GateSchool 2011

LinkedData and Advanced Semantic Annotation

Vassil Momtchev and Konstantin Pentchev

Life Sciences at Ontotext AD
Sirma Group
Sofia, Bulgaria

20.05.2011

Contents

1 Working with RDF **2**
1.1 Convert between different RDF formats . . . 3
1.2 Compare RDF files . . . 3

2 Querying LinkedData **7**
2.1 Get semantic types from LinkedLifeData . . . 7

3 Custom Vocabulary Gazetteer **9**
3.1 Gate Application with LKB Gazetteer . . . 9
3.2 GateDocuments and Corpora . . . 12
3.3 JAPE transformations . . . 14
3.4 Saving the pipeline . . . 14

4 Semantic Annotations and LinkedData **15**
4.1 Annotate data from SPARQL endpoint . . . 15
4.2 Convert annotated documents to RDF . . . 16

5 Mimir **18**

Introduction

In this hands-on session we will be using two applications: *GATE Developer* and *Talend Open Studio*. You will learn how to create Talend workflows for **reading** and **writing RDF data**, **querying LinkedData**, **create annotation pipelines**, **incorporate the GATE pipelines in your workflows** and **index** and **search annotated data** with **MIMIR**.

The GATE family of products is one of the most widely used tools for performing text analysis by both research institutions and commercial users. It is freely available for download from `http://gate.ac.uk/download/` with versions for Linux, Windows and Mac. A detailed manual on the capabilities of the different products is available in the distribution.

Talend will be distributed to you via external HDs or similar. Copy the zip-file to your local hard-drive and extract its contents. Start Talend by running the appropriate executable according to your system (e.g. 32-bit/64-bit; windows/linux). After coming through the registration page(skip), a dialog prompting for username and workspace project will appear. Create a new user by pressing on the ellipsis button next to E-Mail. Enter a valid email and change

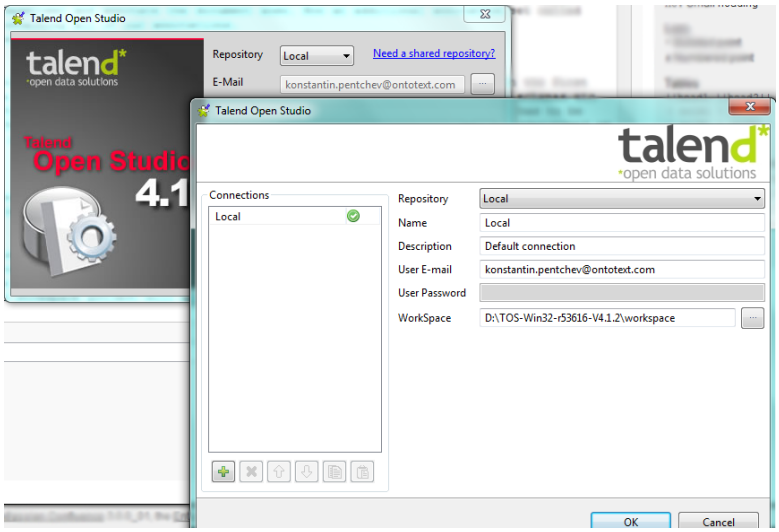

Figure 1

the path to your workspace accordingly. The default workspace is located in the Talend home folder and contains the General project, which comes with some example jobs. If it doesn't appear in the Open field automatically after you create the new user, import it manually. Then press Open load the Talend IDE. At this point it is possible to experience some errors, which prevent the IDE from loading. Restarting the application should solve the issues.

1 Working with RDF

RDF is a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed. RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications. This linking structure forms a directed, labeled graph, where the edges represent the named link between two resources, represented by the graph nodes. This graph view

is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations (for more theory on RDF see the presentation).

1.1 Convert between different RDF formats

In the following task you will learn the different RDF serialization formats, how to read and write them. The task will be done entirely in *Talend*.

Once you have opened the IDE, you will see three panels - the **job tree** on the left, the **general menu** in the bottom and the **component's palette** on the right. The latter will be initially empty. Right-click on the *Job Designs* item in the job tree and select Create job to start a new workflow. Specify a name for the job and press Finish. The main panel in the middle of the screen will now display the design area for the new workflow and the component's palette on the right will initialize.

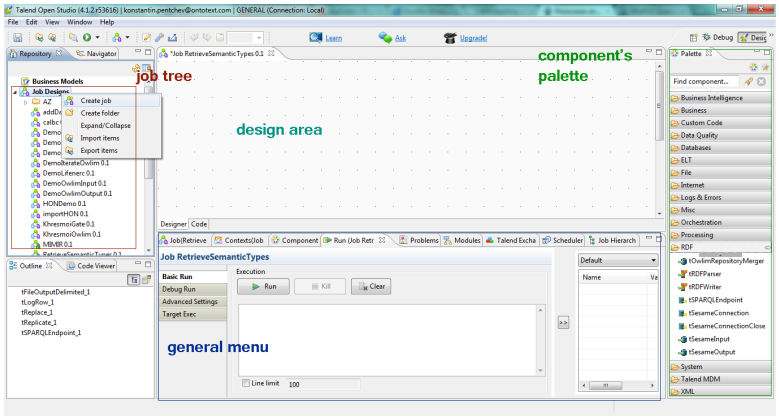

Figure 2

The components for working with semantic data are located under the *RDF tab*. To read from an RDF file use the *tRDFParser*. Drag-and-drop the component to the design area, then double click to view the component's properties. It has 5 parameters: *Schema* allows you to specify the variables to be passed to the next components; *Verify data* determines whether to check the data consistency or not; *Autodetect format* specifies whether the component should try to guess the format or rely on user input; *RDF format* allows the user to select the file format and *RDF file* points to the location of the file. The Schema defines a simple data model as column names. For the RDF component it is the static columns "s", "p", "o" and "g" to the Schema. For most other components you will have to define it yourselves.

Set the RDF Format to "Turtle" and select the file `COPD.ttl` (downloadable with the other materials from).

The next step is to add the *tRDFWriter* component to the workflow, as this is the unit, which will serialize the data. Connect the two components by right-clicking on the tRDFParser component, select *Row->Main*, then point to the tRDFParser.

The last thing you need to do is select a different RDF format. Try out "RDF-XML" and "NTriples". Save your job by pressing CTR+S and execute it from *General menu -> Run -> Run*.

1.2 Compare RDF files

In the following we will identify differences in the information about the drug *Levitra* present in two datasources - *DailyMed* and *DrugBank*. Therefore, the available data has been extracted

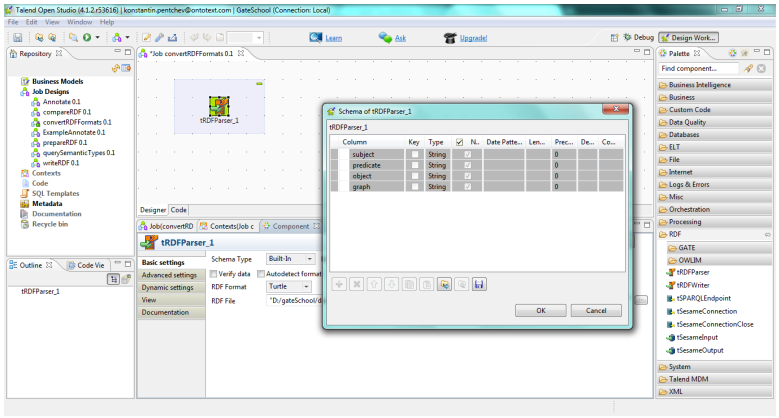

Figure 3

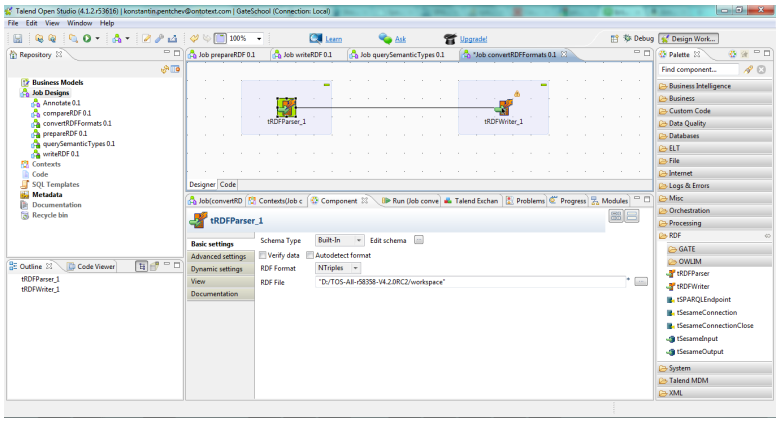

Figure 4

and stored in two RDF files: `dailymedLevitra.n3` and `drugbankLevitra.n3`. You will learn how you can compare the contents of RDF files(and other arbitrary datasources) using Talend.

First add two tRDFParsers to the workflow and configure them to read the two files. Then from the *Processing* family of components add a *tMap* to the workflow. This component is the main and most powerful tool for transforming data and matching schemas. Connect first the dailymed parser and then the drugbank parser as inputs to the tMap.

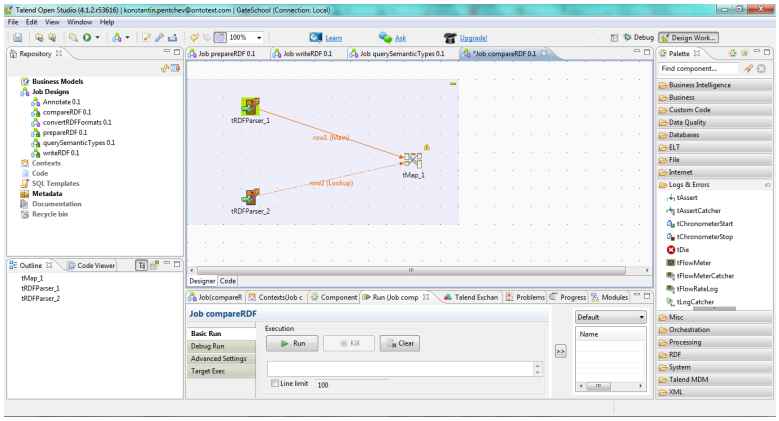

Figure 5

The next step is to add a *tLogRow* component from the *Logs and Errors* tab. This is the unit, which will provide us with output: it will print the statements to the console. Connect the tMap to the tLog, whereby you'll be prompted to enter a name for the link. Do not forget

to add a schema as well and copy it to the link.

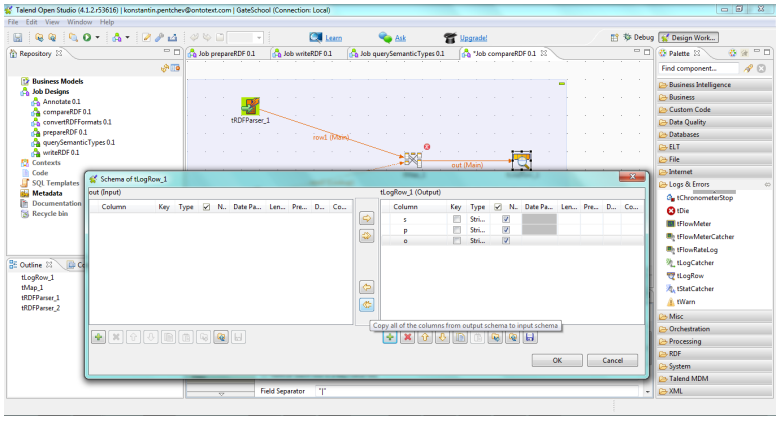

Figure 6

Finally, we need to configure how the data is matched. Double-click on the tMap to open its menu. To the left you will see your input links, grouped with their corresponding columns from the Schemes and to the right the output link. Note that in *row2* there is an additional column called *Expr. key*, which is not present in *row1*. That is so, because the data coming from row2 will be used as the lookup, against which the data from row1 will be matched. How this is done is specified in the *Expr. key* column. For this particular task simply drag-and-drop the "p" and "o" columns of row1 to the corresponding Expr. keys. Then press on the red magnifier next to "row2". This will open the settings for the matching. Set the *Join Model* to "Inner Join".

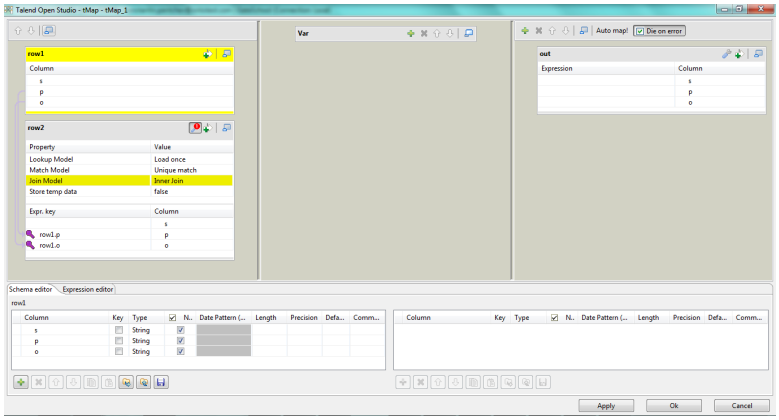

Figure 7

What remains to be done is link the input to the output. To do this again drag-and-drop all columns of row1 to their corresponding counterparts in the output row. Finally, press on the settings button of the output link and set *Catch lookup inner join reject* to "true". This will force only the data from row1, which does not match to be passed to the output.

Press "OK" to close the tMap menu. Now you are ready start the job. The output should return 28 out of 30 statements, meaning only 2 are shared between the datasources. You can switch between outputting the shared/different statements by changing the value for *Catch lookup inner join reject*.

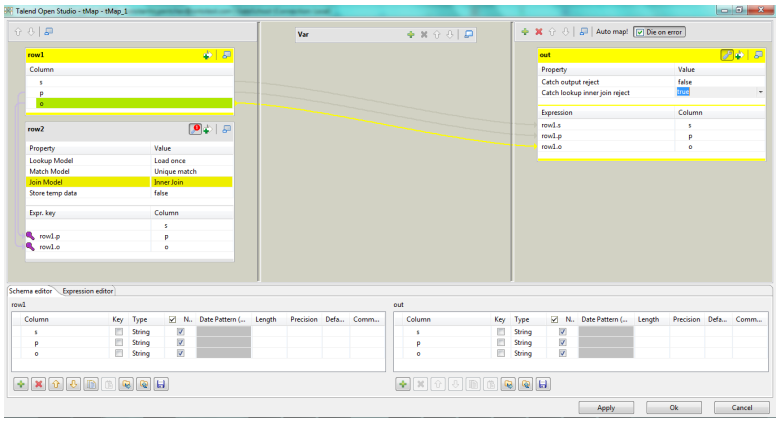
Talend Open Studio - tMap - tMap_1
row1
Column
s
p
o
row2
Property
Value
Lookup Model
Load once
Match Model
Unique match
Join Model
Inner Join
Store temp data
false
Expr. key
Column
row1.p
row1.o
Var
Auto map!
Die on error
out
Property
Value
Catch output reject
false
Catch lookup inner join reject
true
Expression
Column
row1.s
row1.p
row1.o
Scheme editor
Expression editor
row1
out
Column
Key
Type
N..
Date Pattern (...
Length
Precision
Defa...
Comm...
String
Apply
Ok
Cancel

Figure 8

2 Querying LinkedData

2.1 Get semantic types from LinkedLifeData

A common question regarding LinkedLifeData is what semantic times are present in the repository. In the following you will see how to retrieve the *UMLS semantic network types*, do some transformations on them and store them for further use, e.g. MIMIR indexing.

To query data from a remote semantic repository the *tSPARQLEndpoint* is used. Locate the component , select it and use drag-and-drop to add it to your workflow. Then double-click on the icon in the design area to view and edit the component's properties in the general menu. tSPARQLEndpoint has 5 parameters: *URL to repository* specifies the address, where the remote semantic repository is available; *Username* and *Password* can provide credentials if the repository requires authentication; *Schema* allows you to specify the variables to be passed to the next components. These should correspond to the variables selected in the *SPARQL query*. Enter the following values in order to retrive the semantic types:

- URL to repository: `http://linkedlifedata.com/sparql`
- Username and Password: empty
- Schema: one column "tlabel"
- SPARQL query:

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core\#>

SELECT ?tlabel
WHERE {
    ?st <http://linkedlifedata.com/resource/calbc/inGroup> ?group .
    ?st skos:prefLabel ?tlabel
}
```

Next add a component of type tReplace from the Processing tab in the component's palette. Connect the two components, by right-clicking on the tSPARQLEndpoint component, choose Row -> Main and point to the tReplace component (see Fig.9).

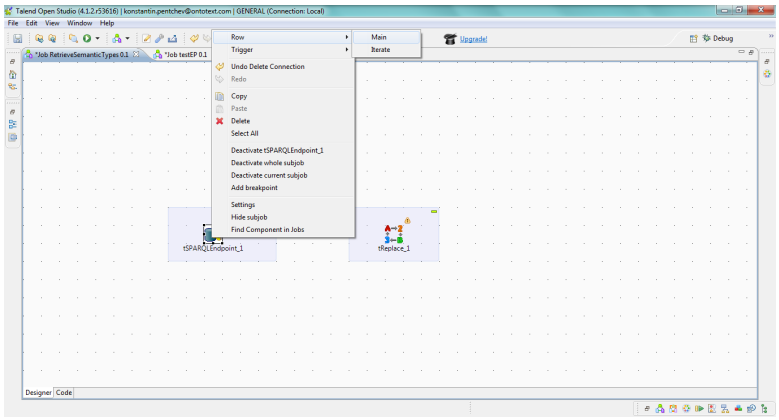

Figure 9

Double-click on the tReplace component and from the component's menu deselect the Simple mode checkbox and and select Advanced mode. Add two rows, with patterns "\\s", "," and replacements "_", "" respectively. The first will convert all whitespace characters to underscores and the second will remove all commas from the semantic type labels. **Do not forget**

to check and alter the components Schema to match the one of its predecessor! Because we want to simultaniously print the semantic types to the console and write them in a file, the next step is to add a tReplicate component from the Orchestration tab. Connect the tReplace component to the tReplicate component. Finally, add a tLogRow(from Logs and Errors) and a tFileOutputDelimited(from File/Output). Add a connections from tReplicate to both of these components. The one remaining thing to do is configure the tFileOutputDelimited. Choose a path, where the file will be saved, set the Row separator to "\n" and the Field separator to "". Save your job by pressing CTR+S and execute it from *General menu -> Run -> Run*.

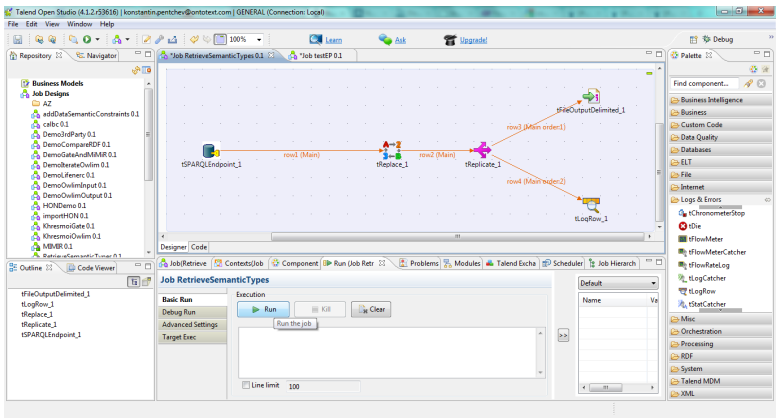

Figure 10

3 Custom Vocabulary Gazetteer

In the following you will see, how you can populate a gazetteer with a custom set of types and entities from a remote semantic repository, create a gate application, execute it and store it for later use. For this you must have GATE Developer installed. Our goal is to annotate documents from pubmed and extract different diseases from the text.

3.1 Gate Application with LKB Gazetteer

Start the GATE Developer. A window with the IDE should appear on screen (see fig.11).

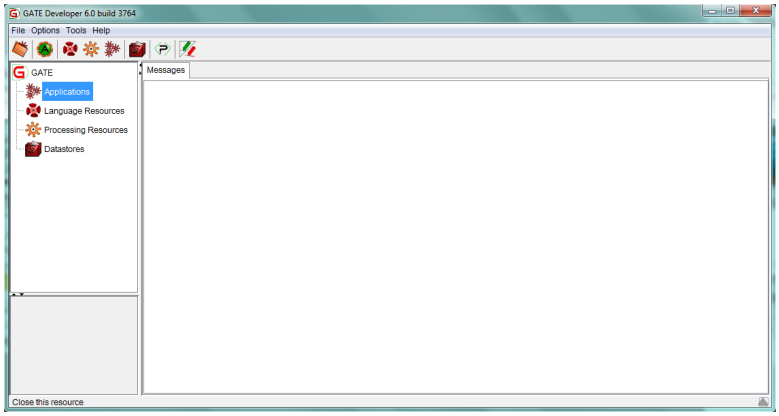

Figure 11

Create a new GATE application by selecting from the left hand-side menu Applications->New->Conditional Corpus Pipeline.

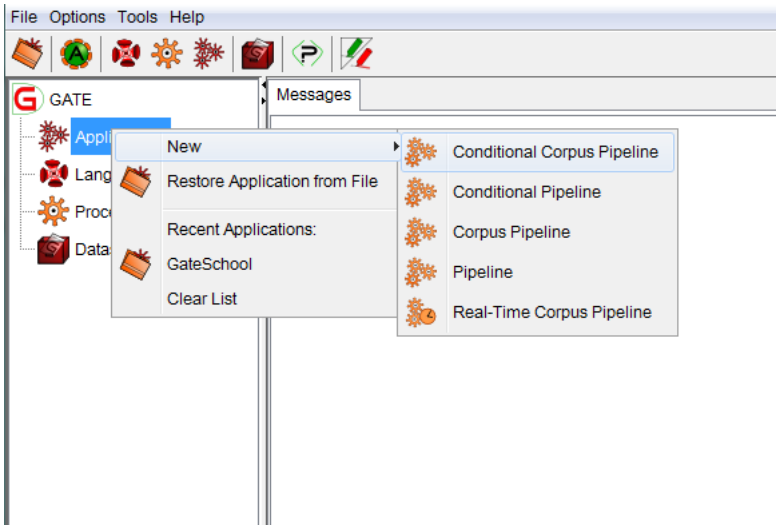

Figure 12

Choose an appropriate name for the application and press "OK". It should appear in the left hand-side menu below Applications.

Next, you'll need to load the Large Knowledge Base(LKB) Gazetteer plugin. Gazetteers are the units in GATE, which perform Named Entity Recognition(NRE). The LKB Gazetteer is special, because it allows you to load your lookup dictionary dynamicaly from a remote semantic repository using SPARQL. To load the plugin press on the questionmark-shaped icon

in the top-side menu labeled Manage CREOLE Plugins, find the Gazetteer_LKB in the list, check the box for Load now and press "OK".

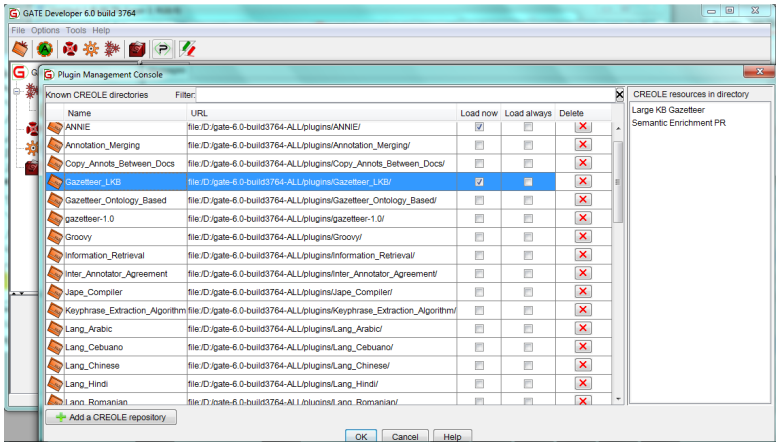

Figure 13

Now if you right-click on the Processing Resources tab in the left hand-side menu, choose New and Gazetteer_LKB you will be able to add a new custom processing resource of the type.

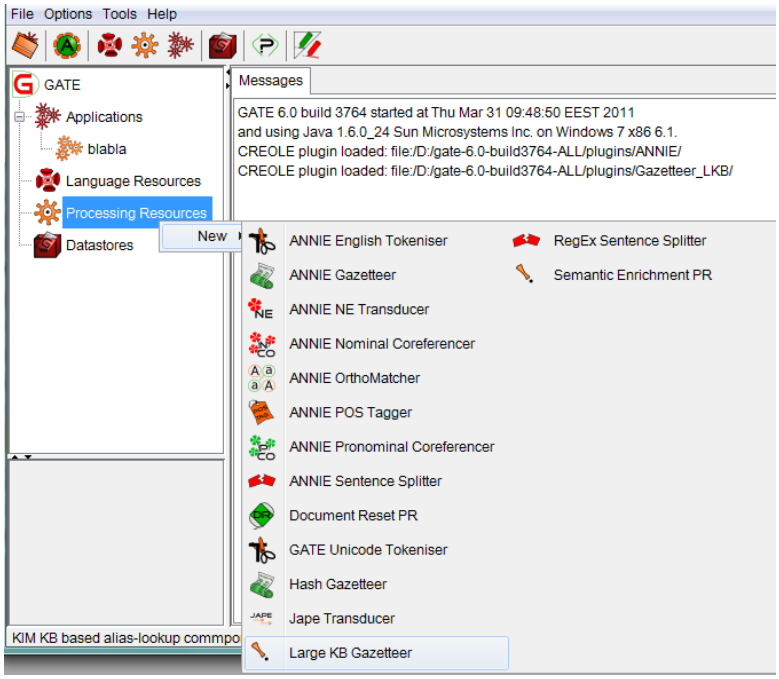

Figure 14

You will be prompted to specify the path(directory) to the Gazetteer's resources. These include a .ttl configuration file and a text file containing the SPARQL query to populate the dictionary. A sample configuration file will look like the following:

```
\# Gazetteer\_LKB dictionary configuration file.
\#
\# %temp% will be automatically replaced with the TEMP folder for the current user
\#
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema\#>.
@prefix rep: <http://www.openrdf.org/config/repository\#>.
@prefix hr: <http://www.openrdf.org/config/repository/http\#>.
@prefix lkbg: <http://www.ontotext.com/lkb\_gazetteer\#>.

\# The dictionary will be loaded from a remote Sesame HTTP repository.
\# Its configuration follows. See the Sesame configuration for detals.
\# http://www.openrdf.org/doc/sesame2/users/ch07.html\#section-repository-config
[] a rep:Repository ;
   rep:repositoryImpl [
      rep:repositoryType "openrdf:HTTPRepository" ;
      hr:repositoryURL <http://linkedlifedata.com/sparql>
   ];
   rep:repositoryID "owlim" ;
   rdfs:label "LinkedLifeData" .

\# The gazetteer-specific options follow.
lkbg:DictionaryConfiguration
   \# Whether the gazeetteer will be sensitive to case.
   \# Possible values: caseinsensitive, casesensitive
   lkbg:caseSensitivity "caseinsensitive" ;

   \# Whether the gazetteer will cache the dictionary after loading it from the data source.
   \# Put "enabled" to enable the cache, any other values will disable it.
   lkbg:caching "enabled" .

   \# The cache will be automatically reloaded on initialization if the configuration have been
   \# modified since the last initialization. However, changes in the underlying datastore
   \# can't be detected and the cache will not be automatically reloaded in that case.
   \# Thus, if you expect changes in your datastore, disable the cache, or make use
   \# of the modifications API.
```

It is important that you set up the repository connection properly. Change the hr:repositoryURL appropriately to the SPARQL endpoint you wish to query. Other parameters you might need to alter are rep:repositoryID and rdfs:label. The next thing you will need to do is specify the SPARQL query in a file called query.txt. It must return 3 variables, specifying the label of the concept(literal), its identifier(URI) and semantic type(URI). These will be included in the annotations later on. For this usecase, we will require a list of diseases for our dictionary. The following SPARQL query will return all concepts of type "Disease or Syndrome" from LinkedLifeData:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns\#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core\#>

SELECT distinct ?literal ?concept ?type
WHERE {
{
\# retrieve all UMLS concepts' preferred labels, classified as "Lung diseases"
    graph ?g1 {?concept rdf:type ?type} .
    graph ?g2 {?type rdf:type <http://linkedlifedata.com/resource/umls/SemanticNetworkConcept>} .
    ?concept skos:broader ?root .
    ?root skos:prefLabel 'Lung diseases' .
    ?concept skos:prefLabel ?literal}
        UNION
{
\# retrieve all UMLS concepts' alternative  labels, classified as "Lung diseases"
    graph ?g1 {?concept rdf:type ?type} .
    graph ?g2 {?type rdf:type <http://linkedlifedata.com/resource/umls/SemanticNetworkConcept>} .
    ?concept skos:broader ?root .
    ?root skos:prefLabel 'Lung diseases' .
    ?concept skos:altLabel ?literal}
}
```

Once you've created your LKB Gazetteer and added it to the Processing resources you have to also add it to your application. Double-click on the latter to open it in the main frame. Then select your LKB Gazetteer and add it by pressing the right-arrow in the middle of the screen.

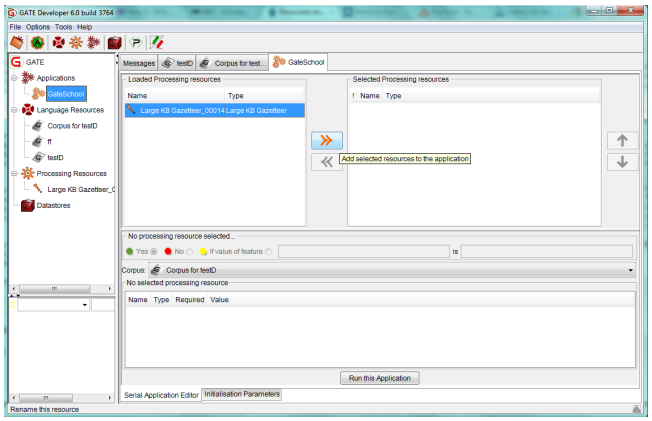

Figure 15

3.2 GateDocuments and Corpora

Your application is now capable of performing NER and annotating the text accordingly. To run the application you'll need to add a Language resource, i.e. a document to be annotated. Do this by right-clicking on Language resources, then select New -> GATE_Document and then select a text resource of your liking (e.g. `http://linkedlifedata.com/resource/pubmed/id/18971902`).

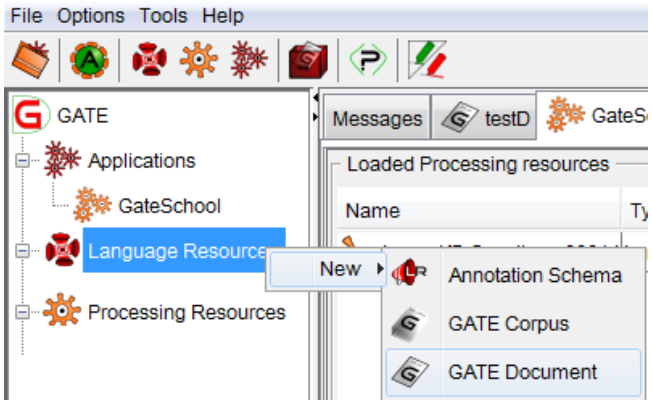

Figure 16

The document title should appear in the left hand-side menu under Language resources. Right-click on it and select New Corpus with this Document. This will create a corpus containing the document, which can now be processed by the application.

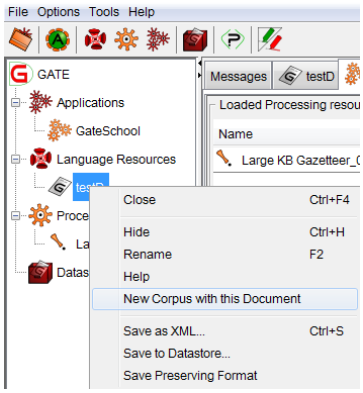

Figure 17

Now from the main application frame select the newly created corpus and annotate the document(s) in it by pressing Run this Application.

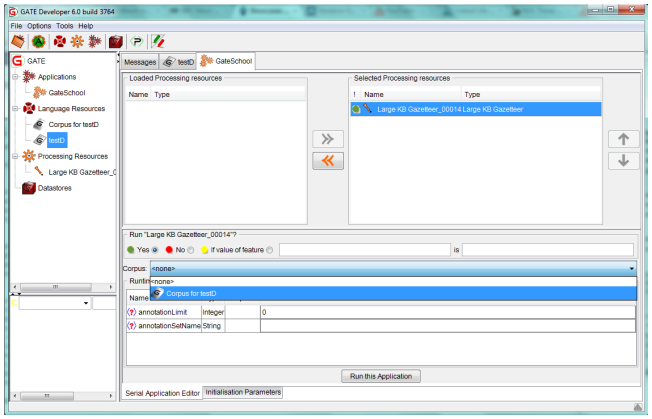

Figure 18

Now double-click on the document title in the left hand-side menu to open it. Then select the tab Annotation sets, which will open on the right hand-side a panel containing the different annotation sets(classes). To view the results of your job select the one available option - lookups.

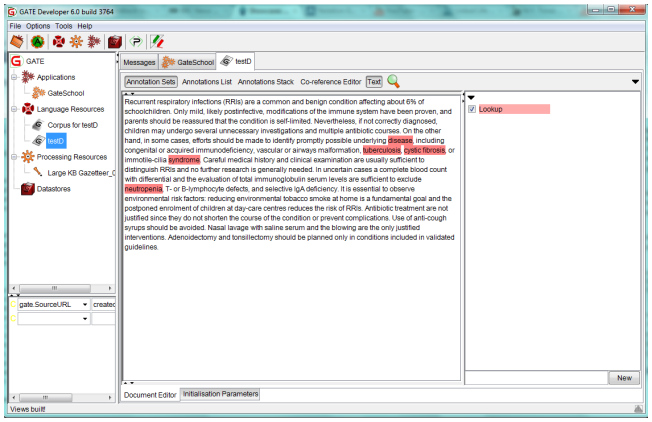

Figure 19

3.3 JAPE transformations

However, this annotaiton set is not particularly informative and will actually impair further processing of the documents, which we would like to perform. In order to enchance the annotation sets, add an additional Processing resource of type Jape transducer:

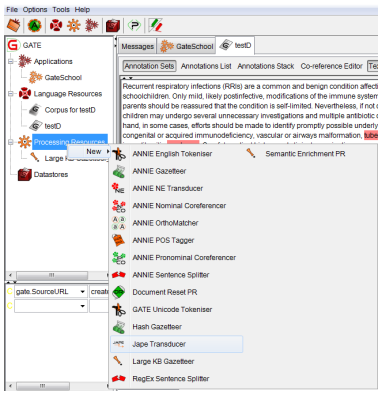

Figure 20

Select the jape called umls_transform_priority.jape from GATE.Home/plugins/gazetteer-0.1/models/gaze/jape. Then add it to the application processing resources similar to how you did with the LKB Gazetteer and annotate the document anew. Now an additional annotation set called Disease_or_Syndrome should be available, containing all of our annotations. What the Jape did is copy the annotations with the annotation type set to the instance type label.

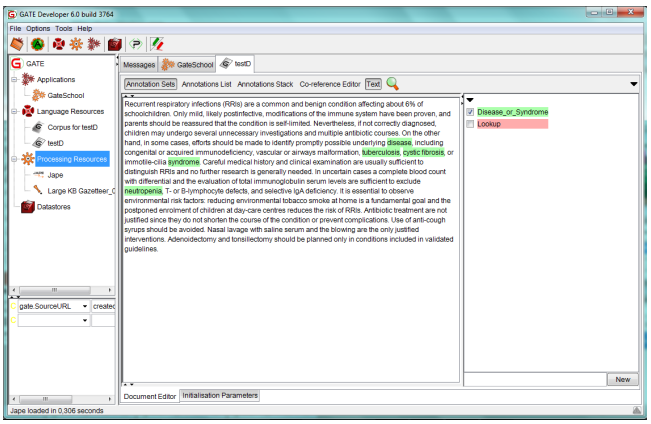

Figure 21

3.4 Saving the pipeline

To reuse the application at a later stage it has to be saved on the hard drive. However, for the sake of the following exercises, please add an additional processing resource to it first - the *ANNIE English Tokenizer*. Then right-click on the application name in the right hand-side menu, select *Save Application State* and specify an appropriate location for the "*.xgapp" file.

4 Semantic Annotations and LinkedData

LinkedData is a virtually endless source for text data, which can be annotated to extract meaning. Components for working with the GATE annotation platform in Talend are provided by Ontotext. This allows fo their seamless integration with semantic data solutions.

4.1 Annotate data from SPARQL endpoint

In the following task you need to query all pubmed articles from LinkedLifeData *published in 2008*, which concern the *therapy* of *lung diseases*. Annotate these articles using the GATE application from *section 3* and view the results.

Start by adding a tSPARQLEndpoint to the new workflow. Configure it using the parameters from *section 2.1* and use the following query:

```
PREFIX pubmed: <http://linkedlifedata.com/resource/pubmed/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema\#>

SELECT ?id ?content
WHERE {
    ?id pubmed:year '2008'.
    ?id pubmed:abstractText ?content .
    ?id pubmed:meshHeading ?mesh .
    ?mesh pubmed:qualifier ?qualifier .
    ?qualifier rdfs:label 'therapy' .
    ?mesh pubmed:mesh ?meshTerm .
    ?meshTerm rdfs:label 'Lung Diseases' .
}
```

Adjust the component schema accordingly. This will be our datasource.

The next step would be to add and configure the annotation components. This will require some initialization prior to the annotation process itself. This can be achieved by adding a *tPreJob* component to the workflow from the *Orchestration* tab. Now add a *tGateInstantiator* and connect the tPreJob component to it using a *On Component Ok* link. Double-click on the the component to edit its properties: it requires a path to the installation folder of GATE and the path to the desired GATE application file (GAPP). If you have already set the environment variable *GATE_HOME* to point to the appropriate directory no further action is required, otherwise uncheck the *Use env variable for GATE_HOME* button and select it manually. Finally, select the GAPP you created in *section 3.4*. Leave the *Number of GAPPs to create in pool* set to "1". This parameter is important when parallel executions of the pipeline are required and should equal the number of threads used.

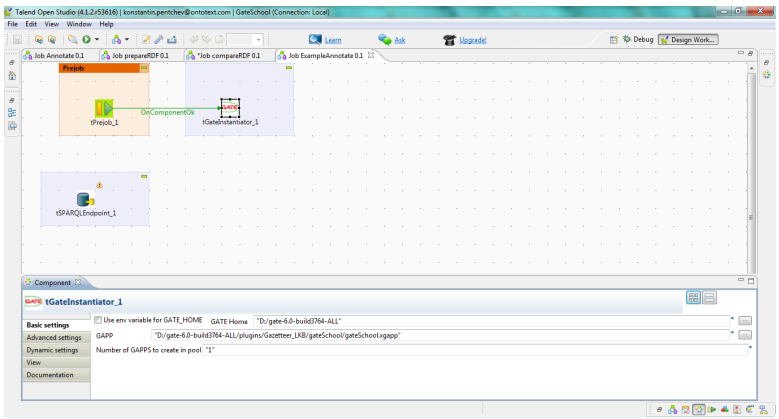

Figure 22

Now the actual annotation component has to be added – the *tGateAnnotator*. It has some specifics though: it accepts only iterate links(because of parellelization features) and retrieves the required document data from an enternal Map object. Therefore, add a *tFlowToIterate* component, which will transform the flow link into an iterate link. Connect tSPARQLEndpoint to it, add tGateAnnotator and connect it as well. Check the *Use preinstantiated Gate* box. Next, to set the *Document id* and *Document content* start typing "tFlowToIterate", press CTRL+SPACE and select *tFlowToIterate_1.id* and *tFlowToIterate_1.content* respectively.

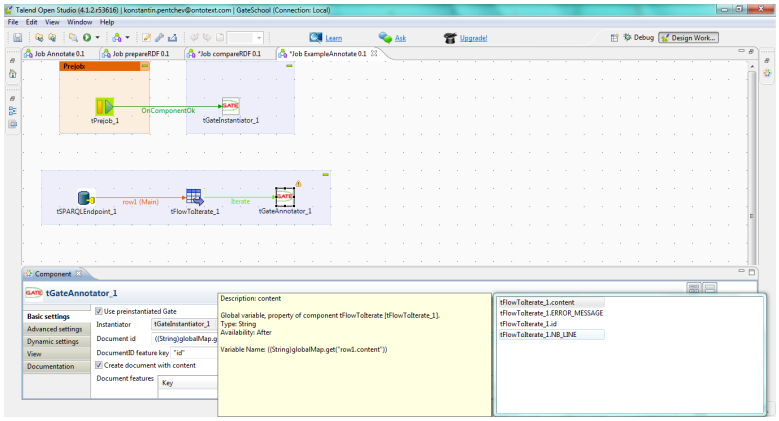

Figure 23

Finally, add a *tGateDocumentToXML* to persist the resulting documents in an XML file. Select a file and check the *Append* box.

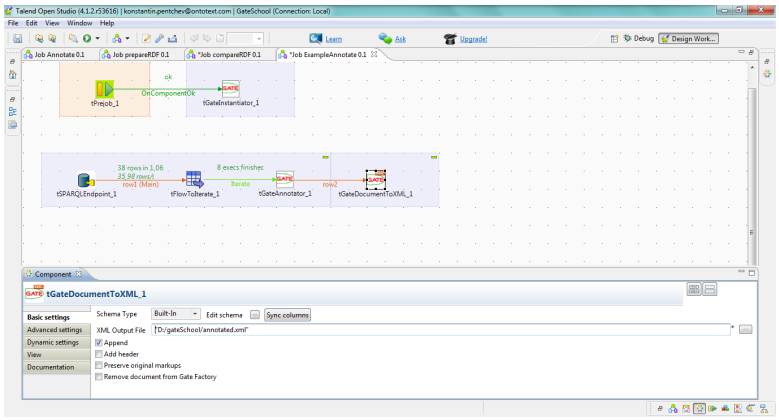

Figure 24

4.2 Convert annotated documents to RDF

The next task is to convert the annotated documents to RDF. This can be done easily immediatly after annotation. Simply replace the tGateDocumentToXML component with a *tGateDocumentToRDF*. When connecting the components, choose not to acquire the schema of the target component. This component will transform all annotations from a specified *annotation set* to rdf triples of the form: `<documentId> <predicate> <annotationinstance>`. Use the component with the default parameter settings.

If you want to persist the RDF statements in a file just add a tRDFWriter component. What is more interesting is to insert these statements into a semantic repository. In order to do this, it will suffice to use the *tSesameOutput* component. However, it is recommended to use

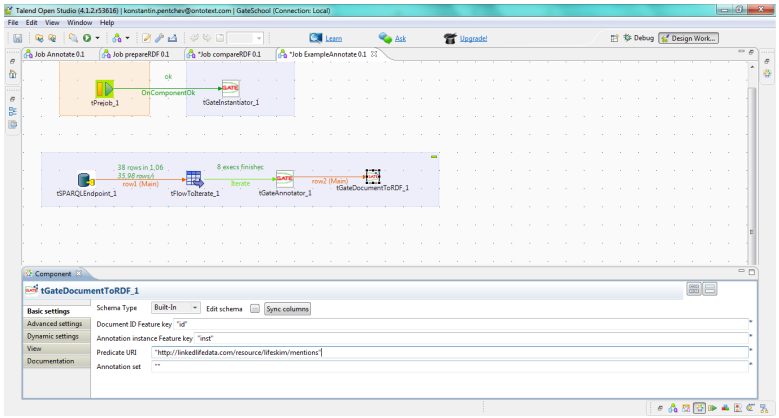

Figure 25

it together with the *tSesameConnection* and *tSesameConnectionClose* components for optimal performance. Add the former to the preJob process by linking tGateInstantiatior to it using *On Component Ok*. Select a storage folder for it and a config file (download from wiki). Next, set the tSesameOutput to use an existing connection. In addition, because the tSesameOutput component uses a quadruple schema and the tGateDocumentToRDF a triple schema you need to use a tMap to map the two. What remains to be done is close the connection when the processing is finished. In order to do this add a *tPostjob* component to the workflow and link it to a tSesameConnectionClose similarly to tPrejob.

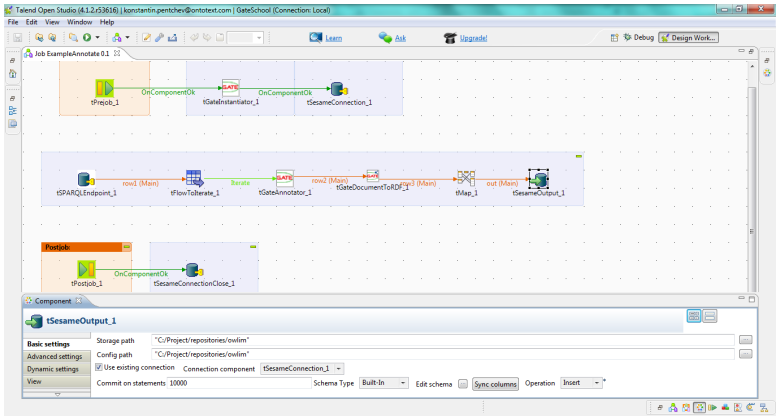

Figure 26

5 Mimir

"Mimir is a multi-paradigm information management index and repository which can be used to index and search over text, annotations, semantic schemas (ontologies), and semantic metadata (instance data). It allows queries that arbitrarily mix full-text, structural, linguistic and semantic queries and that can scale to gigabytes of text. A typical semantic annotation project deals with large quantities of data of different kinds. Mimir provides a framework for implementing indexing and search functionality across all these data type."

Mimir makes use of several plugins for storing(ordi, h2, sesame) and accessing data(sparql). Ontotext provides Talend components for indexing data with Mimir using the sesame plugin.

In the following you will perform indexing with Mimir of annotated documetns. Then you will *import* the *existing index* into your Mimir web application and search it. The web application is available for download from the course materials page.

The components required for indexing with Mimir are *tMimirInitializer, tMimir and tMimirClose*. We will modify the annotation pipeline from *section 5*: replace tSesameConnection and tSesameConnectionClose with tMimirInitializer and tMimirClose respectively. Remove tSesameOutput and tGateDocumentToRDF, add tMimir to the workflow and connect tGateAnnotator to it. Next, edit the properties of the tMimirInitializer component:

- choose an index dir; the folder **must not** exist
- check the *load annotation types from file* box; select the *.txt file created in *section 2.1*
- select the repository config file *owlim.ttl* (download with other materials)

The annotation types file specifies a list of annotation classes, which will be handled by Mimir. All other will be ignored. Also make sure you have chosen the appropriate annotation set (empty string for this tutorial).

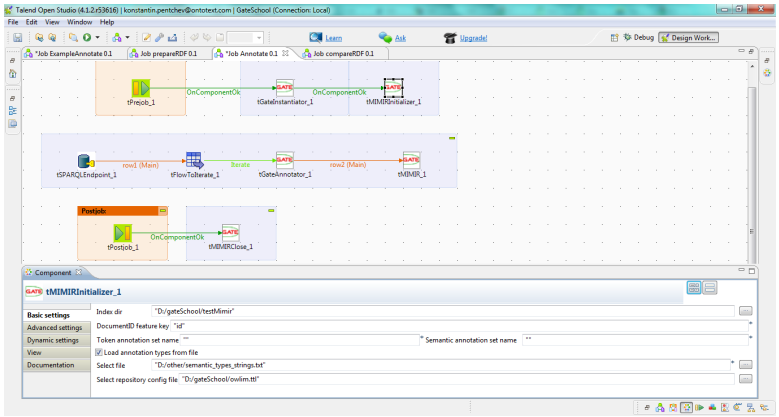

Figure 27

Next, select the tMimir component, open its properties tab and check the *use initialized index* box. You are now ready to index annotated documents.

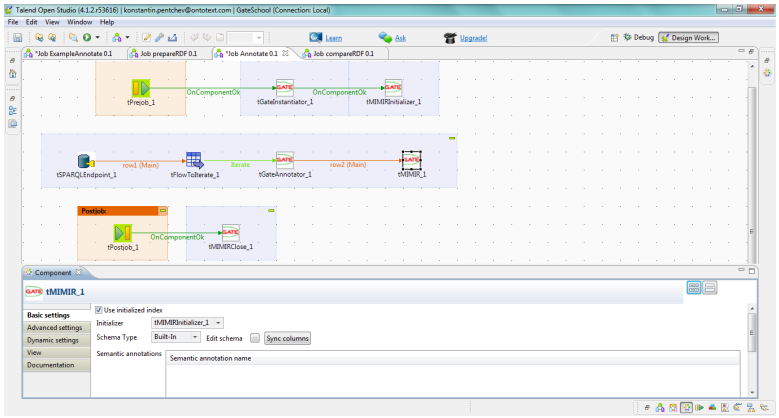

Figure 28

If the job was set up properly, 38 documents will be annotated and indexed.

To search the index, you need to deploy the mimir web application (download from `gate...`; you are also required to have tomcat installed). Copy the *.war* file to the `webapps` folder of tomcat and start the server by running *startup.bat/startup.sh* from the `bin` folder. If you were successful, the mimir web application will be accessible at `http://localhost:8080/mimir-demo-3.2.0-snapshot/admin`. Import the index by selecting *"import an existing index*

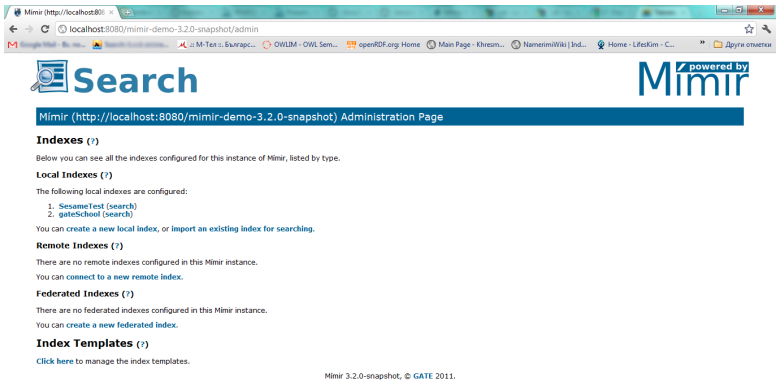

Figure 29

for searching". Choose a name for your index and enter the path to its folder. Check the *Document URIs are external links* box and click on *"Import"*.

Figure 30

Then select *"Search this index using the web interface"* and a search from should appear. The query `{Disease\_or\_Syndrome}` will return all documents(multiple snippets) containing *disease* annotations: The result set should have 29 items. Among the found diseases will be **pneumonia, emphysema, respiratory failure, pulmonary hypertension** and others.

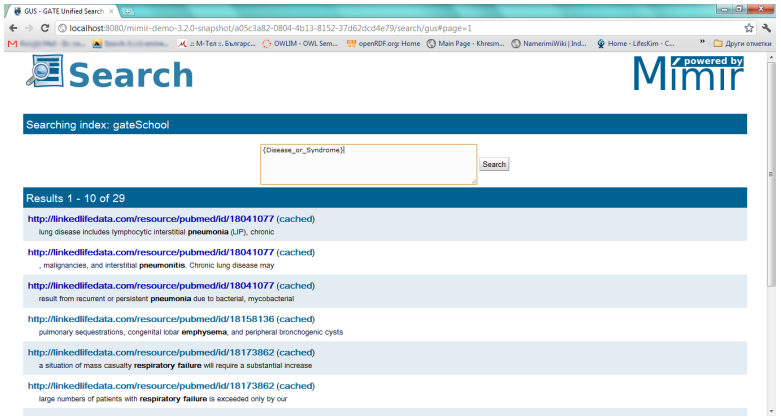

Figure 31

However, such a search is not very specific. Ideally, one should be able to provide additional parameters to constrain the query and receive more specific results. A valid case with regard to our data, is filtering the diseases by symptoms experienced. To do this, we will send a sparql query to LinkedLifeData, which will select a set of diseases related to a symptom. Mimir will then return only matching disease annotations, which are included in the sparql query result set.

An appropriate symptom from UMLS is *Larynx pain*, whose URI is `http://linkedlifedata.com/resource/umls/id/C0549373`. The Mimir query will look like the following:

```
{Disease_or_Syndrome sparql = "SELECT ?inst WHERE {
?inst
<http://linkedlifedata.com/resource/relationontology/hasSymptom>
<http://linkedlifedata.com/resource/umls/id/C0549373>
}"
}
```

This query will yield only 17 results. You will note that some diseases have been filtered out from the results, e.g. respiratory failure.