OwlExporter

Guide for Users and Developers

René Witte
Ninus Khamis

Release 3.2
September 28, 2014

Semantic Software Lab
Concordia University
Montréal, Canada
http://www.semanticsoftware.info

Contents

List of Figures

About this document

This is the documentation for the *OwlExporter*, a component for the GATE architecture to export the results of a text analysis pipeline into an OWL ontology. Please refer to http://www.semanticsoftware.info/owlexporter for more information.

Acknowledgments

The following developers contributed to the design and implementation of the OwlExporter: René Witte, Ninus Khamis, Elian Angius, Qiangqiang Li and Thomas Kappler.

Licenses

[1] AGPL3, http://www.gnu.org/licenses/agpl-3.0.html
[2] LGPL, http://www.gnu.org/licenses/lgpl.html

[3]MPL, http://www.mozilla.org/MPL/

Chapter 1.

Introduction to the OwlExporter

In this chapter, we provide a brief introduction to the concept of *ontology population from text* and explain the major features of our OwlExporter GATE PR. For more background information, please refer to our publication (Witte et al., 2010).

1.1. Ontology Population Foundations

Ontologies in OWL format (Baader et al., 2003) provide for a standardized means of modelling, querying, and reasoning over large knowledge bases. Populated from natural language texts, they offer significant advantages over traditional export formats, such as plain XML.

In particular, ontologies have become a major tool for developing semantically rich applications: Ontology models are capable of representing a large amount of information using a small number of axioms (individuals and relationships). The ontology models provide users with a high level conceptualization of the information, while at the same time enabling them to focus on specific parts of the model. Web ontologies developed in the OWL-DL language also provide a standardized means for querying, linking, and reasoning about knowledge.

As a majority of the world's knowledge is encoded in natural language text, automating the population of these ontologies using results obtained from NLP analysis of documents (Cimiano, 2006) has recently become a major challenge for NLP applications (Figure 1.1).

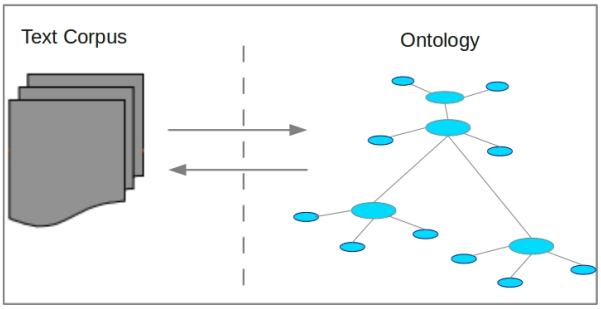


Figure 1.1.: Populating Ontologies from Text

1.2. OwlExporter Overview

The OwlExporter is a reusable component capable of populating OWL ontologies from GATE annotations, which is important for users that need to create more expressive models and would like to benefit from the inferences created by a description logic reasoner, such as Racer (Haarslev and Möller, 2001), Pellet (Sirin et al., 2007), or FaCT++ (Tsarkov and Horrocks, 2006).

The General Architecture for Text Engineering (GATE) is a framework used for the processing of natural language documents.[1] The GATE framework can be extended using plug-ins, known as a Collection of REusable Objects for Language Engineering (CREOLE). GATE's ontology layer is based on OWLIM (Kiryakov et al., 2005), which only supports the less semantically rich OWL-Lite language. In contrast, the OwlExporter makes use of the Protg-OWL libraries and can populate Owl-Lite, OWL-DL, and OWL-Full ontologies.

The OwlExporter is a Processing Resource (PR) that allows to largely automate ontology population from text using a GATE application. With the use of a few simple mapping grammar, the OwlExporter is able to populate an existing ontology using the annotations created by the application (Witte et al., 2010). In addition to domain-specific concepts, the OwlExporter is also capable of exporting NLP concepts, such as Sentence, NP and VP, and integrating reasoning support for entities that reappear within a given text and are linked together using coreference chains.

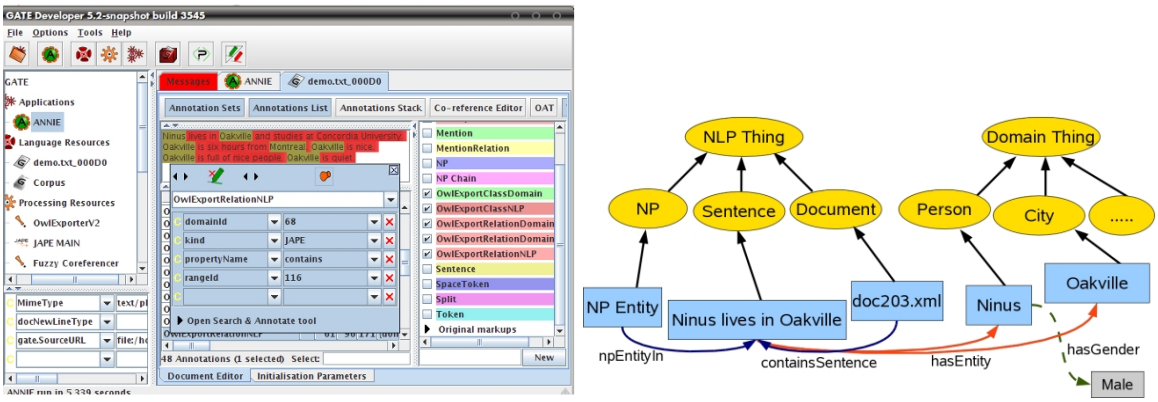


Figure 1.2.: The OwlExporter Processing Resource

1.2.1. OwlExporter Features

The main features of the OwlExporter are:

Exporting individuals: Create OWL individuals using entities of a corpus.

Exporting datatype properties: Create OWL datatype property relationships using information from a corpus.

Exporting object properties: Create OWL object property relationships using the relationships found in a corpus.

Exporting coreference chains: Create coreference chains using *owl:sameAs* for entities that re-appear in different parts of a corpus.

[1] GATE, http://gate.ac.uk

1.3. How to read this documentation

To understand how the OwlExporter is able to export entities of a corpus as individuals and relationships of an ontology, we have included an example GATE pipeline, which is discussed in Chapter 2. Also in Chapter 2, we cover the annotations and features needed to export OWL individuals and relationships using the OwlExporter. In Chapter 3, we discuss additional features that are included with the OwlExporter, such as the ability to export entities that reappear in different parts of a corpus as coreference chains. Finally, in Chapter 4, we discuss the process of compiling and importing OwlExporter, as well as the design decisions of our OwlExporter component.

Chapter 2.

Your First Ontology Population Pipeline

In this chapter, we explain the basic use of the OwlExporter, based on the included example application.

2.1. OwlExporter Installation

The goal of the demo is to show how annotations created by a standard NLP pipeline (here ANNIE) can be exported as instances and relationships of an OWL ontology.

Prerequisites. In addition to the OwlExporter distribution, you must have GATE installed. Please refer to http://gate.ac.uk for instructions on how to download and install GATE. Note that you will need GATE version 7 or better to run the OwlExporter.

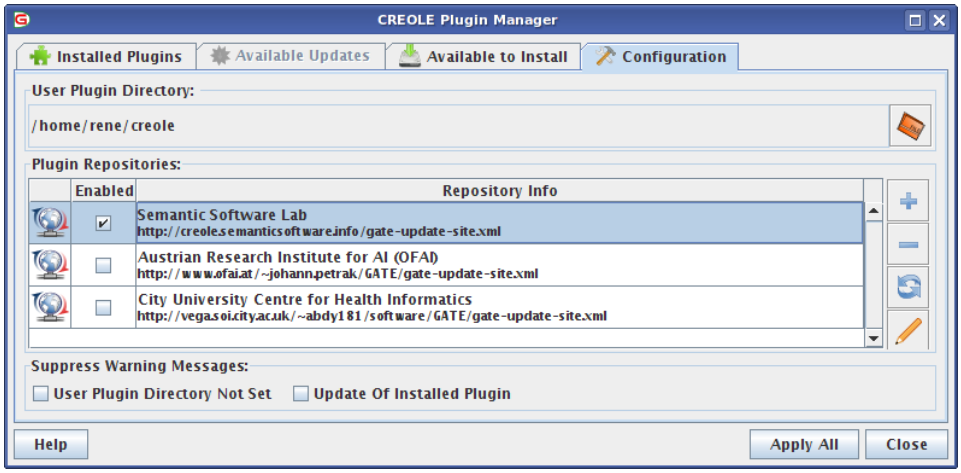


Figure 2.1.: Enabling the Semantic Software Lab repository in GATE's CREOLE Plugin Manager

1. Start GATE, open the Plugin Manager *(File → Manage CREOLE Plugins...)*
2. On the fourth tab *(Configuration)*, select a writable directory for the local plugin installations and enable the "Semantic Software Lab" repository (see Figure 2.1).
3. On the third tab *(Available to install)*, you will now find the OwlExporter System. Select it and click on "Apply All".

4. On the first tab *(Installed Plugins)*, you will now find OwlExporter installed. Select it (either "Load Now" or "Load Always") and click on "Apply All". OwlExporter is now ready.

2.1.1. Loading the Demo Application

To load the system, go to *File* → *Ready Made Applications* → *Semantic Software Lab* (see Figure 2.2).[1].

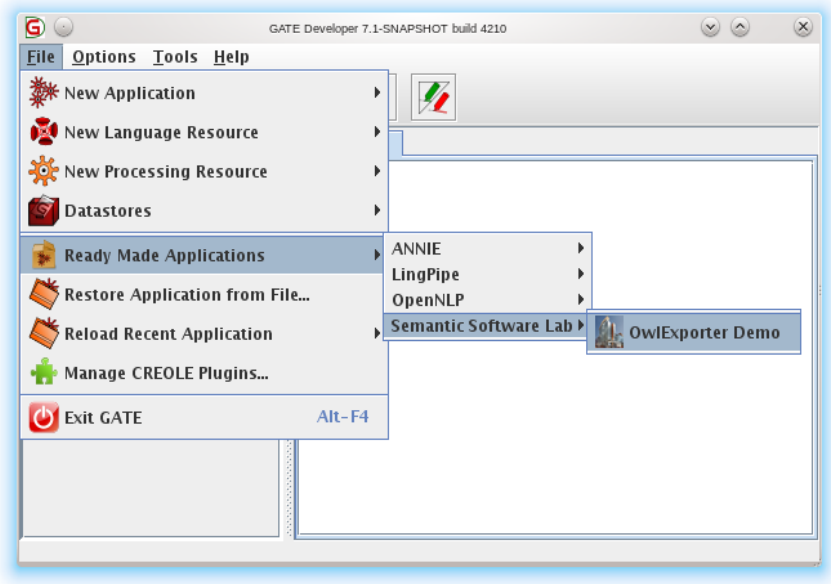


Figure 2.2.: Loading the OwlExporter pipelines in GATE Developer

For the demo described below we use two example ontologies:

ANNIE Ontology: For the domain ontology we used the demo.owl[2] taxonomic classification that models commonly used ontology concepts in GATE such as `Person`, `Organization` and `Location`.

NLP Ontology: A simple NLP ontology that contains common ontology concepts such as Document, Sentence, NP, and relationships such as contains, containsSentence, HEAD and HEAD_START.

2.1.2. Running the Demo Ontology Population Pipeline

In order to populate the ontologies using the demo pipeline (right-click *"OwlExporterDemo"* → *"Run this Application"* The results of the pipeline (shown in Figure 2.4) include the newly created annotations needed by the OwlExporter, namely:

[1] Please refer to the GATE user's guide, http://gate.ac.uk/sale/tao for further details on working with pipelines in GATE.

[2] demo.owl is bundled with GATE and can be accessed from the `$GATE_HOME/plugins/Ontology_Tools/resources` directory

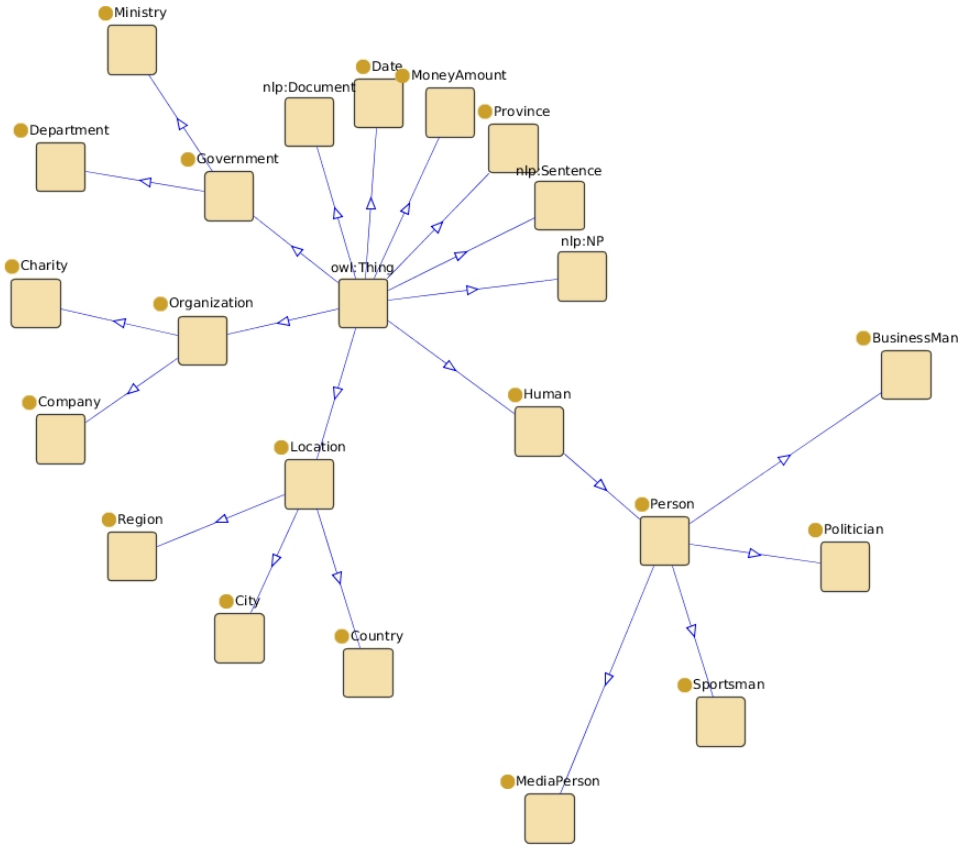


Figure 2.3.: The ANNIE and NLP Ontologies

OwlExportClassDomain: The OwlExporter uses the `OwlExportClassDomain` annotation to identify the domain annotations of a document that need to be exported as *instances* of their related ontology concept in the domain Ontology.

OwlExportRelationDomain: The OwlExporter uses the `OwlExportRelationDomain` annotation to identify the domain annotations of a document that need to be exported as *relationships* of their related *individuals* in the domain Ontology.

OwlExportClassNLP: The OwlExporter uses the `OwlExportClassNLP` annotation to identify the NLP annotations of a document that need to be exported as *instances* of their related ontology concept in the NLP Ontology.

OwlExportRelationNLP: The OwlExporter uses the `OwlExportRelationNLP` annotation to identify the NLP annotations of a document that need to be exported as *relationships* of their related *individuals* in the NLP Ontology.

OwlExportRelationDomainNLP: The OwlExporter uses the `OwlExportRelationDomainNLP` annotation to identify the *object property* relationships that need to be exported between the Domain and NLP ontology.

The output of the pipeline are the two ontologies *domain_out.owl* and *nlp_out.owl*, as specified using the run-time parameters:

exportDomainOntology: Specifies the location of the exported domain ontology.

exportNLPOntology: Specifies the location of the exported NLP ontology.

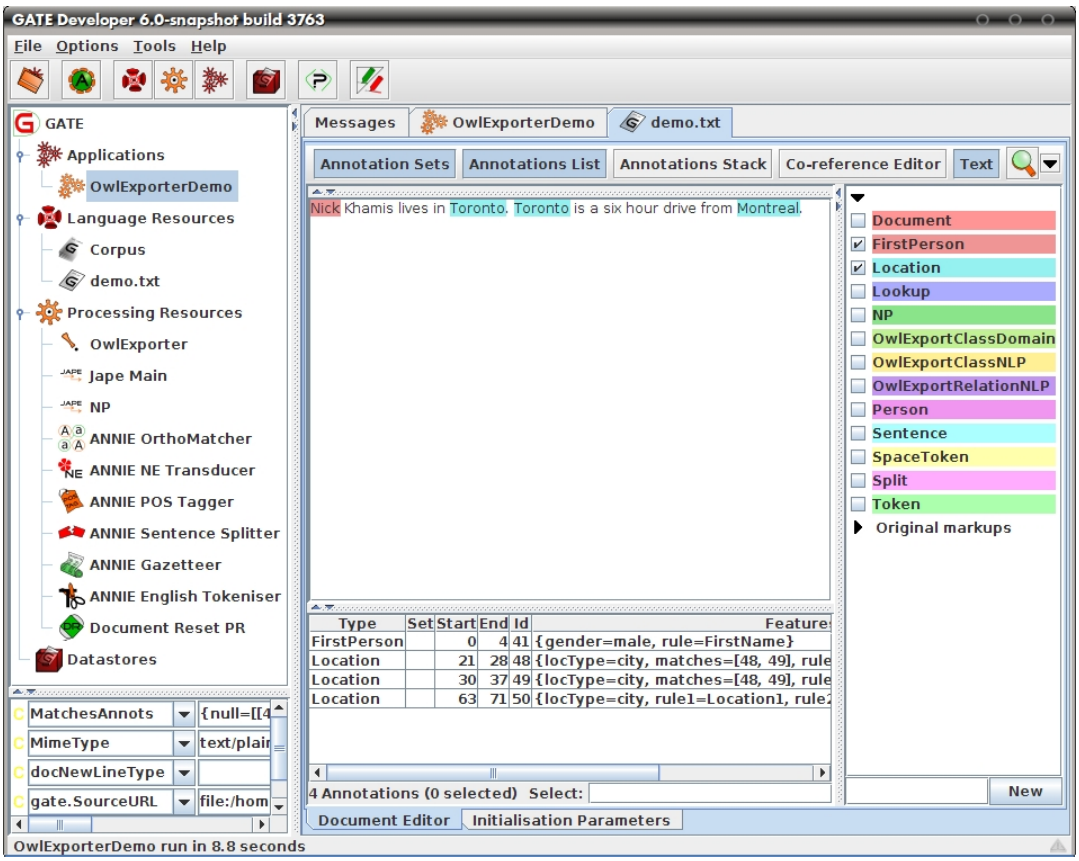


Figure 2.4.: The Demo Document in GATE Developer

By default, you will find them in the sub-directory `OwlExporter/gate/application-resources/ontologies/domain_out.owl`.

2.1.3. Result – The Populated Ontology

The result of the example pipeline are two populated ontologies as shown in Figure 2.5.

2.2. OwlExporter Concepts

We now look at the details on how the OwlExporter example pipeline works, so that you understand how to adapt it to your own applications.

2.2.1. Exporting Domain Individuals

In order to export the entities of a document as individuals of the domain ontology, the annotation types created by a pipeline need to be mapped to the related ontology concepts. The *OwlExporterClassDomain* annotation is used to map annotations created by a pipeline to the relevant domain ontology concepts.

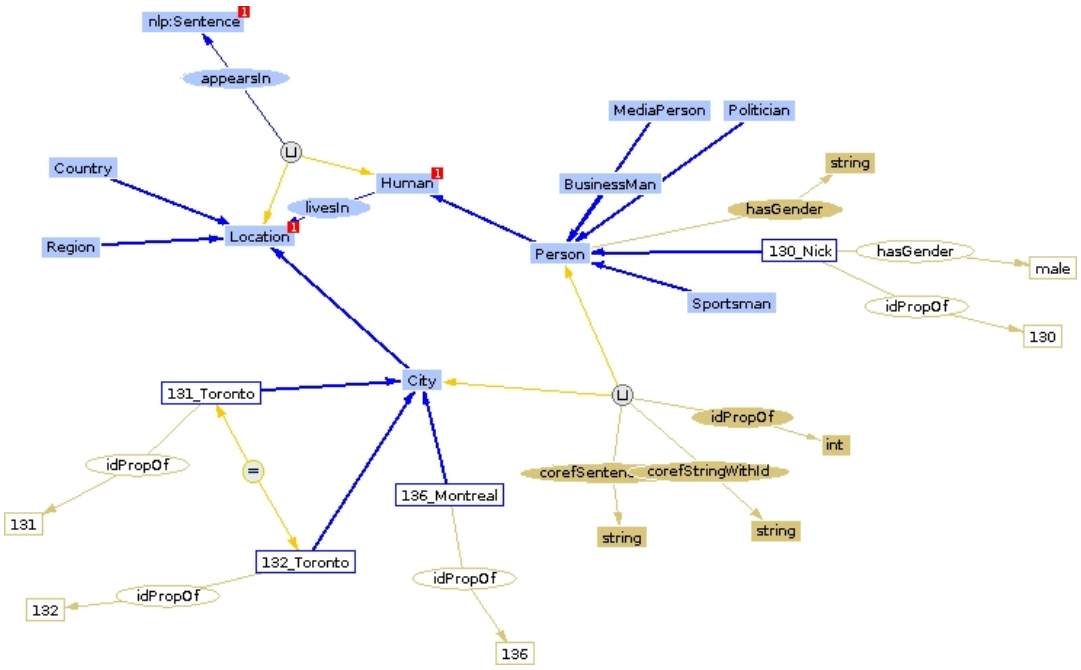
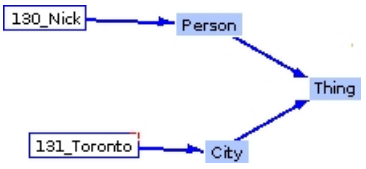


Figure 2.5.: Populated Ontology

```
1  AnnotationSet as = (gate.AnnotationSet)bindings.get("ann");
2  Annotation ann = (gate.Annotation)as.iterator().next();
3  FeatureMap features = ann.getFeatures();
4
5  if(ann.getFeatures().get("majorType").toString().compareToIgnoreCase("person_first") == 0) {
6          features.put("className", "Person");
7  }
8
9  if(ann.getFeatures().get("majorType").toString().compareToIgnoreCase("location") == 0) {
10         String s = ann.getFeatures().get("minorType").toString();
11         features.put("className", Character.toUpperCase(s.charAt(0)) + s.substring(1));
12 }
13
14 String in = doc.getContent().getContent(
15         ann.getStartNode().getOffset(), ann.getEndNode().getOffset()).toString();
16 features.put("instanceName", in);
17 features.put("representationId", ann.getId());
18 features.put("corefChain", null);
19 features.put("kind", "Class");
20 outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassDomain", features);
```

Nick lives in Toronto.

Type	Start	End	Id	Features
FirstPerson	0	4	41	{gender=male, rule=FirstName}
OwlExportClassDomain	0	4	42	{className=Person, corefChain=[null], instanceName=Nick, kind=Class, hasGender=male, representationId=11}
Location	14	21	43	{locType=city, rule1=InLoc1, rule2=LocFinal}
OwlExportClassDomain	14	21	44	{className=City, corefChain=[null], instanceName=Toronto, kind=Class, representationId=12}



Note: The value of the `className` feature for both the `OwlExportClassDomain` and the `OwlExportClassNLP` annotations must match the exact name of the mapped-to concept in the ontology.

Both the `OwlExportClassDomain` and `OwlExportClassNLP` annotations must contain the features shown in Table 2.1

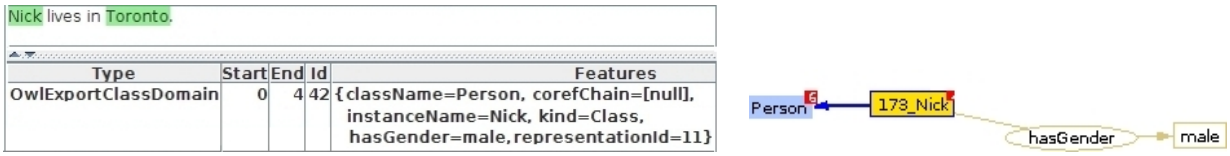
Feature	Description
className	This feature is of type String and must match the name of the Concept in the Ontology that the annotated text in the document is an instances of.
instanceName	This feature is String type and is the normalized form of the annotated text that needs to be exported.
corefChain	This feature is of type integer and is the ID of the coreference chain.
representationId	This feature is of type integer and is the ID of the annotation that needs to be exported.

Table 2.1.: Features needed by the `OwlExportClassDomain` and the `OwlExportClassNLP` annotations

2.2.2. Exporting Domain Datatype Relationships

In order for domain *datatype property* relationships to be exported, all that is required is the creation of the *property* in the domain ontology with a name that matches that of the *feature* of a given annotation that needs to be exported. Additionally, the domain of the *datatype property* must match the annotation type that the feature belongs to and the range of the property must match the literal type of the information being exported.

```
1 features.put("hasGender",
2         ann.getFeatures().get("minorType").toString());
```



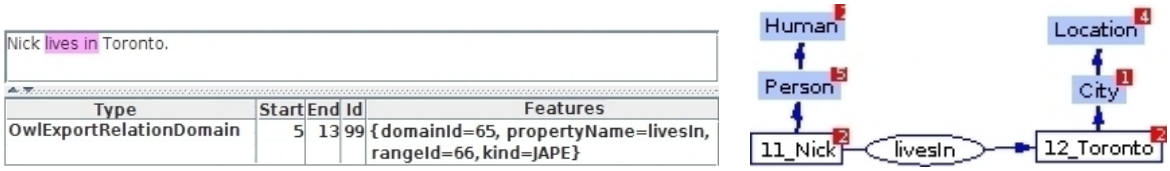
For instance, continuing with the `Person` example Jape grammar created previously, for the `hasGender` information to be exported as a *datatype property* relationship in the domain ontology, we created the *datatype property* `hasGender` that has `Person` as the *domain* and *xsd:string* as the range. Using the features created by the Jape grammar shown above, the `hasGender` *datatype property* relationship is created.

Note: No additional annotations are needed in order for the OwlExporter to create datatype property relationships.

Note: You must make sure the annotation set specified in OwlExporter's `inputASName` runtime parameter contains both the `OwlExportClassDomain`, as well as all the original annotations whose IDs are referenced by the `representationId` feature for datatypes to be exported.

2.2.3. Exporting Domain Object Property Relationships

In order for domain *object property* relationships to be exported, the name of the *property* in the domain ontology must match the *propertyName* feature of the `OwlExportRelationDomain` annotation. Additionally, the *domain* of the *datatype property* must match the annotation type indicated using the *domainId* feature, and the range of the *datatype property* must match the annotation type indicated using the *rangeId*.




```
gate.AnnotationSet domain = (gate.AnnotationSet)bindings.get("dom");
gate.Annotation domainAnn = (gate.Annotation)domain.iterator().next();

gate.AnnotationSet range = (gate.AnnotationSet)bindings.get("ran");
gate.Annotation rangeAnn = (gate.Annotation)range.iterator().next();

gate.AnnotationSet rel = (gate.AnnotationSet)bindings.get("rel");
gate.Annotation relAnn = (gate.Annotation)rel.iterator().next();

String propertyName="livesIn";

gate.FeatureMap features = Factory.newFeatureMap();
features.put("propertyName",propertyName);
features.put("domainId",domainAnn.getFeatures().
                                get("representationId"));
features.put("rangeId",rangeAnn.getFeatures().
                          get("representationId"));
features.put("kind", "JAPE");

outputAS.add(rel.firstNode(), rel.lastNode(),
               "OwlExportRelationDomain",features);
```

Both the `OwlExportRelationDomain` and `OwlExportRelationNLP` annotations must contain the features shown in Table 2.2.

Feature	**Description**
propertyName	This feature is of type String and must match the name of the related object property in the ontology.
domainId	This is feature is of type Integer, it is the id of the annotation that is set as the domain of the related object property.
rangeId	This feature is also of type Integer, it is the id of the annotation that is set as the range of the related object property.
kind	This features is String type and is used by the OwlExporter to identify annotations that need to be exported as relationships. The value of this feature is always JAPE.

Table 2.2.: Features needed by the `OwlExportRelationDomain` and the `OwlExportRelationNLP` annotations

2.2.4. Exporting NLP Individuals

In order to export the entities of a document as individuals of the NLP ontology, the annotation types created by a pipeline need to be mapped to the related ontology concepts. The *OwlExporterClassNLP* annotation is used to map annotations created by a pipeline to the relevant NLP ontology concepts – see Figure 2.6 for the populated NLP ontology for our example text.

```
AnnotationSet as = (gate.AnnotationSet)bindings.get("ann");
Annotation ann = (gate.Annotation)as.iterator().next();

FeatureMap features = Factory.newFeatureMap();

features.put("className", ann.getType().toString());
features.put("representationId", ann.getId());
features.put("corefChain", null);
features.put("kind", "Class");
```

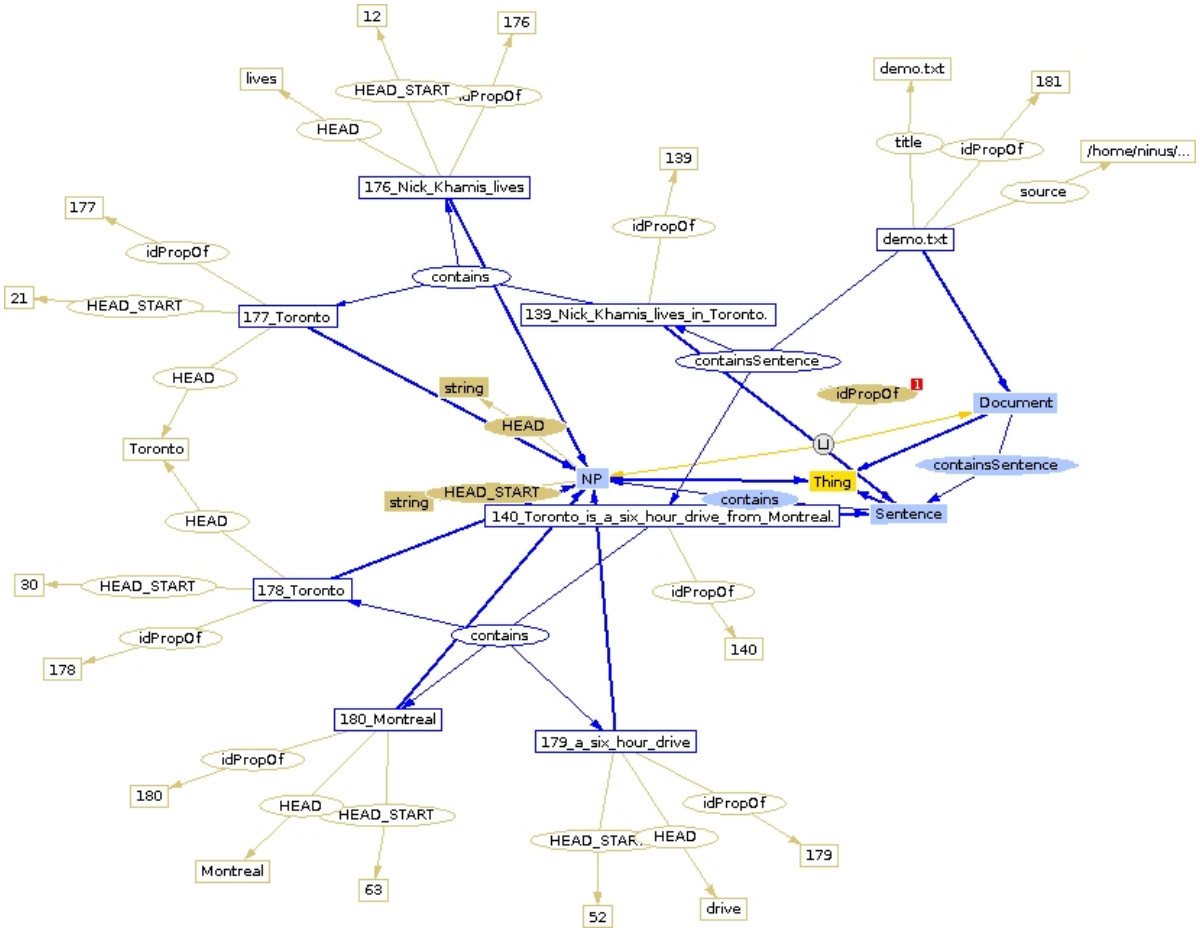


Figure 2.6.: Populated NLP Ontology

```
String in = doc.getContent().getContent(
                ann.getStartNode().getOffset(), ann.getEndNode().getOffset()).toString();

if(ann.getType().toString().compareToIgnoreCase("Document")==0) {
        features.put("instanceName",ann.getFeatures().get("title"));
}
else if (ann.getType().toString().compareToIgnoreCase("Sentence") == 0){
        features.put("content", in);
        features.put("instanceName", in);
        features.put("beginLocation", ann.getStartNode().getOffset());
        features.put("endLocation", ann.getEndNode().getOffset());
}

else if(ann.getType().toString().compareToIgnoreCase("NP") == 0) {
        features.put("instanceName", in);
        features.put("HEAD", ann.getFeatures().get("HEAD"));
        features.put("HEAD_END", ann.getFeatures().get("HEAD_END"));
        features.put("HEAD_START", ann.getFeatures().get("HEAD_START"));
}

outputAS.add(as.firstNode(), as.lastNode(),"OwlExportClassNLP",features);
```

Nick Khamis lives in Toronto. Toronto is a six hour drive from Montreal.

Type	Start	End	Id	Features
Document	0	73	81	{source=/home/ninus/Repository/durm/Applications/OwlExporterDemo/corpora/demo.txt, title=demo.txt}
OwlExportClassNLP	0	73	87	{className=Document, corefChain=[null], instanceName=demo.txt, kind=Class, representationId=81}
Sentence	0	29	39	{}
OwlExportClassNLP	0	29	86	{beginLocation=0, endLocation=29, className=Sentence, kind=Class, corefChain=[null], instanceName=Nick Khamis lives in Toronto. representationId=39}
Sentence	30	72	40	{}
OwlExportClassNLP	30	72	90	{beginLocation=30, endLocation=72, className=Sentence, kind=Class, corefChain=[null], instanceName=Toronto is a six hour drive from Montreal., representationId=40}
NP	0	17	76	{HEAD=lives, HEAD_END=17, HEAD_START=12, MOD=Nick Khamis, MOD_END=11, MOD_START=0}
OwlExportClassNLP	0	17	88	{HEAD=lives, HEAD_END=17, HEAD_START=12, className=NP, kind=Class, corefChain=[null], instanceName=Nick Khamis lives, representationId=76}
NP	21	28	77	{HEAD=Toronto, HEAD_END=28, HEAD_START=21}
OwlExportClassNLP	21	28	89	{HEAD=Toronto, HEAD_END=28, HEAD_START=21, className=NP, kind=Class, corefChain=[null], instanceName=Toronto, representationId=77}
NP	30	37	78	{HEAD=Toronto, HEAD_END=37, HEAD_START=30}
OwlExportClassNLP	30	37	91	{HEAD=Toronto, HEAD_END=37, HEAD_START=30, className=NP, kind=Class, corefChain=[null], instanceName=Toronto. representationId=78}
NP	41	57	79	{DET=a, DET_END=42, DET_START=41, HEAD=drive, HEAD_END=57, HEAD_START=52, MOD=six hour, MOD_END=51, MOD_START=43}
OwlExportClassNLP	41	57	92	{HEAD=drive, HEAD_END=57, HEAD_START=52, className=NP, kind=Class. corefChain=[null], instanceName=a six hourdrive.representationId=79}

Figure 2.7.: Example Annotations for NLP Ontology Export

2.2.5. Exporting NLP Datatype Relationships

Exporting NLP *datatype properties* is similar to exporting domain *datatype properties*. The only difference being the creation of the *property* in the NLP ontology with a name that matches that of the *feature* of a given annotation that needs to be exported.

Nick lives in Toronto.

Type	Start	End	Id	Features
OwlExportClassNLP	21	28	89	{HEAD=Toronto, HEAD_END=28, kind=Class, corefChain=[null], instanceName=Toronto, HEAD_START=21, representationId=77}

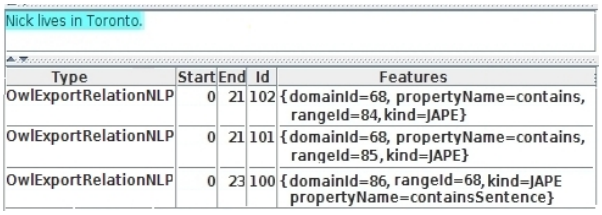

```
1 features.put("HEAD",
2         ann.getFeatures().get("HEAD"));
```

For instance, continuing with the `NP` example Jape grammar created previously, for the `HEAD` information to be exported as a *datatype property* relationship in the NLP ontology, we created the *datatype property* `HEAD` that has `NP` as the *domain* and *xsd:string* as the range. And using the features created by the Jape grammar shown above, the `HEAD` *datatype property* relationship is created.

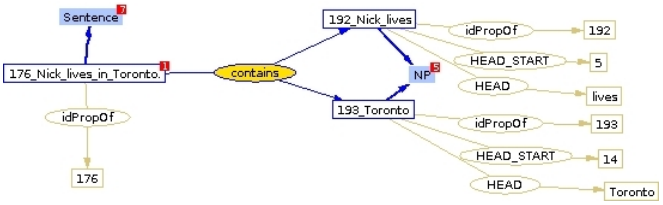
Note: You must ensure the annotation set specified in OwlExporter's `inputASName` run-time parameter contains both the `OwlExportClassNLP` and all the original annotations whose IDs are referenced by the `representationId` feature for datatypes to be exported.

2.2.6. Exporting NLP Object Property Relationships

The process of exporting NLP *object property* relationships is similar to exporting domain *object property* relationships. The name of the *property* in the NLP ontology must match the *propertyName* feature of the `OwlExportRelationNLP` annotation. Additionally, the *domain* of the *datatype property* must match the annotation type indicated using the *domainId* feature, and the range of the *datatype property* must match the annotation type indicated using the *rangeId*.


Nick lives in Toronto.

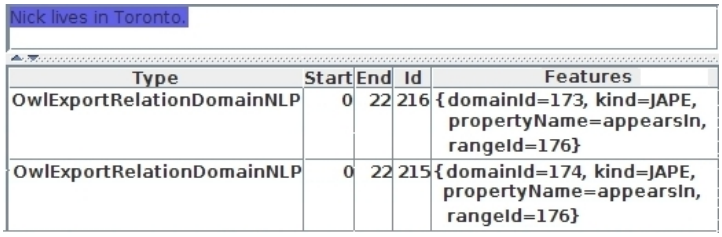
Type	Start	End	Id	Features
OwlExportRelationNLP	0	21	102	{domainId=68, propertyName=contains, rangeId=84, kind=JAPE}
OwlExportRelationNLP	0	21	101	{domainId=68, propertyName=contains, rangeId=85, kind=JAPE}
OwlExportRelationNLP	0	23	100	{domainId=86, rangeId=68, kind=JAPE propertyName=containsSentence}



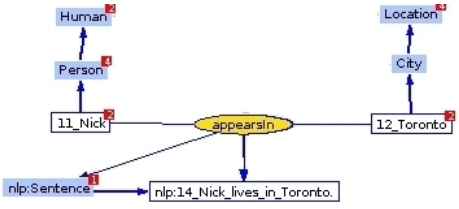

```
gate.AnnotationSet sentAS =
                (gate.AnnotationSet)bindings.get("ann");
gate.Annotation sentAnn =
                (gate.Annotation)sentAS.iterator().next();
AnnotationSet npAS =
inputAS.getContained(sentAS.firstNode().getOffset(),
                     sentAS.lastNode().getOffset()).get("NP");
String propertyName="contains";

for(Annotation a : npAS) {
        gate.FeatureMap features = Factory.newFeatureMap();
        features.put("propertyName",propertyName);
        features.put("domainId", sentAnn.getId());
        features.put("rangeId",a.getId());
        features.put("kind", "JAPE");

        outputAS.add(npAS.firstNode(), npAS.lastNode(),
                     "OwlExportRelationNLP",features);
}
```

2.2.7. Exporting Object Property Relationships across Domain and NLP Ontologies

The OwlExporter is also capable of exporting relationships between the domain and NLP ontologies. Using the demo pipeline, we create an *object property* relationship `appearsIn` that links the domain entities to the sentences they appear in. We begin by modifying the domain ontology to import the NLP ontology. This enables us to create the `appearsIn` relationship in the domain ontology that has `Person`, `Location`, and `Organization` from the domain ontology as the `domain` and `Sentence` from the NLP ontology as the range.


Nick lives in Toronto.

Type	Start	End	Id	Features
OwlExportRelationDomainNLP	0	22	216	{domainId=173, kind=JAPE, propertyName=appearsIn, rangeId=176}
OwlExportRelationDomainNLP	0	22	215	{domainId=174, kind=JAPE, propertyName=appearsIn, rangeId=176}




```
gate.AnnotationSet sentAS =
        (gate.AnnotationSet)bindings.get("ann");
gate.Annotation sentAnn =
        (gate.Annotation)sentAS.iterator().next();
AnnotationSet domAS =
```

```
6         inputAS.getContained(sentAS.firstNode().getOffset(),
7         sentAS.lastNode().getOffset()).get("MentionDomain");
8 gate.Annotation domAnn =
9         (gate.Annotation)domAS.iterator().next();
10
11 String propertyName="appearsIn";
12
13 for(Annotation a : domAS) {
14 gate.FeatureMap features = Factory.newFeatureMap();
15 features.put("propertyName",propertyName);
16 features.put("domainId", a.getFeatures().
17                         get("representationId"));
18 features.put("rangeId",sentAnn.getFeatures().
19                         get("representationId"));
20 features.put("kind", "JAPE");
21
22 outputAS.add(sentAS.firstNode(), sentAS.lastNode(),
23                 "OwlExportRelationDomainNLP",features);
24 }
```

Chapter 3.

OwlExporter Reference

In this chapter, we cover some advanced use cases of the OwlExporter.

3.1. OwlExporter Run-time Parameters

Table 3.1 documents all run-time parameters of the OwlExporter.

Run-time Parameters	Type	Required	Description
corefChainList	ArrayList		The set of coreference chains in the text corpus.
debugFlag	Boolean	•	Run the OwlExporter with or without the debugger. When set to true the OwlExporter prints various messages during different processing stages.
exportDomainOntology	URL	•	The location and filename of the outputted domain ontology.
exportNLP	boolean	•	Export an NLP ontology or not.
exportNLPOntology	URL	•	The location and filename of the outputted NLP ontology.
importDomainOntology	URL	•	The path of the file that contains the pre-existing domain ontology that the OwlExporter will use to populate and create the `exportDomainOntology` output ontology.
importNLPOntology	URL	•	The path of the file that contains the pre-existing NLP ontology that the OwlExporter will use to populate and create the `exportNLPOntology` output ontology.
inputASName	String		The processing annotation set where the OwlExporter will look for the `OwlExportClass` and `OwlExportRelation`.
multiOwlExport	Boolean	•	Output a single Owl file or multiple Owl Files that are all imported by a base output ontology.

Table 3.1.: Summary of OwlExporter Run-time Parameters

3.2. Advanced Features

3.2.1. Document Annotation

In order for the OwlExporter to be aware of the documents that were already processed, a `Document` concept can be created in either the domain or NLP ontology that contains

document instances that store the information of the documents that have been processed. For the exact location of the document, a `sourceUrl` *datatype* property is created that stores the *url* location of the processed document. In Figure 3.1, we show an example of a JAPE grammar that creates a `Document` annotation that includes the required features shown below:

title: Used to store the name of the document

sourceUrl: Used to store the *source url* of the document

```
Phase: mention_doc_info
Input: Token
Options: control = Once

Rule: mention_doc_info

({Token})
:ann
-->
{
        try
        {
                FeatureMap features = Factory.newFeatureMap();

                features.put("title",doc.getName());
                features.put("sourceUrl",doc.getSourceUrl().getFile());
                outputAS.add((long) 0,doc.getContent().size(),
                                "Document",features);
        }
        catch(InvalidOffsetException ioEX) {
                System.out.println(ioEX);
        }
}
```

Figure 3.1.: An Example of the Doc Info Grammar

Note: The OwlExporter uses the `title` and `sourceUrl` information to track which documents have already been processed. If a document has already been processed, a warning message will be displayed.

We modified the NLP Ontology by adding a `Document` concept that the `Document` annotations in the GATE application will be exported to as *individuals* of the ontology.

Note: The `Document` instance can be exported in either the Domain or NLP ontology depending on the the design of the ontology.

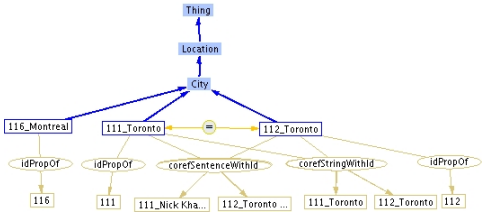
3.2.2. Exporting Coreference Chains

The OwlExporter also supports modelling entities that reappear in different parts of a corpus, and that are linked together using *coreference chains*. For example, a coreferencer such as the one described by (Witte and Bergler, 2003) can identify the nominal and pronominal coreferences between entities and create chains that link them together. When the coreferencer identifies entities of a corpus as being part of the same referent or representative an annotation (for example, `Lookup Chain`) is created that contains a list of entities that make up a coreference. The `OwlExportClassDomain` and `OwlExportClassNLP` annotations accept a `corefChain` feature that contains the ID of the *coreference chain*.

```
AnnotationSet chain = annotations.get("Lookup Chain");
Annotation ann = (gate.Annotation)as.iterator().next();
FeatureMap features = ann.getFeatures();

```

Nick Khamis lives in Toronto. Toronto is a six hour drive from Montreal.

Type	Start	End	Id	Features
OwlExportClassDomain	21	28	174	{className=City, corefChain=167, instanceName=Toronto, kind=Class, majorType=location, minorType=city, representationId=114}
OwlExportClassDomain	30	37	175	{className=City, corefChain=167, instanceName=Toronto, kind=Class, majorType=location,minorType=city, representationId=115}




```
5  if(chain==null||chain.isEmpty())
6  {
7       System.out.println("chain empty");
8  }
9  Integer chainId = null;
10
11 Iterator chainIt = chain.iterator();
12 while(chainIt.hasNext())
13 {
14      Annotation chainAnn = (Annotation)chainIt.next();
15      if(((ArrayList)(chainAnn.getFeatures().
16                  get("IDs"))).contains(ann.getId()))
17      chainId = chainAnn.getId();
18 }
19
20 features.put("corefChain", chainId);
```

By including the `corefChain` feature the OwlExporter creates:

The corefSentenceWithId object property relationship: The corefSentenceWithId relationship associates the referent in a chain with the sentences containing the occurrences of the coreferences.

The corefStringWithId relationship: The corefStringWithId relationship ties the referent to the multiple occurrences of its coreferences.

sameAs relationship: The OwlExporter establishes the links between the individuals in the ontology using the symmetric, transitive and reflexive *owl:sameAs* property (Franz Baader et al., 2007). This allows the related individuals to be classified by an OWL reasoner as being equivalent.

Chapter 4.

OwlExporter Implementation Notes

The OwlExporter is implemented using the Protégé 3.4.2 OWL-API, and the GATE 7 Embedded class library. *Note:* At present, the installation has been tested under Linux and Windows.

4.1. Compilation

In order to compile the OwlExporter:

- Modify "build.properties" in *OWLEXPORTER_HOME* to point to the correct *GATE_HOME* directory on your local machine.

Next, you will need to compile the OwlExporter to include the Protégé libraries needed to run the OwlExporter.

Compiling the OwlExporter:

1. `cd` to OWLEXPORTER_HOME
2. run `ant jar`

After performing the steps mentioned above, a Java Archive called `OwlExporter.jar` is built. This will contain the binaries needed to include the OwlExporter as a CREOLE PR within GATE.

4.2. Importing the OwlExporter

In order to run the OwlExporter, you first need to import the PR into GATE. Plugins are added to GATE using the "Plugin Management Console". The console can be accessed by either selecting "File - Manage CREOLE plugins" from the menu or the "Manage CREOLE plugins" toolbar item.

When in the "Plugin Management Console" screen, click on the "Add a new CREOLE Repository" button and using the "Open File" dialogue box navigate to the directory containing the `OwlExporter.jar` file.

After selecting the directory that contains the jar file, the "OwlExporter" plugin will be added to the list of "Known CREOLE Directories". To begin working with the OwlExporter we will also need to select the "Load now" and/or the "Load Always" options. After completing the

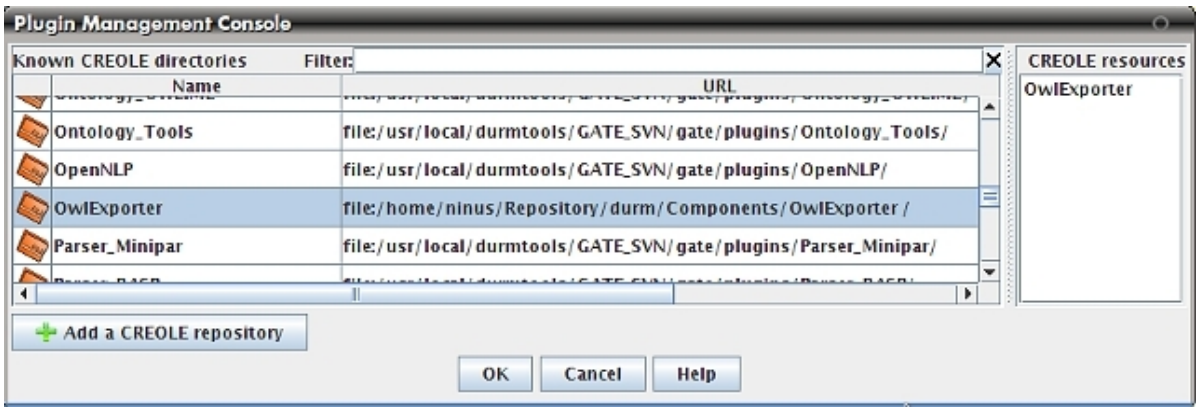


Figure 4.1.: GATE Known CREOLE Window

previous steps, click on the "OK" button in order for the OwlExporter to be successfully added to the GATE Developer.

Note: If the installation of the OwlExporter plug-in was unsuccessful, the stack trace displaying the nature of the error can be viewed in the "Messages" tab of the GATE Developer GUI.

4.3. Developer Notes

In this section, we discuss the various requirements taken into consideration when implementing the OwlExporter. Please not that this is not a formal representation of the OwlExporter implementation.

4.3.1. Duplicate Instances

The OwlExporter exports all *OwlExportClass* annotations from multiple documents and thus there exists the possibility that multiple instances of the same type get exported. We handle this by underscoring the id of the instance (id_x), and we also use owl:sameAs to mark all occurrences of x the same.

4.3.2. Annotation Types versus Annotation Sets

As explained earlier, the OwlExporter only looks for two annotation types, called *OwlExportClassDomain/NLP* and *OwlExportRelationDomain/NLP*, to create *instances* and *relationships* expressed by *object properties*, respectively. Handling a temporary annotation makes exporting entities much simpler than, for example, using an ontology-aware JAPE transducer that requires entire input annotation sets. The UI gets simpler because the user doesn't have to use the GATE UI to add the relevant annotation sets. Authors of components have total flexibility concerning what is exported to the ontology, because only after all their analysis steps are done, they decide which instance (annotation) of their annotation set should become the representation, i.e., the exported individual.

OwlExporter Appendix

A. Main Multi-Phased Jape

```
MultiPhase: Mention
Phases:
docinfo
mention_map_domain_entities
mention_map_nlp_entities
mention_map_domain_relation
mention_map_nlp_relation
mention_map_domain_nlp_relation
```

B. Doc Info Jape

```
Phase: docInfo
Input: Token
Options: control = Once

Rule: docInfoRule

({Token})
:ann
-->
{
        try
        {
                FeatureMap features = Factory.newFeatureMap();

                features.put("source",doc.getSourceUrl().getFile());
                features.put("title",doc.getName());
                outputAS.add((long) 0,doc.getContent().size(),
                                "Document",features);
        }
        catch(InvalidOffsetException ioEX) {
                System.out.println(ioEX);
        }
}
```

C. Mention Map Domain Entities

```
Phase: mention_map_domain_entities
Input: Document Sentence Lookup
Options: control = all  debug = true

Rule: mention_map_domain_entities
(
        {Lookup}
)
:ann
-->
{
```

```
        try     {
                AnnotationSet as = (gate.AnnotationSet)bindings.get("ann");
                Annotation ann = (gate.Annotation)as.iterator().next();
                FeatureMap features = ann.getFeatures();

                if(ann.getFeatures().get("majorType").toString().compareToIgnoreCase("person_first") == 0) {
                        features.put("className", "Person");
                        features.put("hasGender", ann.getFeatures().get("minorType").toString());
                }

                if(ann.getFeatures().get("majorType").toString().compareToIgnoreCase("location") == 0) {
                        String s = ann.getFeatures().get("minorType").toString();
                        features.put("className", Character.toUpperCase(s.charAt(0)) + s.substring(1));
                }

                if(ann.getFeatures().get("majorType").toString().compareToIgnoreCase("person_first") == 0 ||
                        ann.getFeatures().get("majorType").toString().compareToIgnoreCase("location") == 0) {
                        String in = doc.getContent().getContent(
                                        ann.getStartNode().getOffset(), ann.getEndNode().getOffset()).toString();
                        features.put("instanceName", in);
                        features.put("representationId", ann.getId());
                        features.put("corefChain", null);
                        features.put("kind", "Class");
                        outputAS.add(as.firstNode(), as.lastNode(), "OwlExportClassDomain", features);

                }

        }
        catch(Exception e){
                e.printStackTrace();
        }
}
```

D. Mention Map NLP Entities

```
Phase: mention_map_nlp_entities
Input: Document Sentence NP
Options: control = all

Rule: mention_map_nlp_entities
(
        {Document} |
        {Sentence} |
        {NP}
):ann
-->
{
        try {
                AnnotationSet as = (gate.AnnotationSet)bindings.get("ann");
                Annotation ann = (gate.Annotation)as.iterator().next();

                FeatureMap features = Factory.newFeatureMap();

                features.put("className", ann.getType().toString());
                features.put("representationId", ann.getId());
                features.put("corefChain", null);
                features.put("kind", "Class");

                String in = doc.getContent().getContent(
                                ann.getStartNode().getOffset(), ann.getEndNode().getOffset()).toString();

                if(ann.getType().toString().compareToIgnoreCase("Document")==0) {
                        features.put("instanceName",ann.getFeatures().get("title"));
                }
                else if (ann.getType().toString().compareToIgnoreCase("Sentence") == 0){
                        features.put("content", in);
                        features.put("instanceName", in);
                        features.put("beginLocation", ann.getStartNode().getOffset());
                        features.put("endLocation", ann.getEndNode().getOffset());
                }

                else if(ann.getType().toString().compareToIgnoreCase("NP") == 0) {
                        features.put("instanceName", in);
```

```
                        features.put("HEAD", ann.getFeatures().get("HEAD"));
                        features.put("HEAD_END", ann.getFeatures().get("HEAD_END"));
                        features.put("HEAD_START", ann.getFeatures().get("HEAD_START"));
                }

                outputAS.add(as.firstNode(), as.lastNode(),"OwlExportClassNLP",features);
        }
        catch(Exception e)
        {
                e.printStackTrace();
        }
}
```

E. Mention Map Domain Relation

```
Phase: mention_map_domain_relation
Input: Token MentionDomain
Options: control = all

Rule: Person_livesIn_Place
(({MentionDomain.className == "Person"}):dom)
({Token.string == "lives"} {Token.string == "in"}): rel
(({MentionDomain.className == "City"}):ran)
-->
{
        gate.AnnotationSet domain = (gate.AnnotationSet)bindings.get("dom");
        gate.Annotation domainAnn = (gate.Annotation)domain.iterator().next();

        gate.AnnotationSet range = (gate.AnnotationSet)bindings.get("ran");
        gate.Annotation rangeAnn = (gate.Annotation)range.iterator().next();

        gate.AnnotationSet rel = (gate.AnnotationSet)bindings.get("rel");
        gate.Annotation relAnn = (gate.Annotation)rel.iterator().next();

        String propertyName="livesIn";

        gate.FeatureMap features = Factory.newFeatureMap();
        features.put("propertyName",propertyName);
        features.put("domainId",domainAnn.getFeatures().get("representationId"));
        features.put("rangeId",rangeAnn.getFeatures().get("representationId"));
        features.put("kind", "JAPE");

        outputAS.add(rel.firstNode(), rel.lastNode(),"OwlExportRelationDomain",features);
}
```

F. Mention Map NLP Relation

```
Phase: mention_map_nlp_relation
Input: Document Sentence Token Mention
Options: control = all

Rule: Document_containsSentence_Sentence
(
{Document}
)
:ann
-->
{
        gate.AnnotationSet docAS = (gate.AnnotationSet)bindings.get("ann");
        gate.Annotation docAnn = (gate.Annotation)docAS.iterator().next();
        AnnotationSet sentAS = inputAS.getContained(docAS.firstNode().getOffset(),
                                        docAS.lastNode().getOffset()).get("Sentence");
        gate.Annotation sentAnn = (gate.Annotation)sentAS.iterator().next();

        String propertyName="containsSentence";

        for(Annotation a : sentAS) {
                gate.FeatureMap features = Factory.newFeatureMap();
```

```
                features.put("propertyName",propertyName);
                features.put("domainId",docAnn.getId());
                features.put("rangeId",a.getId());
                features.put("kind", "JAPE");

                outputAS.add(docAS.firstNode(), docAS.lastNode(),"OwlExportRelationNLP",features);
        }
}

Rule: Sentence_contains_NP
(
{Sentence}
)
:ann
-->
{
        gate.AnnotationSet sentAS = (gate.AnnotationSet)bindings.get("ann");
        gate.Annotation sentAnn = (gate.Annotation)sentAS.iterator().next();
        AnnotationSet npAS = inputAS.getContained(sentAS.firstNode().getOffset(),
                                                  sentAS.lastNode().getOffset()).get("NP");
        String propertyName="contains";

        for(Annotation a : npAS) {
                        gate.FeatureMap features = Factory.newFeatureMap();
                        features.put("propertyName",propertyName);
                        features.put("domainId", sentAnn.getId());
                        features.put("rangeId",a.getId());
                        features.put("kind", "JAPE");

                        outputAS.add(npAS.firstNode(), npAS.lastNode(),"OwlExportRelationNLP",features);
        }
}
```

G. Mention Map Domain NLP Relation

```
Phase: mention_map_domain_nlp_relation
Input: MentionNLP
Options: control = all

Rule: Domain_appearsIn_Sentence
(
{MentionNLP.className == "Sentence"}
)
:ann
-->
{
        gate.AnnotationSet sentAS = (gate.AnnotationSet)bindings.get("ann");
        gate.Annotation sentAnn = (gate.Annotation)sentAS.iterator().next();
        AnnotationSet domAS = inputAS.getContained(sentAS.firstNode().getOffset(),
                                        sentAS.lastNode().getOffset()).get("MentionDomain");
        gate.Annotation domAnn = (gate.Annotation)domAS.iterator().next();

        String propertyName="appearsIn";

        for(Annotation a : domAS) {
                gate.FeatureMap features = Factory.newFeatureMap();
                features.put("propertyName",propertyName);
                features.put("domainId", a.getFeatures().get("representationId"));
                features.put("rangeId",sentAnn.getFeatures().get("representationId"));
                features.put("kind", "JAPE");

                outputAS.add(sentAS.firstNode(), sentAS.lastNode(),"OwlExportRelationDomainNLP",features);
        }
}
```

Bibliography

F. Baader et al. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003. ISBN 9780521781763.

Philipp Cimiano. *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387306323.

Diego Calvanese Franz Baader, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2007.

V. Haarslev and R. Möller. RACER System Description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy*, pages 701–705. Springer-Verlag, 2001.

Atanas Kiryakov, Damyan Ognyanov, and Dimitar Manov. OWLIM – A Pragmatic Semantic Repository for OWL. In Mike Dean, Yuanbo Guo, Woochun Jun, Roland Kaschek, Shonali Krishnaswamy, Zhengxiang Pan, and Quan Z. Sheng, editors, *WISE Workshops*, volume 3807 of *Lecture Notes in Computer Science*, pages 182–192. Springer, 2005. ISBN 3-540-30018-X. URL http://dblp.uni-trier.de/db/conf/wise/wise2005w.html#KiryakovOM05.

E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, June 2007. ISSN 15708268. doi: 10.1016/j.websem.2007.03.004. URL http://dx.doi.org/10.1016/j.websem.2007.03.004.

Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006*, pages 292–297. Springer, 2006.

René Witte and Sabine Bergler. Fuzzy Coreference Resolution for Summarization. In *Proceedings of 2003 International Symposium on Reference Resolution and Its Applications to Question Answering and Summarization (ARQAS)*, pages 43–50, Venice, Italy, June 23–24 2003. Università Ca' Foscari. http://www.semanticsoftware.info/biblio/fuzzy-coreference-resolution-summarization.

René Witte, Ninus Khamis, and Juergen Rilling. Flexible Ontology Population from Text: The OwlExporter. In *International Conference on Language Resources and Evaluation (LREC)*, Valletta, Malta, 05/2010 2010. ELRA. http://www.semanticsoftware.info/biblio/flexible-ontology-population-text-owlexporter.