Grant Agreement Number: 257528

KHRESMOI

www.khresmoi.eu

Initial prototype for semantic annotation of the Khresmoi literature

Deliverable number	*D1.2*
Dissemination level	*Public*
Delivery date	*18th May 2012*
Status	*Final*
Author(s)	*Mark A. Greenwood, Angus Roberts, Niraj Aswani and Phil Gooch*

This project is supported by the European Commission under the Information and Communication Technologies (ICT) Theme of the 7th Framework Programme for Research and Technological Development.

Abstract

This deliverable describes the prototype for the semantic annotation of the Khresmoi document collection. The application consists of a number of connected GATE applications which annotation the documents. The annotations are exposed to other Khresmoi components via a GATE Mímir index.

Table of Contents

List of Abbreviations

HON: Health On The Net

IDF: Inter-Document Frequency

NE: Named Entity

POS: Part-of-Speech

PR: Processing Resource

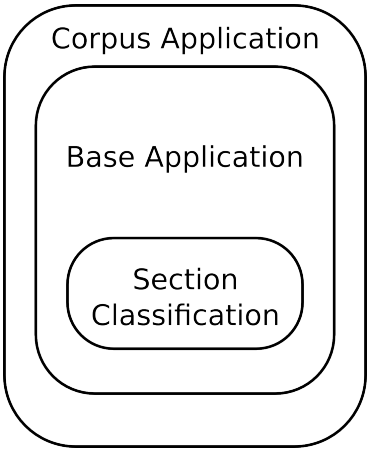
TF: Term Frequency

UMLS: Unified Medical Language System

URI: Uniform Resource Identifier

1 Executive Summary

This document accompanies the main software deliverable and gives a biref overview of the prototype Khresmoi semantic application developed for processing the textual documents that make up the Khresmoi knowledge base, and the GATE Mímir indexed which can be produced to allow access to the processed documents.



The Khresmoi application is a complex GATE pipeline consisting of three sub applications arranged as shown.

- The **Corpus Application** is the main application used to semantically annotate documents. It is responsible only for the extraction of corpus specific metadata etc.
- The **Base Application** is the largest of the sub applications and is responsible for the majority of the annotations produced.
- The **Section Classification** application only processes the main content of each page and assigns a classification to each as well as the document as a whole.

The rest of this document presents details of each application (at the level of processing resources) and the annotations which result.

The output from this deliverable is therefore a prototype semantic annotation application which can process the Khresmoi literature to produce a searchable index that can be integrated with other Khresmoi components in order to provide both flexiable and powerful search across the indexed documents.

2 Application Details

Each of the semantic applications contain a number of processing resources (PRs) which are run in sequence. PRs can be roughly divided into two groups; generic linguistic pre-processing and domain specific processing. The following sections detail the PRs used by each of the sub applications. For more details of the different PRs please consult the GATE user guide [2].

2.1 Corpus Application

While the majority of the semantic annotation is common to all document collections, some metadata extraction (titles, authors, sections, etc.) is specific to each corpus. This is taken care of by book-ending the base application (see following section) with corpus specific processing.

A **document reset** PR to remove annotations from any previous processing – this is mostly needed for testing to ensure that documents are returned to their initial state before processing.

The **Base Application** (see below) is embedded to provide the main processing.

The **metadata collector** is an instance of the Groovy Script PR and is used to extract and store metadata as document features (this is the PR which requires modification for each corpus type).

Currently we provide a single "web" corpus application. If, in the future, we want to make use of documents which are available in some other form then other applications can easily be generated as required.

2.2 Base Application

The base application is responsible for producing most of the semantic annotations, although it should only be used from within a corpus application. The application consists of the following processing resources:

The **sentence splitter** is a cascade of finite-state transducers which segments the text into sentences. This module is required for the tagger. Both the splitter and tagger are generally domain and application-independent.

The **tokeniser** splits text into simple tokens, such as numbers, punctuation, symbols, and words of different types (e.g. with an initial capital, all upper case, etc.), adding a "Token" annotation to each. It does not need to be modified for different applications or text types.

The **POS tagger** is a modified version of the Brill tagger [2], which adds a part-of-speech tag as a feature to each Token annotation. Neither the splitter nor the tagger is a mandatory part of the NE system, but the annotations they produce can be used by the semantic tagger (described below), in order to increase its power and coverage.

The **morphological analyser** considers one token and its part of speech tag, one at a time, it identifies its lemma and an affix. These values are than added as features on the Token annotation. We also generate a stem feature using the **snowball stemmer**[1].

The application contains two **ontology based gazetteers** (for full words and abbreviations) which are used to annotate the documents with UMLS concepts via the Khresmoi Large Scale Biomedical Knowledge Base as described in deliverable D5.2.

The gazetteers return URI objects which we convert to strings (via a JAPE grammar – **TurnURIsIntoStrings**) to makle later processing/indexing easier. For debugging purposes we also convert the lookups into named annotations via a second JAPE grammar (**AssignTypesFromTUIs**).

The **HON Tag Parser** assigns HON classifications (based upon the domain from which the page originated) to each document.

Whilst not complete the **Drug Gazetteer** is used to annotate the names of many common drugs (the list includes all drugs that can be prescribed the UK's National Health Service)

Boilerpipe is used to perform **Content Detection**. This allows us to both focus the section classification to only those sections within the main article content, and also to allow searches to be restricted to article content if required by indexing the resultant annotation (this is more flexiable than only indexing annotations within the main content, although does require more overhead during indexing).

The **Section-By-Section** PR is used to run the Section Classification application detailed below.

The final phase of the application involves three inter-related **Section Classification** PRs. This involves; the creation of TF/IDF indexes of the keyword/phrase category and subcategory schema occurrences in each document, classification of sentences based on the within-sentence TF/IDF keyword/phrase scores, and linguistic patterns based on Hearst and the UMLS SemRep, clustering of similarly classified sentences into sections, which then inherit the classification of their constituent

[1] http://snowball.tartarus.org/

sentences, and finally document classification from the TF/IDF index of the keyword/phrase document schema categories. Twenty three of the forty two different document categories are currently covered by the app, If the application can't assign a document category it uses the section category/subcategory IDF maps. For the missing document categories, these are similar (e.g. medicines and treatment vs drugs and medicinal substances, sport and leisure vs. exercise and weight loss, insurance vs health insurance, nutrition vs balanced diet etc). The more ambiguous categories will be implemented in further prototypes.

2.3 Section Classification

The section classification application is responsible for identifying keywords that are clues to the section classification. This consists of three gazetteers; **Categories and Individuals Gazetteer**, **Ontology Lookup** and a **Neoclassical Gazetteer**.

The gazetteers are followed by a **Verb Phrase Chunker** which aids with the final PR which performs **Keyword Detection**.

3 Mímir Index

The annotations produced by the semantic annotation pipeline described above are exposed to other components of the Khresmoi system via a GATE Mímir index. See Appendix A for details on how to produce an index.

The index contains the following annotation types, and features, that can be used to construct a search query :

- **Publication**: This is a document level annotation, which means that the features are available at search time and as document metadata)
    - title: the title of the publication
    - date: the date of the document, stored as a number in the form yyyymmdd
    - authors: the authors of the publication stored as a set (specifically an instance of LinkedHashSet). When searching this is essentially treated as a comma separated list of authors, but the underlying set can be accessed via the RemoteQueryRunner API for easier handling at display time. If accessed via the RESTful web service then the indexed string is returned as the interface doesn't support the return of arbitary objects.
    - url: the original URL of the publication. This is where users should be directed for access to the full publication
    - publisher: the publisher of the publication
    - corpus: the name of the corpus in which this publication can be found
    - id: an internal id of the the publication
    - idType: in conjunction with the corpus featue this tells interfaces how to interpret the id feature
    - containsImages: true if the document contains images, false otherwise
    - images: an array of strings, each of which is the URL of one of the images contained within the document (only guaranteed to exist if containsImages is true). Currently this feature is only available as document metadata and can't be used within a query
    - HONLabel.targetAudience: the target audience of the article as determined by HON

  - HONLabel.language: the language of the article as determined by HON
  - HONLabel.tags: the type of content as determined by HON (note this isn't the classification determined by the application)
  - classification: a set of zero or more document level classifications
  - sections: this a list that holds the information about sections (see the Section annotation below), in a form that can be used at result display time. The list has the format: start offset, end offset, set of types, set of subtypes. Note that this is only available as part of the results and should not be used for restricting searches.

- **ArticleContent**: spans the main content of the article. Restricting searches to these sections allow text appearing in menus, headers etc. to be ignored.
- **Lookup**: spans text looked up via a gazetteer built from the Khresmoi knowledge base
  - class: a string version of the class URI from the Khresmoi knowledge base
  - inst: a string version of the instance URI from the Khresmoi knowledge base
- **Medication**: spans the names of drugs within the documents that are produced by the drug gazetteer in the Base Application.
- **Section**: sections identified by the automatic section classifier
  - type: a set of top level automatically assigned HON categories
  - subtype: a set of sub level automatically assigned HON categoies

4 Conculsions

This deliverable has outlined the current version of the prototype for the semantic annotation of the Khresmoi literature. The specific components within the application have been documented and described in some detail. These annotations are exposed to the rest of the Khresmoi system via a GATE Mímir index, the structure (and hence the queries which can be issued) of which has also been described. Together with the prototype this deliverable should allow for easy integration of the software with other Khresmoi components, such as ezDL.

Whilst we believe that the performance of the prototype is adequate to the task, a thorough evaluation is planned for the next period which will be reported in deliverable D1.3.

5 References

[1] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, and C. Ursu. 2002. The GATE User Guide. `http://gate.ac.uk/`

[2] Mark Hepple. Independence and commitment: Assumptions for rapid training and execution of rule-based POS taggers. In Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000), Hong Kong, October 2000.

A. Installation and Usage Guide

This appendix aims to give a brief introduction to using the Khresmoi prototype, both for testing purposes and for large scale processing. Note that this appendix is not a replacement for the GATE user guide [1] which should be referred to for more details.

a. Testing Within GATE Developer

Testing the prototype within GATE Developer is very straightforward. Simple follow the user guide [1] to download, install and run GATE Developer. Once running choose to restore an existing application, and load the "web-application.xgapp" file from the prototype. This will result in the loading of a large number of PRs and three applications. The application of interest is called "Khresmoi (Web)". This application can be run over any corpus of documents (although best results will come from web documents previously classified by HON). Once the application has been run, opening any document will enable you to see the annotations produced by the application. Note that a large number of annotations are produced not all of which are treated as the final output of the application, but they are retained to aid in debugging and future development.

Note that this application, due to it's size and complexity, requires more RAM than GATE is by default configured to use. We recommend configuring GATE with at least 2GB of RAM for this application to function correctly.

b. Large Scale Processing

Whilst testing via GATE Developer is useful, large scale processing requires a different approach. For this we recommend using the GATE Cloud Paralleliser[1], either installed locally or via GATECloud.net. In either case as well as the application you will need a GATE Mímir index template which defines which annotations are included in the final searchable index. This is included with the application (mimir-index-template.groovy). Details on how to run a large scale processing job are outside the scope of this deliverable. For full details please see the appropriate documentation.

[1] `http://gate.svn.sourceforge.net/viewvc/gate/gcp/trunk/`