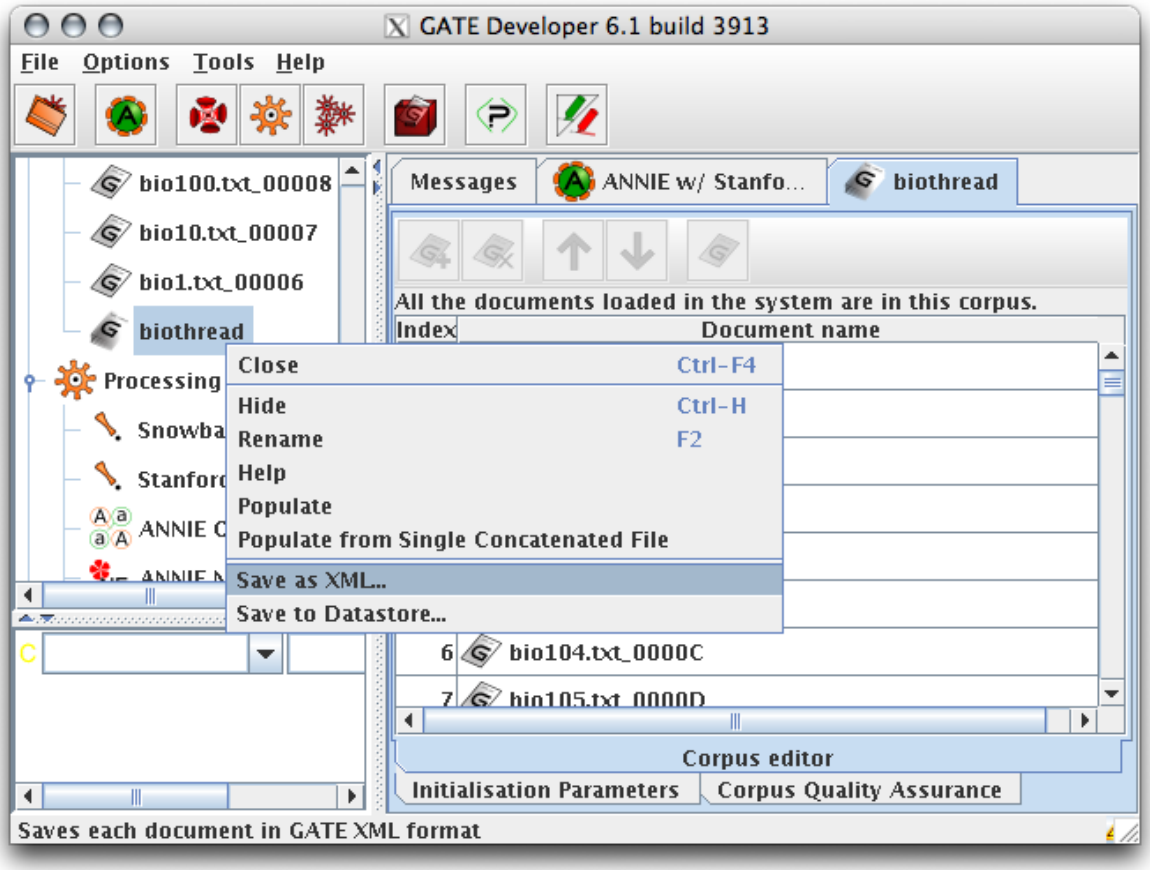

Figure 13: Choose Save as XML...

3.4.6 Summary: Dependencies among GATE Components

Tractor uses 12 GATE components. The order dependencies among them are:

Document Reset PR must precede all other components if the application is to be rerun.

ANNIE English Tokenizer must precede the Sentence Splitter and the Gazetteer.

ANNIE Sentence Splitter must follow the Tokenizer and precede the Pre-Processor.

Syntactic Pre-Processor must follow the Sentence Splitter and precede the Parser.

Stanford Parser must follow the Syntactic Pre-Processor and, due to its production of part-of-speech tags, must precede the NE Transducer and the Morphological Analyzer.

Syncoin Gazetteer must follow the Tokenizer and precede the NE Transducer.

ANNIE NE Transducer must follow the Gazetteer and the Parser, and must precede the Orthomatcher.

ANNIE OrthoMatcher must follow the NE Transducer, and must precede the Nominal Coreferencer and the Pronominal Coreferencer.

ANNIE Nominal Coreferencer must follow the OrthoMatcher and precede the Coreference Editor.

ANNIE Pronominal Coreferencer must follow the OrthoMatcher and precede the Coreference Editor.

GATE Morphological Analyser must follow the Parser.

Coreference Editor must follow the OrthoMatcher, and the Nominal and Pronominal Coreferencers.

3.5 Converting to Propositional Graphs

At this point, a directory with the name `Mmmdd/XMLParses/` contains a set of files of the output of the GATE Tractor Application, represented in XML. Each of those files contains a set of annotations of one message, including the annotations that represent the dependency parse of the message.

The purpose of this step is to translate these files into files of SNePS 3 syntactic propositional graphs that represent the dependency parses of the messages, with additional information derived from the annotations. The program, `xml2sneps3`, performs several operations: annotation merging, correction of minor errors in syntactic categories, canonicalization of dates and times, and processing the structured portion of semi-structured messages.

3.5.1 Annotation Merging

The GATE XML files contain annotations for each annotation type in in the message as determined by GATE. Annotation types include entities such as Person, Location, and Organization, the dependency relations, the results of lookups from the "gazetteer", and the tokens themselves. Annotations each have a unique ID, start and end positions, and features specific to the annotation type. Multiple annotations of different types can occupy the same region of text.

In order to associate all of the annotation data about a span of text, the Propositionalizer merges all entity, lookup, and token annotations which have the same start and end position. The result of this is a set of annotations each with unique start and end positions, and each with a unique identifier.

3.5.2 Correction of Syntactic Categories

Annotations for syntactic categories are assigned by the Stanford Parser in GATE for each token (that is, each textual region identified by the tokenizer, including punctuation). This leaves entities which span more than one token without a syntactic category. In many cases this is desired, but for entities consisting of tokens which are all proper nouns, it's useful to identify the entity as a proper noun.

Punctuation is also given a syntactic category by the parser. In many cases the category is equal to the string representation of the punctuation. In rare cases, the punctuation may be given another part-of-speech, such as NNP (proper noun). This is likely erroneous, and since these categories are not used, we remove all syntactic categories from punctuation.

3.5.3 Canonicalization

For the purposes of easier comparison in later phases, dates, times, and durations are converted to ISO8601 format. For dates this is YYYYMMDD. That is, the four digit year, followed by the two digit month, and two digit day. For times the canonicalized form is HHMM. That is the two digit hours (in 24 hour format) followed by the two digit minutes. The duration form is PTMMSS. That is, the string "PT" followed by the two digit minutes, and two digit seconds. ISO8601 defines longer, more detailed, versions of each of these formats, but these are sufficient for our current needs.

3.5.4 Automatic "Manual" Co-referencing

Optionally, the Propositionalizer can replace the co-references found by GATE with those from a provided file. The provided file should be tab delimited, with the first column containing a numeric unique identifier,

the second column containing the message number, the fourth column containing the beginning text offset of the co-referenced word or phrase, and finally the fifth column contains the length, in characters, of the co-referenced word. The third column is not used.

It is required that the message numbers used in the co-reference file are contained in the names of the processed file. For example, if the provided file has entries for message number 54, it will match the input file `syn054.txt.xml`. In some configurations of Tractor it is understood that original filenames may not be preserved. In this case, place the co-reference text file in `$TractorHome/Syntax/xml2sneps3/TruthFiles`, and create a folder with the same name (but missing the `.txt` extension). In that new folder, place the messages with filenames matching the numbers in the co-reference file. The system will then be able to match incoming messages with existing ones to recover the original filenames.

3.5.5 Structured Headers

Tractor is capable of processing messages which are semi-structured. Semi-structured headers are defined by creating a header definition file and placing it in the `"$TractorHome$/Syntax/xml2sneps3/HeaderDefinitions"` folder. A header definition file has the following format:

```
clojureCode         =  //Clojure function definitions
clojureCodeRegion   =  '{' , newline , clojureCode , newline '}' ;
matchConditionFn    =  //Anonymous Clojure function of one argument
actionFn            =  //Anonymous Clojure function of one argument
ruleBody            =  (matchConditionFn | (ruleName ('*' | '+' | '?')?)+) ,
                       '-->' , actionFn?  ;
ruleDefinition      =  '#Rule' , ruleName , ':'  , ruleBody ;
headRuleDefinition  =  '#HeadRule' , '(' , priorityNumber , ')' , ruleName , ':'  ,
                       ruleBody ;
header              =  clojureCodeRegion?  , (headRuleDefinition | ruleDefinition)+ ;
```

There are two rule types - regular rules and head rules. Head rules are "starting points" for the matching, since it's not meaningful to just match any rule against a set of annotations. Head rules have priorities which indicate which rules should be tried first. Rules with higher priority numbers are attempted before those with lower priority numbers. Rules with the same priority may be attempted in any order.

Either type of rule can be either a base rule or a recursive rule. A base rule is is matched by first

identifying the annotations which start at the current position of interest in the message, then by filtering the relevant annotations by the `matchConditionFn` defined for that rule. A matchConditionFn should take a single annotation as an argument, and return true if the annotation matches some condition, and false or nil otherwise. If multiple annotations match the rule, the algorithm is greedy - taking the longest matching annotation. A recursive rule is matched by successively matching each rule named in its left hand side.

The result of a successful match is a tree each of whose interior nodes are rule names, and whose leaf nodes are annotations which match the rule sequence starting with the head rule (at the root of the tree) and ending at the rule whose name is the value of the node directly above the annotation of interest. This is analogous to a compiler's Concrete Syntax Tree (CST).

At match time the `actionFn`, if given, is executed. An `actionFn` is given as an argument the tree derived at and below that node. On regular rules this is useful for debug purposes to output the current state of the match. For head rules, the result of the `actionFn` is written to the SNePS 3 output file, so the `actionFn` will likely be a call to some Clojure function defined earlier in the definition file to translate the CST to SNePS 3.

Currently only one structured header of any complexity is included in Tractor. It is for messages representing communication intercepts, defined below.

```
messageHeader          =  messageNumber , '.'  , date ('-')?  ,
                          (('ET' | 'RT') , ':')?  , time?  , ('-' | '--') ;
commInfo               =  '|C:|' , commDuration?  , '|' , commInterceptor?  , '|' ,
                          commMedium , '|' ;
participantInfo        =  participantID , '|' , participantPhoneNum?  , '|' ,
                          participantLocationName?  , '|' , participantMGRS? , '|' ;
communicatorInfo       =  '|P1| | communicator |' , participantInfo ;
addresseeInfo          =  '|P2| | addressee |' , participantInfo ;
analystText            =  quotedFreeText ;
conversationComponent  =  ('|P1|' | '|P2|') , quotedFreeText ;
conversation           =  conversationComponent+ ;
commMessage            =  messageHeader , commInfo , communicatorInfo , addresseeInfo ,
                          conversation?  , analystText?  ;
```

Other included structured headers are for the standard message formats used in SynCOIN, the Bomber Buster Scene, the STEF messages, and the Ali Baba data set.

3.5.6 Running The Propositionalizer

Make sure that you have run `"$TractorHome/makedirectories"` and `/projects/snwiz/bin/gate&`, saved the files of annotations in `Mmmdd/XMLParses`, and made `MMMdd` your current working directory. Then execute the Unix command,

```
$ "$TractorHome/Syntax/xml2sneps3/xml2sneps3"
```

The directory `Mmmdd/SNEPSParses` will now contain one file for each message. Each file will contain a series of SNePS 3 assertions which represent the dependency parse of, and other information about, one message.

There are several command line arguments you may wish to use when running the Propositionalizer:

`--compare-equivs-from-tab <filename>`	Compares co-references from the given tab file but does not replace them. Warnings and errors will be presented for non-matching co-references.
`--replace-equivs-from-tab <filename>`	Replaces co-references from the given tab file. Errors will be displayed if there are text spans in the tab file which do not map to annotations from GATE.
`--auto-replace-equivs`	Replaces co-references using the files in `$TractorHome/Syntax/xml2sneps3/TruthFiles`.

3.6 Enhancing and Performing Semantic Analysis

3.6.1 Creating Semantic Propositional Graphs

The purpose of the Syntax-Semantics Mapper is to convert a directory of SNePS 3 files of syntactic propositional graphs into SNePS 3 and SNEPSXML files of semantic propositional graphs. For each file in the `Mmmdd/SNEPSParses` directory, the syntax/semantics mapper performs the following steps.

1. Load the file into a SNePS 3 KB.
2. Collect all the nouns and verbs that are in the KB, and send them to the CBIR process[11], which will return ontological and geographical assertions about them. Statistics about what CBIR accomplished are written to a file named `Mmmdd/cbirStats.txt`.

[11]The CBIR process requires an Allegro Common Lisp image with a large heap size (about 1500MB). We've installed our Lisp image with this large heap as `/util/acl/composer1500`.

3. Add these assertions to the SNePS 3 KB. The KB is now called an enhanced syntactic propositional graph, and is written to a file in the directory `Mmmdd/EnhancedSyntacticGraphs`.

4. Execute a set of syntax-semantics mapping rules that are in the file `$TractorHome/Propositionalizer/sneps3mapper`. Counts of the number of times each syntax-semantics mapping rule fired on each message graph are written into the file named `Mmmdd/ruleUsage.csv`.

5. Write all the asserted propositions from the KB into a `Mmmdd/SNEPSPropGraphs` file in SNePS 3 format.

6. Write all the asserted propositions from the KB into a `Mmmdd/SNEPSXML` file in XML format.

7. Write a formalized English description of each entity and event in the graph into a file in the directory `Mmmdd/Descriptions`.

8. Write comma-separated values files of descriptions of all the entities and events represented in the graph into a file in the directory `Mmmdd/Answers`

9. Add a line recording the number of syntactic assertions, the number of semantic assertions, and the percent of assertions that are semantic into a file named `Mmmdd/synsemNumbers.csv`.

To run the Syntax-Semantics Mapper:

1. In a separate shell, change into the `$TractorHome/CBIR/ClojureCBIR` directory, and run

   ```
   lein run
   ```

2. In the original shell, make sure that the working directory is `Mmmdd`, and that the two output directories `Mmmdd/SNEPSPropGraphs`, and `Mmmdd/SNEPSXML` exist, and then run the Unix command

   ```
   $ composer -L "$TractorHome/Propositionalizer/propositionalizer"
   ```

The directories `Mmmdd/EnhancedSyntacticGraphs`, `Mmmdd/SNEPSPropGraphs`, `Mmmdd/SNEPSXML`, `Mmmdd/Descriptions`, and `Mmmdd/Answers`, and the files `cbirStats.txt`, `Mmmdd/ruleUsage.csv`, and `Mmmdd/synsemNumbers.csv` will now be populated as described above.

However, if the environment variable `CloudStorageDirectory` exists and is a path to a directory, presumably in the cloud, the directories `SNEPSXML`, `Answers`, and `Descriptions`, instead of being subdirectories of `Mmmdd`, will be subdirectories of the path which is the value of `CloudStorageDirectory`, and within the directories `CloudStorageDirectory/SNEPSXML`, `CloudStorageDirectory/Answers`, and

`CloudStorageDirectory/Descriptions` there will be directories named `yyyymmdd-hhmmss`. Those latter directories will contain one file for each message.

3.6.2 CBIR

One step of the Syntax-Semantics Mapper is the use of CBIR, which adds ontological/taxonomic information about the nouns and verbs in the message, along with additional information about locations. When CBIR is run in its separate shell by default it does the following

1. reads the WordNet and VerbNet dictionaries into memory.
2. opens a connection to a SQL server containing data from the NGA GEOnet Names Server running on `muri3.cse.buffalo.edu`;
3. opens a server port for communicating with the Syntax-Semantics Mapper;
4. receives a sequence of nouns and verbs from the Syntax-Semantics Mapper;
5. for each noun, n, CBIR:
   (a) looks up the synsets for the noun in WordNet;
   (b) if the word was found, for each synset s, sends the assertion `(Type` n s`)` to the Syntax-Semantics Mapper, where s stands for a unique identifier for the synset;
   (c) crawls recursively up the hypernym relation for each synset, sending Type assertions back to the Syntax-Semantics Mapper.
6. for each verb, v, CBIR:
   (a) looks up the verb classes for the verb in VerbNet;
   (b) if the verb was found, for each verb class c, sends the assertion `(Type` n c`)` to the Syntax-Semantics Mapper, where c stands for a unique identifier for the verb class;
   (c) crawls recursively up the parent relation for each verb class, sending Type assertions back to the Syntax-Semantics Mapper;
   (d) when the top of the verb class hierarchy has been reached, mappings to WordNet are used from the member verbs of that class to then crawl recursively up the hypernym relation for each mapped synset, sending Type assertions back to the Syntax-Semantics Mapper.

7. for each proper noun, CBIR additionally:
   (a) uses the SQL connection to look up proper nouns in the NGA GEOnet Names Server database;
   (b) sends the assertions found in the NGA GEOnet Names Server to the Syntax-Semantics Mapper;
8. for each MGRS coordinate, CBIR additionally:
   (a) uses the SQL connection to look up the MGRS coordinate in the NGA GEOnet Names Server database;
   (b) if GEOnet returns no results, converts the MGRS coordinate to Latitude and Longitude using NASA's World Wind software;
   (c) calculates the radius of the MGRS coordinate based on the length of the coordinate string;
   (d) sends the assertions from NGA GEOnet or NASA World Wind, along with the MGRS radius to the Syntax-Semantics Mapper;
9. When all the words from a client have been processed, statistics about how many of each word type were looked up in WordNet or VerbNet are sent to the client, displayed to the screen there, and written to the file `Mmmdd/cbirStats.txt`.

Some of the above actions can be controlled by the configuration file located at `$TractorHome/CBIR/ClojureCBIR/src/CBIR/configuration.clj`. In this configuration file you can declare:

- Whether or not pedigree assertions should be produced for each assertion produced by CBIR;
- Whether or not to use the GEOnet server, along with username, password, server address, and table information for the SQL server where the GEOnet data is stored;
- Where the WordNet dictionary files are located;
- Where the VerbNet dictionary files are located.

3.6.3 Using CBIR Interactively

CBIR may be used interactively to either display the hierarchies of verb classes and synsets, or to produce the SNePS 3 assertions which will be transmitted across the network.
To display the hierarchies, do the following:

1. `$ cd ‘‘$TractorHome/CBIR/ClojureCBIR’’`
2. `$ lein repl`
3. one of:
   (a) `(print-noun-tree` *`string`*`)`
   (b) `(print-verb-tree` *`string`*`)`

   where *`string`* is a string containing the item to be looked up

To interactively perform CBIR on a word, and see the SNePS 3 assertions do:

1. `$ cd ‘‘$TractorHome/CBIR/ClojureCBIR’’`
2. `$ lein repl`
3. one of:
   (a) `(handle-common-noun` *`string token`*`)`
   (b) `(handle-proper-noun` *`string token`*`)`
   (c) `(handle-person-name` *`string token`*`)`
   (d) `(handle-verb` *`string token`*`)`
   (e) `(handle-mgrs` *`string token`*`)`

   where *`string`* is a string containing the item to be looked up, and *`token`* is a string containing the token identifier, such as `n34`.

This will print the WordNet or VerbNet translation of the *`word`*, and return a set of assertions found by CBIR.

3.7 Descriptions

Each file in the `Mmmdd/Descriptions` directory contains a sequence of lines for each entity represented in the corresponding semantic propositional graph, where an entity is a set of coreferring terms that are an instance of some class and have an associated text start and end position in the message. Note that in this context, events are also considered entities. For each entity, the lines printed are:

- Its “best name”

- Its "best class(es)"
- A set of lines describing
  - Its attributes
  - The relations it participates in
- One line for each coreferring term, showing
  - The actual term
  - The token-start and token-end positions
  - The message text in that range

  ordered by token-start position.

3.8 Answers

Each file in the Answers directory is a comma-separated values file, which can be loaded into a spreadsheet, containing lists of the entities and events, and their attributes and relations for use in grading the performance of Tractor.[12]

The file is in two sections:

1. a section of entries for individual entities and events;
2. a section of entries for groups.

The first line of each section contains header fields. The subsequent lines contain entries for the individual entities, events, groups, their attributes, relations, and text ranges from the corresponding message.

The columns in the section for individual entities and events are:

BestName: The "best name" of the entity or event, or a triple giving an attribute of the individual or a relation the individual participates in;

Type: The category of the entity or event, chosen from the list: `Person, Organization, Location, Event, Thing`;

LGType: One or a set of least general categories the entity or event is an instance of;

[12] *See* Stuart C. Shapiro, A Gradng Rubric for Soft Information Understanding, University at Buffalo, 2013.

Mentions: A list of text strings where the entity or event, or co-referring entities or events are mentioned in the text. Each of these consists of the range of character positions followed by the actual text string.

The columns in the section for groups are:

GroupBestName: The "best name" of the group, or a triple giving an attribute of the group or a relation the group participates in;

InstancesOfType: The categories in which the members of the group are;

FillersOfRole: The roles which the members of the group fill;

Mentions: A list of text strings where the group, or co-referring groups are mentioned in the text. Each of these consists of the range of character positions followed by the actual text string.

3.9 Summary of Processing Steps

A summary of the steps for running Tractor is:

1. `cd` into the directory to be `$TractorHome`.
2. `$ source anchorTractor`
3. `cd` into the directory containing the message set.
4. `$ "$TractorHome/makedirectories"`
5. `$ cd` *`Mmmdd`*
6. `$ cp ../Messages/* Messages/`
7. `$ /projects/snwiz/bin/gate &`
   (a) Restore the `tractor.gapp` application.
   (b) Set up and populate the corpus.
   (c) "Run this Application".
   (d) Use the Co-reference Editor.
   (e) "Save as XML" to the directory `XMLParses`.
8. `$ "$TractorHome/Syntax/xml2sneps3/xml2sneps3"`

9. In a separate shell, switch into the `$TractorHome/CBIR/ClojureCBIR` directory and run `lein run`.

10. `$ composer -L "$TractorHome/Propositionalizer/propositionalizer"`

11. (Optional) If CBIR was configured to allow multiple client connections, type `e` followed by enter in the CBIR terminal to exit CBIR. (Sometimes this will need to be done several times.)

3.10 Automated Batch Processing

Some of the above steps (7, 8, and 11) have been scripted so Tractor can be run with a single command. To use the automated system, follow the following steps:

1. Set the `GATE_HOME` environment variable to `/path/to/Gate-7.0`.

2. `cd` into the directory to be `$TractorHome`.

3. `$ source anchorTractor`

4. `cd` into the directory containing the message set.

5. `$ "$TractorHome/makedirectories"`

6. `$ cd` *Mmmdd*

7. `$ cp ../Messages/* Messages/`

8. In a separate shell, switch into the `$TractorHome/CBIR/ClojureCBIR` directory and run `lein run`.

9. `$ "$TractorHome/runTractor"`

4 Tools

4.1 Introduction

In this section are described a set of tools that are useful for examining or manipulating the information in the files created by Tractor, but are not part of the main-line Tractor processing stream.

4.2 Examining Propositional Graphs

4.2.1 Examining the SNePS 3 Propositional Graphs

To examine any of the propositional graphs in any of the directories, `Mmmdd/SNEPSParses`, `Mmmdd/EnhancedSyntacticGraphs`, or `Mmmdd/SNEPSPropGraphs` in SNePS 3, make sure that `TractorHome` is defined correctly, and run the SNePS 3 GUI by executing the Unix command

```
$ composer -L /projects/snwiz/Sneps3/GUI
```

select `File --> Load --> Load to KB`, and choose the file you want to display in the GUI.

When you Quit the GUI, there will still be a running Lisp REPL (Read-Evaluate-Print Loop). You will need to exit Lisp manually, for example by entering `:ex` at the Lisp prompt.

4.3 Operating on SNePS 3 Propositional Graphs

4.3.1 Getting Started

Either: use the Lisp REPL (Read-Evaluate-Print Loop) in the SNePS 3 GUI; or use the Lisp REPL that is started when you start the GUI as directed in §4.2.1; or start an independent Lisp REPL, load SNePS 3 by evaluating `(load "/projects/snwiz/Sneps3/sneps3")`, and put the Lisp Reader into the `:snuser` package by evaluating `(in-package :snuser)` . Then set the generalized place `(sys:getenv "TractorHome")` to `TractorHome`, or make sure that it is already set correctly.

4.3.2 Starting the GUI

If the REPL is running, but the GUI is not, you can start the GUI by first making sure that the Lisp reader is in the `:snuser` package, and then evaluating `(startGUI)`.

4.3.3 Loading a Graph

Any graph in any of the directories `Mmmdd/SNEPSParses`, `Mmmdd/EnhancedSyntacticGraphs`, or `Mmmdd/SNEPSPropGraphs` may be loaded into the Lisp image in one of two ways. The preferred way is to load it into the SNePS 3 GUI as instructed in §4.2.1. Alternatively, you can load it directly by use of the `load` function.

4.3.4 Running the Mapper

If a graph in one of the directories `Mmmdd/SNEPSParses` or `Mmmdd/EnhancedSyntacticGraphs` is loaded into the Lisp image, you can run the syntax-semantics mapper on it by first making sure that the program

`$Tractor/Propositionalizer/sneps3mapper` is loaded, and then evaluating `(synsemMap)`.

The operation of the mapping rules can be followed more closely by setting the variable `*pauseRules*` to a non-null value. If it is set to a list of rule names, then the mapper will pause and open a `cerror` REPL

- just before each named rule is tested,
- and just after each named rule fires.

If `*pauseRules*` is set to `t`, it will be as if the name of every rule were given. This behavior is particularly useful if the SNePS 3 GUI is running, making it easy to see the effect of each named rule.

4.3.5 Describing Instances

If a semantic propositional graph is loaded in a Lisp image, you can have descriptions of the entities and events in the graph printed by first loading `$TractorHome/describe` and then evaluating `(describeAllInstances)`. The result will be a series of lines like those in the `Descriptions` files.