www.kconnect.eu

Semantic Annotation Toolkit (Final Version)

Deliverable number	*D1.5*
Dissemination level	*Public*
Delivery date	*25 July 2016*
Status	*Final*
Author(s)	*Genevieve Gorrell, Ian Roberts, Pascal Vaillant, Zoltan Varju, Fredrik Axelsson, Angus Roberts*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644753 (KConnect)

Executive Summary

This deliverable comprises the KConnect GATE[1] application Bio-YODIE, language-configurable code to prepare the extensive resources required by the application at runtime, a toolkit enabling further language customization of the application and a manual providing instructions (Appendix A). It builds on Deliverable 1.1, in which the first version of the language customization toolkit was presented in the context of the Khresmoi application.

Deliverable 1.1 provided a) tools to support porting the basic linguistic processing (sentence splitting, part of speech tagging, etc.) and b) terminology lookup components of the Khresmoi application to other languages. The tools for porting basic linguistic components remain part of this final version of the customization toolkit, and instructions for their use provided in D1.1 are included in the manual delivered here in Appendix A. Terminology lookup components (gazetteers) have been superseded by new work.

Bio-YODIE provides more sophisticated disambiguation of mentions that match multiple entities than the Khresmoi application did, and for this reason it is a larger, more complex application, requiring various informational resources that can be used to select the right interpretation. For this reason, preparation of resources from UMLS is fully scripted, including gazetteer preparation, and it has been possible to parameterize this for language. This is the reason why the earlier work on gazetteer porting included in D1.1 has been superseded.

Deliverable 1.1 also provided some preliminary evaluation for the French case study, extended here, and reports from work porting to Swedish and Hungarian, which again we follow up. Deliverable 1.1 indicated that this deliverable would also include tools for configuring indexes of annotations created by the resulting applications. These are delivered in the form of a Mimir template that can be used as-is. This is included in Appendix B.

1 https://gate.ac.uk/

Table of Contents

Executive Summary 2
1 Software Description 4
2 French Case Study 5
2.1 System overview 5
2.2 Evaluation 6
3 Hungarian Application 7
3.1 Components adapted 7
3.2 Next steps 7
4 Swedish Application 8
4.1 Adapting basic text processing 8
4.2 Terminology dictionaries and heuristics 8
4.3 Other modifications 9
References 9
Appendix A—Bio-YODIE Language Adaptation Manual 10
Preparation of Resources 10
Toolkit for Language Adaptation 12
Tokeniser, POS tagger, morphological analysis 12
Stopword list 13
GATE Simple Manual Annotator 14
The Bio-YODIE Application 18
Getting Bio-YODIE and Providing the Resources Directory 18
Preprocessing Sub-Application 19
Gazetteer Sub-Application 19
Main Application 20
Appendix B—Mimir Template 21

1 Software Description

Accompanying this report are the following;

- yodie-bio-preparation – the repository containing language-parameterized scripts required to prepare resources for the Bio-YODIE GATE[2] application from UMLS, as well as some shared resources. This is available to download at http://www.dcs.shef.ac.uk/~genevieve/D1.5/yodie-bio-preparation.zip
- Language adaptation toolkit, available at http://www.dcs.shef.ac.uk/~genevieve/D1.5/toolkit.zip, including:
  - annotate-stopwords – the tool for creating morphologically aware GATE stopword gazetteers, as described in D1.1
  - gate-SimpleManualAnnotator – an annotation tool created for KConnect facilitating the preparation of the corpora required to create disambiguation resources in the target language, in the form of annotated GATE documents
- yodie-pipeline – the repository containing the base GATE application, including a script to facilitate extension to other languages. This can be downloaded at http://www.dcs.shef.ac.uk/~genevieve/D1.5/yodie-pipeline.zip.

2 https://gate.ac.uk/

2 French Case Study

Producing a French version of the system formed the test case for developing the machinery presented in this deliverable, as was the case in the first version of the toolkit, delivered as D1.1. The full UMLS was downloaded, and the scripts delivered here were used to subset it for French. The process of adding a new language to Bio-YODIE was tested on French.

2.1 System overview

A stopword list was taken from Snowball[3], and was found comprehensive enough that morphologically aware stopword matching was unlikely to be of benefit, so was used as-is. GATE's French tokeniser and sentence splitter were used. GATE's generic French gazetteer was also included in case it might be found beneficial in future, though that gazetteer doesn't contain the medical entities we are looking for in this application. Likewise Stanford's French POS tagger was included though not currently used by the system.

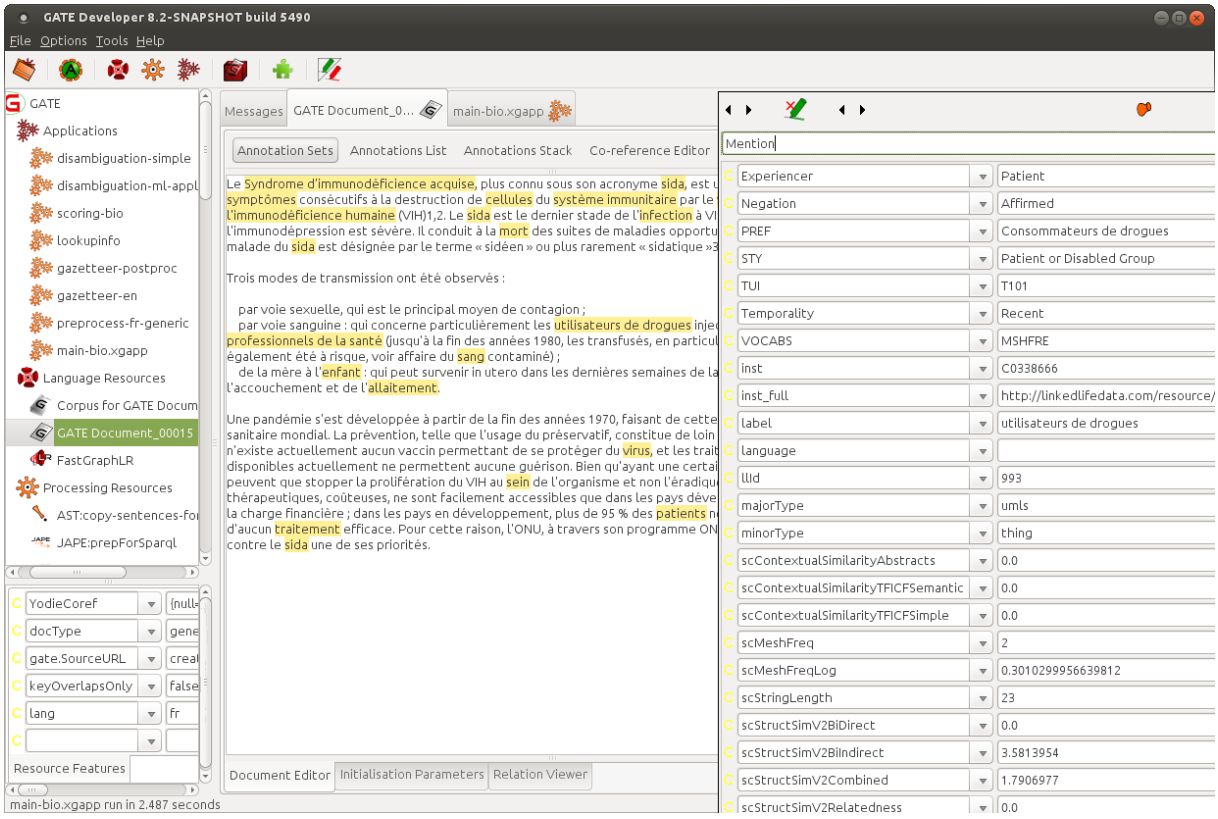


Figure 1: French Bio-YODIE

Figure 1 shows a screenshot of French Bio-YODIE, having been run on an example of French medical-domain text. It is apparent that “utilisateurs de drogues” has been annotated with the CUI C0338666 from the vocabulary MESHFRE (French Medical Subject Headings). “Consommateurs de drogues” is a preferred label for this CUI.

3 https://github.com/snowballstem

2.2 Evaluation

As in deliverable 1.1, we present some preliminary evaluation figures here on a subset of the QUAERO corpus. The figures shown are the average of strict (exact span matches only considered correct) and lenient results. The figures show a slight improvement on those reported in D1.1, suggesting some contribution from the disambiguation technology, though only minor. This might perhaps result from the inclusion of the MeSH CUI frequency resource, which although compiled on English data, is likely to have some utility for other languages. However, most disambiguation resources aren't included for French, so it is unsurprising that the result is very similar to that shown in D1.1, since little has been done that might improve the result. We suggest that the reason recall is low is that the French subset of UMLS is more sparse than the English, and that supplementing with additional vocabularies, in particular for drug names, would be the most important next step in improving the result.

Precision	Recall	F1
0.72	0.24	0.36

Table 1: Preliminary evaluation of French Bio-YODIE on the QUAERO subset

3 Hungarian Application

The review of the available resources for the Hungarian language is described in D1.1. Since a new version of magyarlanc, the wrapped morphological analyzer and dependency parser, has been released, the Hungarian pipeline has been updated with it.

3.1 Components adapted

The new version of magyarlanc (3.0) provides tools for basic NLP tasks (sentence splitting, tokenisation, lemmatisation, part-of-speech tagging, and dependency parsing)[4]. The new version uses the purePOStagger[5] and it now uses the Penn Treebank Tagset instead of the KR codes.

Before adapting the new version into the GATE framework, the original version of magyarlanc has been split into four main modules, namely sentence splitter, tokenizer, lemmatizer and part-of-speech tagger. The dependency parser has not been modularized yet; it is very unlikely that KConnect services will use it.

As in the previous version, each component extends AbstractLanguageAnalyzer and implements GateProcessingResource from the GATE API, and the string constants follow the naming conventions of ANNIE.

A basic medical gazetteer was generated from the Hungarian MeSH and has been integrated into the Hungarian pipeline.

Currently, the Hungarian pipeline is using a stopword list based on the word frequency lists of the Hungarian National Corpus (Magyar Nemzeti Szövegtár)[6]. The basic stopword list contains only function words; the extended version contains the most frequent words from other categories too.

3.2 Next steps

As described in Section 4.3 of D3.2 (Prototypes of Vertical Search Solutions), the Hungarian pipeline works well on Hungarian data, but it produces some systematic mistakes. An evaluation study has been designed and started to understand the issues. The above-mentioned report describes the evaluation study in Section 4.1.

Adapting the Context algorithm for determining negation, experiencer, and temporal status described in Harkema et al (2009) for the Hungarian language is planned.

4 http://www.inf.u-szeged.hu/rgai/magyarlanc

5 https://github.com/ppke-nlpg/purepos

6 http://mnsz.nytud.hu/

4 Swedish Application

In Deliverable D1.1, components and resources available for Swedish were described, along with the first adaptation of the toolkit to parse Swedish text. This section describes the final adaptation of this version of the toolkit to Swedish.

4.1 Adapting basic text processing

The Swedish basic language pipeline makes use of the following components, which were described in deliverable D1.1. In this version of pipeline the part-of-speech-tagger has been removed.

- Swedish sentence splitter in openNLP-plugin,
- Swedish Tokenizer in openNLP-plugin,
- Swedish Snowball stemmer,
- Findwise Swedish lemmatizer,
- Findwise Swedish Compound word splitter,
- Basic medical named entity recognition based on gazetteers with dictionaries from Snomed CT SE, SweMeSH and ICD-10.

All components but the lemmatizer and compound word splitter are available as part of existing GATE plugins. The lemmatizer returns the base form of a word based on a morphological lexicon.

Based on a morphological lexicon, the Findwise Swedish compound word splitter knows the different forms of a word which can start and end a compound word. Given a string representation of a word the components provide possible splittings of the given word, with the lemma for the separate parts of the word.

The stop word list is based on the snowball stemmer list[7], with minor adaptions.

4.2 Terminology dictionaries and heuristics

The terminology dictionaries for Swedish are constructed by extracting Swedish labels directly from the KConnect knowledge base. The label names are extracted along with their corresponding UMLS CUIs and TUIs, in the tab separated format required by the adaptation toolkit. Two lists are extracted one containing labels from Drugbank and one containing all other labels from UMLS.

Synonym lists for MeSH terms have been extracted from the SweMeSH distribution with UMLS CUI and TUI to their corresponding term.

The dictionaries are processed by the basic application and create case insensitive gazetteers from the following word features:

- Lemmatisation
- Stemming
- Lemma of last part in the compound word

The lemmatisation and stemming are to find different inflections of the labels mentioned in the text. The lemma of the compound word is for finding a broader concept when a specific is missing in the semantic repository, i.e. « cancerdoktor » would be annotated as « doktor » if the more specified annotion « cancerdoktor » is missing.

The default disambiguation rules are used to removing overlapping matches and preferring the match with the highest CUI in the case of multiple annotations with the same span.

7 http://snowballstem.org/algorithms/swedish/stop.txt

4.3 Other modifications

In addition to creating the Swedish pipeline with the toolkit, a script is used to clean the pipeline bundle from superfluous files, to minimise the footprint when bundling the processing pipeline in a web application.

References

Harkema, H., Dowling, J.N., Thornblade, T. and Chapman, W.W., 2009. ConText: an algorithm for determining negation, experiencer, and temporal status from clinical reports. *Journal of biomedical informatics*, *42*(5), pp.839-851.

Appendix A—Bio-YODIE Language Adaptation Manual

This manual describes how to port the KConnect medical named entity linking GATE application Bio-YODIE to various languages. The manual is divided into three parts. The first part, "Preparation of Resources", focuses on the preparation of databases, word lists and other components required by the application at runtime. Next, in "Toolkit for Language Adaptation", we acquire/adapt language-specific components such as part of speech tagging for the desired language. Finally, in "The Bio-YODIE Application", we install the base GATE application and configure it to use the resources and subcomponents we have just created for the target language. A basic level of GATE expertise is assumed; GATE tutorials are available on the GATE website[8], as is the latest GATE download.

Preparation of Resources

Several resources are required by the Bio-YODIE application, and our first task is to prepare these resources. These are:

- A language-specific gazetteer containing all the terms required to be recognized in the text;
- A language-specific H2 database containing the entities we wish to recognize. There could be several of these to be matched to a particular label in text. This database indexes them on label. For each entity, concept unique identifier (CUI) and type (TUI) are included, as well as any information that will later be required for disambiguation. For example, a language-independent pre-prepared resource mapping CUIs to occurrence frequencies derived from MeSH terms is included at this stage;
- A database of abstracts in the target language for the entities we want to recognize, that we can use to see whether language relevant to the entity occurs in the context of the mention, thus gauging the likelihood that this is the correct interpretation;
- A graph derived from term co-occurrence data (the "MRCOC" table) that allows us to judge the likelihood of two entities occurring near each other in text using the PageRank algorithms. This is a language-independent resource. It contains a dictionary but that will contain labels for everything in the UMLS subset;
- A graph derived from formal relationships between entities that allows us to judge the likelihood of two entities occurring near each other in text using Milne and Witten's (2008) relatedness metric. Again, this is a language-independent resource, containing no labels.

In order to do this, the following need to be available:

- UMLS[9]. You will need to obtain a license to access UMLS. These are free to obtain, with some reasonable conditions attached. There's no need to subset it for language, though if you are working in English, which is unhelpfully rich and abundant, it might be a good idea to use the NLP subset rather than the complete resource. Subsetting can be done with the

8 https://gate.ac.uk/

9 https://www.nlm.nih.gov/research/umls/licensedcontent/umlsknowledgesources.html

Metamorphosys tool. There is also a possibility to narrow to the NLP subset later, in the scripts;

- The additional table MRCOC[10], which is a large resource containing CUI co-occurrences derived from a publications corpus;
- Structure and resources expected by the scripts, most notably the table with MeSH frequencies by CUI, which are provided as part of the toolkit. You may simply unzip this from the provided archive, yodie-bio-preparation.tgz;
- Scala[11], H2[12] and UKB[13] installed;
- The scripts are shell scripts tested on Linux, though they should work in any Unix-like environment.

Having unzipped yodie-bio-preparation.tgz, you may change into that directory. Your working directory is the top level directory where you have the scripts, for example containing the “bin” and “sql” directories.

The preparation script begins by loading UMLS into a H2 database, from where it can be queried to create the smaller, bespoke resources Bio-YODIE requires at runtime. This is quite a big job. MRCOC will also be added to it, as will the MeSH resource that USFD are sharing. A number of helper tables will also be created in this database. From there, resources required by Bio-YODIE are extracted.

You will need the following directories or symbolic links in your working directory:

- databases -> should point to the directory where you will store your new UMLS H2 database. You will need plenty of space available, maybe around 100GB depending on whether you subsetted UMLS and how many languages you included. (This database won't form part of the Bio-YODIE application resources, which are much smaller.)
- output -> this is the directory that will be populated with the new Bio-YODIE resources. You have been supplied with some contents and structure in this directory already.
- srcs -> should point to where you have your UMLS download, MeSH frequencies etc. It expects a directory called "umls" containing "2015AB" (this is hardcoded in bin/loadUMLS.sh) and directories named Mesh-freq and mrcoc containing the respective files. Again, you have been provided with some structure in this directory to indicate how it should be and include the MeSH frequency resource that USFD is sharing as well as a couple of items that need to be there for historical reasons.
- tmpdata -> this is a temporary space for intermediate files, and just needs to be somewhere you have space.

10 https://mbr.nlm.nih.gov/MRCOC.shtml

11 www.scala-lang.org/

12 http://www.h2database.com/html/main.html

13 http://ixa2.si.ehu.es/ukb/

You can then run bin/all.sh. You can comment this script using "#" to take it a step at a time while you are first getting it working. The slowest parts are loading UMLS and adding the MRCOC table.

You should change the environmental variable LANGS export statement in config/getvars.sh to the appropriate two letter code for the language you want to run prep for, for example "fr". It will default to English. If you want to do multiple languages, separate them with a space. Be sure to put inverted commas around the string, for example;

```
export LANGS="en fr"
```

The list of language codes is visible in config/getvars.sh. This script is also where you can switch to a different type subset if you wish. We include only entities of certain types, such as drugs, anatomical parts etc. in Bio-YODIE, and were we to include all of them, overhead would increase only for us to find many entities in text that we aren't interested in. By default we use the KConnect type set, and you are unlikely to need to change that. You can also opt to impose the NLP subset at this stage, in this script, though note that this is only relevant to English. For English, it is an important way to reduce the number of spurious matches as well as reduce overhead from including many unhelpful entities in the resources, and it is strongly recommended.

The scripts rely on Scala to work. You'll need Scala or a link to Scala in a directory called "yodie-preparation/scala_latest_version" that sits alongside your working directory. You'll need UKB on your class path for the UKB graph creation part to work.

Having run all of the above, you should find that your output directory contains a directory called "databases", one called "ukb" and one called "fastgraph", all containing language-independent resources (although ukb contains a dictionary, it should include all the labels in your UMLS download regardless of language), and one (or several) named after the two letter code of your target language containing language-specific resources, namely the gazetteer, abstracts and entity information databases.

Toolkit for Language Adaptation

The first stage of the application involves annotating the text with basic annotations such as tokens and sentences, which are required by several of the downstream components. This basic linguistic processing is language-specific, and requires adaptation. Furthermore a list of stopwords is made use of, to reduce the number of spurious matches. For example, "no" is far more likely to be the common negator than a reference to nitrous oxide, so simply excluding it is far more likely to help than hinder. In the next two sections we prepare resources that we will then integrate into the main application.

In order to evaluate, as well as adapt and extend the disambiguation resources for the target language, an annotated corpus is required in the target language. In the following section we describe how to use the GATE Simple Manual Annotator, a GATE tool facilitating rapid annotation of a disambiguation corpus.

Tokeniser, POS tagger, morphological analysis

Tokeniser, part of speech (POS) tagger and morphological analysis components closely interact. Tokeniser in particular is required by the pre-processing sub-pipeline of Bio-YODIE, with the other components being of particular value for morphologically rich languages, and though not currently

used by default in English, are candidates for future inclusion. For many western languages the standard GATE Unicode tokeniser will produce sensible results, but if you wish to use some other POS tagger or morphological analyser then the tokenisation may need to be adjusted to match that expected by the tagger. In English, for example, the Hepple POS tagger (HepTag) expects "don't" to be two tokens, "do" and "n't" where the plain Unicode tokeniser creates three, "don", apostrophe, and "t", so the English tokeniser has a post-processing step to correct this.

GATE includes support for several different POS taggers by default:

- GATE's own native HepTag tagger (the Hepple PPOS tagger)
- The OpenNLP maximum entropy POS tagger[14]
- The Stanford log-linear POS tagger[15]

The latter two taggers offer pre-trained models for a number of different languages, as well as training tools to allow you to create your own model from a manually-tagged training corpus.

For morphology, GATE includes a native morphological analyser for English and also support for the Snowball stemmer[16], providing basic stemming in 16 different languages.

Since this sub-pipeline is potentially tightly customised to the target language it is difficult for us to provide tool support for adaptation. If one or other of the components described above are not suitable, then a suitable equivalent for the target language must be found, and wrapped for the GATE API, as described in the GATE user guide[1]. The final pre-processing pipeline must generate the following annotations and features:

- Token: individual words
    - string: the plain string of the token
    - kind: word, number, symbol or punctuation
    - orth: lowercase, upperInitial (first letter uppercase, the rest lowercase), allCaps (all uppercase) or mixedCaps (any other mixture of upper and lower case)
    - category: part-of-speech tag (optional)
    - root: morphological root (optional)
    - stem: stem (optional)
- SpaceToken: spaces between words
- Sentence: sententences, including the terminating punctuation if any

Other annotations such as Split (for the sentence boundaries) as created by the default GATE components are acceptable but not strictly required by downstream components.

Any custom plugins that are not provided with the default GATE distribution should be placed in the top level "plugins" directory in the toolkit template structure.

Integration into the application will be discussed later in this manual, but the principle is to create a new "preprocessing" subpipeline based on the existing English or French ones, and modify it as needed.

Stopword list

The gazetteer sub-application includes a "stopword" list of words that should be ignored when doing dictionary lookups. Typically these are single letter words, closed class words such as prepositions, conjunctions, etc. If a similar list is available for the target language it can be used in the same way. It is strongly recommended to obtain or create a stopword list to reduce spurious matching. The easiest

14 http://opennlp.apache.org

15 http://nlp.stanford.edu/software/tagger.shtml

16 http://snowball.tartarus.org

way to include your stopword list is simply to replace the stopword list "stopwords.lst" in gazetteer-nn (nn indicating the two-letter language code) with the list for your language. The list contains stopwords each on their own separate line. This approach is adequate for English, but for many languages with richer morphology, it makes sense to take a more sophisticated approach. Stopwords will typically be matched using exact string matching, but if a morphological analyser is included in the pre-processing sub-pipeline then you may additionally wish to match stopwords based on their morphological roots.

A tool is provided to take a stopword list and convert it into the form required to do this. The tool is in the form of a command line script which takes its configuration from an XML file. In the "annotate-stopwords" folder of the adaptation toolkit, you will find an example configuration file "config-sample.xml" – make a copy of this file as "config.xml" and edit it appropriately. The configurable options are:

- language: the ISO 639 two letter language code for the target language, "en" for English, "fr" for French, etc.
- input: the path to the stopword list file, e.g. "stopwords.txt"
- preprocessingApplication: the path to the GATE saved application you containing your basic linguistic processing components for your language, which will perform the tokenisation and POS tagging/morphological analysis
- wordAnnotationType: the annotation type representing words, which should be left as the default value "Token" unless you are sure you need to change it
- spaceAnnotationType: the annotation type representing spaces between words, which should be left as the default value "SpaceToken" unless you are sure you need to change it

The remaining "feature" elements should name the annotation features you want to use for matching. Typically the first one will be "string", which denotes exact string matching. If you are using a stemmer or morphological analyser then you should add "root" or "stem" as appropriate.

To construct the morphologically aware gazetteer, you will need to use the pre-processing application that you will shortly construct from components you prepared in the previous section. In the next section, we will talk more about assembling this application, so it might make sense to come back to this part after that. The principle is that you will use this pre-processing application to process your stopword list into morphological roots that can then be used to match against the morphological roots of words in text, to more flexibly identify them as stopwords. Note that in the English pre-processing application, preprocessing-en, a morphological analyser is not by default included. You will need to ensure one is present for your language.

To run the tool, set your GATE_HOME and JAVA_HOME environment variables to the locations where you have installed GATE Developer (8.1) and Java JDK (7 or later), and run the `stopwordsApp.sh` script. This will generate a number of new files in a "resources" folder, and a top level `application.xgapp` file containing a saved GATE application that will match stopwords based on all the features you specified in the configuration. We'll include this functionality in Bio-YODIE in the next section.

GATE Simple Manual Annotator

The GATE Simple Manual Annotator annotation tool allows you to quickly annotate a directory of GATE documents stored locally, and therefore is quick and convenient to use with confidential medical data, where it may not be possible to use web-based tools that require data to be sent over the internet. It is suitable for tasks such as accepting or rejecting automatically generated annotations in text or selecting from several options. It presents each annotation to be reviewed in a clear and simple way with key press options allowing one of a short list of options to be selected quickly, without mouse use, facilitating an annotation rate in the region of one every couple of seconds depending on task complexity. Annotation choices are written back onto the GATE documents.

In order to make use of the tool to create a manually annotated corpus of medical data with entities linked to CUIs, you will need to prepare it by automatically finding named entity mentions in the text and attaching candidate interpretations to them. This can be achieved using the Bio-YODIE application. For corpus preparation, scoring and disambiguation sub-applications of Bio-YODIE may be turned off, as we require only that the spans be found and the candidates be added. Note that in order to present an option to an annotator, a readable label is constructed by Bio-YODIE based on heuristics for selecting the best label that are tailored to English. For languages very different to English, some work may be required at the preparation stage (described above), specifically the SQL script that adds YODIE tables to the UMLS database.

Configuring the Task

Your corpus of GATE documents needs to be prepared with annotations that describe what annotations you want to be presented to the annotator (hereafter referred to as "mentions"), and what options they should be presented with. Three modes are available for you to choose between depending on your task and data. The task is defined in the configuration file, which is passed to the annotation tool on startup, an example of which is provided, and in here you will choose your mode and specify the required options. We begin by listing options that pertain to all three modes, before discussing each mode separately.

Options that need to be set in all modes

- autoadvance - This option specifies whether the tool should automatically move on to the next mention when the user selects an options (quicker) or wait for the user to manually move on to the next mention (which some users may prefer).
- inputASName - The name of the input annotation set from which the mentions and options are to be drawn. For Bio-YODIE, candidates are found in the default annotation set.
- contextType - Each mention is presented to the annotator with some context, to enable them to form a more accurate judgement. Your corpus needs to be prepared with suitable context annotations, such as sentences.
- mentionType - The annotation type of the mentions that will be presented to the annotator to form a judgement about. For Bio-YODIE, use LookupList
- outputASName - Once the annotator has made their choice, an output annotation will be generated recording this. The output annotation set name indicates the annotation set you want these judgements to go into. “Key” is suggested, as this is conventionally where manual annotations go. Alternatively you may wish to use the name of your annotator if you plan on having your data multiple-annotated.
- includeNoneOfAbove - In addition to the choices you provide, do you want the annotator to also see an automatically generated "none of the above" option? This would be used to indicate that the mention is valid, but that it was impossible to choose the correct choice from the ones provided. This is recommended for KConnect disambiguation annotation, since the correct CUI may not be available.
- includeSpurious - In addition to the choices you provide, do you want the annotator to also see an automatically generated "spurious" option? This would be used to indicate that the mention is not a valid example of the type being annotated. For KConnect disambiguation annotation, this is recommended, since it often happens that a mention is found due to having a string match to a CUI of interest, but that mention isn't actually relevant to KConnect.

OPTIONSFROMSTRING

In this mode, the options to be presented are specified as a semicolon-separated list in the configuration file. The output feature is also specified in the configuration file. This option is not likely to be relevant to KConnect work, but is included here for completeness.

- options - The options are listed here, separated by semicolons.
- outputFeat - The output annotations appear in the output annotation set, as described above, and are of the same type as the mention. The choice the user makes appears in the outputFeat, the name of which is specified here.

OPTIONSFROMFEATURE

This option indicates that the choices to be provided to the annotator should be taken from a set provided in a feature on the mentions. This should be a Set object containing String objects. This option is included here for completeness, but this format is not produced by Bio-YODIE.

- optionsFeat - The feature on the mention that contains the choices.
- outputFeat - The feature on the output annotations to contain the choice.

OPTIONSFROMTYPEANDFEATURE

This option indicates that choices are to be found on a separate annotation type. For example, you may have a "Mention" type defining the locations to be annotated, but your options are taken from "Candidate" annotations that are co-located with the mentions.

- optionsType - The annotation type that defines the choice. Bio-YODIE creates “Lookup” annotations for each candidate.
- optionsFeat = The feature indicating the textual string to be presented describing the choice. In our case, Lookup annotations have a feature called "Readable".

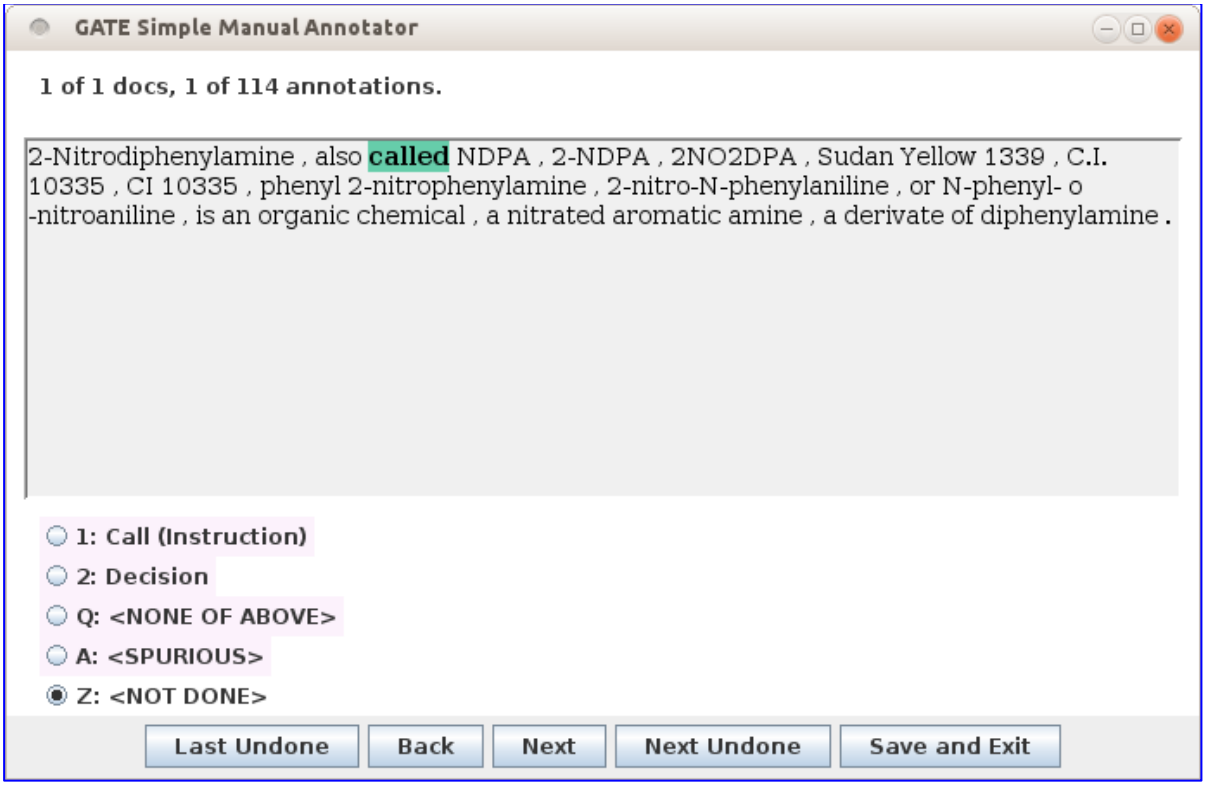
Running the Tool

To run, the annotator requires the GATE jar as well as the GATE lib directory. The following command assumes you have the GATE_HOME environment variable set, giving the location of your GATE installation:

```
java -cp gate-
SimpleManualAnnotator.jar:$GATE_HOME/bin/gate.jar:$GATE_HOME/lib/*
gate.tools.SimpleManualAnnotator <config file> <directory of GATE
documents>
```

A shell script is provided with the distribution.

Usage



The tool will automatically find the next un-done mention in the corpus to begin with. The position we are currently at in the corpus is indicated at the top. In the screenshot, we are on the first mention in the first document.

The mention is highlighted in teal and the context (in the screenshot, it is the sentence) is presented surrounding it.

The choice list is presented below the mention. Numbers are assigned to each choice, and letters are assigned to autogenerated options. This means you can use key presses to annotate quickly. If you want to use the number pad on the right of the keyboard, you need to press "Num Lock" on your keyboard first. You are of course welcome to use the mouse to select your option too.

Navigation options allow you to move forward and back through the mentions, or jump to the last or next un-done mention. You can also use right or down arrows to move to the next mention, or left or up arrows to move to the previous one. If you are using the autoadvance option, when you select a choice the tool will automatically move to the next mention, so you may never need to use the navigation buttons.

A "Save and Exit" button is provided for your reassurance, but in fact closing the window will also save your work. Every time you move to a new document, your work on that document is saved. If your computer crashed, you would lose the work on the current document, but not on completed documents. There is no way to not save your work. However you can quickly remove a large number of annotations by holding down Z with autoadvance switched on. If you are concerned that you might not want to save your work, you are recommended to make a back-up.

The Bio-YODIE Application

Having prepared our resources, we can now turn to the application itself. Bio-YODIE is a biomedical domain variant of a more general application, YODIE, originally designed to link named entities into DBpedia. YODIE exists as a Git repository currently located in Sheffield. You have received it in the form of the file "yodie-pipeline.tgz" for convenience. YODIE is a research system and a work in progress, so it's quite complicated, and includes files that are relevant to the DBpedia version, not the KConnect bio-version, which can be confusing at times.

YODIE is a modular, configurable GATE pipeline, and extending it to new languages (or indeed domains) involves changing configuration settings where that will do the job, and adding new components where strictly necessary. The top level contains a directory, "main-bio", containing main-bio.xgapp. This is the main application. Parallel directories contain smaller applications that do part of the job, that are called from the main application. In some cases we can configure existing sub-applications, and in some cases we need to make new sub-applications. You will see, for example, that "main-bio" has a French version, indicated by the "-fr" suffix, as do the gazetteer and preprocessing application directories.

The gazetteer and preprocessing applications are designed to work with the DBpedia YODIE, and have been adapted to serve the needs of Bio-YODIE using configuration file settings. For this reason, there are PRs included that we don't want to use, and ones we do that are turned off! We use the configuration file to turn on the ones we want, and turn off the ones we don't. Without the configuration file, it defaults to the behaviour required by DBpedia YODIE. The configuration file is consulted at runtime, so when you load the application, even if you passed the configuration file, you may notice that the PRs that are turned on and the ones that are turned off aren't the ones you indicated in the configuration file, until you run it.

In summary, extending to a new language should be done as follows;

- Get and set up Bio-YODIE, and link the resources directory created in part 1
- Use the script "bin/add-new-language.sh" to make new components for your language
- Update the new gazetteer sub-application to include the stopwords for your language, created in part 2
- Update the new pre-processing sub-application to use the components you chose/created in part 2

Getting Bio-YODIE and Providing the Resources Directory

The Bio-YODIE pipeline is available as yodie-pipeline.tgz. Having unzipped it you can change into the top level directory and review the structure. You should see several directories, such as "main-bio" and "plugins". Make a symbolic link to the application resources directory you created above (or include it directly) called bio-yodie-resources in this top level directory.

Bio-YODIE expects to find a PageRank microservice running locally on port 9090. To include this, you'll need to have Docker installed. The UKB microservice docker file is on gateservice2.dcs.shef.ac.uk, and can be cloned from here:

`https://hub.docker.com/r/johannpetrak/ms-ukb/`

Documentation is included with the microservice, and describes how to indicate the location of the UKB resources you created above, which are now located in your resources directory, as well as how to run it. An example of how to run it would be:

```
docker run --name ms-ukb1 -it -p 9090:9090 -v /home/you/yodie-
pipeline/bio-yodie-resources/ukb/:/ukbdata  ms-ukb:latest
```

If you don't run the microservice, the application will still run, but will throw an exception for each document, and disambiguation performance may be harmed.

A script has been provided to do some of the work in making new components for a new language. The script takes a language suffix, which is recommended to be the two-letter ISO code for your new language. For example, for Bulgarian you would run this like so;

```
bin/add-new-language.sh bg
```

If you already have components for this language extension, namely “main-bio-nn”, “gazetteer-nn” or “preprocess-nn” directories (where nn is your language code), you'll need to clear them out before you run it, as the script won't overwrite existing files.

You now have a new top-level application, main-bio-nn/main-bio.xgapp which should run. However, you need to include language-appropriate resources in the gazetteer and preprocessing applications in order to complete the job of tailoring it to your language.

Preprocessing Sub-Application

You will find that you now have a sub-application called preprocess-nn (where nn is your two letter language code). You are welcome to modify this application according to your needs, by including the tokenisers, POS taggers and morphological analysers you gathered for your language earlier. This application is designed to work with both the DBpedia version of YODIE as well as Bio-YODIE, so for this reason it contains components that we don't use in addition to the ones we do. In your config file (main-bio-nn/main-bio.config.yaml) you will see extensive settings intended to adapt the pre-processing application to the needs of Bio-YODIE, whilst leaving the application by default to do what it needs to do for DBpedia YODIE. Therefore it may seem rather complicated. However, in replacing components, so long as input and output annotation sets and types are preserved, there should be no problem.

Gazetteer Sub-Application

If you don't want to use morphologically aware matching for stopwords, you can simply drop in a new stopwords.lst file into gazetteer-nn/gazetteer. **Be sure to delete “stopwords.gazbin”**. The gazbin is a compressed gazetteer that is created from the lst file, and is only recreated if it is missing, so replacing the lst file alone will not update the gazetteer.

If you want to match stopwords based on morphology, you will need to add a morphological analyser for your language to your pre-processing application. You may also/instead wish to create a stem-based gazetteer, for which again you will need to include a stemmer. You can then run the script as described above to create stopword gazetteers. The script will create an application for you containing the new gazetteers, which you may include as a module. Alternatively you can copy the resources across to your gazetteer-nn directory and add new gazetteers to this sub-application. You will need to

ensure that your stopword gazetteers output annotations to the type “Stopword” in the annotation set “GazetteerEN”.

Main Application

The main application file is located in the main-bio-nn subdirectory (where nn is your two letter language code), and is called main-bio.xgapp. This also contains its configuration file, main-bio.config.yaml. **Whenever you use Bio-YODIE, you need to pass the configuration file as an argument, otherwise it won't work!** You can use a Java argument for this: -Dat.ofai.gate.modularpipelines.configFile. For example you can use this argument to run GATE from the gate.sh script in your GATE installation to run test data in the GUI. (If you don't have GATE installed already, you will need to install it!)

It might be helpful to review the configuration file, in order to understand how you may turn PRs on and off, which might be helpful during development, for example when adding and removing gazetteers, and also in order to understand what is actually going to be run when you run the application, since the configuration file is only consulted at runtime, so a PR may appear in the GUI to be turned on, despite being configured not to run.

Running the application should give you named entity annotations in the "Bio" annotation set.

Appendix B—Mimir Template

```
import gate.creole.ANNIEConstants
import gate.mimir.index.OriginalMarkupMetadataHelper
import gate.mimir.util.DocumentFeaturesMetadataHelper
import gate.mimir.sparql.SPARQLSemanticAnnotationHelper as SPARQLHelper
import gate.mimir.db.DBSemanticAnnotationHelper as DefaultHelper
import gate.mimir.util.NormalizingTermProcessor
import gate.mimir.SemanticAnnotationHelper.Mode
import gate.mimir.sparql.RequestMethod as RM
import gate.nesta.LabelKindMentionDescriber

// The token annotations are in the default set
tokenASName = ""
tokenAnnotationType = ANNIEConstants.TOKEN_ANNOTATION_TYPE
tokenFeatures = {
  string(directIndex:true)
  category(directIndex:true)
  kind(directIndex:true)
  root(directIndex:true)
}


def sparqlHelper(params) {

  // this is a DB helper
  def delegateHelper = new DefaultHelper(params)
  // increase the cache sizes:
  // L1: will only ever have 1 value, as there are no nominal feature,
  // L2, L3: there will be a small number of million entries, so we try and
cache most of them
  delegateHelper.setCacheSizes(16, 2* 1024 * 1024, 2* 1024 * 1024)

  return new SPARQLHelper(
//    sparqlEndpoint:
"http://mimir:semantic2015@195.113.21.16:8080/repositories/owlim",
//    sparqlEndpoint:
"http://mimir:semantic2015@services.gate.ac.uk/asasqeacwce",
    sparqlEndpoint: "http://kconnect-kb.s4.ontotext.com/v1/sparql",
    sparqlEndpointUser: "APIKEYINBASE64",
    sparqlEndpointPassword: "SECRETINBASE64",
    queryPrefix:('PREFIX :<http://linkedlifedata.com/resource/umls/> ' +
                 'PREFIX khr:<http://khresmoi.eu/resource/umls/> ' +
                 'PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#> ' +
                 'PREFIX xsd:<http://www.w3.org/2001/XMLSchema#> ' +
                 'PREFIX owl:<http://www.w3.org/2002/07/owl#> ' +
```

```
                  'PREFIX psys:
<http://proton.semanticweb.org/2006/05/protons#> ' +
                  'PREFIX gene:
<http://linkedlifedata.com/resource/entrezgene/> ' +
                  'PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
'),
    delegate:delegateHelper
  )
}


// See also: http://wiki.khresmoi.eu/index.php5/Mimir_Index_Schema


semanticASName = "Shef"
semanticAnnotations = {

  index {
    annotation helper:new DefaultHelper(annType:'Sentence')
  }

  index(directIndex:true) {
    annotation helper:sparqlHelper(
        annType:'Mention',
        textFeatures:['inst_full','tui_full','inst','TUI','PREF'],
        nominalFeatures:['STY','Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))
  }

  index(directIndex:true) {
    annotation helper:sparqlHelper(
        annType:'Vocabulary',
        textFeatures:['vocab'],
        mentionDescriber:new LabelKindMentionDescriber('vocab', null))
  }

  index(directIndex:true) {
    annotation helper:sparqlHelper(
        annType:'Body_Location_or_Region',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Body_Part_Organ_or_Organ_Component',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
```

```
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Body_Space_or_Junction',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Body_System',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Tissue',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Anatomical_Abnormality',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Acquired_Abnormality',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Congenital_Abnormality',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Finding',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Injury_or_Poisoning',
```

```
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Pathologic_Function',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Disease_or_Syndrome',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Mental_or_Behavioral_Dysfunction',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Cell_or_Molecular_Dysfunction',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Experimental_Model_of_Disease',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Sign_or_Symptom',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))

    annotation helper:sparqlHelper(
        annType:'Neoplastic_Process',
        textFeatures:['inst_full','inst','PREF'],
        nominalFeatures:['Experiencer','Negation','Temporality'],
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))
```

```
annotation helper:sparqlHelper(
    annType:'Diagnostic_Procedure',
    textFeatures:['inst_full','inst','PREF'],
    nominalFeatures:['Experiencer','Negation','Temporality'],
    mentionDescriber:new LabelKindMentionDescriber('PREF', null))

annotation helper:sparqlHelper(
    annType:'Laboratory_Procedure',
    textFeatures:['inst_full','inst','PREF'],
    nominalFeatures:['Experiencer','Negation','Temporality'],
    mentionDescriber:new LabelKindMentionDescriber('PREF', null))

annotation helper:sparqlHelper(
    annType:'Laboratory_or_Test_Result',
    textFeatures:['inst_full','inst','PREF'],
    nominalFeatures:['Experiencer','Negation','Temporality'],
    mentionDescriber:new LabelKindMentionDescriber('PREF', null))

annotation helper:sparqlHelper(
    annType:'Research_Activity',
    textFeatures:['inst_full','inst','PREF'],
    nominalFeatures:['Experiencer','Negation','Temporality'],
    mentionDescriber:new LabelKindMentionDescriber('PREF', null))

annotation helper:sparqlHelper(
    annType:'Molecular_Biology_Research_Technique',
    textFeatures:['inst_full','inst','PREF'],
    nominalFeatures:['Experiencer','Negation','Temporality'],
    mentionDescriber:new LabelKindMentionDescriber('PREF', null))

annotation helper:sparqlHelper(
    annType:'Clinical_Drug',
    textFeatures:['inst_full','inst','PREF'],
    nominalFeatures:['Experiencer','Negation','Temporality'],
    mentionDescriber:new LabelKindMentionDescriber('PREF', null))

annotation helper:sparqlHelper(
    annType:'Pharmacologic_Substance',
    textFeatures:['inst_full','inst','PREF'],
    nominalFeatures:['Experiencer','Negation','Temporality'],
    mentionDescriber:new LabelKindMentionDescriber('PREF', null))

annotation helper:sparqlHelper(
    annType:'Antibiotic',
    textFeatures:['inst_full','inst','PREF'],
    nominalFeatures:['Experiencer','Negation','Temporality'],
```

```
        mentionDescriber:new LabelKindMentionDescriber('PREF', null))
  }

  index(directIndex:true) {
    // Metadata features
    annotation helper:new DefaultHelper(annType:'Document',
mode:Mode.DOCUMENT,
        integerFeatures:['publicationDate'])
  }

}

documentRenderer = new OriginalMarkupMetadataHelper()
documentFeaturesHelper = new
DocumentFeaturesMetadataHelper("publicationDate","id")
documentMetadataHelpers = [documentRenderer, documentFeaturesHelper]
```