New Challenges For NLP Frameworks Programme

A workshop at LREC 2010, Valletta, Malta, May 22
http://nlpframeworks2010.semanticsoftware.info

9:15 – 9:30 Welcome

9:30 – 10:30 Invited Talk: Jean-Marie Favre

10:30 – 11:00 Coffee break

11:00 – 13:00 Talks

Kalina Bontcheva, Hamish Cunningham, Ian Roberts and Valentin Tablan: *Web-based Collaborative Corpus Annotation: Requirements and a Framework Implementation*

Cartic Ramakrishnan, William A. Baumgartner Jr., Judith A. Blake, Gully APC Burns, K. Bretonnel Cohen, Harold Drabkin, Janan Eppig, Eduard Hovy, Chun-Nan Hsu, Lawrence E. Hunter, Tommy Ingulfsen, Hiroaki Onda, Sandeep Pokkunuri, Ellen Riloff, Christophe Roeder and Karin Verspoor: *Building the Scientific Knowledge Mine (SciKnowMine): a community-driven framework for text mining tools in direct service to biocuration*

Adam Funk and Kalina Bontcheva: *Effective Development with GATE and Reusable Code for Semantically Analysing Heterogeneous Documents*

Manuel Fiorelli, Maria Teresa Pazienza, Steve Petruzza, Armando Stellato and Andrea Turbati: *Computer-aided Ontology Development: an integrated environment*

13:00 – 14:30 Lunch break

14:30 – 15:30 Invited Talk: Michael Tanenblatt

15:30 – 16:30 Poster Session

Ralf Krestel, René Witte and Sabine Bergler: *Predicate-Argument EXtractor (PAX)*

Radim Řehůřek and Petr Sojka: *Software Framework for Topic Modelling with Large Corpora*

Ninus Khamis, Juergen Rilling and René Witte: *Generating an NLP Corpus from Java Source Code: The SSL Javadoc Doclet*

Nicolas Hernandez, Fabien Poulard, Matthieu Vernier, Jérôme Rocheteau: *Building a French-speaking community around UIMA, gathering research, education and industrial partners, mainly in Natural Language Processing and Speech Recognizing domains*

Elena Beisswanger and Udo Hahn*: JULIE Lab's UIMA Collection Reader for WIKIPEDIA*

16:00 – 16:30 Coffee Break

16:30 – 17:30 Panel Discussion: New Challenges for NLP Frameworks

17:30 – 17:45 Conclusions

Workshop Organisers

René Witte
Concordia University, Montréal, Canada

Hamish Cunningham
University of Sheffield, UK

Jon Patrick
University of Sydney, Australia

Elena Beisswanger
University of Jena, Germany

Ekaterina Buyko
University of Jena, Germany

Udo Hahn
University of Jena, Germany

Karin Verspoor
University of Colorado Denver, USA

Anni R. Coden
IBM T.J. Watson Research Center, USA

Workshop Programme Committee

Aaron Kaplan (Xerox, France)
Adam Funk (University of Sheffield)
Angus Roberts (University of Sheffield)
Anni R. Coden (IBM T.J. Watson Research Center)
Claude Roux (Xerox Research Labs)
Diana Inkpen (University of Ottawa)
Diana Maynard (University of Sheffield)
Dietmar Rösner (University of Magdeburg)
Dragan Gasevic (University of Athabasca)
Ekaterina Buyko (University of Jena)
Elena Beisswanger (University of Jena)
Epaminondas Kapetanios (University of Westminster)
Eric W. Brown (IBM T.J. Watson Research Center)
Graham Wilcock (University of Helsinki)
Guergana K. Savova (Mayo Clinic)
Hamish Cunningham (University of Sheffield)
Horacio Saggion (University of Sheffield)
Iryna Gurevych (University of Darmstadt)
Jian Su (I2R, Singapore)
Jochen Leidner (Thomson Reuters)
Jon Patrick (University of Sydney)
Juergen Rilling (Concordia University, Montréal)
Kalina Bontcheva (University of Sheffield)
Karin Verspoor (University of Colorado)
Katrin Tomanek (University of Jena)
Kevin B. Cohen (University of Colorado School of Medicine/MITRE)
Leila Kosseim (Concordia University, Montréal)
Leo Ferres (University of Concepcion)
Marc Light (Thomson Corp. R&D)
Michael Tanenblatt (IBM T.J. Watson Research Center)
Nancy Ide (Vassar College)
Nicolas Hernandez (University of Nantes)
Philip V. Ogren (University of Colorado)
Ralf Krestel (L3S Research Center, Hannover)
René Witte (Concordia University, Montréal)
Richard Eckart de Castilho (University of Darmstadt)
Sameer Pradhan (BBN)
Stefan Geißler (TEMIS GmbH)
Steven Bethard (Stanford University)
Thilo Götz (IBM Germany)
Udo Hahn (University of Jena)
Valentin Tablan (University of Sheffield)
Yoshinobu Kano (University of Tokyo, Tsujii Lab)

Table of Contents

Author Index

Effective Development with GATE and Reusable Code for Semantically Analysing Heterogeneous Documents

Adam Funk, Kalina Bontcheva

Department of Computer Science
University of Sheffield
Regent Court, Sheffield, S1 4DP, UK
a.funk@dcs.shef.ac.uk, k.bontcheva@dcs.shef.ac.uk

Abstract

We present a practical problem that involves the analysis of a large dataset of heterogeneous documents obtained by crawling the web for information related to web services. This analysis includes information extraction from natural-language (HTML and PDF) and machine-readable (WSDL) documents using NLP and other techniques, classifying documents as well as services (defined by sets of documents), and exporting the results as RDF for use in the back-end of a portal that uses Web 2.0 and Semantic Web technology. Triples representing manual annotations made on the portal are also exported back to our application to evaluate parts of our analysis and for use as training data for machine learning (ML). This application was implemented in the GATE framework and successfully incorporated into an integrated project, and included a number of components shared with our group's other projects.

1. Introduction

The Service-Finder project addresses the problem of web service discovery for a wide audience through a portal[1] which presents automatic semantic descriptions of a wide range of publicly available web services and also enables service consumers (not just providers) to enrich them according to Web 2.0 principles (manual annotation according to the project ontology, tagging, and wiki-like editing of free text fields). In this project, the Service Crawler (SC) carries out focused crawling for web services and archives WSDL files and related HTML files, then passes monthly batches of these data to the Automatic Annotator (AA), which analyses them to produce semantic annotations to the Conceptual Indexer and Matchmaker (CIM), the semantic repository and back end for the web portal. Additional components include the portal itself and the clustering engine that provides recommendations. (**?**)

Here we present the implementation of the Automatic Annotator using the versatile GATE[2] (**?**) framework for NLP and related applications.

1.1. Input and output

The SC (Service Crawler) component delivers to the AA a monthly batch of data, consisting of a number of compressed Heritrix (**?**) Internet Archive files (up to 100 MB each), along with an index of all the documents in the batch. The documents for each service include one or more WSDL files, an abstract[3], and zero or more HTML and PDF files (especially those with contact details, links to WSDL files, pricing information, terms and conditions, and other useful information).

Figure 1 shows a sample extract from the index, which lists every document in the crawler output, along with the archive file number and offset where it can be found and type codes (e.g., `w` for WSDL or `a` for abstract). Each stanza is headed by a service URI[4], which is also used in the Heritrix archives as the URL of the abstract. The same service URI may occur several times in this index with different documents listed below it.

This collection of files is downloaded onto one of our servers for processing, as described in the rest of the paper. The results are exported as RDF-XML to the CIM. Table 1 summarizes the input and output of a typical monthly batch of data.

Input from the SC	
Number of `.arc.gz` files	5
Total size of compressed files	441 MB
Number of documents	$\sim$ 250 000
Output to the CIM	
Number of RDF-XML files	30
Total size of compressed files	40 MB
Number of RDF triples	$\sim$ 4 500 000
Number of Providers	$\sim$ 8 700
Number of Services	$\sim$ 25 000

Table 1: Typical AA input and output

1.2. Annotation tasks

The Automatic Annotator's principal tasks are as follows:

- analyse WSDL files to produce *Endpoint*, *Interface*, and *Operation* instances as well as properties associating them with each other and with the relevant *Service* instances;

[1] `http://demo.service-finder.eu/`

[2] `http://gate.ac.uk/`

[3] The *abstract* is an HTML file that the SC compiles from various elements' and attributes' strings (the service name, documentation, operation names, input and output parameters, etc.) in the best WSDL file for the service.

[4] The crawler generates URIs for instances of the *Service* and *Provider* classes. The service URI always consists of the provider URI followed by one path-element, so the provider URI can be easily obtained from the service URI.

```
http://seekda.com/providers/dp2003.com/FileService  1  100684561  a
 http://dp2003.com/filews/filews.asmx?WSDL          1  19796469   w
 http://dp2003.com/filews/ListUser                  3  29130320   f o
 http://dp2003.com/filews/Logout                    3  29131388   f o
 http://microsoft.com/wsdl/mime/textMatching/       4  104841754  f o
 http://dp2003.com/filews/UserInfo                  3  29132932   f o
```

Figure 1: Excerpt from the input index

- classify documents by type (e.g., documentation, pricing, contact details) and rate them as low-, medium-, or high-interest;
- carry out information extraction to identify providers' addresses, phone numbers, e-mail addresses, etc.;
- carry out information extraction over services to identify service level agreements, free trials, etc.;
- categorize each service in one or more of the 59 subclasses of *ServiceCategory*.

Some tasks require information to be amalgamated across various sets of documents, and all the output is expressed as RDF-XML according to the project ontology. Figure 2 highlights several types of keywords and annotated pieces of information in GATE's GUI.

2. Implementation

We implemented the Automatic Annotator tools using GATE, a versatile, extensible library and framework for NLP and text processing. We used the *GATE Developer* GUI environment to develop some of the pipelines (using the ANNIE information extraction pipeline as a starting point), GATE's gazetteer and JAPE tools for rapid development of processing resources (PRs) to mark keywords and phrases with weights according to context, and PRs for more complicated functions functions that could not be easily coded in JAPE, along with control programs, both written in Java using the *GATE Embedded* library. (**?**)

2.1. Preprocessing

First of all, we recognized the need to split the large input dataset into manageable chunks and took advantage of the independent nature of the data about each provider. We developed a preprocessor in Java that can be run from the command-line or a shell script in GNU screen[5] on a server. This preprocessor reads the index file into memory, creates several (typically 30) serially numbered GATE Serial DataStores[6], then iterates through all the documents in the input archive files. For each document, it checks the HTTP status code and discards the document if the code is 3xx, 4xx, or 5xx; otherwise, it uses the index to identify the document type (WSDL, abstract, HTML, PDF) and the service and provider URIs. It then calculates the MD5 hash (**?**) of the document and checks the list of already hashed documents; in case of a match, it reloads the existing serialized GATE document, merges the additional document URL and service URI into it, re-saves it, and goes on to the next document. Table 2 shows the effects of these suppression and de-duplication steps, and Table 3 shows the striking "before and after" effects on performance of making more effort in the preprocessor to eliminate unnecessary documents from analysis (on a Xeon X3220 server with Java's `-Xmx 4000m` setting). The preprocessing time increased sixfold but the total time decreased by 53%.

Stage		Approx. number of documents
Input	total documents	250 000
Preprocessing reductions		
	HTTP error codes	49 000
	unwanted provider IDs	5 000
	empty documents	3 000
	reduced duplicates	23 000
	faulty XML	< 30
Output	HTML	37 000
	WSDL	110 000
	abstract	25 000
total (31% reduction)		173 000

Table 2: Typical results of preprocessing

	Time in hours	
Tool	Before	After
Preprocessor	3.0	18.3
Analysis	72.9	18.1
Archiving	6.5	2.5
Total	82.4	38.9

Table 3: Examples of AA performance times

For each valid, non-duplicate document, the preprocessor instantiates a GATE document in a mark-up-aware manner, with the plain-text content, the HTML mark-up or XML tags, and the metadata all stored in the appropriate parts of GATE's document model (the document content, *Original markups* annotation set, and document feature map, respectively). (GATE uses a modified form of the TIPSTER and Atlas formats (**?**; **?**) as stand-off mark-up, as shown in Figure 2.) (WSDL documents are also analysed by software developed mainly by another project partner, seekda[7], integrated so that it stores its results as an RDF-XML document feature.) It adds this document to the provider's corpus (which it creates when it first encounters that provider). The

[5] `http://www.gnu.org/software/screen/`

[6] A Serial DataStore provides disk-based persistence for documents and corpora using Java serialization.

[7] `http://seekda.com/`

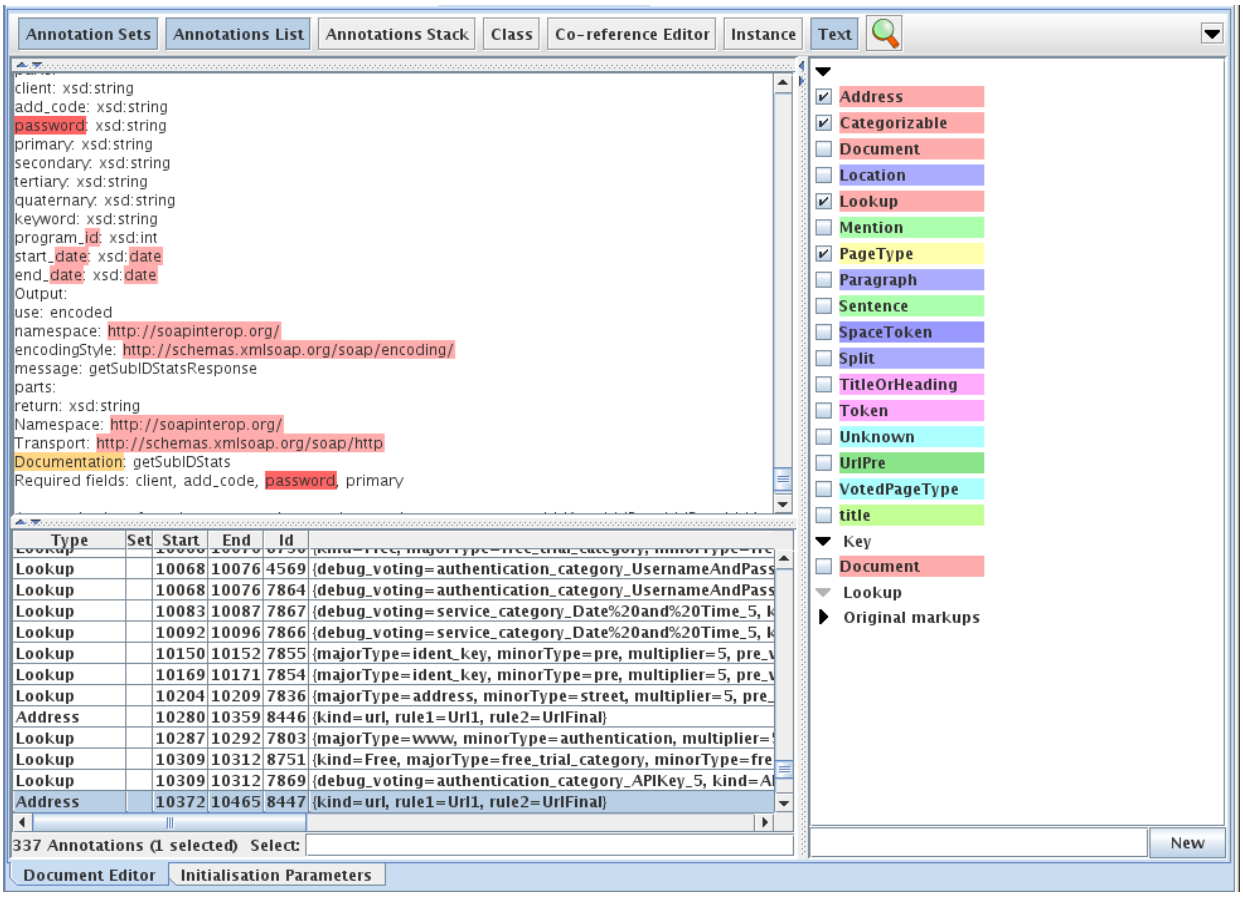


Figure 2: Annotated document in GATE

corpora are allocated in a loop over the datastores so that the latter end up with roughly the same sizes. The provider–corpus and document–MD5 mappings are kept in memory since they are used so frequently. Finally, the preprocessor closes all the corpora (synchronizing them on disk), closes the datastores, and writes several index files that indicate which providers and services are in which corpora and datastores.

2.2. Analysis

We then run a shell script that carries out the main analysis tasks and generates the consolidated RDF-XML file separately over each datastore.

As mentioned earlier, we took ANNIE as the starting point for the information extraction tasks, but modified it so that most of its PRs run only on HTML and PDF documents.

Each datastore produced by the preprocessor is processed through a Java tool that includes ANNIE[8] with additional gazetteers and rules developed within Service-Finder to identify relevant terms in the web service domain and annotate interesting sections and documents, consolidate the information from various documents for each provider, merge in the RDF-XML snippets generated by the preprocessor and attached to the corpora and documents, and produce one large RDF-XML file for each block (datastore).

2.2.1. Overview

The analysis pipeline consists of the following series of processing resources (PRs), as illustrated in Figure 3.

1. Standard ANNIE components tokenize and sentence-split the HTML and PDF documents (creating *Token* and *Sentence* annotations on the document). Abstracts and WSDLs are processed with a *source-code tokenizer* (developed in the TAO project[9]), a version of the ANNIE tokenizer with the rule files modified to split camel-cased strings (e.g., `getUnsplicedSequence` → `get Unspliced Sequence`) as well as tokens separated by whitespace.

2. The ANNIE gazetteers and NER (named-entity recognition) module (consisting of JAPE transducers) identify and annotate a range of entities such as *Date*, *Person*, *Organization*, and *Address*.

3. Gazetteers developed for this application mark keywords relating to web services, such as those used to indicate free trials, terms and conditions, pricing, and categories of services. Figure 4 shows keywords as-

[8] ANNIE is the information extraction system supplied with GATE; it includes standard NLP tools for English (such as a tokenizer and POS tagger) and gazetteers and rule-based tools for named entity recognition.

[9] `http://www.tao-project.eu/`

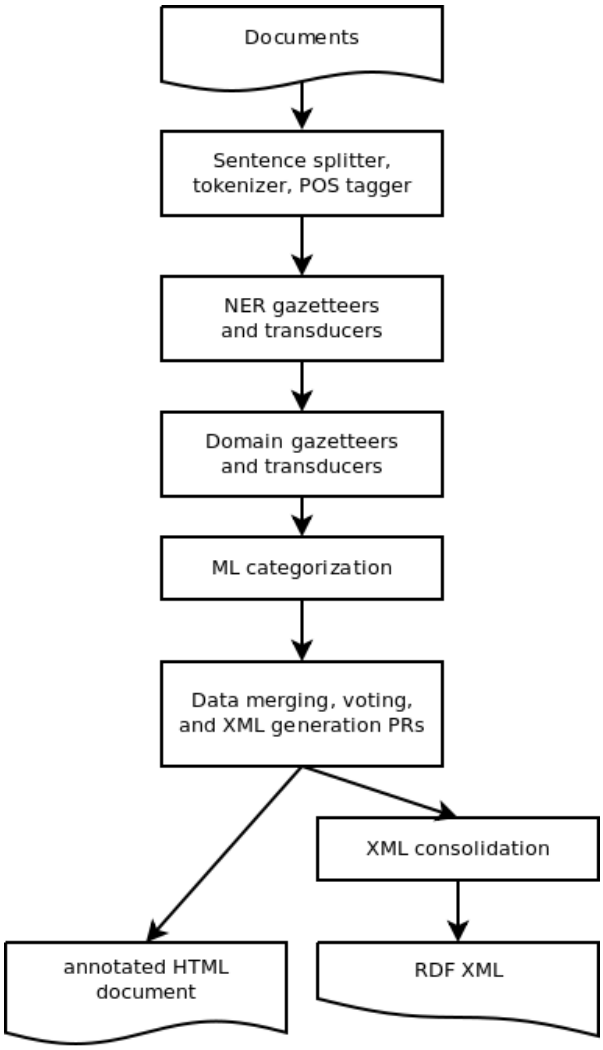


Figure 3: Overview of the IE pipeline

terms and conditions	user agreement
terms & conditions	terms of use
licence agreement	TOU
license agreement	T&C
licencing agreement	AUP
licensing agreement	T&Cs
acceptable use policy	TOS
terms of service	

Figure 4: Examples of keywords that vote for *TermsAndConditionsPage*

sociated with the *TermsAndConditionsPage* document type.

4. A series of custom JAPE transducers compare the annotations produced by ANNIE and the custom gazetteers with the HTML mark-up in order to evaluate their role and set a *multiplier* feature accordingly, which is used by the voting processor in step 6. (For example, keywords and named-entity annotations in `title` or `h1` elements are more important than elsewhere in the document, and `p` elements that begin with the keyword *Description* or *Price* are more likely to relate to the description of the service or its pricing. Such annotations are assigned a *multiplier* feature $1 \leq m$, typically $m < 6$.) Other JAPE rules try to identify company details such as country of origin.

   These rules also have access to document features containing metadata such as the service, provider, and document URIs, which they can copy into features on the annotations they create. They also adjust an *interesting* feature[10].

5. An instance of the GATE machine learning PR (**?**) aims to label each document with a class from the service category ontology. Section 2.2.2 describes the integration of this component in more detail.

6. "Voting" PRs (extensions of *AbstractLanguageAnalyser* from the *GATE Embedded* library) compare the weighted frequency of significant keywords to assign certain ontology classes and properties as follows.

   - For each WSDL document and interesting (see step 4 above) textual (i.e., HTML or PDF) document, create a potential instance of the general class *Document* or one of its subclasses (*DocumentationPage*, *TermsAndConditionsPage*, etc.) and assert the properties *hasSize*, *hasTitle*, and *retrievedAt*; also create a *DocumentAnnotation* associating the document with a service or provider. (Some document types, such as *ContactDetailsPage*, are associated with providers; others are associated with services.)
   - For each service, assertions of the properties *supportsAuthenticationModel*, *hasServiceLevelAgreement*, *allowsFreeTrials*, etc.; these properties are not asserted if no votes (keywords) are found.
   - For each service, the two best categories for *CategoryAnnotation* (or just one category if all the votes were for the same one); the categories assigned by machine-learning outweigh those generated by keywords and rules (as §2.2.2 explains in detail).
   - For each provider, one or two best values for *hasHomepage*, *fromCountry*, *hasEmail*, *hasAddress*, and *hasTelephone* (these properties are not asserted when no values are found in the documents).

   All GATE PRs override the `execute` method, which is called on each document in the corpus. Each voting PR uses additional hooks, as shown in Figure 5, to initialize the set of "ballots" at the beginning of the `execute` on the first document in the corpus; and to compute the results, generate the corresponding RDF-XML, and store it as a corpus feature at the

[10] $0.0 \leq i \leq 3.0$, interpreted as low ($0.0 \leq i < 1.0$), medium ($1.0 \leq i < 2.0$), or high ($2.0 \leq i \leq 3.0$). All documents start with 0.0.

```
public void execute() throws ExecutionException {
   if (corpus.indexOf(document) == 0) {
      // Before analysing the first document: initialize the data for the corpus
   }

   // Analyse this document and record the votes

   if (corpus.indexOf(document) == (corpus.size() - 1)) {
      // After analysing the last document: compute the results,
      // generate RDF-XML, and store it as a corpus feature
   }
}
```

Figure 5: Hooks in a voting PR's *execute()* method. Note that *CorpusController.execute()* iterates through the corpus's documents in list order: opening each document, calling each PR's *execute()* method, and closing the document.

end of the `execute` on the last document in the corpus. (The RDF-XML is generated by matching and filling in templates, as described in §2.2.3.)

7. Some important annotations on the document are translated into RDF-XML according to a set of templates based on the features of each annotations. (This RDF-XML is also generated from templates.)

8. All the RDF-XML snippets are collected from the document and corpus features and consolidated into one output file for the datastore.

The "pipeline" outlined above actually consists of two GATE *SerialAnalyserController* instances (conditional corpus pipelines) serialized as `gapp` files and an XML configuration file for GATE's *Batch Learning PR* (**?**). Like the preprocessor, the main analysis tool is a command-line Java program that can be run in GNU screen on a server; it instantiates the two pipelines from the `gapp` files and creates a separate pipeline for the learning PR (which it instantiates from the configuration file); it then iterates through the corpora in a datastore, runs the pipelines in the correct sequence (taking advantage of GATE's serialization to save memory: only one document is loaded at a time), and finally collects together all the RDF-XML snippets stored as document and corpus features in the datastore and consolidates them into one RDF-XML file, which constitutes this tool's output to the CIM for that block (datastore).
A modified version is also used in the NeOn project[11] for batch processing large datastores through pipelines (both specified by command-line arguments) on a server.

2.2.2. Service categorization

Although the portal's Web 2.0 features encourage users to add, correct, and otherwise improve the category annotations of services, it is important to provide a number of reasonably good ones to start with so users will find the portal useful and interesting and then contribute to it—otherwise they would have to face 23 000 uncategorized services with only keyword searching. We therefore placed strong emphasis in the AA on rapid development and then refinement of service categorization. In this paper, we will summarize this task and describe the integration of the components into the pipeline above. Our scientific results here are interesting in their own right and are published in detail elsewhere (**?**).
Briefly, the task is to annotate each service with one or more of the 59 subclasses of *Category* for web services, such as *Genetics*, *Address Information*, and *Media Management*, arranged in a shallow tree (down to three levels below the top class) in seven main branches, such as *Business*, *Consumer*, and *Science* (**?**; **?**).
In the early stages, we had no training data but needed to generate some category annotations quickly for the portal, so we created *ad hoc* gazetteers of keywords and phrases based on the category names, synonyms, and related words. These were affected by the multipliers described in step 4 in §2.2.1. The IE pipeline at this stage was as shown in Figure 3 but without the machine learning (ML) categorization PR (step 5 in §2.2.1); the voting process treated keyword or phrase match as m (the multiplier) votes for the relevant category for each service associated with the document, and annotated each service with the two highest-scoring categories (an arbitrary limit agreed within the project).
After the first release of the portal, we manually annotated a few hundred services with the same portal features that users have for manually adding or correcting categories, and used these annotations to evaluate the AA's categorization and then as training data for machine-learning. We trained the ML PR to classify documents, carried out evaluations with various ML parameters, and integrated the ML PR into the pipeline by assigning a very high multiplier ($m = 100$) to the ML annotations so that in the subsequent voting PR, they will outweigh any gazetteer-based categorizations, although the latter can still be used to make the total number of categories per service up to two if the ML PR fails to classify any of a service's documents or provides only one category, since it is more user-friendly for us to provide approximate categorizations than none at all. We have published elsewhere (**?**) the full technical details and results of our ML experiments.

2.2.3. Approaches to ontology development and population

One can develop and populate ontologies from GATE applications using the GATE Ontology API (**?**), and we have

[11] `http://www.neon-project.org/`

used this technique in our SPRAT and SARDINE[12] applications (**?**; **?**), which use ontology design patterns to recognize new concepts, instances, and properties, and add them to the seed ontology (which can be empty) used to initialize the application. The principal output of both applications is the extended ontology produced from a corpus of documents. As an evaluated example, SPRAT processed 25 Wikipedia articles and produced 1058 classes, 659 subclasses, 23 instances, and 55 properties as output.

The CLOnE[13] and RoundTrip Oontology Authoring software (**?**; **?**) developed and used in SEKT[14] and NEPOMUK[15], also used the GATE Ontology API.

In the Service-Finder and MUSING tasks, however, the ontologies' class and property structures are fixed and our applications only need to create instances and property assertions, specifically in RDF-XML (as requested by other developers in the projects). The volume of data generated is also much larger and more time-consuming (as shown in Tables 1 and 3), so we use a template-filling technique which requires relatively little memory.

We originally developed this GATE PR for generating XML (principally RDF-XML) in the MUSING[16] business intelligence research project, and have used modified and improved versions of it in Service-Finder and CLARIN[17]. (A version of it will probably be integrated into GATE in the future, once we have settled the list of configurable features.) The PR's configuration file consists mainly of a series of template specifications, as Figure 6 shows. Each entry lists the annotation features that are required for the template to match, and the values of those features are substituted for the variables in the template. (The annotation can have other features, which are ignored.) The PR's output for a matching annotation consists of the "filled-in" copy of the element(s) in the `template` element. Figures 7 and 8 show the characteristics of a matching annotation and the resulting XML snippet, respectively. (In addition to the `feature` element, the entry can have a `generated` specification, which names a variable for which a UUID string is substituted. This is useful for generating unique `rdf:id` values.)

When the configuration file is loaded (during initialization of the PR), the order of the entries is preserved, so that for each annotation, the first match is used. It is therefore possible to use a template that requires an annotation with three features, followed by a simpler template which requires only two of the three (and uses a default value for the third, for example); the first template will "fire" whenever all three features appear, but the PR will drop back to the second one if the first does not match, and so on. (Another version of this component takes a *Map<String, String>* rather than a GATE Annotation; the voting PR generates the maps from the winning ballots and obtains the corresponding RDF-XML from this generator.)

2.3. Miscellaneous tools

We developed two other separate tools for the Service-Finder AA. The archiving tool iterates through the documents in the GATE datastores after the analysis tool has been run (leaving its annotations on the serialized documents) and produces Heritrix archives of the plain-text content of "interesting" (see step 4 in §2.2) HTML and PDF documents and selected strings from WSDLs and abstracts; the CIM produces Lucene indexes from these files to support keyword searches for services on the portal. The quantitative evaluation tool loads the RDF-XML files into a Sesame repository, executes a series of SERQL or SPARQL queries specified in a control file, and produces an output file containing the number of results for each query or a list of those results (according to the specifications in the control file).

To provide a datastore suitable for training the ML classifier, we added extra features to the preprocessor (§2.1), activated by additional command-line options, so that it reads a file of manual category annotations (exported from the CIM) as well as a set of archives from the SC and produces a datastore containing only the documents related to the manually annotated services, with document features representing the categories. We also developed a pipeline for training the classifier, which carries out steps 1 through 4 in §2.2 and treats the annotated documents as instances, and then saves the learned model for use in the complete integrated analysis tool.

3. Conclusion

The immediate result of the development presented here was its contribution to the successful completion of the Automatic Annotator tasks and their integration in the Service-Finder project, which received good intermediate and final project reviews. The relevant public deliverables (**?**; **?**) describe the AA software in much greater detail. We evaluated the AA software itself in two ways: IE measures (precision, recall, and F_1) for the especially important and scientifically interesting service categorization task, which we present in detail elsewhere (**?**); and quantitative measures of the instances and property assertions created at various stages of development (**?**).

The broader results included the dissemination of GATE as a tool for semantically annotating the results of focused web crawling—in particular at a *Future Internet Symposium* tutorial on web service crawling and annotation (**?**), where we demonstrated the suitability of the *GATE Developer* IDE and *GATE Embedded* library for rapid application development and effective code re-use—and the development of useful, reusable code shared with other projects (NeOn, MUSING, and CLARIN).

4. Acknowledgements

This research is partially supported by the EU Sixth Framework Program projects NeOn (IST-2005-027595) and MUSING (FP6-027097) and the Seventh Framework Program project Service-Finder (FP7-215876).

[12] Semantic Pattern Recognition and Annotation Tool; Species Annotation and Recognition and Indexing of Named Entities; both developed in NeOn.

[13] Controlled Language for Ontology Editing.

[14] `http://www.sekt-project.com/`

[15] `http://nepomuk.semanticdesktop.org/`

[16] `http://www.musing.eu/`

[17] `http://www.clarin.eu/`

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns:sfso="http://www.service-finder.eu/ontologies/ServiceOntology#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema#" >
...
<entry id="provider_homepage_link">
<feature name="provider_URL"  />
<feature name="homepage_link" />
<template>
  <sfso:Provider  rdf:about="${provider_URL}">
    <sfso:hasHomepage  rdf:datatype="xsd:string">${homepage_link}</sfso:hasHomepage>
  </sfso:Provider>
</template>
</entry>
...
</root>
```

Figure 6: Excerpt from the RDF-XML template file. To match this template, at least the `provider_URL` and `homepage_link` features must be provided.

Type	`Mention`
Start and end offsets	117–121
Underlying text	`Home`
Features	`doc_URL="http://www.serviceobjects.net/products/dots_phone_exchange_details.asp"`
	`homepage_link="http://www.serviceobjects.net/default.asp"`
	`provider_URL="http://seekda.com/providers/serviceobjects.com"`
	`service_URL="http://seekda.com/providers/serviceobjects.com/DOTSPhoneExchange"`

Figure 7: Annotation matching the template in Figure 6

5. References

S. Bird and M. Liberman. 1999. A Formal Framework for Linguistic Annotation. Technical Report MS-CIS-99-01, Department of Computer and Information Science, University of Pennsylvania. `http://xxx.lanl.gov/abs/cs.CL/9903003`.

K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham. 2004. Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, 10(3/4):349—373.

S. Brockmans, M. Erdmann, and W. Schoch. 2008. Hybrid matchmaker and Service-Finder ontologies (alpha release). Deliverable D4.2, Service-Finder Consortium.

Saartje Brockmans, Irene Celino, Dario Cerizza, Daniele Dell'Aglio, Emanuele Della Valle, Michael Erdmann, Adam Funk, Holger Lausen, and Nathalie Steinmetz. 2010. Final report on assessment of tests for beta release. Deliverable D7.5, Service-Finder Consortium, January.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*.

Brian Davis, Ahmad Ali Iqbal, Adam Funk, Valentin Tablan, Kalina Bontcheva, Hamish Cunningham, and Siegfried Handschuh. 2008. Roundtrip ontology authoring. In *Proceedings of the 7th International Semantic Web Conference (ISWC)*, Karlsruhe, Germany, October.

Emanuele Della Valle, Dario Cerizza, Irene Celino, Andrea Turati, Holger Lausen, Nathalie Steinmetz, Michael Erdmann, and Adam Funk. 2008. Realizing Service-Finder: Web service discovery at web scale. In *European Semantic Technology Conference (ESTC)*, Vienna, September.

A. Funk and K. Bontcheva. 2010. Ontology-based categorization of web services with machine learning. In *Proceedings of the seventh international conference on Language Resources and Evaluation (LREC)*, Valetta, Malta, May.

A. Funk, V. Tablan, K. Bontcheva, H. Cunningham, B. Davis, and S. Handschuh. 2007. CLOnE: Controlled Language for Ontology Editing. In *Proceedings of the 6th International Semantic Web Conference (ISWC 2007)*, Busan, Korea, November.

Adam Funk, Holger Lausen, and Nathalie Steinmetz. 2009a. Automatic semantic annotation component—beta release. Deliverable D3.4, Service-Finder Consortium, November.

Adam Funk, Holger Lausen, Nathalie Steinmetz, and Kalina Bontcheva. 2009b. Automatic semantic annotation research report—version 2. Deliverable D3.3, Service-Finder Consortium, October.

R. Grishman. 1997. TIPSTER Architecture Design Document Version 2.3. Technical report, DARPA. `http://www.itl.nist.gov/div894/894.02/-related_projects/tipster/`.

Y. Li, K. Bontcheva, and H. Cunningham. 2009. Adapting SVM for Data Sparseness and Imbalance: A Case Study on Information Extraction. *Natural Language Engineering*, 15(2):241–271.

```
<sfso:Provider rdf:about="http://seekda.com/providers/serviceobjects.com">
  <sfso:hasHomepage
    rdf:datatype="xsd:string">http://www.serviceobjects.com/default.asp
      </sfso:hasHomepage>
</sfso:Provider>
```

Figure 8: RDF-XML snippet output produced from the template in Figure 6 and the annotation in Figure 7

D. Maynard, A. Funk, and W. Peters. 2009a. Nlp-based support for ontology lifecycle development. In *Workshop on Collaborative Construction, Management and Linking of Structured Knowledge (CK 2009) at ISWC 2009*, October.

D. Maynard, A. Funk, and W. Peters. 2009b. Using lexico-syntactic ontology design patterns for ontology creation and population. In *Workshop on Ontology Patterns (WOP 2009) at ISWC 2009*, October.

R. Rivest. 1992. The MD5 message-digest algorithm. RFC 1321, Internet Engineering Task Force, April.

Kristinn Sigurðsson, Michael Stack, and Igor Ranitovic. 2008. Heritrix user manual. Software documentation, Internet Archive.

N. Steinmetz, A. Funk, and M. Maleshkova. 2009. Web service crawling and annotation (tutorial). In *Future Internet Symposium (FIS 2009)*, September.

Building the Scientific Knowledge Mine (SciKnowMine[1]): a community-driven framework for text mining tools in direct service to biocuration

Cartic Ramakrishnan[3], William A. Baumgartner Jr.[1], Judith A. Blake[2], Gully APC Burns[3], K. Bretonnel Cohen[1], Harold Drabkin[2], Janan Eppig[2], Eduard Hovy[3], Chun-Nan Hsu[3], Lawrence E. Hunter[1], Tommy Ingulfsen[3], Hiroaki 'Rocky' Onda[2], Sandeep Pokkunuri[4], Ellen Riloff[4], Christophe Roeder[1], Karin Verspoor[1]

Affiliation information:
[1] University of Colorado Denver, PO Box 6511, MS 8303, Aurora, CO 80045, USA
[2] The Jackson Laboratory, 600 Main Street, Bar Harbor, Maine 04609 USA
[3]Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292, USA
[4] University of Utah, 50 S. Central Campus Drive, Rm 3190 MEB, Salt Lake City, UT 84112-9205

E-mail: cartic@isi.edu, William.Baumgartner@ucdenver.edu, judith.blake@jax.org, gully@usc.edu, kevin.cohen@gmail.com, hjd@informatics.jax.org, jte@informatics.jax.org, hovy@isi.edu, chunnan@isi.edu, Larry.Hunter@ucdenver.edu, tommying@isi.edu, honda@informatics.jax.org, sandeepp@cs.utah.edu, riloff@cs.utah.edu, Chris.Roeder@ucdenver.edu, Karin.Verspoor@ucdenver.edu

Abstract

Although there exist many high-performing text-mining tools to address literature biocuration (populating biomedical databases from the published literature), the challenge of delivering effective computational support for curation of large-scale biomedical databases is still unsolved. We describe a community-driven solution (the SciKnowMine Project) implemented using the Unstructured Information Management Architecture (UIMA) framework. This system's design is intended to provide knowledge engineering enhancement of pre-existing biocuration systems by providing a large-scale text-processing pipeline bringing together multiple Natural Language Processing (NLP) toolsets for use within well-defined biocuration tasks. By working closely with biocurators at the Mouse Genome Informatics[2] (MGI) group at The Jackson Laboratory in the context of their everyday work, we break down the biocuration workflow into components and isolate specific targeted elements to provide maximum impact. We envisage a system for classifying documents based on a series of increasingly specific classifiers, starting with very simple surface-level decision criteria and gradually introducing more sophisticated techniques. This classification pipeline will be applied to the task of identifying papers of interest to mouse genetics (primary MGI document triage), thus facilitating the input of documents into the MGI curation pipeline. We also describe other biocuration challenges (gene normalization) and how our NLP-framework based approach could be applied to them.

1. Introduction

In biomedical research, organizations such as the NIH funded model organism databases or the Cochrane Collaboration systematically scan, read, evaluate and organize the published literature to provide formally structured resources that summarize individual fields of research. This effort, termed 'literature biocuration', is widely recognized as important to the scientific community (Bourne et al. 2006), and the promise that text-mining systems will be able to assist biocuration is long standing and well supported (Rebholz-Schuhmann et al. 2005).

Although suitable natural language processing (NLP) methods that can support biocuration (Hersh et al. 2005) exist, we assert that authentic computational support for biocuration work has not yet been delivered to the places where it is most needed. In this position paper, we describe the possible role that a framework-based approach might play in accomplishing this goal. In collaboration with the Mouse Genome Informatics group (MGI) (Bult et al. 2010), we seek to provide the necessary scalable computational support to speed up MGI biocuration. Every month, the biocuration staff process the contents of roughly 200 separate scientific journals in order to determine if each paper needs to be read in more depth or can be discarded as irrelevant to MGI's mission (this process is known as 'document triage'). Despite some success in measures of utility, systems developed in shared evaluations (Hersh, Cohen et al. 2005) have not been incorporated into the MGI curation workflow. We describe a community-based cyberinfrastructure project (called 'SciKnowMine') specifically designed *to accelerate biocuration*.

2. Goals of BioNLP development

Biomedical NLP (BioNLP) development should involve the creation of novel algorithms, systems and solutions that satisfy well-established global metrics. Such metrics permit the community to evaluate which methods are the most effective for a given task (as it does now).

[1] The SciKnowMine project is funded by NSF grant #0849977 and supported by U24 RR025736-01, NIGMS: RO1-GM083871, NLM: 2R01LM009254, NLM:2R01LM008111, NLM:1R01LM010120-01, NHGRI:5P41HG000330
[2] http://www.informatics.jax.org

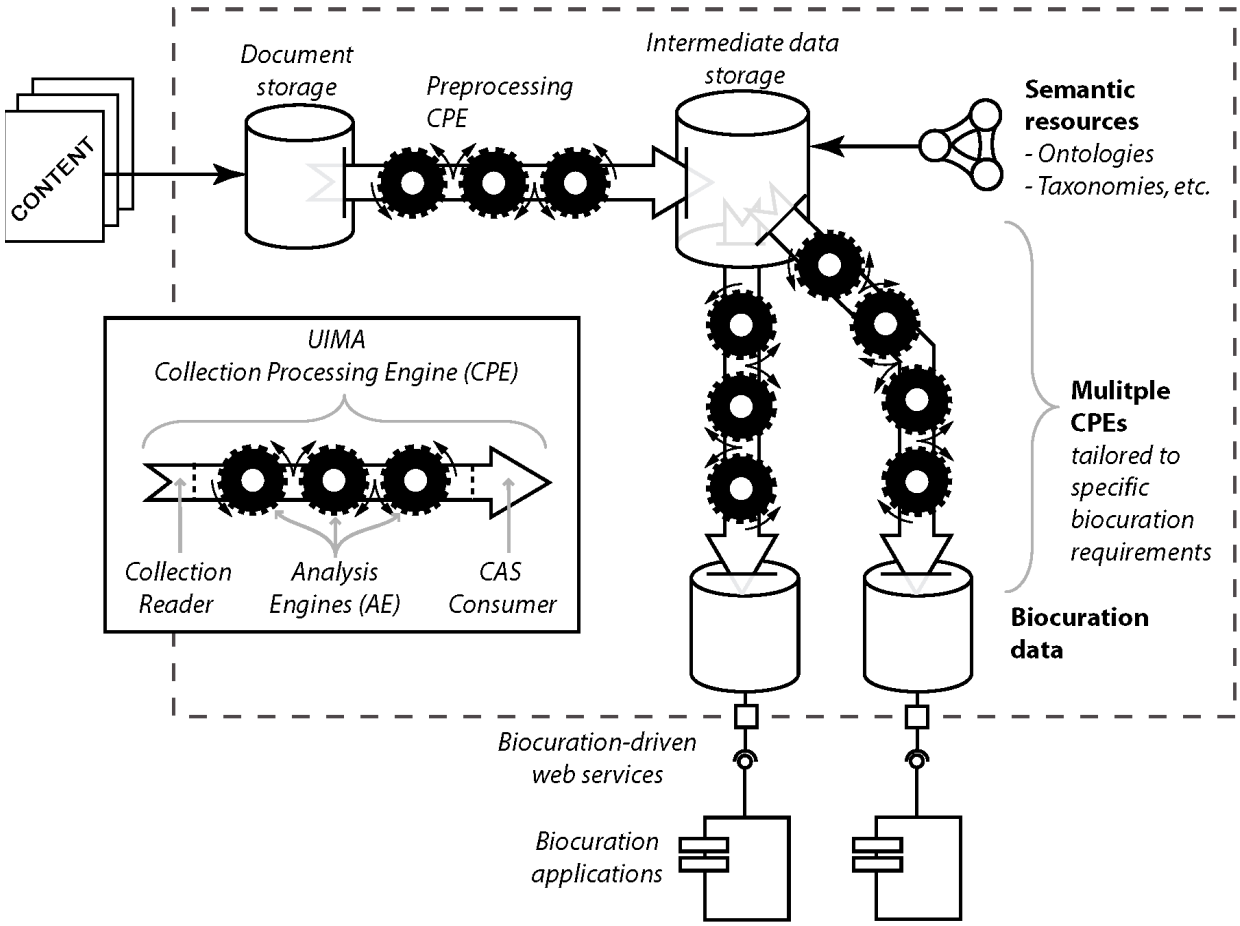


Figure 1 High-level design of SciKnowMine

We argue that this is not enough. It is essential that there exist a practical methodology for these systems to be used *in-situ* within biomedical databases' biocuration teams. There are a variety of individual components with similar high-level functions (*e.g.,* the classification of text spans within documents) from different research groups. Examples of such tools are the NCBO Annotator (Jonquet et al. 2009) that enables annotation of arbitrary concepts in text from existing biomedical ontologies, GOPubMed which organizes PubMed abstracts according to MeSH and Gene Ontology terms contained within them (Doms et al. 2005), the proprietary ProMiner system for terminology recognition in text (Hanisch et al. 2005), and the TextPresso system which recognizes terms based on regular expression (Müller et al. 2004). While each of these tools is potentially useful, various challenges have been encountered when trying to integrate them into a biocuration workflow, ranging from technical or licensing difficulties to inherent limitations in the applicability of the tools to the relevant full-text publications the biocurators work with.

A reliable middleware infrastructure providing a common platform where different components can easily be deployed together to develop a complete tool for an individual biocuraton task is therefore necessary.

The infrastructure used to support community evaluations is closely related to this idea, since these similar components must run on a shared system in order to be compared accurately. For example, the BioCreative meta-server[3] (Leitner et al. 2008) and the U-compare framework[4] (Kano et al. 2009) used in the BioNLP'09 evaluation each provide a mechanism for evaluating multiple systems against a common task, However, it has been recently argued that systems that perform well in tasks with intrinsic measures (such as F-Score, Precision and Recall) do not necessarily accelerate biocuration (Alex et al. 2008; Caporaso et al. 2008). This situation is being addressed in future evaluations (such as the BioCreative III workshop), but this raises the question of how performance can be measured. Extrinsic measures require direct measurement of usefulness within the context of a real-world use case whereas evaluation measures used in these shared tasks must be a computable substitute for the 'true measure'. In developing such a measure, it is therefore imperative that, at some stage, these systems are made to run within the direct context of a live biocuration task in a biomedical database.

A crucial barrier to success in this project is access to the full text biomedical literature for automated processing (Dowell et al. 2009). Although electronic access to individual publications is widely available through resources such as the National Library of Medicine's PubMedCentral and commercial subscription

[3] http://bcms.bioinfo.cnio.es/
[4] http://u-compare.org/bionlp2009.html

servers such as Elsevier ScienceDirect®, these mechanisms do not provide bulk access to entire collections, as is necessary for automated processing of the full range of publications relevant to biocuration. Attempts to use APIs or web interfaces to get systematic access to entire collections are blocked, despite the apparent rights of subscribers, or in the case of PubMedCentral, the public, to this material. A critical need is to reconcile the requirements for automated processing of publications in bulk with the concerns and technological capabilities of publishers and the National Library of Medicine. Our approach of reusing existing NLP components as open-source software for use by the BioNLP community will make NLP expertise available for publishers who would otherwise not have access to this technology. Since access to full-text articles for these text-mining efforts is a stumbling block, discussions regarding these issues are underway with Elsevier labs. We are building a single demonstration implementation that can produce measurable acceleration of the biocuration process at MGI coupled with active development of novel NLP technology. This work requires a collaborative effort between multiple groups (U. Colorado, USC ISI, U. Utah and JAX).

3. Goals of BioNLP development

SciKnowMine is a large-scale text-processing pipeline based on the BioNLP-UIMA project (Baumgartner et al. 2008). UIMA is the Unstructured Information Management Architecture (Ferrucci et al. 2004), and is available as an Apache open source project. As shown in Figure 1, SciKnowMine brings together multiple BioNLP toolsets in a UIMA implementation. UIMA provides a way of defining 'Collection Processing Engines' (CPEs). Each CPE is defined in a three stage cascade consisting of a 'Collection Reader' (which iterates over documents to initiate processing), a series of 'Analysis Engines' (which add meta-data, often in the form of text annotations, to the CAS or 'Common Analysis Structure' UIMA document representation), and finally a 'CAS Consumer' (which formats the final CAS data and writes it to output). The first step of our processing is designed to upload remote files, store them locally, execute a set of standard preprocessing tasks (*e.g.* tokenization, sentence splitting, etc.) and then store a local set of partially annotated data. We propose to wrap several different implementations of these standard preprocessing tasks as AEs (see also Section 3.4). We will then develop a library of CPEs specifically tailored for biocuration tasks that operate on this set of pre-annotated texts. These CPEs will be made available as web-services that specifically deliver biocuration functionality to end-user systems in a structured way. Our objectives are to prototype, develop and scale up this infrastructure in order to convert the entire primary research literature into an online resource that can then be uniformly accessed by both BioNLP researchers and biocuration teams, enabling the development of powerful new accelerated methods of biocuration.

3.1 A Repository of UIMA Analysis Engines

SciKnowMine takes a community-centered approach to building its biocuration workflows. We use publicly available tools when applicable and construct new tools when needed and release these new tools as open source software. We take advantage of existing public repositories containing UIMA components including (1) the BioNLP-UIMA framework (2) resources from the Julie lab[5] (3) the UIMA sandbox and (4) U-Compare (Kano, Baumgartner et al. 2009). Every UIMA component is dependent on a defined set of data types that are specified by that component's 'type system'. One challenge is that we must integrate across multiple UIMA type systems since components that use different type systems cannot work with each other. The SciKnowMine infrastructure uses a generic, domain-independent type system capable of expressing the wide range of types necessary to support biocuration (Verspoor et al. 2009). It is easily adaptable to external type systems, and is already compatible with U-Compare. With this approach, as SciKnowMine progresses, it will not only result in a series of biocuration workflows, but will also amass a collection of UIMA components compatible with a single type system that could be made publicly available.

Almost all the components discussed in this paper already exist, mostly taken from well-known external resources. Our planned contribution is assembling them into a single framework and extending the modules where necessary to obtain a useful, scalable and seamless end-to-end support system for MGI. This will be made available to the MGI biocuration team as a web service. We will make public Collection Processing Engines (CPEs) for certain exemplar text processing tasks along with the component Analysis Engines (AE) and collection readers for such tasks. These will serve as templates for the community to deploy and test their implementations of individual AEs on a large corpus of biomedical text relevant to focused biomedical research groups like MGI.

3.2 Scaling up SciKnowMine

We will explore three ways to scale up processing in order to meet the goal of analyzing large collections of text in reasonable amount of time. Methods for scaling up UIMA processing range from adding more threads to the CPE processor (the Collection Processing Manager, CPM), to scaling out to more hardware using UIMA-AS, to Hadoop cloud computing[6]. Given a large, shared-memory, multi-processor machine, a lot can be accomplished by specifying more threads in a CPE. The pipeline's primary data structure, the CAS, is shared between engines in memory. Running more than one instance of the pipeline allows for parallel execution of each engine, including any relatively slow engines. As a scaling method in UIMA, this is effective, but limited.

[5] http://www.julielab.de/Resources/Software
[6] http://hadoop.apache.org

Each engine in the pipeline is duplicated for each thread, so engines that consume a large amount of memory would also be duplicated.

Since the discrete nature of document processing allows for independent processing, a small cluster of single CPU, multi-core machines can be as effective as a single more powerful shared-memory machine, at a fraction of the cost. Using UIMA-AS (Asynchronous Scale-out) allows different analysis engines from a single CPE to be located on different machines. In this way, slow engines that need multiple instances can be duplicated, while faster engines can be run with single instances with engines connected through the use of message queues. This allows for asynchronous access and makes distributing work and collecting results easy. UIMA has been adapted to Hadoop, a MapReduce (Dean et al. 2004) implementation, in a project called Behemoth[7]. It makes use of UIMA Processing Engine Archive (PEAR) packaging so that Hadoop and the Hadoop Distributed File System (HDFS) can manage distributing the code and data files across a much larger cluster. Not everyone has access to thousands of nodes, but Hadoop cluster time is available for rent on Amazon's Elastic Compute Cloud (EC2)[8] as Elastic MapReduce[9]. The economics make it worth considering.

3.3 Access to PDF content

Given the ubiquity and familiarity of PDF documents, building effective methods for extracting and processing text from PDF files is a high priority. We use a combination of machine-learning and rule-based approaches to render and extract text as a UIMA Collection Reader. This approach is an open-source component that has been used in text mining studies of neuroanatomical experiments (Burns et al. 2007). We plan to make this PDF extraction technology available as an open-source UIMA analysis engine.

3.4 Incorporating new NLP research

We intend to incorporate novel NLP approaches into our system by using UIMA as a central representational framework and working with NLP researchers to build methods to wrap their tools as UIMA components. We follow the general approach of having each NLP system produce annotations that are attached to the text(s) in appropriate places, resulting in a steady accretion of information within and around each text. We follow the stand-off model of annotation which is inherent to the UIMA data structures. Annotations produced by multiple components – even annotations of the same type, e.g. different tokenizations or gene mention annotations – can exist in parallel and be made available for downstream analysis. Each downstream component can choose to use whichever annotations it believes to be the most useful for its task, perhaps even using multiple sets of annotations of the same kind (e.g., a component could utilize the annotations produced by three different named entity recognizers to maximize coverage).

[7] http://code.google.com/p/behemoth-pebble
[8] http://aws.amazon.com/ec2/
[9] http://aws.amazon.com/elasticmapreduce/

4. Knowledge Engineering Study of the MGI Biocuration Workflow

A key feature of this project is our approach to understanding the biocuration workflow being used at MGI. This approach is modeled loosely after the CommonKADS methodology (Schreiber et al. 1999). Using the UIMA framework it is possible to deploy an automated biocuration engine with relative ease. However, given the well-known shortcomings of automating biocuration shown in previous work, we plan to use the system as a human aide, and thus to integrate it with the human curators' workflow. In order to minimally disrupt the existing well-honed procedures and to obtain as much guidance for automated processing as possible, this integration requires careful consideration of several issues.

- At which point during the manual biocuration should the intermediate results of automated curation be made known to the biocurators?
- How should the system inform the biocurator of these results so as to be least intrusive?

These issues have motivated us to conduct studies in modeling workflows of manual biocuration. We used UML 2.0 activity diagrams to model the activities of different curator teams to extract information from the literature. Although this approach is not strictly formal, it does provide a useful framework for exploring questions such as: 'Which tasks take the longest?' 'Where are the most prominent curation bottlenecks?' Our preliminary investigations have helped us identify three MGI curation operations that are candidates for acceleration via computational support.

4.1 MGI Triage Automation Tools

We view the triage task as a document classification task that ranks documents in order of likelihood of interest for further analysis. Biocurators can then vary parameters to learn how they characterize the likelihood thresholds to include documents in the system or not. Our approach is to build a series of increasingly specific classifiers, starting with very simple surface-level decision criteria and gradually introducing more sophisticated NLP. The current baseline is that a document is included if it contains the words *mouse, mice* or *murine* (unless the words appear in the Bibliography section only). Subsequent levels involve setting zone-specific classification decisions (such as the presence of 'stigma words' within methods sections, *etc.*), the use of word combinations (bigrams, trigrams, *etc.*) in these decisions, the use of topic model signatures derived from language modeling, and at the highest level the development of structured linguistic information extraction frames. In keeping with our objective of minimal disruption of the manual biocuration process, our automated triage system will rank the documents downloaded and provide for

each one its classification suggestion(s), together with an indication of its confidence. Human curators will use this to determine the confidence level at which the system's judgments are trustworthy.

4.2 Gene Normalization Tools

'Gene normalization' refers to a mapping of mentions of genes or proteins in text to an appropriate database identifier. This is challenging due to species ambiguity in the text (genes in different organisms often share names) and the widespread use of acronyms and abbreviations. Solutions to this problem could be integrated into a biocuration process to help curators assess the relevance of a particular paper to their target area, as well as focus the curator's attention to specific parts of the text that mention particular genes. Gene normalization has been the focus of several recent challenge tasks in BioCreative II (Krallinger et al. 2008) and II.5 (Mardis et al. 2009). The state-of-the-art performance is currently achieved by the GNAT system (Hakenberg et al. 2008). Currently, MGI is incorporating gene normalization tools independently of the triage process. Our task would be to incorporate such a tool into the triage task.

4.3 Event Recognition Tools

Protein-protein interactions have been the most common candidate for biological event extraction from the earliest studies (Blaschke et al. 1999; Craven et al. 1999), to the latest competitions like BioCreative II and II.5. Research has also extended to other types including those focused on in the recent BioNLP'09 challenge (Kim et al. 2009): (a) gene expression, (b) transcription, (c) protein catabolism, (d) protein localization, (e) binding, (f) phosphorylation, (g) regulation, (h) positive regulation, and (i) negative regulation. The needs of MGI will require extension to novel composite semantic types, such as 'phenotype'. Phenotypes are observable attributes of an organism caused by myriad underlying factors. Identifying them requires extracting information on chromosomal locations, polymorphisms, Gene Ontology terms, protein domains, and experimental assays; all of these information extraction tasks are either novel or demonstrably difficult but if solved, could have a large impact. We have begun experiments with information extraction pattern learning (Riloff 1996) in order to address some of these tasks.

5. Conclusion

The work described in this paper is currently in the preliminary stages. We have collected a representative corpus of documents to serve as training data for classifiers within the biocuration pipeline, and begun the design of the classifier. We have also engaged the MGI biocurators in a requirement elicitation process to build models of their workflows. Experiments are also underway to tune our PDF extraction system to extract text from the MGI journals.

We have described a fundamental (even formative) unsolved challenge in the field of BioNLP and present a community driven approach that directly leverages NLP Frameworks to solve it. SciKnowMine is an effort to leverage the BioNLP community's expertise to solve that challenge in a general way that can be used across different biocuration systems.

6. References

Alex, B., C. Grover, et al. (2008). "Assisted curation: does text mining really help?" Pacific Symposium Biocomputing: 556-67.

Baumgartner, W., B. Cohen, et al. (2008). "An open-source framework for large-scale, flexible evaluation of biomedical text mining systems." Journal of Biomedical Discovery and Collaboration 3: 1.

Blake, J., J. Eppig, et al. (2006). "The Mouse Genome Database (MGD): updates and enhancements." Nucleic Acids Research 34(suppl_1): D562-567.

Blaschke, C., M. A. Andrade, et al. (1999). "Automatic extraction of biological information from scientific text: protein-protein interactions." ISMB: 60-67.

Bourne, P. E. and J. McEntyre (2006). "Biocurators: contributors to the world of science." PLoS Computational Biology 2(10): e142.

Bult, C. J., J. A. Kadin, et al. (2010). "The Mouse Genome Database: enhancements and updates." Nucleic Acids Research 38(Database issue): D586-92.

Burns, G., D. Feng, et al. (2007). Infrastructure for Annotation-Driven Information Extraction from the Primary Scientific Literature: Principles and Practice. 1st IEEE Intl. Workshop on Service Oriented Technologies for Biological Databases and Tools (SOBDAT 2007), Salt-Lake City.

Caporaso, J. G., N. Deshpande, et al. (2008). "Intrinsic evaluation of text mining tools may not predict performance on realistic tasks." Pacific Symposium Biocomputing: 640-51.

Clement Jonquet, Nigam H. Shah, Mark A. Musen, The Open Biomedical Annotator, AMIA Summit on Translational Bioinformatics, p. 56-60, March 2009, San Francisco, CA, USA.

Craven, M. and J. Kumlien (1999). Constructing biological knowledge-bases by extracting information from text sources. Proceedings of the Seventh ISMB.

Dean, J. and S. Ghemawat (2004). MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004.

Doms, A. and M. Schroeder (2005). "GoPubMed: exploring PubMed with the Gene Ontology." Nucleic Acids Research 33(suppl_2): W783-786.

Dowell, K. G., M. S. McAndrews-Hill, et al. (2009). "Integrating text mining into the MGI biocuration workflow." Database : The Journal of biological databases and curation 2009(0).

Ferrucci, D. and A. Lally (2004). "Building an example application with the unstructured information management architecture." IBM Syst. J. 43(3): 455-475.

Hakenberg, J., C. Plake, et al. (2008). "Inter-species normalization of gene mentions with GNAT." Bioinformatics 24(16): i126-132.

Hanisch, D., K. Fundel, et al. (2005). "ProMiner: rule-based protein and gene entity recognition." BMC Bioinformatics 6(Suppl 1): S14.

Hersh, W., A. Cohen, et al. (2005). "TREC 2005 Genomics Track Overview."

Kano, Y., W. Baumgartner, et al. (2009). "U-Compare: share and compare text mining tools with UIMA." Bioinformatics 25(15): 1997-1998.

Kim, J.-D., T. Ohta, et al. (2009). Overview of BioNLP'09 shared task on event extraction. Proceedings of the Workshop on BioNLP: Shared Task. Boulder, Colorado, ACL.: 1-9.

Krallinger, M., A. Morgan, et al. (2008). "Evaluation of text-mining systems for biology: overview of the Second BioCreative community challenge." Genome Biology 9(Suppl 2).

Leitner, F., M. Krallinger, et al. (2008). "Introducing meta-services for biomedical information extraction." Genome Biology **9**(Suppl 2): S6.

Müller, H.-M., E. Kenny, et al. (2004). "Textpresso: An Ontology-Based Information Retrieval and Extraction System for Biological Literature." PLoS Biol **2**(11): e309

Mardis, S., F. Leitner, et al. (2009). BioCreative II.5: Evaluation and ensemble system

Rebholz-Schuhmann, D., H. Kirsch, et al. (2005). "Facts from text--is text mining ready to deliver?" PLoS Biol 3(2): e65.

Riloff, E. (1996). Automatically Generating Extraction Patterns from Untagged Text. (AAAI-96), 1996, pp. 1044-1049.

Schreiber, G., H. Akkermans, et al. (1999). Knowledge Engineering and Management: The CommonKADS Methodology, {The MIT Press}.

Verspoor, K., W. Baumgartner, et al. (2009). Abstracting the Types away from a UIMA Type System. From Form to Meaning: Processing Texts Automatically. C. Chiarcos, Eckhart de Castilho, Stede, M.: 249-256.

JULIE Lab's UIMA Collection Reader for WIKIPEDIA

Elena Beisswanger **Udo Hahn**

Jena University Language & Information Engineering (JULIE) Lab
Friedrich-Schiller-Universität Jena
Fürstengraben 30, 07743 Jena, Germany

{elena.beisswanger|udo.hahn}@uni-jena.de

Abstract

WIKIPEDIA, a huge, collaboratively built Web encyclopedia, is gaining increasing importance as a lexico-semantic resource for a large variety of natural language processing tasks. However, other than 'well-defined' and pre-formatted resources such as WORDNET, the ease of usability of its articles for text analytics is severely hampered due to underspecified document structure descriptions. To overcome this shortcoming, we here introduce a JAVA-based collection reader for WIKIPEDIA articles that is fully integrated in the Unstructured Information Management Architecture (UIMA). It imports articles from a WIKIPEDIA database, parses their raw text, composes a cleansed document text version and retains the original document structure in terms of UIMA annotations. We describe the structure and design of the WIKIPEDIA Reader and introduce the tools we incorporated, *viz.* UKP Lab's JWPLDataMachine for setting up the database and the JWPL parser for parsing the wiki markup. In addition, we briefly introduce a scheduling system (in which the WIKIPEDIA Reader is integrated) that enables running several NLP pipelines in parallel, each with its own instance of the reader.

1. Introduction

In this resource-greedy age of human language technology, we currently witness a move from first-generation resources (often small, yet carefully crafted by guideline-trained linguistic and domain experts) to second-generation resources. These are hosted on the Web, typically huge, and – according to the Web 2.0 spirit – collaboratively produced by a crowd of people with varying professional training background and domain competence, without strictly adhering to agreed-upon quality standards.

This move holds, in particular, for semantic resources. After years of exploitation of lexical resources from the WORDNET family[1] for a plethora of NLP tasks lots of language engineers, additionally, are considering Web resources such as WIKIPEDIA[2] as an information-rich and large-coverage alternative. The tasks being dealt with are also pretty diverse, ranging from text categorization (Janik and Kochut, 2008) and grading semantic relatedness between texts (Gabrilovich and Markovitch, 2009) to large-scale taxonomy learning (Ponzetto and Strube, 2007) using WIKIPEDIA.

While WORDNET comes as a structured lexicon database, with short definitional phrases (glosses) and explicit semantic relations (synonymy, hyponymy, partonymy, etc.), WIKIPEDIA can be characterized as a set of Web documents forming a hyperlinked encyclopedia, whose semantic ties are far less evident and uniform. Either they are made explicit through links between pages or descriptor categories manually assigned to pages, or they are implicitly buried in the full text body of an article (which, apart from tables and table-style WIKIPEDIA infoboxes, lacks further structure). Still, from a content perspective, WIKIPEDIA articles compared with WORDNET entries seem richer, broader and deeper in thematic scope and are certainly more up to date.

Yet, there is no free lunch – WIKIPEDIA articles require an in-depth NLP analysis to make use of this implicit contents, semantic relations going beyond plain category labels, in particular. Even more, prior to making use of the information contained in WIKIPEDIA articles they have to be reformatted such that the original wiki markup with embedded hyperlinks is converted into a structurally richer text format that can then be submitted to linguistic analytics.

In this paper, we deal with such a document converter, a collection reader in UIMA speech.

2. WIKIPEDIA

2.1. WIKIPEDIA Features

WIKIPEDIA is a non-profit, multilingual, Web-based encyclopedia supported by the Wikimedia Foundation. Currently, 272 official WIKIPEDIAs exist, each written in a different natural language, also differing considerably in size, scope and quality. WIKIPEDIA articles are collaboratively written and updated by members of the Web community, with some authoring policies in mind to which all collaborators should commit (e.g., the neutrality of views and plurality of opinions), and with authoritative editors resolving conflicts and generating some level of consensus.[3]

WIKIPEDIA is a hypermedia system which allows to integrate additional media types into an article and to complement plain text by tables, figures, etc. It also provides means for cross-linking articles. At a meta level of content description, the articles may (but need not necessarily) contain user defined descriptor categories intended to semantically describe the field or topics an article deals with. The English WIKIPEDIA is by far the largest one, with over 3 million article pages, tagged with around 500,000 different content categories, followed by the German WIKIPEDIA with over 1 million articles. WIKIPEDIA is rapidly growing, day by day, and due to the vast number

[1] http://wordnet.princeton.edu/

[2] http://www.wikipedia.org/

[3] See, for example, http://meta.wikimedia.org/wiki/Wikimedia_principles

of active users and collaborators is up to date at a level unmatched so far in the encyclopedic domain.

2.2. WIKIPEDIA Challenges

Given these editorial principles and the distribution of responsibilities among many authors, WIKIPEDIA also inherits some technical challenges and even suffers from obvious pitfalls. Below we summarize them from a technical, a linguistic and a content quality perspective and we describe how our WIKIPEDIA Reader copes with them.

Technically, WIKIPEDIA articles adhere to the (idiosyncratic) MEDIAWIKI syntax. Since it lacks a clear formal definition and, meanwhile, numerous extensions have been defined, the development of full-coverage parsers is intrinsically difficult. This distinguishes WIKIPEDIA sources from ones in fairly common data formats such as XML, where the user community enjoys much more reliable, fully operational high-performance document parsers.

In the implementation phase for our WIKIPEDIA Reader, we decided on purpose against using the only full-text parser for wiki markup that we know about, the PHP-based parser of the MEDIAWIKI project itself, since it suffers from "labyrinthine code"[4] and is known for being quite slow. Instead, we chose the JWPL parser[5] developed by the Ubiquitous Knowledge Processing (UKP) Lab at Technische Universität Darmstadt. It is a lightweight, JAVA-based parser with reasonable performance that represents the wiki markup structure in terms of JAVA objects. However, we had to modify and enhance the parser to serve our needs.

Most WIKIPEDIA articles contain textual content in different formats – paragraphs of text, but also lists, caption headings, tables, and MEDIAWIKI templates. Hence, from a linguistic perspective, one not only encounters (sometimes very complex and long) sentences but also noun phrases or even just lists of keywords. Considering all these varieties of utterances puts high demands on NLP-style syntactic analytics. To alleviate this issue, the WIKIPEDIA Reader omits potentially problematic stretches of text originating from templates. It may also be configured to skip table bodies, too (keeping only the table captions though), by setting the descriptor parameter **SkipTableContents** to true.

With regard to WIKIPEDIA contents, one has also got to be selective. Within one WIKIPEDIA article, sections differ as to the relevance for further analysis. There are a number of section types that one might reasonably want to exclude systematically from content-oriented analysis. Thus we introduced the descriptor parameter **SectionsToSkip** allowing for a customized list of designators of sections to be excluded, such as "References", "External links", "Further readings", "See also", or "Footnotes". Also different names for the same type of section to be skipped may be listed, such as "References", "Bibliography" or "Literature" for the reference section.

Besides proper article pages, WIKIPEDIA also contains so-called disambiguation pages which help distinguish between different topics that share a common name. They mainly consist of links redirecting to the appropriate article pages. To let the reader skip disambiguation pages the configuration parameter **OnlyArticles** is required.

WIKIPEDIA contains a particularly high proportion of articles dealing with popular culture (music, movies, artists, actors, etc.). Running an NLP pipeline on the whole of WIKIPEDIA might, therefore, introduce an unwarranted bias in the analysis results. One might also speculate whether this abundance of culture-specific issues and names will increase the likelihood of additional lexical ambiguities. To intentionally restrict the input of the WIKIPEDIA Reader to a subset of WIKIPEDIA the corresponding article pages must be marked in the database from which the WIKIPEDIA Reader solicits its data (see the WIKIPEDIA Reader documentation). Alternatively, especially for defining smaller fractions of WIKIPEDIA, the configuration parameters **PageList** and **CategoryList** can be used (see Section 4.3.).

Notwithstanding the concerns raised above, the varying quality of WIKIPEDIA articles with respects to coverage, granularity, category assignment, trustworthiness, language, layout and style, remains a fundamental problem that we cannot address further since it is clearly beyond the scope of our WIKIPEDIA Reader.

3. UIMA

The Unstructured Information Management Architecture[6] (UIMA) is a middleware platform for structuring unstructured data such as natural language, images, movies or music via some analysis pipeline (Ferrucci and Lally, 2004). The structures being assigned by such analytics are metadata, so-called annotations.

In UIMA pipelines, two major types of components are distinguished. First, Collection Readers account for the input data being processed by importing documents from a remote repository and representing them in a UIMA-internal data structure, the Common Analysis Structure (CAS). The CAS is the central data structure through which all UIMA components communicate. Second, Analysis Engines perform the actual structure-building analysis (such as, for NLP tasks, tokenization, POS tagging or parsing) by creating annotations and adding them to the CAS. Which annotation types a UIMA component can create depends on the type system(s) being used. Analysis components work on a per CAS basis. Yet, UIMA provides additional support for applying analysis components to data collections.

4. WIKIPEDIA Reader

The WIKIPEDIA Reader is designed as a UIMA collection reader, i.e., it implements the UIMA `org.apache.uima.collection.CollectionReader` interface. In procedural terms, it reads WIKIPEDIA pages from a database (see Section 4.4.), preprocesses the raw page text (see Section 4.2.), parses it, composes the markup-free document text, creates annotations marking text fractions as paragraph, list, caption, etc., and adds the document text together with the annotations to a new CAS (see Section 4.1.). The CAS becomes the UIMA internal representation of the

[4] `http://www.mediawiki.org/wiki/Alternative_parsers`

[5] `http://www.ukp.tu-darmstadt.de/software/jwpl/jwpl-parser/`

[6] `http://incubator.apache.org/uima/`

Parameter	Data Type	Description	Mandatory	Default
DataBaseServerURL	String	database server URL	yes	-
DataBase	String	name of the database	yes	-
DataBaseUser	String	database user name	yes	-
DataBasePassword	String	database user password	yes	-
DataBaseDriver	String	database driver	no	`com.mysql.jdbc.Driver`
BatchSize	Integer	number of documents fetched from the DB at once	no	100
OnlyArticles	Boolean	omit disambiguation pages	no	true
PageList	String list	pages to be considered	no	-
CategoryList	String list	categories to be considered	no	-
ImageIdentifiers	String list	WIKIPEDIA image identifiers	no	"Image"
CategoryIdentifiers	String list	WIKIPEDIA category identifiers	no	"Category"
SkipTableContents	Boolean	omit table contents, while keeping the captions	no	true
SectionsToSkip	String list	omit listed sections	no	"References", "Footnotes", etc. (see documentation)
Language	String	language of the WIKIPEDIA used	no	"en"
AddMetaData	Boolean	create descriptor annotation	no	false

Table 1: WIKIPEDIA Reader configuration parameters.

WIKIPEDIA page. The reader is configurable via an XML descriptor (see Section 4.3.).

4.1. From WIKIPEDIA Pages to CASes

To create a UIMA-internal representation of a WIKIPEDIA page, our WIKIPEDIA Reader traverses through the following steps subsequently discussed in more detail:

- Retrieve page from the WIKIPEDIA database.
- Create header and descriptor annotations.
- Parse the wiki markup of the page.
- Compose the document text.
- Create annotations for text segments and links.

First of all, the page information (id, text, meta information) is retrieved from the database. Second, a header annotation is created, including the page id, page type (article or disambiguation page), the title of the page, and the language of the chosen WIKIPEDIA article. If the value of the configuration parameter **addMetaData** has been set to true, a descriptor annotation is created providing meta information about the page, such as the category labels the page is tagged with. Next, the WIKIPEDIA Reader parses the wiki markup of the page. The parser outputs JAVA objects corresponding to the different content items of the WIKIPEDIA page (sections, paragraphs, lists, tables, and image captions). Then content items are handled in consecutive order. From each item the text is extracted, cleansed (see Section 4.2.) and added to the CAS document text. If the parameter **SkipTableContents** is switched on, table bodies are omitted and only table captions are considered. To preserve the information from which content item a particular text fragment has been derived from, an appropriate UIMA annotation is attached to it. The annotation types being used are defined in an extended version of the JULIE Lab type system (Hahn et al., 2007). For example, to the text derived from a paragraph an annotation of type `de.julielab.jules.types.Paragraph` is attached. In addition, for all WIKIPEDIA internal links occurring in any of the text segments `de.julielab.jules.types.wikipedia.Link` annotations are created holding the link target as value of the `target` feature.

4.2. Text Cleansing

To render a properly processable document text to subsequent text analytics the WIKIPEDIA Reader processes the WIKIPEDIA page text prior to and after parsing.
Parsing MEDIAWIKI documents is a tricky job. Besides the proper MEDIAWIKI markup elements, also some HTML tags are contained. In addition, numerous extensions to the MEDIAWIKI syntax have been developed, some of them defining their own XML tags. The parser the WIKIPEDIA Reader incorporates has limited abilities to deal with XML and HTML tags. We use a configuration of the parser that drops all tags while keeping the enclosed text.
While this is a reasonable choice for presentational and structural markup, it may cause problems for XML tags from MEDIAWIKI extensions marking special, from our perspective irrelevant, contents. Examples are "<ref>" from the "Cite" extension marking references, and "<timeline>" from the "EasyTimeline" extension marking wiki markup from which embedded images are produced. The WIKIPEDIA Reader deletes such contents prior to parsing.
After parsing, yet before the text of particular content items of a WIKIPEDIA page becomes part of the document text, it is cleansed and adapted as described in the WIKIPEDIA Reader documentation. For example, empty brackets and quotes and invalid XML characters are removed.

4.3. Configuration Parameters

The WIKIPEDIA Reader is configurable in a flexible way. Five parameters are provided to specify the connection to the WIKIPEDIA database (**DataBaseServerURL**,

DataBase, **DataBaseUser**, **DataBasePassword**, and **DataBaseDriver**). Another parameter, **BatchSize**, controls how many WIKIPEDIA pages the WIKIPEDIA Reader fetches at once. To restrict the analysis to a subset of WIKIPEDIA pages, the parameters **OnlyArticles**, **PageList** and **CategoryList** can be used. If **OnlyArticles** is valid, only proper articles are processed, while disambiguation pages (that are also contained in the database) are skipped. In case **PageList** contains page titles, only those pages which match these titles are considered for analysis, ignoring the **CategoryList** parameter. However, if **PageList** is empty, but **CategoryList** contains category titles, only those pages tagged with at least one of the listed categories are considered. If neither pages, nor categories are specified, all pages in the database are considered by the WIKIPEDIA Reader. Note that there is another possibility to restrict the WIKIPEDIA Reader to particular WIKIPEDIA pages: the pages can be marked in the database directly.

The parameters **ImageIdentifiers** and **CategoryIdentifiers** contain information for the wiki markup parser, specifying which (language-dependent) image and category link identifiers apply for the chosen WIKIPEDIA source. The parameter **SkipTableContents** allows to control the inclusion or exclusion of table bodies. Table captions are considered independent from this parameter value. Via the parameter **SectionsToSkip** titles of sections can be specified that should be excluded from the analysis. The parameter **Language** selects which WIKIPEDIA will be accessed. Each CAS will get the specified language id as part of its meta information. Setting the parameter **AddMetaData** to true lets the reader create a descriptor annotation for each CAS, holding meta data of the WIKIPEDIA page such as associated category labels. All these parameters are summarized in Table 1.

4.4. WIKIPEDIA Database

Basically, the WIKIPEDIA database contains the id, title and text of WIKIPEDIA pages plus some additional information such as the category labels associated with the pages. We set up the database in two steps. First, we used the UKP Lab's JWPLDataMachine tool. It supports the import of WIKIPEDIA articles into a database (only pages in the article name space are considered) and establishes the required table structure, taking publicly available WIKIPEDIA XML dumps as input (containing the last revision of pages only). Second, we slightly extended the table structure to be able to mark WIKIPEDIA pages for inclusion / exclusion for further analysis and to track the processing status of pages. Details are given in the WIKIPEDIA Reader documentation.

4.5. Embedding in a Scheduling System

In order to process a large document collection such as WIKIPEDIA it is desirable to run several pipelines in parallel. When multiple instances of a collection reader simultaneously access the same document collection, a scheduling system is required to keep track of the processing status of each document. We established such a scheduler based on setting flags in the WIKIPEDIA database ("raw", "in process", and "processed"). In essence, it runs as follows:

- In the initial state, all WIKIPEDIA pages in the database are marked as "raw".
- When a pipeline is running, the WIKIPEDIA Reader queries the database for raw pages, batch-wise. If raw pages are still available, the reader retrieves them and changes their status from "raw" to "in process".
- When all analysis components of the pipeline have terminated processing a CAS, a listener component attached to the pipeline keeps track of the termination and changes the status of the corresponding page in the database from "in process" to "processed".
- When all pages have been marked as "processed", the pipeline eventually stops.

Further information about the scheduling system is contained in the WIKIPEDIA Reader documentation.

4.6. Availability and Usage

The WIKIPEDIA Reader is available as UIMA PEAR package from our website at `http://www.julielab.de/`. The PEAR package includes the required UIMA types and a detailed documentation of the component. The tools provided by the JULIE Lab are licensed under the terms of the Common Public License. However, note that the JWPL library used by our reader is freely available for academia but not open source yet (March 2010).[7]

To run the WIKIPEDIA Reader, a database containing WIKIPEDIA data is required. We propose to use the JWPL-DataMachine to set up the database. Instructions are given in the JWPL tutorial.[8] After populating the database the table structure needs to be adapted as described in the WIKIPEDIA Reader documentation. When the reader is included in a UIMA pipeline an appropriate UIMA component descriptor with configuration information is needed. A descriptor is already contained in the PEAR package that may be adapted. Finally, to provide the pipeline with a fully functional scheduling system, besides the WIKIPEDIA Reader an appropriate listener component is required (for further information consult the WIKIPEDIA Reader documentation).

5. Discussion

Our work builds on software developed by the Ubiquitous Knowledge Processing Lab. They have published the JAVA WIKIPEDIA Library (JWPL), a JAVA-based API that provides structured access to WIKIPEDIA data, such as articles, disambiguation pages, categories, redirects and link structure (Zesch et al., 2008). The WIKIPEDIA Reader profits from the JWPL API in two ways. First, it uses the JWPLDataMachine tool to transform the publicly available WIKIPEDIA XML dumps into SQL dumps to be loaded into a database. Second, it processes WIKIPEDIA pages

[7] According to the developers at UKP Lab the JWPL API is currently in the process of becoming open source.

[8] `http://www.ukp.tu-darmstadt.de/software/jwpl/documentation/`

using JWPL's parser API. (A comparison with other JAVA-based wiki markup parsers, such as WIKIMODEL[9] generating XHTML and XML output is still pending.)

As far as the division of responsibilities between JWPL and our WIKIPEDIA Reader is concerned, JWPL is an API to access WIKIPEDIA data in a structured way and to parse the MEDIAWIKI syntax, while the WIKIPEDIA Reader is a configurable UIMA collection reader that makes use of the JWPLDataMachine and the JWPL parser. A similar reader component (though lacking both, the ability to represent the WIKIPEDIA article structure in terms of UIMA annotations and the integration into a scheduling system) has been published by the UKP Lab as part of the Darmstadt Knowledge Processing Repository (DKPro) (Gurevych et al., 2007). However, in the current version of DKPro (March 2010) this reader does not seem to be available anymore.

Composing the document text for a WIKIPEDIA page the WIKIPEDIA Reader applies a cleansing procedure to text derived from different content items and skips all WIKIPEDIA template information (infoboxes are but one example). Both design decisions serve the goal to provide appropriate input for syntactic and semantic analysis components (Hahn et al., 2008), the backbone of text analytics for semantic search engines, information extraction and text mining as developed at the JULIE Lab (Buyko et al., 2009). Yet, other applications might take advantages from the uncleansed text or from including the whole spectrum of textual information blocks, including content from templates. Such scenarios would require a code adaptation of the current version of the WIKIPEDIA Reader.

While we think that UIMA components should be type system independent whenever possible, and if not possible, should preferably reuse existing types systems, the current version of the WIKIPEDIA Reader is dependent on our own lab-grown type system. This is due to the fact that the reader is designed to create detailed annotations of different content items of WIKIPEDIA pages, a task that requires very specific annotation types and features we were unable to trace in any other existing type system. Article three of the Charter of the OASIS UIMA Technical Committee (TC)[10] addresses a type system base model that we would like to extend with our WIKIPEDIA types. However, so far this base model is work in progress.

The WIKIPEDIA Reader accesses a database holding WIKIPEDIA information, and, by design, cannot access WIKIPEDIA online. As a consequence, edits on WIKIPEDIA are not realized by the reader, until the database is updated with a fresh WIKIPEDIA dump. However, there are several advantages of the database approach over Web server-based retrieval (a list borrowed from the JWPL documentation): computational efficiency enabling large-scale NLP tasks (nearly constant retrieval time for each article), reproducibility of research results (since a fixed database dump is used instead of the continuously changing online WIKIPEDIA), and the easy-to-use, object-oriented programming interface.

[9] `http://code.google.com/p/wikimodel/`

[10] `\http://www.oasis-open.org/committees/uima/charter.php`

6. Conclusions

We presented an overview of JULIE Lab's WIKIPEDIA Reader and its integration into the UIMA framework. This Reader builds upon the availability of parts of UKP Lab's JWPL API which supports loading WIKIPEDIA data into a database and parsing the WIKIMEDIA syntax of WIKIPEDIA articles.

Our WIKIPEDIA Reader is flexibly configurable to account in a general way for different forms of and requirements for text analytics. The current version of the WIKIPEDIA Reader is available as a UIMA PEAR package from our lab's web page at `http://www.julielab.de`.

Acknowledgments

We want to thank our student Bernd Weigl for implementing parts of the WIKIPEDIA Reader. This research was partially funded by the EC within the CALBC project (FP7-231727).

7. References

Ekaterina Buyko, Erik Faessler, Joachim Wermter, and Udo Hahn. 2009. Event extraction from trimmed dependency graphs. In *Proceedings of the BioNLP 2009 Workshop Companion Volume for Shared Task*, pages 19–27. Boulder, CO, USA.

David Ferrucci and Adam Lally. 2004. UIMA: An architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348.

Evgeniy Gabrilovich and Shaul Markovitch. 2009. WIKIPEDIA-based semantic interpretation for natural language processing. *Journal of Artificial Intelligence Research*, 34(1):443–498.

Iryna Gurevych, Max Mühlhäuser, Christof Müller, Jürgen Steimle, Markus Weimer, and Torsten Zesch. 2007. Darmstadt Knowledge Processing Repository based on UIMA. In *Proceedings of the 1st Workshop on Unstructured Information Management Architecture at GLDV 2007*. Tübingen, Germany.

Udo Hahn, Ekaterina Buyko, Katrin Tomanek, Scott Piao, John McNaught, Yoshimasa Tsuruoka, and Sophia Ananiadou. 2007. An annotation type system for a data-driven NLP pipeline. In *LAW at ACL 2007 – Proceedings of the Linguistic Annotation Workshop*, pages 33–40. Prague, Czech Republic.

Udo Hahn, Ekaterina Buyko, Rico Landefeld, Matthias Mühlhausen, Michael Poprat, Katrin Tomanek, and Joachim Wermter. 2008. An overview of JCORE, the JULIE lab UIMA Component Repository. In *Proceedings of the LREC'08 Workshop 'Towards Enhanced Interoperability for Large HLT Systems:* UIMA *for NLP'*, pages 1–7, Marrakech, Morocco.

Maciej Janik and Krys Kochut. 2008. WIKIPEDIA in action: Ontological knowledge in text categorization. In *ICSC 2008 – Proceedings of the 2nd IEEE International Conference on Semantic Computing*, pages 268–275. Santa Clara, CA, USA.

Simone Paolo Ponzetto and Michael Strube. 2007. Deriving a large scale taxonomy from WIKIPEDIA. In *AAAI 2007 – Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pages 1440–1445. Vancouver, B.C., Canada.

Torsten Zesch, Christof Müller, and Iryna Gurevych. 2008. Extracting lexical semantic knowledge from WIKIPEDIA and WIKTIONARY. In *LREC'08 – Proceedings of the 6th International Language Resources and Evaluation Coonference*. Marrakech, Morocco.

Web-based Collaborative Corpus Annotation: Requirements and a Framework Implementation

Kalina Bontcheva, Hamish Cunningham, Ian Roberts, Valentin Tablan

Natural Language Processing Group
Department of Computer Science, University of Sheffield
211 Portobello, Sheffield S1 4DP, UK
Initial.Surname@dcs.shef.ac.uk

Abstract

In this paper we present Teamware, a novel web-based collaborative annotation environment which enables users to carry out complex corpus annotation projects, involving less skilled, cheaper annotators working remotely. It has been evaluated by us through the creation of several gold standard corpora, as well as through external evaluation in commercial annotation projects.

1 Introduction

For the past ten years, NLP development frameworks such as OpenNLP, GATE, and UIMA have been providing tool support and facilitating NLP researchers with the task of implementing new algorithms, sharing, and reusing them. At the same time, Information Extraction (IE) research and computational linguistics in general has been driven forward by the growing volume of annotated corpora, produced by research projects and through evaluation initiatives such as MUC (Marsh and Perzanowski, 1998), ACE[1], DUC (DUC, 2001), and CoNLL shared tasks. Some of the NLP frameworks (e.g., AGTK (Maeda and Strassel, 2004), GATE (Cunningham et al., 2002)) even provide text annotation user interfaces. However, much more is needed in order to produce high quality annotated corpora: a stringent methodology, annotation guidelines, inter-annotator agreement measures, and in some cases, annotation adjudication (or data curation) to reconcile differences between annotators.

Current tools demonstrate that annotation projects can be approached in a collaborative fashion successfully (see Section 2). However, we believe that this can be improved further by providing a unified environment that provides a multi-role methodological framework to support the different phases and actors in the annotation process. The multi-role support is particularly important, as it enables the most efficient use of the skills of the different people and lowers overall annotation costs through having simple and efficient annotation web-based UIs for non-specialist annotators. This also enables role-based security, project management and performance measurement of annotators, which are all particularly important in corporate environments.

This paper presents Teamware, a web-based software suite and a methodology for the implementation and support of complex annotation projects[2]. In addition to its research uses, it has also been tested as a framework for cost-effective commercial annotation services, supplied either as in-house units or as outsourced specialist activities. Teamware is based on GATE: a widely used, scalable and robust open-source language processing framework.

The rest of the paper is structured as follows. We first discuss related work in Section 2 and then motivate the requirements which need to be met (Section 3). Our architecture and implementation are discussed in Section 4, followed by a discussion of three practical applications in Section 5 and conclusions.

2 Related Work

Tools for collaborative corpus annotation enable researchers to work together on annotating corpora regardless of their physical location. This problem can be decomposed into two major tasks: (i) provide users with access to distributed corpora over the web; and (ii) provide visualisation and editing tools that require no installation effort and are easy to use. Some of the most sophisticated tools in this area are those developed by the Linguistic Data Consortium, due to their need to run large-scale annotation projects. The AGTK toolkit (Maeda and Strassel, 2004) provides a shared relational database model for storing and accessing corpora on a shared server, as well as being a framework for development of collaborative annotation tools based on these shared corpora. One example is the specialised ACE annotation tool, which also comes with an accompanying tool for annotation adjudication. Maeda *et al* (2008) describe ACK (Annotation Collection Kit) which is web based and uses comma-separated CSV files to define the questions which an annotator has to answer (e.g., what are the possible parts of speech of this word). In the context of machine translation, they also discuss a workflow system for post-editing machine translation results which supports different user roles (editors in this case) and the communication between them. While the LDC tool set is very impressive, the various annotation tasks are covered by *separate, independent* tools and in some cases, these tools are specific to a particular annotation project (e.g., ACE, GALE).

One way to generalise the annotation tools is to support their customisation to a specific task by means of annota-

Authors are listed alphabetically.

[1] http://www.ldc.upenn.edu/Projects/ACE/

[2] Teamware is currently available as a hosted service over the web for use by both researchers and companies. If you are interested in using the Teamware service or want to install it locally, please contact the first author.

tion schemas, e.g., Callisto (Day et al., 2004), GATE (Cunningham et al., 2002). A somewhat more complex, but more powerful approach is to model the different linguistic annotation tasks with ontologies, e.g., Knowtator (Ogren, 2006). All these tools also provide support for measuring inter-annotator agreement, while Knowtator also supports semi-automatic adjudication and the creation of a consensus annotation set. However, they all require installation on the user's machine and are designed primarily for expert annotators. All except GATE, also do not provide support for bootstrapping the manual annotation by running an automatic NLP system first.

With respect to workflows, these have been studied predominantly in the context of configuring a set of NLP modules into an application. For example, both GATE (Cunningham et al., 2002) and UIMA(Ferrucci and Lally, 2004) support workflows[3], but not for asynchronous annotation tasks. Neither do they provide business process monitoring (e.g., time spent by annotators, their performance levels, progress, overall costs).

Recently researchers began experimenting with using Amazon's Mechanical Turk to recruit non-expert annotators. Experiments (Snow et al., 2008) show that on average 10 unskilled annotators are needed in order to reach the quality of an expertly annotated gold standard. Multi-role collaborative annotation environments like ours have the potential to lower the number of required low-skilled annotators through the involvement of an expert curator, assisted with automatic bootstrapping and adjudication tools. In addition, there are also many cases when reliable gold-standard creation is still required and/or when the data cannot be released on the web for public annotation.

To summarise, in comparison to previous work Teamware is a novel general purpose, web-based annotation framework, which:

- structures the roles of the different actors involved in large-scale corpus annotation (e.g., annotators, editors, managers) and supports their interactions in an unified environment;
- provides a set of general purpose text annotation tools, tailored to the different user roles, e.g., a curator management tool with inter-annotator agreement metrics and adjudication facilities and a web-based document tool for in-experienced annotators;
- supports complex annotation workflows and provides a management console with business process statistics, such as time spent per document by each of its annotators, percentage of completed documents, etc;
- offers methodological support, to complement the diverse technological tool support.

3 Requirements for Multi-Role Collaborative Annotation Environments

As discussed in the previous section, collaborative corpus annotation is a complex process, which involves different kinds of actors (e.g., annotators, editors, managers) and also requires a diverse range of pre-processing, a user interface, and evaluation tools. Here we structure all these into a coherent set of key requirements, which arise from our goal to provide cost-effective corpus annotation.

Firstly, due to the multiple actors involved and their complex interactions, a collaborative environment needs to **support these different roles** through user groups, access privileges, and corresponding user interfaces. Secondly, since many annotation projects manipulate hundreds of documents, there needs to be a **remote, efficient data storage**. Thirdly, significant cost savings can be achieved through pre-annotating corpora automatically, which in turns requires support for **automatic annotation services** and their flexible configuration. Last, but not least, a **flexible workflow engine** is required to capture the complex requirements and interactions.

Next we discuss the four high-level requirements in finer-grained details.

3.1 Typical Division of Labour

Due to annotation projects having different sizes and complexity, in some cases the same person might perform more than one role or new roles might be needed. For example, in small projects it is common that the person who defines and manages the project is also the one who carries out quality assurance and adjudication. Nevertheless these are two distinct roles (manager vs editor), involving different tasks and requiring different tool support.

Annotators are given a set of annotation guidelines and often work on the same document independently. This is needed in order to get more reliable results and/or measure how well humans perform the annotation task (see more on Inter-Annotator Agreement (IAA) below). Consequently, manual annotation is a slow and error-prone task, which makes overall corpus production very expensive. In order to allow the involvement of less-specialised annotators, the manual annotation user interface needs to be simple to learn and use. In addition, there needs to be an automatic training mode for annotators where their performance is compared against a known gold standard and all mistakes are identified and explained to the annotator, until they have mastered the guidelines.

Since the annotators and the corpus editors are most likely working at different locations, there needs to be a communication channel between them, e.g., instant messaging. If an editor/manager is not available, an annotator should also be able to mark an annotation as requiring discussion and then all such annotations should be shown automatically in the editor console. In addition, the annotation environment needs to restrict annotators to working on a maximum of n documents (given as a number or percentage), in order to prevent an over-zealous annotator from taking over a project and introducing individual bias. Annotators also need to be able to save their work and, if they close the annotation tool, the same document must be presented to them for completion the next time they log in.

From the user interface perspective, there needs to be support for annotating document-level metadata (e.g., language identification), word-level annotations (e.g., named

[3] Called controllers in GATE.

entities, POS tags), and relations and trees (e.g., coreference, syntax trees). Ideally, the interface should offer some generic components for all these, which can be customised with the project-specific tags and values via an XML schema or other similar declarative mechanism. The UI also needs to be extensible, so specialised UIs can easily be plugged in, if required.

Editors or curators are responsible for measuring Inter-Annotator Agreement (IAA), annotation adjudication, gold-standard production, and annotator training. They also need to communicate with annotators when questions arise. Therefore, they need to have wider privileges in the system. In addition to the standard annotation interfaces, they need to have access to the actual corpus and its documents and run IAA metrics. They also need a specialised adjudication interface which helps them identify and reconcile differences in multiply annotated documents. For some annotation projects, they also need to be able to send a problematic document back for re-annotation by different annotators.

Project managers are typically in charge of defining new corpus annotation projects and their workflows, monitoring their progress, and dealing with performance issues. Depending on project specifics, they may work together with the curators and define the annotation guidelines, the associated schemas (or set of tags), and prepare and upload the corpus to be annotated. They also make methodological choices: whether to have multiple annotators per document; how many; which automatic NLP services need to be used to pre-process the data; and what is the overall workflow of annotation, quality assurance, adjudication, and corpus delivery.

Managers need a project monitoring tool where they can see:

- Whether a corpus is currently assigned to a project or, what annotation projects have been run on the corpus with links to these projects or their archive reports (if no longer active). Also links to the the annotation schemas for all annotation types currently in the corpus.
- Project completion status (e.g., 80% manually annotated, 20% adjudicated).
- Annotator statistics within and across projects: which annotator worked on each of the documents, what schemas they used, how long they took, and what was their IAA (if measured).
- The ability to lock a corpus from further editing, either during or after a project.
- Ability to archive project reports, so projects can be deleted from the active list. Archives should preserve information on what was done and by whom, how long it took, etc.

3.2 Remote, Scalable Data Storage

Given the multiple user roles and the fact that several annotation projects may need to be running at the same time, possibly involving different, remotely located teams, the data storage layer needs to scale to accommodate large, distributed corpora and have the necessary security in place through authentication and fine-grained user/group access control. Data security is paramount and needs to be enforced as data is being sent over the web to the remote annotators. Support for diverse document input and output formats is also necessary, especially stand-off ones when it is not possible to modify the original content. Since multiple users can be working concurrently on the same document, there needs to be an appropriate locking mechanism to support that. The data storage layer also needs to provide facilities for storing annotation guidelines, annotation schemas, and, if applicable, ontologies. Last, but not least, a corpus search functionality is often required, at least one based on traditional keyword-based search, but ideally also including document metadata and linguistic annotations.

3.3 Automatic annotation services

Automatic annotation services can reduce significantly annotation costs (e.g., annotation of named entities), but unfortunately they also tend to be domain or application specific. Also, several might be needed in order to bootstrap all types that need to be annotated, e.g., named entities, coreference, and relation annotation modules. Therefore, the architecture needs to be open so that new services can be added easily. Such services can encapsulate different IE modules and take as input one or more documents (or an entire corpus). The automatic services also need to be scalable, in order to minimise their impact on the overall project completion time. The project manager should also be able to choose services based on their accuracy on a given corpus.

Machine Learning (ML) IE modules can be regarded as a specific kind of automatic service. A mixed initiative system (Day et al., 1997) can be set up by the project manager and used to facilitate manual annotation behind the scenes. This means that once a document has been annotated manually, it will be sent to train the ML service which internally generates an ML model. This model will then be applied by the service on any new document, so that this document will be partially pre-annotated. The human annotator then only needs to validate or correct the annotations provided by the ML system, which makes the annotation task significantly faster (Day et al., 1997).

3.4 Workflow Support

In order to have an open, flexible model of corpus annotation processes, we need a powerful workflow engine which supports asynchronous execution and arbitrary mix of automatic and manual steps. For example, manual annotation and adjudication tasks are asynchronous. Resilience to failures is essential and workflows need to save intermediary results from time to time, especially after operations that are very expensive to re-run (e.g. manual annotation, adjudication). The workflow engine also needs to have status persistence, action logging, and activity monitoring, which is the basis for the project monitoring tools.

In a workflow it should be possible for more than one annotator to work on the same document at the same time, however, during adjudication by editors, all affected anno-

tations need to be locked to prevent concurrent modifications. For separation of concerns, it is also often useful if the same corpus can have more than one active project. Similarly, the same annotator needs to be able to work on several annotation projects.

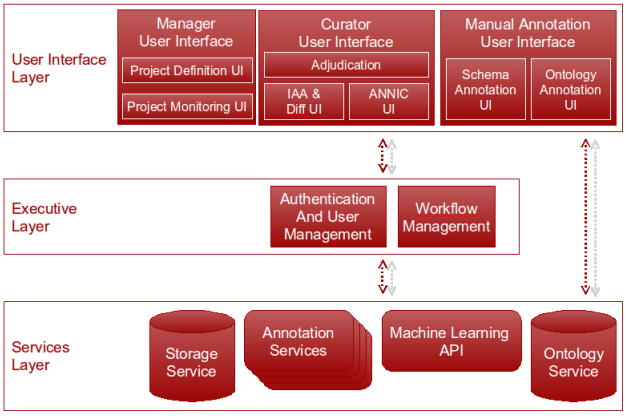


Figure 1: Teamware Architecture Diagram

4 Teamware: Architecture, Implementation, and Examples

Teamware is a web-based collaborative annotation and curation environment, which allows unskilled annotators to be trained and then used to lower the cost of corpus annotation projects. Further cost reductions are achieved by bootstrapping with relevant automatic annotation services, where these exist, and/or through mixed initiative learning methods. It has a service-based architecture which is parallel, distributed, and also scalable (via service replication) (see Figure 1).

As shown in Figure 1, the Teamware architecture consists of SOAP web services for data storage, a set of web-based user interfaces (UI Layer), and an executive layer in the middle where the workflows of the specific annotation projects are defined. The UI Layer is connected with the Executive Layer for exchanging command and control messages (such as requesting the ID for document that needs to be annotated next), and also it connects directly to the services layer for data-intensive communication (such as downloading the actual document data, and uploading back the annotations produced).

4.1 Data Storage and Ontology Services

The storage service provides a distributed data store for corpora, documents, and annotation schemas. Input documents can be in all major formats (e.g. plain text, XML, HTML, PDF), based on GATE's comprehensive support. In all cases, when a document is created/imported in Teamware, the format is analysed and converted into a single unified, graph-based model of *annotation*: the one of the GATE NLP framework. Then this internal annotation format is used for data exchange between the service layer, the executive layer and the UI layer. Different processes within Teamware can add and remove annotation data within the same document concurrently, as long as two processes do not attempt to manipulate the same subset of the data at the same time. A locking mechanism is used to ensure this and prevent data corruption. The main export format for annotations is currently stand-off XML, including XCES (Ide et al., 2000). Document text is represented internally using Unicode and data exchange uses the UTF-8 character encoding, so Teamware supports documents written in any natural language supported by the Unicode standard (and the Java platform).

Since some corpus annotation tasks require ontologies, these are made available from a dedicated ontology service. This wraps the OWLIM (Kiryakov, 2006) semantic repository, which is needed for reasoning support and consequently justifies having a specialised ontology service, instead of storing ontologies together with documents and schemas.

4.2 Annotation Services

The Annotation Services (GAS) provide distribution of compute-intensive NLP tasks over multiple processors. It is transparent to the external user how many machines are actually used to execute a particular service. GAS provides a straightforward mechanism for running applications, created with the GATE framework, as web services that carry out various NLP tasks. In practical applications we have tested a wide range of services such as named entity recognition (based on the freely-available ANNIE system (Cunningham et al., 2002)), ontology population (Maynard et al., 2009), patent processing (Agatonovic et al., 2008), and automatic adjudication of multiple annotation layers in corpora.

The GAS architecture is itself layered, with a separation between the web service endpoint that accepts requests from clients and queues them for processing, and one or more *workers* that take the queued requests and process them. The queueing mechanism used to communicate between the two sides is the Java Messaging System (JMS)[4], a standard framework for reliable messaging between Java components, and the configuration and wiring together of all the components is handled using the Spring Framework [5].

The endpoint, message queue and worker(s) are conceptually and logically separate, and may be physically hosted within the same Java Virtual Machine (VM), within separate VMs on the same physical host, or on separate hosts connected over a network. When a service is first deployed it will typically be as a single worker which resides in the same VM as the service endpoint. This may be adequate for simple or lightly-loaded services but for more heavily-loaded services additional workers may be added dynamically without shutting down the web service, and similarly workers may be removed when no longer required. All workers that are configured to consume jobs from the same endpoint will transparently share the load. Multiple workers also provide fault-tolerance – if a worker fails its in-progress jobs will be returned to the queue and will be picked up and handled by other workers.

4.3 The Executive Layer

Firstly, the executive layer implements authentication and user management, including role definition and assignment.

[4] http://java.sun.com/products/jms/

[5] http://www.springsource.org/

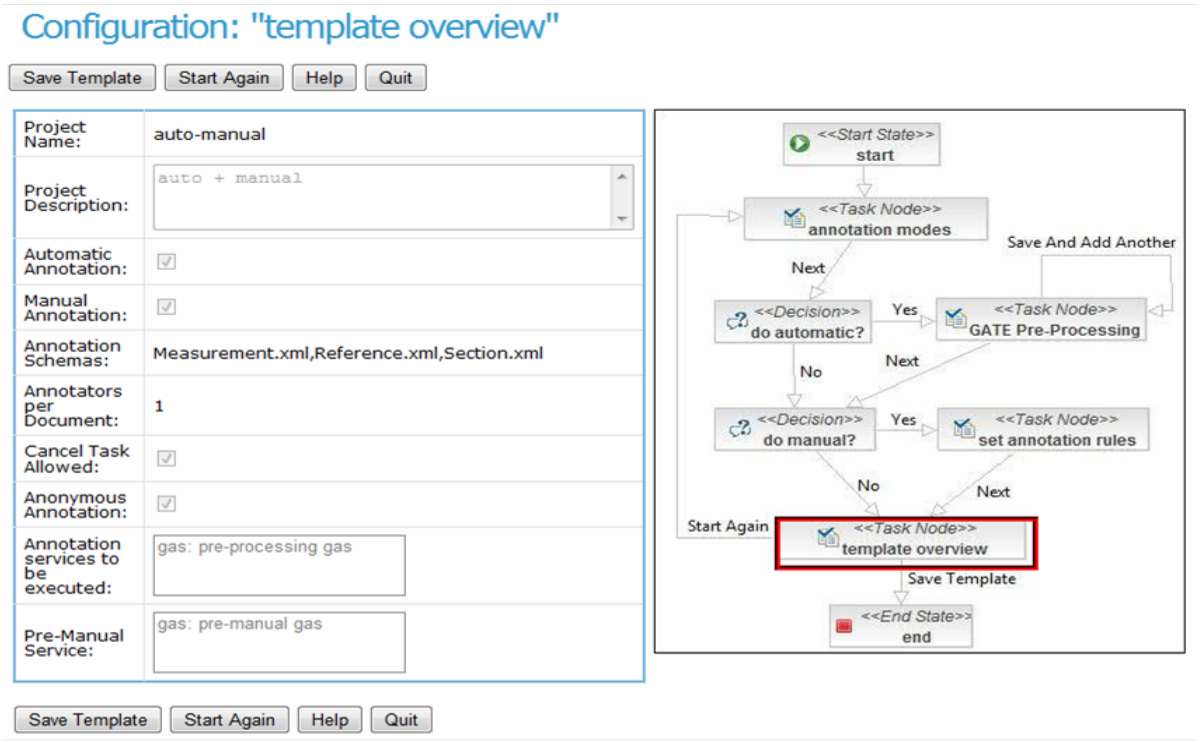


Figure 2: Dynamic Workflow Configuration: Example

In addition, administrators can define here which UI components are made accessible to which user roles (the defaults are shown in Figure 1).

The second major part is the workflow manager, which is based on JBoss jBPM[6] and has been developed to meet most of the requirements discussed in Section 3.4 above. Firstly, it provides dynamic workflow management: create, read, update, delete (CRUD) workflow definitions, and workflow actions. Secondly, it supports business process monitoring, i.e., measures how long annotators take, how good they are at annotating, as well as reporting the overall progress and costs. Thirdly, there is a workflow execution engine which runs the actual annotation projects. As part of the execution process, the project manager selects the number of annotators per document; the annotation schemas; the set of annotators and curator(s) involved in the project; and the corpus to be annotated.

Figure 2 shows an example workflow template. The diagram on the right shows the choice points in workflow templates - whether to do automatic annotation or manual or both; which automatic annotation services to execute and in what sequence; and for manual annotation – what schemas to use, how may annotators per document, whether they can reject annotating a document, etc. The left-hand side shows the actual selections made for this particular workflow, i.e., use both automatic and manual annotation; annotate measurements, references, and sections; and have one annotator per document. Once this template is saved by the project manager, then it can be executed by the workflow engine on a chosen corpus and list of annotators and curators. The workflow engine will first call the automatic annotation service to bootstrap and then its results will be corrected by human annotators.

The rationale behind having an executive layer rather than defining authentication and workflow management as services similar to the storage and ontology ones comes from the fact that Teamware services are all SOAP web services, whereas elements of the executive layer are only in part implemented as SOAP services with the rest being browser based. Conceptually also the workflow manager acts like a middleman that ties together all the different services and communicates with the user interfaces.

4.4 The User Interfaces

The Teamware user interfaces are web-based and do not require prior installation. They either rendered natively in the web browser or, for more complex UIs, a Java Web Start wrapper is provided around some Swing-based GATE editors (e.g., the document editor and the ANNIC viewer (Aswani et al., 2005)). After the user logs in, the system checks their role(s) and access privileges to determine which interface elements they are allowed to access.

4.4.1 Annotator User Interface

When manual annotators log into Teamware, they see a very simple web page with one link to their user profile data and another one – to start annotating documents. The generic schema-based annotator UI is shown in Figure 3 and it is a visual component in GATE, which is reused here via Java Web Start[7]. This removes the need to install GATE on the annotator machines and instead they just click on a link to download and start a web application.

[6] http://www.jboss.com/products/jbpm/

[7] http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp

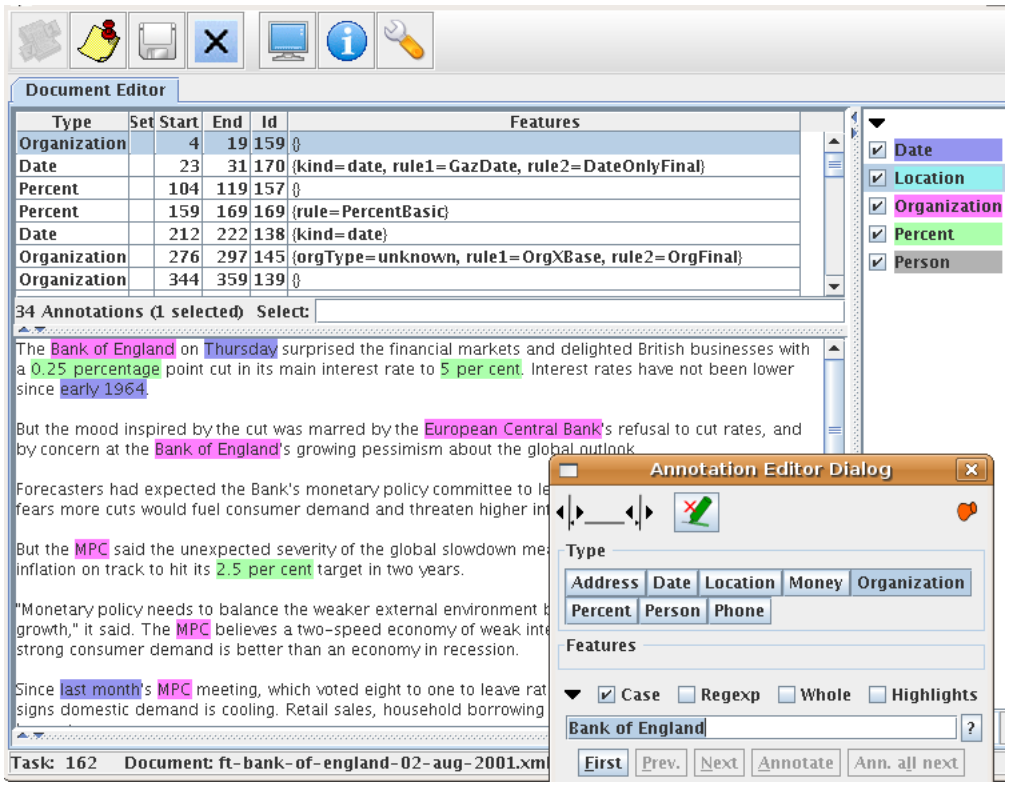


Figure 3: The Schema-based Annotator UI

The annotation editor dialog shows the annotation types (or tags) valid for the current project and optionally their features (or attributes). These are generated automatically from the annotation schemas assigned to the project by its manager. The annotation editor also supports the modification of annotation boundaries, as well as the use of regular expressions to annotate multiple matching strings simultaneously. To add a new annotation, one selects the text with the mouse (e.g., "Bank of England") and then clicks on the desired annotation type in the dialog (e.g., Organization). Existing annotations are edited by hovering over them, which shows their current type and features in the editor dialog.

The toolbar at the top of Figure 3 shows all other actions which can be performed. The first button requests a new document to be annotated. When pressed, a request is sent to the workflow manager which checks if there are any pending documents which can be assigned to this annotator. The second button signals task completion, which saves the annotated document as completed on the data storage layer and enables the annotator to ask for a new one (via the first button). The third (save) button stores the document without marking it as completed in the workflow. This can be used for saving intermediary annotation results or if an annotator needs to log off prior to completing a document. The next time they login and request a new task, they will be given this document to complete first.

Ontology-based document annotation is supported in a similar fashion, but instead of having a flat list of types on the right, the annotator is shown the type hierarchy and when they select a particular type (or class), they can then optionally choose an existing instance or add a new one. This UI also supports the annotation of relations by modelling them as properties in the ontology and allowing annotators to instatiate their values in the UI (not shown due to lack of space). Similar to the schema annotation editor, the ontology-based editor is a visual plugin from GATE delivered via Java Web Start.

4.4.2 Curator User Interface

As discussed in Section 3.1, curators (or editors) carry out quality assurance tasks. In Teamware the curation tools cover IAA metrics (e.g. precision/recall and kappa) to identify if there are differences between annotators; a visual annotation comparison tool to see quickly where the differences are per annotation type (Cunningham et al., 2002); and an editor to edit and reconcile annotations manually (i.e., adjudication) or by using external automatic services.

The key part of the manual adjudication UI is shown in Figure 4: the complete UI shows also the full document text above the adjudication panel, as well as lists all annotation types on the right, so the curator can select which one they want to work on. In our example, the curator has chosen to adjudicate Date annotations created by two annotators and to store the results in a new consensus annotation set. The adjudication panel has on top arrows that allow curators to jump from one difference to the next, thus reducing the required effort. The relevant text snippet is shown and below it are shown the annotations of the two annotators. The curator can easily see the differences and correct them, e.g., by dragging the correct annotation into the consensus set.

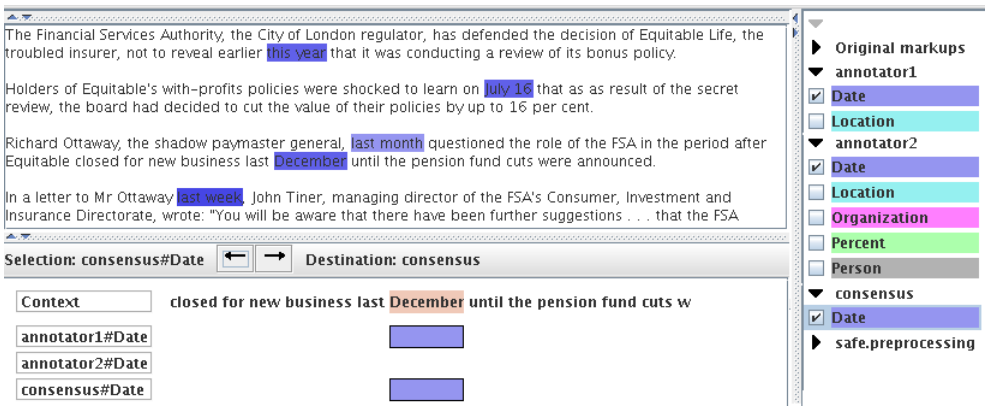


Figure 4: Part of the Adjudication UI

4.4.3 Project Manager Interface

The project manager web UI is the most powerful and multi-functional one. It provides the front-end to the executive layer (see Section 4.3 and Figure 2). In a nutshell, managers upload documents and corpora, define the annotation schemas, choose and configure the workflows and execute them on a chosen corpus. The management console also provides project monitoring facilities, e.g., number of annotated documents, number in progress, and yet to be completed (see Figure 5). Per annotator statistics are also available – time spent per document, overall time worked, average IAA, etc. These requirements were discussed in further detail in Section 3.1 above.

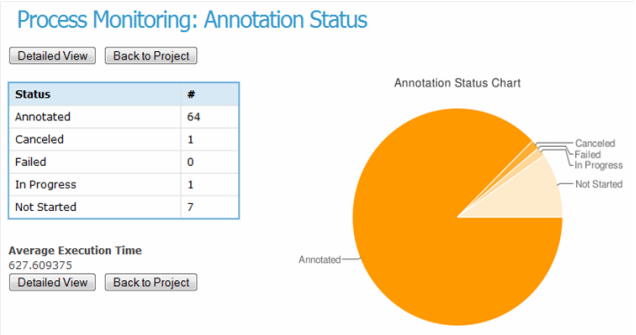


Figure 5: Project Progress Monitoring UI

5 Practical Applications

Teamware has already been used in practice in several corpus annotation projects of varying complexity and size – due to space limitations, here we focus on three representative ones. Firstly, we tested the robustness of the data layer and the workflow manager in the face of simultaneous concurrent access. For this we annotated 100 documents, 2 annotators per document, with 60 active annotators requesting documents to annotate and saving their results on the server. There were no latency or concurrency issues reported.

Once the current version was considered stable, we ran several corpus annotation projects to produce gold standards for IE evaluation in three domains: business intelligence, fisheries, and bio-informatics. The latter involved 10 bio-informatics students which were first given a brief training session and were then allowed to work from home. The project had 2 annotators per document, working with 6 entity types and their features. Overall, 109 Medline abstracts of around 200-300 words each were annotated with average annotation speed of 9 minutes per abstract. This project revealed several shortcomings of Teamware which will be addressed in the forthcoming version 2:

- IAA is calculated per document, but there is no easy way to see how it changes across the entire corpus.
- The datastore layer can sometimes leave the data in an inconsistent state following an error, due to the underlying binary Java serialisation format. A move towards XML file-based storage is being investigated.
- There needs to be a limit on the proportion of documents which any given annotator is allowed to work on, since one over-zealous annotator ended up introducing a significant bias by annotating more than 80% of all documents.

The most versatile and still ongoing practical use of Teamware has been in a commercial context, where a company has two teams of 5 annotators each (one in China and one in the Philippines). The annotation projects are being defined and overseen by managers in the USA, who also act occasionally as curators. They have found that the standard double-annotated agreement-based approach is a good foundation for their commercial needs (e.g., in the early stages of the project and continuously for gold standard production), while they also use very simple workflows where the results of automatic services are being corrected by annotators, working only one per document to maximise volume and lower the costs. In the past few months they have annotated over 1,400 documents, many of which according to multiple schemas and annotation guidelines. For instance, 400 patent documents were doubly annotated both with measurements (IAA achieved 80-95%) and bio-informatics entities, and then curated and adjudicated to create a gold standard. They also annotated 1000 Medline abstracts with species information where they measured average speed of 5-7 minutes per document. The initial annotator training in Teamware was between 30 minutes and one hour, following which they ran several small-scale experimental projects to train the annotators in the particular

annotation guidelines (e.g., measurements in patents). Annotation speed also improved over time, as the annotators became more proficient with the guidelines – the Teamware annotator statistics registered improvements of between 15 and 20%. Annotation quality (measured through inter-annotator agreement) remained high, even when annotators have worked on many documents over time.

From a Teamware implementational perspective, the diverse user needs and practical experience with remote annotator teams exposed several weaknesses in the current implementation:

- Different sets of annotators need to be able to work on the same corpus simultaneously, as part of separate projects, so that each team can specialise in a small number of annotation types. This requires support for merging the results of the separate projects into one consistent corpus, which is currently not achieved easily within the Teamware environment.
- The annotator UI needs to be highly responsive to maximise the time annotators spend actually working on the documents. Consequently the data storage layer and the workflow need to minimise further network traffic, e.g., allow access to document-level metadata without also loading the entire document content.
- Execution speed of the annotation workflows needs to be optimised further, e.g., by avoiding unnecessary network traffic generated by temporary results being saved to the data store.
- Annotation of relations as well as manual annotation with medium- to large-size ontologies are required in many projects and the corresponding UIs need to be improved to support faster annotation.

6 Conclusion and Future Work

In this paper we have described a multi-role web-based annotation environment, which supports customised annotation workflows and provides methodological support to the different actors involved in the process. Evaluation with distributed annotator teams working on a wide range of corpus annotation projects is still ongoing. We have already identified some minor issues, mostly requiring optimisations in the workflow and data layers, as well as usability improvements in the user interfaces. All these will be addressed in the forthcoming second version. We have also planned controlled experiments that compare annotation times with Teamware against other annotation tools, although obtaining statistically significant results would be difficult, expensive, and would require large teams of annotators.

Acknowledgements: This work has been supported by a Matrixware/IRF research grant. We also wish to thank Matthew Petrillo, Jessica Baycroft, Angus Roberts, and Danica Damljanovic for running the Teamware distributed annotation experiments and allowing us to report the results here. Many thanks also to Milan Agatonovic who worked on the Teamware executive layer, while being a researcher at Sheffield.

7 References

M. Agatonovic, N. Aswani, K. Bontcheva, H. Cunningham, T. Heitz, Y. Li, I. Roberts, and V. Tablan. 2008. Large-scale, parallel automatic patent annotation. In *Proc. of 1st International CIKM Workshop on Patent Information Retrieval - PaIR'08.*.

N. Aswani, V. Tablan, K. Bontcheva, and H. Cunningham. 2005. Indexing and Querying Linguistic Metadata and Document Content. In *Proceedings of Fifth International Conference on Recent Advances in Natural Language Processing (RANLP2005)*, Borovets, Bulgaria.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*.

D. Day, J. Aberdeen, L. Hirschman, R. Kozierok, P. Robinson, and M. Vilain. 1997. Mixed-Initiative Development of Language Processing Systems. In *Proceedings of ANLP-97*.

D. Day, C. McHenry, R. Kozierok, and L. Riek. 2004. Callisto: A configurable annotation workbench. In *Int. Conf. on Language Resources and Evaluation*.

NIST. 2001. *Proceedings of the Document Understanding Conference*, September 13.

D. Ferrucci and A. Lally. 2004. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*.

N. Ide, P. Bonhomme, and L. Romary. 2000. XCES: An XML-based Standard for Linguistic Corpora. In *Proceedings of the Second International Language Resources and Evaluation Conference (LREC)*.

A. Kiryakov. 2006. OWLIM: balancing between scalable repository and light-weight reasoner. In *Proc. of WWW2006*, Edinburgh, Scotland.

K. Maeda and S. Strassel. 2004. Annotation Tools for Large-Scale Corpus Development: Using AGTK at the Linguistic Data Consortium. In *Proc. of 4th Language Resources and Evaluation Conference*.

K. Maeda, H. Lee, S. Medero, J. Medero, R. Parker, and S. Strassel. 2008. Annotation Tool Development for Large-Scale Corpus Creation Projects at the Linguistic Data Consortium. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*.

E. Marsh and D. Perzanowski. 1998. Muc-7 evaluation of ie technology: Overview of results. In *Proceedings of the Seventh Message Understanding Conference*.

D. Maynard, A. Funk, and W. Peters. 2009. SPRAT: a tool for automatic semantic pattern-based ontology population. In *International Conference for Digital Libraries and the Semantic Web*, Trento, Italy, September.

P. Ogren. 2006. Knowtator: A Protege Plug-In For Annotated Corpus Construction . In *HLT-NAACL - Demos*.

R. Snow, B. O'Connor, D. Jurafsky, and A. Y. Ng. 2008. Cheap and fast—but is it good?: Evaluating non-expert annotations for natural language tasks. In *EMNLP '08*.

Computer-aided Ontology Development: an integrated environment

Manuel Fiorelli, Maria Teresa Pazienza, Steve Petruzza, Armando Stellato, Andrea Turbati

ART Research Group, Dept. of Computer Science,
Systems and Production (DISP) University of Rome, Tor Vergata
Via del Politecnico 1, 00133 Rome, Italy
{pazienza, stellato, turbati}@info.uniroma2.it
{manuel.fiorelli, steve.petruzza}@gmail.com

Abstract

In this paper we introduce CODA (Computer-aided Ontology Development Architecture), an Architecture and a Framework for semi-automatic development of ontologies through analysis of heterogeneous information sources. We have been motivated in its design by observing that several fields of research provided interesting contributions towards the objective of augmenting/enriching ontology content, but that they lack a common perspective and a systematic approach.
While in the context of Natural Language Processing specific architectures and frameworks have been defined, time is not yet completely mature for systems able to reuse the extracted information for ontology enrichment purposes: several examples do exist, though they do not comply with any leading model or architecture. Objective of CODA is to acknowledge and improve existing frameworks to cover the above gaps, by providing: a conceptual systematization of data extracted from unstructured information to enrich ontology content, an architecture defining the components which take part in such a scenario, and a framework supporting all of the above through standard implementations.
This paper provides a first overview of the whole picture, and introduces UIMAST, an extension for the Knowledge Management and Acquisition Platform *Semantic Turkey*, that implements CODA principles by allowing reuse of components developed inside UIMA framework to drive semi-automatic Acquisition of Knowledge from Web Content.

1. Introduction

A number of tasks focused on ontology development as well as on augmentation or refinement of their content through reuse of external information has been defined in the last decade. The nature of these tasks is manifold: from the automation of ontology development processes to their facilitation through innovative and effective solutions for human-computer interaction. In some cases their assessment has produced a plethora of (often contrasting) methodologies and approaches (as in the case of *ontology and lexicon integration* (Buitelaar, et al., 2006; Cimiano, Haase, Herold, Mantel, & Buitelaar, 2007; Pazienza & Stellato, 2006; Pazienza & Stellato, Linguistic Enrichment of Ontologies: a methodological framework, 2006)); in other ones, such as *ontology learning*, it has lead to founding entire new branches of research (Cimiano, 2006)
The "external information" we are interested in, mostly refers to diverse forms of "narrative information sources", such as text documents (or other kind of media, such as audio and video) or to more structured knowledge content, like the one provided by machine readable linguistic resources. These latter comprise lexical resources (e.g. rich lexical databases such as WordNet (Miller, Beckwith, Fellbaum, Gross, & Miller, 1993), bilingual translation dictionaries or domain thesauri), text corpora (from pure domain-oriented text collections to annotated corpora of documents), or other kind of structured or semi-structured information sources, such as frame-based resources (Baker, Fillmore, & Lowe, 1998; Shi & Mihalcea, 2005).
With the intent of providing a definition covering all of the previously cited tasks and addressing the interaction they have with the above resources, we coined the expression COD (Computer-aided Ontology Development), with this acronym covering all processes for enriching ontology content through exploitation of external resources, by using (semi)automatic approaches.
In this paper, we lay the basis for an architecture (CODA: COD Architecture), supporting Computer-aided Ontology Development, then introduce UIMAST, an extension for the Knowledge Management and Acquisition Platform *Semantic Turkey*, implementing CODA principles by allowing reuse of components developed inside the UIMA framework to drive semi-automatic Acquisition of Knowledge from Web Content.

2. State-of-the-art and Motivation

Motivations and ideas for supporting fulfillment of the above tasks' objectives, have been often supported through proof-of-concept systems, tools and in some cases open platforms (Cimiano & Völker, 2005) developed inside the research community, laying the path and showing the way for future industrial follow-up.
Until now basic architectural definitions and interaction modalities have been defined in detail fulfilling industry-standard level for processes such as:

- *ontology development* with most recent ontology development tools following the path laid by Protégé (Gennari, et al., 2003)
- *text analysis* starting from the TIPSTER architecture (Harman, 1992), its most notable implementation GATE (Cunningham, Maynard, Bontcheva, & Tablan, 2002) and the recently approved OASIS standard UIMA (Ferrucci & Lally, 2004).

On the contrary a comprehensive study and synthesis of an architecture for supporting ontology development driven by knowledge acquired from external resources, has not been formalized until now.
What lacks in all current approaches is an overall perspective on the task and a proposal for an architecture

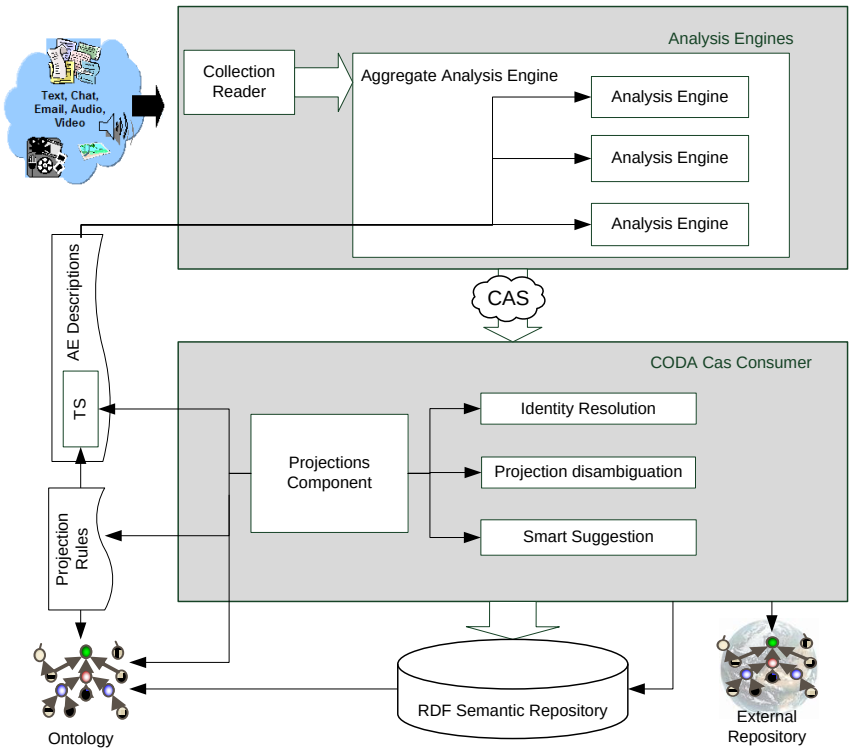


Figure 1. CODA Architecture (overview of components related to tasks 1 and 2)

providing instruments for supporting the entire flow of information (from acquisition of knowledge from external resources to its exploitation) to enrich and augment ontology content. Just scoping to ontology learning, OntoLearn (Velardi, Navigli, Cucchiarelli, & Neri, 2005) provides a methodology, algorithms and a system for performing different ontology learning tasks, OntoLT (Buitelaar, Olejnik, & Sintek, 2004) provides a ready-to-use Protégé plugin for adding new ontology resources extracted from text, while the sole Text2Onto (Cimiano & Völker, 2005) embodies a first attempt to realize an open architecture for management of ontology learning processes.

If we consider ontology-lexicon integration, previous studies dealt with how to represent this integrated information (Peters, Montiel-Ponsoda, Aguado de Cea, & Gómez-Pérez, 2007; Buitelaar, et al., 2006; Cimiano, Haase, Herold, Mantel, & Buitelaar, 2007), other have shown useful applications exploiting onto-lexical resources (Basili, Vindigni, & Zanzotto, 2003; Peter, Sack, & Beckstein, 2006) though only few works (Pazienza, Stellato, & Turbati, 2008) dealt with comprehensive framework for classifying, supporting, testing and evaluating processes for integration of content from lexical resources with ontological knowledge.

3. Objectives

Considering these expectations, we worked with the objective of acknowledging and improving existing frameworks for Unstructured Information Management, thus providing:

– a *conceptual systematization* of the tasks covering reuse of data extracted from unstructured information to improve ontology content
– an *architecture* defining the components which take part in such a scenario
– a *framework* supporting all of the above through standard implementations

We provide here requirements and objectives which characterize COD tasks, the COD Architecture, and a CODA Framework

3.1. COD Tasks

Given the definition of COD provided at the beginning, we sketch here major related tasks:

1. **(Traditional) Ontology Learning tasks**, devoted to augmentation of ontology content through discovery of new resources and axioms. They include discovery of new concepts, concept inheritance relations, concept instantiation, properties/relations, domain and range restrictions, mereological relations or equivalence relations etc…
2. **Population of ontologies with new data:** a rib of the above, this focuses on the extraction of new ground data for a given (ontology) model (or even for specific concepts belonging to it)
3. **Linguistic enrichment of ontologies**: enrichment of ontological content with linguistic information coming from external resources (eg. text, linguistic resources etc…)

3.2. CODA Architecture

COD Architecture (CODA, from now on) defines the components (together with their interaction) which are needed to support tasks above. This architecture builds on top of existing standard for Unstructured Information Management UIMA (UIM Architecture) (tasks 1&2) and, for task 3, on the Linguistic Watermark (Pazienza,

Stellato, & Turbati, 2008) suite of ontology vocabularies and software libraries for describing linguistic resources and the linguistic aspects of ontologies. Figure 1 depicts the part of the architecture supporting tasks 1 and 2. Tiny arrows represent the *use/depends on* relationship, so that the Semantic Repository owl:imports the reference ontologies, the projection component *invokes* services from the other three components in the CODA CAS Consumer as well as *is driven* by the projection document and TS and reference ontology. Large arrows represent instead the flow of information.

While UIMA already foresees the presence of CAS Consumers[1] for projecting collected data over any kind of repository (ontologies, databases, indices etc…), COD Architecture expands this concept by providing ground anchors for engineering ontology enrichment tasks, decoupling the several processing steps which characterize development and evolution of ontologies. This is our main original contribution to the framework. Here follows a description of the presented components.

Projection Component

This is the main component which realizes the projection of information extracted through traditional UIM components (i.e. UIMA *Annotations*).

The Unstructured Information Management (UIM) standard foresees data structures stored in a CAS (Common Analysis System). CAS data comprises a *type system*, i.e. a description – represented through feature structures (Carpenter, 1992) – of the kind of entities that can be manipulated in the CAS, and the *data* (modeled after the above type system) which is produced over processed information stream.

This component thus takes as input:

- A *Type System* (TS)
- A reference *ontology* (we assume the ontology to be written in the RDFS or OWL W3C standard)
- A projection document containing projection rules from the TS to the ontology
- A CAS containing annotation data represented according to the above TS

and uses all the above in order to project UIMA annotations as data over a given Ontology Repository.

The language for defining projections allows for:

- *Projecting CAS feature structures (FS) as instances of a given class*. FeaturePaths can be used to project arbitrary feature values as instance names
- *Projecting FSs as values of datatype properties*. Note that this requires ontology instances to be elected as subjects for each occurrence of this property annotation. The domain class which will be used to look for instance can be specified in the projection rule. Note that, by default, the domain of the property is inherited from the ontology, though it may be further restricted for the specific rule. So, for example, if property date has owl:Thing as its domain (i.e. no domain restriction), the outcome of a specific Analysis Engine, which is able to capture dates for conference events (or which is being used in a given setting for this purpose), can be restricted in the projection rule to automatically search for instances of the restricted domain. The use that is made of the above information is partially demanded to the application context, in order to properly select the right instances to be associated to the valued property.
- *Projecting complex FSs as custom graph patterns*. Some TS provide complex extraction patters which contains much more than plain text annotations; they possibly provide *facts* with explicit semantics which only need to be properly imported into the ontology. In this case, custom RDF graph patterns can be defined to create new complex relations inside the ontology. GRAPH Patterns are sets of RDF triples, in this case enriched by the presence of bindings to TS elements (again, in the form of FeaturePaths). When this projection is being applied, the feature path bindings are resolved and the ground pattern is used in a SPARQL CONSTRUCT query to generate new RDF triples in the Semantic Repository).

The Projection Component can be used in different scenarios (from massively automated ontology learning/population scenarios, to support in human centered processes for ontology modeling/data entry) and its projection processes can be supported by the following components.

Identity Resolution Component

Whenever an annotation is projected towards ontology data, the services of this component are invoked to identify potential matches between the annotated info which is being reified into the semantic repository, and previously recognized resources already present inside it.

If the Identity Resolution (IR) component discovers a match, then the new entry is merged into the existing one; that is, any new data is added to the resource description while duplicated information (probably the one which helped in finding the match) is discarded.

The IR component may look up on the same repository which is being fed by CODA though also external repositories of LOD (linked open data) can be accessed. Eventually, entity naming resolution provided by external services – such as the Entity Naming System (ENS) OKKAM (Bouquet, Stoermer, & Bazzanella, 2008) – may be combined with internal lookup on the local repository.

Input for this component are:

- External RDF repositories (providing at least indexed approximate search over their resources)
- Entity Naming Systems access methods
- Other parameters needed by specific implementation of the component

Projection Disambiguation Component(s)

These components may be invoked by the Projection Component to disambiguate between different possible projections. Projection documents may in fact describe more than one projection rule which can be applied to given types in the TS. These components are thus, by definition, associated to entries in Projection Documents

[1] UIMA terminology is widely adopted along the paper: though some explanations are provided here, we refer non-proficient readers to the *UIMA Glossary* inside the *UIMA Overview & SDK Setup* document, which is available at: http://incubator.apache.org/uima/documentation.html

and are automatically invoked when more than a rule is matched on the incoming CAS data.
This component has access by default to the current Semantic Repository (and any reference ontology for the Projection rules), to obtain a picture of the ongoing process which can contribute to the disambiguation process.

Smart Suggestion Component(s)

These components help in proposing suggestions on how to fill empty slots in projection rules (such as subject instances in datatype property projections or free variables in complex FS to graph-pattern projections). As for Disambiguation Components, these components can be written for specific Projection Documents and associated to the rules described inside them, as supporting computational objects.

3.3. CODA Framework Objectives

CODA Framework is an effort to facilitate development of systems implementing the COD Architecture, by providing a core platform and highly reusable components for realization of COD tasks.
Main objectives of this architectural framework are:

1. Orchestration of all processes supporting COD tasks
2. Interface-driven development of COD components
3. Maximizing reuse of components and code
4. Tight integration with available environments, such as UIMA for management of unstructured information from external resources (e.g. text documents) and Linguistic Watermark (Pazienza, Stellato, & Turbati, 2008) for management of linguistic resources
5. Minimizing required LOCs (lines of code) and effort for specific COD component development, by providing high level languages for matching/mapping components I/O specifications instead of developing software adapters for their interconnection
6. Providing standard implementations for components realizing typical support steps for COD tasks, such as management of corpora, user interaction, validation, evaluation, production of reference data (oracles, gold standards) for evaluation, identity discoverers etc…

In the specific, with respect to components described in section 3.2, CODA Framework will provide the main Projection Component (and its associated projection language), a basic implementation of an Identity Resolution Component, and all the required business logic to fulfill COD tasks through orchestration of COD components.

4. Possible application scenarios

Willing to fulfill these objectives, we envision several application scenarios for CODA. We provide here a description of a few of them.

Fast Integration of existing UIMA components for ontology population

By providing projections from CAS type systems to ontology vocabularies, one could easily embed standard UIMA AEs (Analysis Engines) and make them able to populate ontology concepts pointed by the projections, without requiring developing any new software component. These projections, which are part of objective 5 above, will be modeled through a dedicated language which will be part of the CODA framework. Moreover (objective 6 above), standard or customized identity discoverers will try to suggest potential matches between entities annotated by the AE and already existing resources in the target ontology, to keep identity of individual resources and add further description to them. In this scenario, given an ontology and a AE, only the projection from the CAS type system of the AE to the ontology is needed (and optionally, a customized identity discoverer). Everything else is assumed to be automatically embedded and coordinated by the framework.

Rapid prototyping of Ontology Learning Algorithms

This is the opposite situation of the scenario above. CODA, by reusing the same chaining of UIMA components, ontologies, CAS-to-Ontology projections, identity discoverers etc… , will provide:

- a preconfigured CAS type system (Ontology Learning CAS Type System) for representing information to be extracted under the scope of standard ontology learning tasks (i.e. the ones discussed in section 3.1)
- preconfigured projections from above CAS type system to learned ontology triples
- extended interface definitions for UIMA analysis engines dedicated to ontology learning tasks: available abstract adapter classes will implement the standard UIMA AnalysisComponent interface, interacting with the above Ontology Learning CAS type system and exposing specific interface methods for the different learning tasks

In this scenario, developers willing to rapidly deploy prototypes for new ontology learning algorithms, will be able to focus on algorithm implementation and benefit of the whole framework, disburdening them from corpora management and generation of ontology data. This level of abstraction far overtakes the *Modeling Primitive Library* of Text2Onto (i.e. a set of generic modeling primitives abstracting from specific ontology model adopted and being based on the assumption that the ontology exposes at least a traditional object oriented design, such as that of OKBC (Chaudhri, Farquhar, Fikes, Karp, & Rice, 1998)). In fact CODA does not even leaves to the developer the task of generating new ontology data, while just asks for specific objects to be associated and thus produced for given ontology learning tasks. For example, pairs of terms could be produced by taxonomy learners, which need then to be projected as IS-A or type-of relationships by the framework.

Plugging of algorithms for automatic linguistic enrichment of ontologies

In such a scenario, the user is interested in enriching ontologies with linguistic content originated from external lexical resources. The Linguistic Watermark library - which is already been used in tools for (multilingual) linguistic enrichment of ontologies (Pazienza, Stellato, & Turbati, 2010) and which constitutes a fundamental module of CODA - supports uniform access to

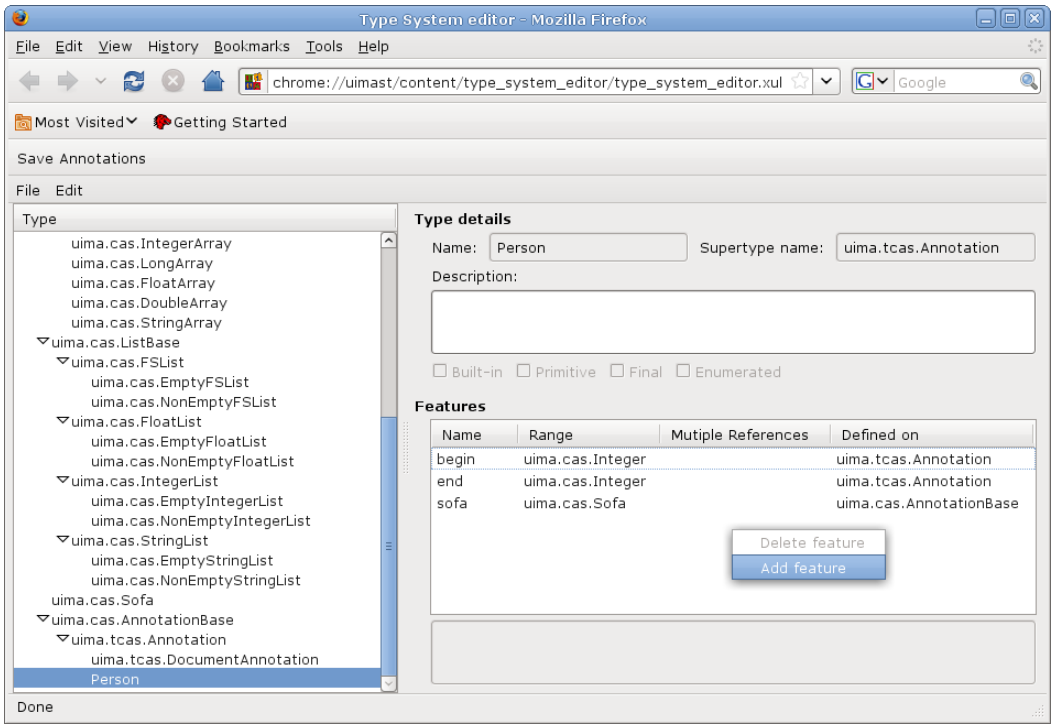


Figure 2. UIMAST Type System Editor

heterogeneous resources wrapped upon a common model for lexical resource definition, allows for their integration with ontologies and for evaluation of the acquired information. Once more, the objective is to relieve developers from technical details such as resource access, ontology interaction and update, by providing standard facilities associated to tasks for ontology-lexicon integration/enrichment, and thus leaving up to them the sole objective of implementing enrichment algorithms.

User Interaction for Knowledge Acquisition and Validation

User interaction is a fundamental aspect when dealing with decision-support systems. Prompting the user with compact and easy-to-analyze reports on the application of automated processes, and putting at his hands instruments for validating choices made by the system can dramatically improve the outcome of processes for knowledge acquisition as well as support supervised training of these same processes. CODA front-end tools should thus provide CODA specific applications supporting training of learning-based COD components, automatic acquisition of information from web pages visualized through the browser (or management of info previously extracted from entire corpora of documents) and editing of main CODA data structures (such as UIMA CAS types, projection documents and, obviously, ontologies). Interactive tools should support iterative refinement of massive production of ontology data as well as human-centered process for ontology development/evolution.

This last important environment is a further very relevant objective, and motivated us to define and develop UIMAST, an extension for Semantic Turkey (Griesi, Pazienza, & Stellato, 2007; Pazienza, Scarpato, Stellato, & Turbati, 2008), - a Semantic Web Knowledge Acquisition and Management platform[2] hosted on the Firefox Web Browser - to act as a CODA front-end for doing interactive knowledge acquisition from web pages.

5. UIMAST: A CODA-based tool supporting dynamic ontology population

The UIMAST Project[3] originated in late 2008, with the intent of realizing a system for bringing UIMA support to Semantic Turkey's functionalities for Knowledge Acquisition. The project has been organized around two main milestones:

- Supporting manual production of UIMA CAS compliant annotations
- Reuse UIMA annotators to automatically extract information from web pages and project them over the edited ontology

Milestone 1 has been reached in early 2009, with the first release of UIMAST. This release features:

1. *A UIMA Type System Editor* (figure 2 above), more intuitive to use than the Eclipse-based one bundled with UIMA, in that it provides a taxonomical view of edited Feature Structures, showing explicit and inherited attributes for each Type.
2. *Interactive UIMA annotator*: Semantic Annotations taken through Semantic Turkey can be projected as

[2] https://addons.mozilla.org/it/firefox/addon/8880 is the official page on Firefox add-ons site addressing Semantic Turkey extension, while http://semanticturkey.uniroma2.it/ provides an inside view about Semantic Turkey project, with updated downloads, user manuals, developers support and access to ST extensions.

[3] http://semanticturkey.uniroma2.it/extensions/uimast/. The idea for the project has been awarded with IBM UIMA Innovation Award http://download.boulder.ibm.com/ibmdl/pub/software/dw/university/innovation/2007_uima_recipients.pdf

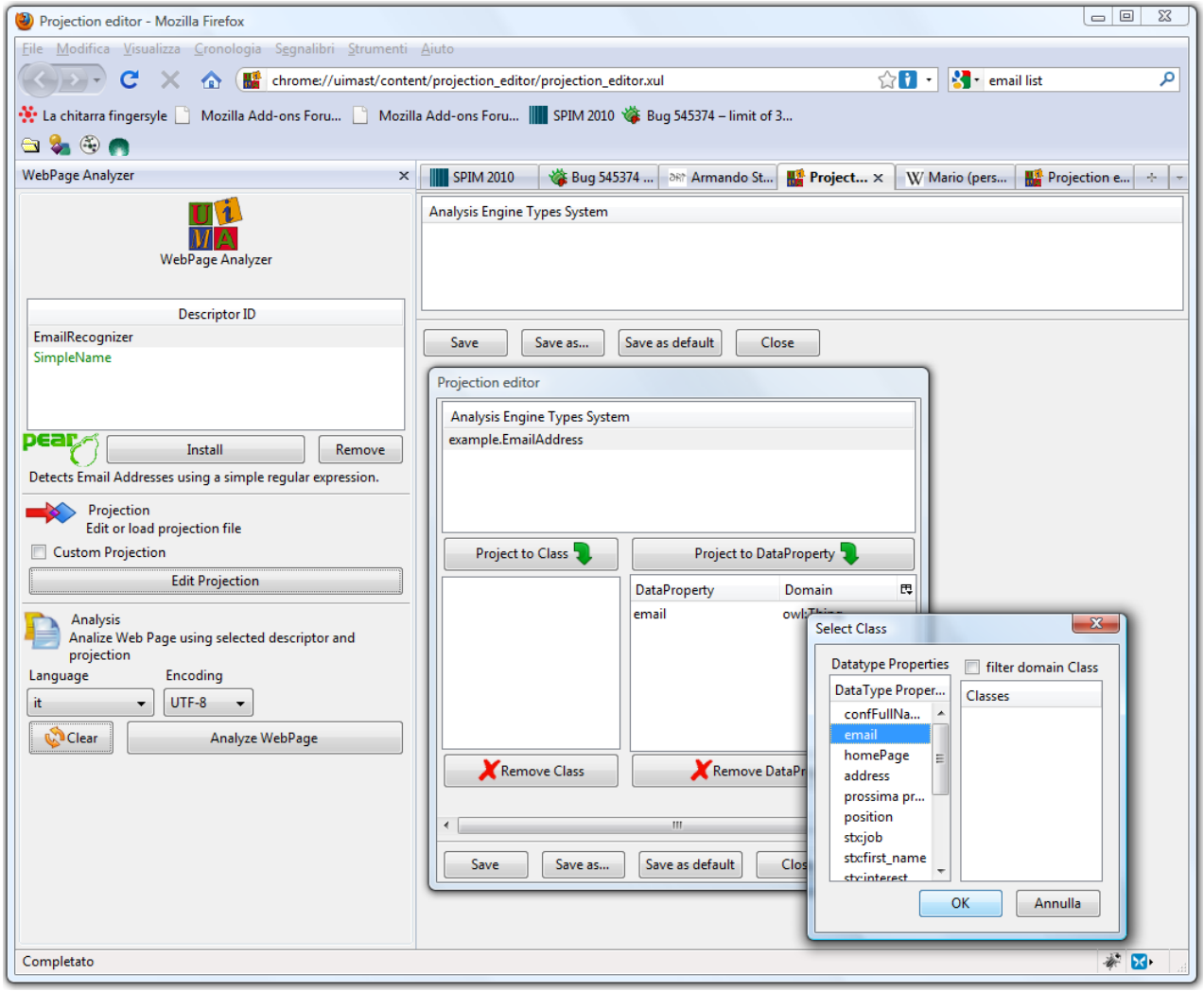


Figure 3. Editing Projections in UIMAST: from simple TS feature EmailAddress to ontology Datatype property email

UIMA Annotations. A xml based projection language[4]allows to project standard annotations taken against any domain ontology with respect to a given Type System. Currently there is no system supporting manual production of UIMA annotations from Web Pages. Annotations taken by human annotators can be reused to train machine-learning based AEs as well as to evaluate the output of AEs by producing golden-standard annotated documents.

Annotations taken through feature 2 can be exported in different formats, providing that their content can be projected according to *begin*/*end* attributes of UIMA AnnotationBase feature. By default, UIMAST exploits x-pointer annotations taken through the RangeAnnotator[5] extension of Semantic Turkey.

During Milestone 2, we produced a cross-SOFA[6] annotator which is able to parse content of specific document formats (such as HTML, PDF etc…) and produce cross-annotations setting links between pure raw-text surrogates of analyzed documents and their original source formats. An HTML version of this annotator thus accepts HTML documents, stores their content in a dedicated HTML SOFA, then runs an HTML SAX parser erupting raw-text content which is stored in a dedicated SOFA and cross-linked with the tag elements of the former one.

X-pointer annotations taken over the HTML page can thus be easily aligned with annotations taken over raw-text. This alignment allows to produce standard char-offset annotations starting from those manually taken with the interactive UIMA annotator, as well as to project automatically generated annotations produced by UIMA AEs (which usually work over raw text content) over X-Pointer references; as a consequence they can be visualized inside the same web page under analysis (which is the objective of milestone 2).

Currently, the new release of UIMAST provides:

1. A projection editor (figure 3: supporting only simple Class and Property projections)
2. A UIMA pear installer, able to load UIMA pear packages
3. The Visual Knowledge Acquisition Tool (KA Tool or simply KAT).

KAT provides visual anchors for users willing to semi-automatically import textual information present inside

[4] http://semanticturkey.uniroma2.it/extensions/uimast/schemas/projection-20081117.xsd

[5] http://semanticturkey.uniroma2.it/extensions/rangeannotator/

[6] SOFA: Subject OF Analysis, a perspective over a (multimodal) artifact, see UIMA User guide

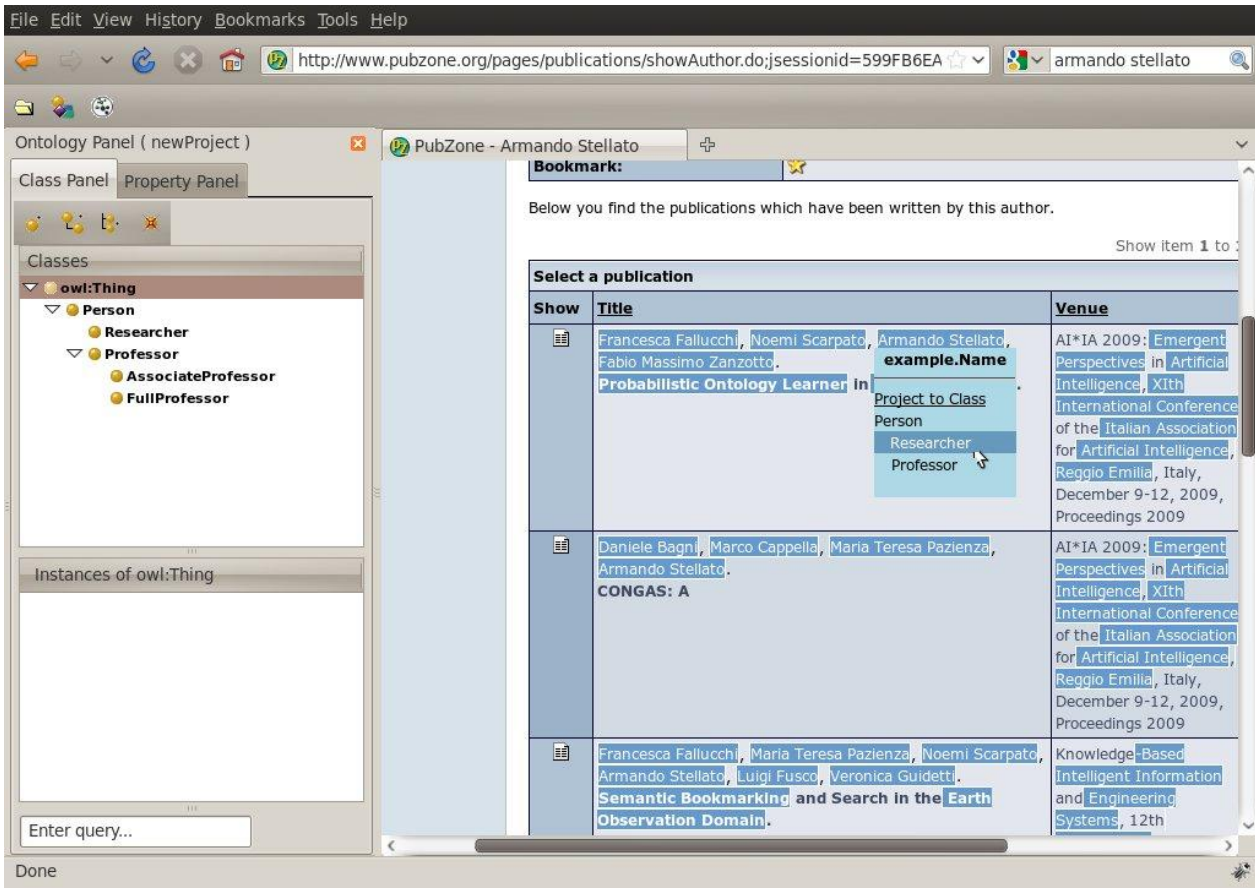


Figure 4. Knowledge Acquisition with UIMAST

web pages into the current working ontology of Semantic Turkey. While knowledge acquisition in standard Semantic Turkey requires the user to perform manual work (discovery of useful info and annotation) and decision making (produce data from annotated elements), UIMAST KAT heavily exploits the background knowledge available from the Type System of the loaded AEs and the projection document, as well as beneficiating of support coming from CODA components in the form of smart suggestions, resolved identities etc…) thus speeding up the acquisition process in the direction of automatizing the task.

In an ordinary KA session, the user starts by defining the tool setup: this implies loading one or more UIMA pears[7] through the *pear installer* and then loading a projection document associated to the currently edited ontology and the imported pears (all of the above may be stored as default settings for the ontology project being edited so that this process will not need to be repeated each time).

After tool setup, the user can immediately inspect web pages containing interesting data which can be extracted by the loaded AEs. The KAT then highlights all the text sections of the web page which have been annotated by the AEs. Each of these dynamically added highlights is not a purely visual alteration of the underlying HTML, but an active HTML component providing fast-to-click acceptance of proposed data acquisitions as well as more in-depth decision making procedures.

As an example of integrated process involving different resources in a user defined application, see figure 4 where a simple Named Entity Recognizer (the one bundled with UIMA sample AEs) has been projected towards ontology class Person. The AE has been launched and named entities discovered over the page have been highlighted. When the user passes with the mouse over one of these highlighted textual occurrences, the operation available from the projection doc is shown, and the user can right away either authorize its execution, or modify its details. In the example in the figure, the user has been prompted with the subtree rooted in the projected ontology class, and the user chooses to associate selected name to class Researcher instead of the more general Person. Should an identity resolution component discover that the given text may correspond to an existing resource (from the same edited ontology or from an external ENS), then he may choose to associate the taken annotation to it or reject it and create a new one.

6. Conclusion

The engineering of complex processes involving manipulation, elaboration and transformation of data and synthesis of knowledge is a recognized and widely accepted need, which lead in these years to the reformulation of tasks in terms of processing blocks other than (more than?) resolution steps. While traditional research fields such as Natural Language Processing and Knowledge Representation/Management have now found their standards, cross-boundary disciplines between the two need to find their way towards real applicability of approaches and proposed solutions. CODA aims at filling this gap by providing on the one hand a common

[7] A UIMA components package

environment for ontology development through knowledge acquisition, and on the other one by reusing the many solutions and technologies which years of research on these fields made easily accessible .

We hope that the ongoing realization of CODA will lead to a more mature support for research in the fields of both ontology learning and ontology/lexicon interfaces, .

7. References

Baker, C., Fillmore, C., & Lowe, J. (1998). The Berkeley FrameNet project. *COLING-ACL*. Montreal, Canada.

Basili, R., Vindigni, M., & Zanzotto, F. (2003). Integrating Ontological and Linguistic Knowledge for Conceptual Information Extraction. *IEEE/WIC International Conference on Web Intelligence.* Washington, DC, USA.

Bouquet, P., Stoermer, H., & Bazzanella, B. (2008). An Entity Naming System for the Semantic Web. *In Proceedings of the 5th European Semantic Web Conference (ESWC 2008).* Springer Verlag.

Buitelaar, P., Declerck, T., Frank, A., Racioppa, S., Kiesel, M., Sintek, M., et al. (2006). LingInfo: Design and Applications of a Model for the Integration of Linguistic Information in Ontologies. *OntoLex06.* Genoa, Italy.

Buitelaar, P., Olejnik, D., & Sintek, M. (2004). A Protégé Plug-In for Ontology Extraction from Text Based on Linguistic Analysis. *Proceedings of the 1st European Semantic Web Symposium (ESWS).* Heraklion, Greece.

Carpenter, B. (1992). *The Logic of Typed Feature Structures. Cambridge Tracts in Theoretical Computer Science* ((hardback) ed., Vol. 32). Cambridge University Press.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P., & Rice, J. P. (1998). OKBC: A programmatic foundation for knowledge base interoperability. *In Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)* (pp. 600-607). Madison, Wisconsin, USA: MIT Press.

Cimiano, P. (2006). *Ontology Learning and Population from Text Algorithms, Evaluation and Applications* (Vol. XXVIII). Springer.

Cimiano, P., & Völker, J. (2005). Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery. *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems*, (pp. 227-238). Alicante.

Cimiano, P., Haase, P., Herold, M., Mantel, M., & Buitelaar, P. (2007). LexOnto: A Model for Ontology Lexicons for Ontology-based NLP. *In Proceedings of the OntoLex07 Workshop (held in conjunction with ISWC'07).*

Cunningham, H., Maynard, D., Bontcheva, K., & Tablan, V. (2002). GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. *In Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02).* Philadelphia.

Ferrucci, D., & Lally, A. (2004). Uima: an architectural approach to unstructured information processing in the corporate research environment. *Nat. Lang. Eng. , 10* (3-4), 327-348.

Gennari, J., Musen, M., Fergerson, R., Grosso, W., Crubézy, M., Eriksson, H., et al. (2003). The evolution of Protégé-2000: An environment for knowledge-based systems development,. *International Journal of Human-Computer Studies , 58* (1), 89–123.

Griesi, D., Pazienza, M., & Stellato, A. (2007). Semantic Turkey - a Semantic Bookmarking tool (System Description). In E. Franconi, M. Kifer, & W. May (A cura di), *The Semantic Web: Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007, Proceedings. Lecture Notes in Computer Science. 4519*, p. 779-788. Springer.

Harman, D. (1992). The DARPA TIPSTER project. *SIGIR Forum , 26* (2), 26-28.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. (1993). *Introduction to WordNet: An On-line Lexical Database.*

Pazienza, M. T., & Stellato, A. (2006). Exploiting Linguistic Resources for building linguistically motivated ontologies in the Semantic Web. *Second Workshop on Interfacing Ontologies and Lexical Resources for Semantic Web Technologies (OntoLex2006), held jointly with LREC2006.* Magazzini del Cotone Conference Center, Genoa, Italy.

Pazienza, M. T., Stellato, A., & Turbati, A. (2010). A Suite of Semantic Web Tools Supporting Development of Multilingual Ontologies. In G. Armano, M. de Gemmis, G. Semeraro, & E. Vargiu (Eds.), *Intelligent Information Access. Studies in Computational Intelligence Series.* Springer-Verlag.

Pazienza, M., & Stellato, A. (2006). Linguistic Enrichment of Ontologies: a methodological framework. *Second Workshop on Interfacing Ontologies and Lexical Resources for Semantic Web Technologies (OntoLex2006).* Genoa, Italy.

Pazienza, M., Scarpato, N., Stellato, A., & Turbati, A. (2008). Din din! The (Semantic) Turkey is served! *Semantic Web Applications and Perspectives.* Rome, Italy.

Pazienza, M., Stellato, A., & Turbati, A. (2008). Linguistic Watermark 3.0: an RDF framework and a software library for bridging language and ontologies in the Semantic Web. *Semantic Web Applications and Perspectives, 5th Italian Semantic Web Workshop (SWAP2008).* FAO-UN, Rome, Italy.

Peter, H., Sack, H., & Beckstein, C. (2006). SMARTINDEXER – Amalgamating Ontologies and Lexical Resources for Document Indexing. *Workshop on Interfacing Ontologies and Lexical Resources for Semantic Web Technologies (OntoLex2006).* Genoa, Italy.

Peters, W., Montiel-Ponsoda, E., Aguado de Cea, G., & Gómez-Pérez, A. (2007). Localizing Ontologies in OWL. *In Proceedings of the OntoLex07 Workshop (held in conjunction with ISWC'07).*

Shi, L., & Mihalcea, R. (2005). Putting Pieces Together: Combining FrameNet, VerbNet and WordNet for Robust Semantic Parsing. *CICLing 2005*, (pp. 100-111). Mexico.

Velardi, P., Navigli, R., Cucchiarelli, A., & Neri, F. (2005). Evaluation of ontolearn, a methodology for automatic population of domain ontologie. In *Ontology Learning from Text: Methods, Applications and Evaluation.* IOS Press.

Building a French-speaking community around UIMA, gathering research, education and industrial partners, mainly in Natural Language Processing and Speech Recognizing domains

Nicolas Hernandez, Fabien Poulard, Matthieu Vernier, Jérôme Rocheteau

LINA (CNRS - UMR 6241) – University of Nantes
2 rue de la Houssinière – B.P. 92208, 44322 NANTES Cedex 3, France
first.last@univ-nantes.fr

Abstract

We report on the efforts we have made to build a UIMA French-speaking community both in Natural Language Processing and Speech Recognizing domains that would bring together researchers, industrials and educational interests. We present the services we set up as well as the resources we distribute freely under open licences to accomplish this objective. Most of them are currently available on the `uima-fr.org` Web Portal.

1. Introduction

Nowadays, it is crucial for a NLP (Natural Language Processing) laboratory which aims at playing a role within the national and the international community to acquire a robust middleware backbone to support its research and engineering activities. The issues are numerous:

- to ensure interoperability among the team members and with the project partners;
- to reuse existing software tools and consequently to benefit from preceding development efforts;
- to go beyond the prototype stage and to make possible the transfer toward industrial releases; In particular by taking into account scalable issues and distributed architecture;
- to be able to build more complex business applications;
- to be promptly operational and responsive to answer scientific and engineering national and international project calls;
- to demonstrate its know-how by deploying its software results as web services for example;
- to extend its business activities to data processing other than text such as audio or video.

In France, although GATE[1], NLTK[2] and Nooj[3] have been used as educational and research tools, there is no common agreement on the use of a particular platform for these purposes. Instead, it is not rare than researchers produce ad hoc solutions for their workflow management and language engineering problems.

Since December 2007, the NLP team of the LINA lab. at the University of Nantes (Nantes Atlantic Computer Science Laboratory) has explored the Apache UIMA framework as a middleware architecture to support its educational, research and engineering projects.
By comparison with the above-cited platforms, the Apache UIMA (Unstructured Information Management Architecture[4]) framework (Ferrucci and Lally, 2004) suits our needs for several reasons: Apache UIMA is an open, industrial-strength, scalable and extensible platform for creating, integrating and deploying unstructured (or semi-structured) information (text, audio, video...) management applications which help to build the bridge from unstructured information to structured knowledge. Apache UIMA dissociates the engineering middleware problems from NLP issues, its principles (semantic search and content analytics) result from a standardization effort at OASIS[5], its international community is very active, its Apache license fits research objectives and allows collaboration with industrial partners, and it is integrated in the Eclipse IDE (Integrated Development Environment).
So far the French-speaking community has paid little attention to this framework, mainly because, since Apache UIMA is a recent framework (the first Apache release is from December 2007), there were few NLP components compared to other frameworks like GATE; In particular for processing French. It is also correct to say that although several tools were available from the first release (such as a standalone workflow manager, an annotation viewer or a web REST service UIMA workflow deployer), the graphic user interfaces of these tools have remained quite basic.
In this paper, we report on the efforts we have made to build a UIMA French-speaking community both in Natural Language Processing and Speech Recognizing domains that would bring together researchers, industrials and educational interests. Our intentions of building this community are twofold:

1. to encourage the French-speaking academic and in-

[1] `gate.ac.uk`
[2] `www.nltk.org`
[3] `www.nooj4nlp.net`
[4] `incubator.apache.org/uima`
[5] `www.oasis-open.org/committees/uima`

dustrial organizations which have not yet adopt a middleware solution to use UIMA as a common development framework and middleware architecture for their research and engineering projects;

2. to improve the collaborative development of common UIMA-based NLP tools and components for processing French.

We present the services we have set up as well as the resources we distribute freely under open licences to accomplish this objective. Most of them are currently available on the `uima-fr.org` Web Portal. They consist of:

- A web portal to discuss and exchange information about UIMA;
- A bundle of scripts and resources for automatically installing the whole of the Apache UIMA SDK;
- A bundle of UIMA-based components including some French NLP preprocessing components, a type mapper and a semantic rule-based analyser;
- A bundle of UIMA tools including an Analysis Engine Apache Maven archetype and an advanced web rest server;
- Course and training materials.

Among similar efforts all around the world (Germany, Japan, UK, USA), we count the LTI repository at the Carnegie Mellon University[6], the Apache repository[7], the DKPro repository at the Darmstadt University [8] (Gurevych et al., 2007), the Julie lab repository[9] at the Jena University(Hahn et al., 2007) and the U-Compare project repository[10] (Kano et al., 2009).
These efforts are mainly dedicated to host UIMA tools and components. In comparison, we also focus on services and resources to help colleagues to quickly be productive with UIMA. Nevertheless it is important to notice that the project aims at encouraging the creation of a French-speaking community but it is not strictly dedicated to process French.
This project has been supported by both an IBM Unstructured Information Analytics 2008 Innovation Award and several LINA's research projects. The LINA have been using actively the UIMA framework in several ANR (National Research Agency) and local research projects. In the PIITHIE[11] project, we develop and deploy semantic and discourse analyzers as web services, in order to detect plagiarism and text reuse. In the Blogoscopie[12] project, we develop a component for opinionmining in blogs. In the C-Mantic[13] project, we aim at developing a semantic search engine with UIMA for the semantic analysis parts. In the Miles[14] project, UIMA will be used as the architecture to connect various geographically distributed components for speaker recognizing in text transcription. All these projects involve various academic and industrial partners.

[6] `uima.lti.cs.cmu.edu`
[7] `incubator.apache.org/uima`
[8] `www.ukp.tu-darmstadt.de/software/dkpro`
[9] `www.julielab.de/Resources/Software/NLP_Tools.html`
[10] `u-compare.org/components`
[11] `www.piithie.com` financed by the ANR under the Software Technologies Program 2006–2008
[12] `www.blogoscopie.org` financed by the ANR under the Software Technologies Program 2006–2008

2. The UIMA concepts

In the UIMA jargon, the *Common Analysis Structure* (CAS) is the data structure which is exchanged between the UIMA workflow components. It includes the data, subject of analysis and called the *Artefact*, and the metadata, in general simply called the *Annotations*, which describe the data. The annotations are stored in an index within the CAS. The annotations structure is called the *Types System* (TS) and consist of an implementation of a given annotation scheme. A UIMA workflow is made of three types of components: the *Collection Reader* (CR) which imports the data to process (for example from the Web or from the file system...) and turns it into a CAS. The *Analysis Engines* (AE) which literally process the data (including but not restricted to NLP analysis tasks); The annotations result from AE processing. And lastly the *CAS Consumer* (CC) which exports the annotations (for example to a database or to an XML representation of the analysis results).

3. The `uima-fr.org` web portal services

As part of the efforts, we count the launch of a French-speaking web portal about UIMA, `uima-fr.org`. This portal aims at developing a UIMA French-speaking community by providing services for French-speaking users and developers, researchers or professionals from both academic and industrial organizations to discuss and exchange information about UIMA.
Currently, the portal offers three main services allowing anyone to inform and to share informations about UIMA.

- a discussion list `sympa.univ-nantes.fr/wws/info/discussion-uima-fr`;
- a feed aggregator designed to collect posts from the blogs of any members of the community and display them on a single page `uima-fr.org/planet`;
- and a resource repository available under open license;

The first two services, the discussion list and the feed aggregator, aim at collecting FAQ explanations and HOWTO procedures in French. They act as a first step toward a more structured version of the content that a wiki could offer for example. Topics cover both users and developers interests, dealing with install, use, teaching and development issues both with the Apache UIMA framework and the third-part tools and components.
The third service aims at freely distributing the documentation, the tutorial and the education resources as well as the ready-to-use UIMA-based components and UIMA tools we created.

[13] `www.c-mantic.org` financed by the ANR under the Data mining and knowledge 2007–2009
[14] Regional project, "Pays de Loire" 2007–2009

4. Scripts and resources for installing the Apache UIMA SDK

The Apache UIMA Software development kit (SDK) is made of several tools and dependencies namely the Java Sun Development Kit (JDK), the Eclipse Integrated Development Environment (IDE), some Eclipse plugins and the Apache UIMA framework itself. This basic environment can be extended to include Apache Tomcat or other third-part tools or uima-based components.
Despite the fact that it exists a well-made documentation to help the installation and the use of all these tools and dependencies, it is not always easy to get into it because each tool has it own installation instructions, because it also may require a few engineer skills or sensitivity, because it can take some times to handle all of that...
To avoid all these disheartening aspects, we decided to provide some scripts to assist the download, the configuration, the installation of the UIMA SDK as well as its run within the Eclipse IDE.
We worked out that Eclipse would support the use of UIMA workflows and the development of UIMA-based components. The scripts were dedicated to run on Debian-like systems. They were validated on Ubuntu 8.10 Hardy and 9.04 Jaunty. Currently two versions of them are distributed:

- a light version which requires the launch of a download script to retrieve the tools and the dependencies of the UIMA SDK;
- and a standalone version which directly includes all the required tools as a resources package. The 20100207 version integrated the JDK 6u17, Eclipse Galileo 3.5 IDE, the subeclipse Subversion and the m2eclipse Maven eclipse plugins, the Apache UIMA 2.3.0-incubating framework, the Apache Tomcat 6.0.20 web server and the OpenNLP v1.3 toolkit.

In order to follow the progress of the resources, the scripts were written to work with a property file where the version and the url of the tools to use can be set up. These scripts are distributed under GPLv3 license.

5. UIMA-based components

We present below some French NLP preprocessing components as well as a type mapper and a semantic rule-based components.

5.1. French NLP preprocessing components and type system

In order to open the French NLP community to UIMA, we focused on the development and the distribution of preprocessing components which are commonly used in most of the NLP applications. The underlying idea was to offer a base from which collegues could directly start to work on their own applications and issues without losing times.
The components we worked out and distribute now permit the following processing: URI-based data import, MIME type recognition, text extraction, language recognition, NLP preprocessing (tokenization, stemming, POS tagging, lemmatization). We decided to wrap widely known and used tools whenever it was possible at least for three main reasons: First, it is a tremendous work to redevelop from scratch and we did not have the time neither the fund to do it. Second, so that novice people in UIMA but not in NLP wouldn't be too lost. And last but not the least, in order to enjoy the evolutions of other tools progressing independently.
Namely, we wrap the following tools and libraries: Apache Tika[15] (toolkit for detecting and extracting metadata and structured text content from various documents using existing parser libraries), nGRAMj (a Java based library providing robust and state of the art language recognition/guessing) [16], the Snowball library[17] (French stemmer), Brill (POS tagger) [18], TreeTagger Schmidt (POS tagger and lemmatizer) [19], Flemm (lemmatizer) [20]. In future work, we will include wrappers for TreeTagger and Minipar chunkers.
Our Tika Annotator acts in a complementary way with the Apache one since it works with URI as input and provided Dublin Core compliant markup annotations whenever the types of information are available. Compared to the TreeTagger wrapper of DKPro Repository, we map the TreeTagger outputs to the MulText annotation scheme.
Based on the capabilities of the available preprocessing tools for lemmatization, postagging and chunking in French, we have designed a Multext[21] compliant type system to annotate morphosyntaxic informations, with a particular attention to French language. This type system is currently experimented in our projects. Similarly to the Julie lab and U-compare project's type systems (Hahn et al., 2007; Kano et al., 2009), we intend to make the type system as generic as possible. Our type system is compliant with them as well as offering a specialization for French language.

5.2. A Type Mapper Component

One of the major issue dealing with any workflow management frameworks is the components interoperability. UIMA components only exchange data. So the data structure of the shared data is important since it ensures the interoperability.
Currently, there are at least three proposals for a tool- and domain-independent type system: The CCP meta model type system (Verspoor et al., 2009), the Julie lab's type system (Hahn et al., 2007) and the U-Compare project's type system (Kano et al., 2009). The former consists of a simple annotation hierarchy where the domain semantics is captured through pointers into external resources. The second and the third roughly consist of an abstract hierarchy of NLP concepts covering the various linguistic analysis levels.

[15] incubator.apache.org/tika/
[16] ngramj.sourceforge.net
[17] snowball.tartarus.org
[18] en.wikipedia.org/wiki/Brill_tagger
[19] www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger
[20] www.univ-nancy2.fr/pers/namer/Telecharger_Flemm.htm
[21] aune.lpl.univ-aix.fr/projects/MULTEXT and nl.ijs.si/ME/ Multilingual POS Tagset projects

According to us, tool and domain-independent type systems should be used whenever it is possible, at least as an example frame. But in our opinion, it will always be necessary to develop software converters from/to the proposed standard solution at least for two main raisons: First, in order to ensure the compatibility with pre-existing annotated data and processing softwares (which may have taken considerable time and funding to develop); this directly concerns the current tool- and domain-independent type systems[22]. Second, the need for new ad hoc type system will always exist for some problem adequacy reasons (unexplored application/domain/language, new or opposite theoretical approach, language dependency) and economic specifications (software resources to use imperatively).
For all these reasons, we decided to develop a generic converter Analysis Engine, called the Type Mapper, to permit the mapping of types or features values into others. The precondition for type mapping is that all the concerned types inherit from *uima.tcas.Annotation* so that they could have a begin and end offset. For each annotation of a given type, our Type Mapper component creates one or several annotations at the same offset. The code is totally generic except it requires both to put in the build path the various type systems classes involved as well as to set up the input/output capabilities of the converter. The mapping rules are specified in an external file as a parameter.
The preprocessing components package described in Section 5.1. includes a version for mapping the POS taggers and lemmatizer outputs to the Multext scheme.

5.3. A semantic rule-based analyser

Analysis is one of the major NLP tasks. A semantic rule-based analyser should enable to create or update annotations according to rules expressed over other annotations.
Currently, the Apache Regular Expression Annotator (RegexAnnotator) is an analysis engine which can create or update annotations but unfortunately it is dedicated to text surface analysis at the character level. The Apache Lucene CAS indexer (Lucas) are CAS consumers that stores annotations in an external index. It is then possible to make some structured searches in a Lucene application. The IBM Semantic Search, also referenced in the Apache UIMA repository, works similarly. These solutions are external applications and are not part of a UIMA workflow (at least natively). The Apache Configurable Feature Extractor (CFE) enables feature extraction from a CAS according to rules expressed using the Feature Extraction Specification Language (FESL). This is assurely a step toward the solution we are looking for but the CFE is a Cas Consumer and it does not enable natively annotation creation or update. The TextMarker component (Kluegl et al., 2009)[23] is a very appealing project but so far it remains very complex to use and quite dependent of the Eclipse environment.
Based on the Type Mapper we presented in Section 5.2., we started to develop a semantic rule-based analysis engine. We start from the Type Mapper since a semantic rule can be considered as a mapping operation from a contextually constrained source type. Currently, we are maintaining two development branches.
The former is based on the Apache UIMA API (Application Programming Interface). From a formal language we defined with ANTLR (ANother Tool for Language Recognition)[24], constraints are dynamically generated over the annotations, then the annotations are filtered according to the constraints. So far, the constraint language permits to specify annotations features and covered text values.
The latter branche follows an alternative approach. The idea was to transpose the problem to another domain where a request language over structured data and its processing engine are already available. The Lucas Annotator and the IBM Semantic Search developers chose to transpose to a database request problem. We chose to transpose the CAS analysis problem to a XML analysis problem. The XPath language offers to express constraints over a structure somehow quite similar to the text structure with the possibility to specify directions within. Futhermore it has several functions, in particular String functions. The major drawback is that XML is by definition a tree structure but not necessary the CAS. We solve the problem by using the JXPath[25] library which applies XPath expressions to graphs of objects of all kinds by setting a context node. So far, the two branches progress at the same level. This work is hosted at the Google forge[26].

6. UIMA tools

Below we present some tools we have been developed during our projects and we wish to distribute to the community.

6.1. An Advanced Web Rest Server

Among the tools available in the Apache repository, the Simple Server permits to provide UIMA analysis as a REST service. The server processes text raw data attached in HTTP request and outputs analysis results in an XML ad hoc format.
For the need of the PIITHIE project, we extended this version in three ways: First, the input source does not need to be attached to the HTTP request but can be specified by an URI (Universal Reference Identifier). The Server automatically upload the resource from the URI thanks to the JAVA API. Second, URI can refer to any resource formats. We included the Tika library in the server side. Third, the server can process XMI data in input and provides XMI in output. The server can turn the XMI into a CAS as long as the described annotations belong to a type system available in its classpath.
Recently, we decided to distribute this work and to extend it with new features such as PEAR (Processing Engine ARchive), collection and access rights management. This project is hosted at the Google forge[27].

[22] The U-Compare project comes with some ad hoc Type System converters from CCP, OpenNLP and Apache which turn them into the U-Compare Type System. It offers also some U-Compare Type System to OpenNLP.
[23] tmwiki.informatik.uni-wuerzburg.de
[24] www.antlr.org
[25] commons.apache.org/jxpath
[26] code.google.com/p/uima-type-mapper
[27] code.google.com/p/advanced-uima-web-rest-server

6.2. An Analysis Engine Maven Archetype

Apache Maven[28] is a build manager for Java projects. One of its features is the *archetype* facility which offers a way to define template project.
We built an Analysis Engine Maven Archetype in order to save the best practices we defined as well as to enable new developers jumping board as quickly as possible.
The archetype creates a repository project dedicated to the development of an Analysis Engine adding the UIMA nature to it. It comes with an annotator code and descriptor templates which include generic parameters. These parameters are meant to specify the input/output views/types/encoding. The archetype also creates some basic files such as a README and a LICENSE files.

7. Course and training materials

In order to animate and to train the community to the UIMA framework, we organized a training session during the 10th edition of the LSM/RMLL[29] (Libre Software Meeting) conferences 2009. This session presented how to build and to carry out processing chains and to develop his own UIMA components. For this purpose, we built tutorial-handouts, exercices and answers codes, and videos. We have also used Apache UIMA as a framework for educational purpose. We wrote course materials for Master's programs. We focused on writing UIMA components and interfacing UIMA with WEKA (Machine Leaning Library). All these resources are referenced on the `uima-fr.org` web portal.

8. Conclusion and future works

Many of the mentioned resources are currently dispatched on several web pages (LINA web pages[30], `uima-fr.org` planet and repository, Google forge projects). We are currently setting up a main index in the `uima-fr.org` repository. The UIMA-based components and UIMA tools are distributed under Apache 2.0[31] or GPLv3 [32] license. The training resources are distributed under a double License CC-by-sa fr 2.0[33] and GNU FDL[34]. One of our future works will concern the distribution of the components across an Apache Maven repository since it handles automatically the download of package dependencies.
The list of components we presented remains uncomplete. Indeed, our components bundle includes a Command Line analysis engine which performs on a given view the shell command specified and get the result in a dedicated annotation. This annotator is useful to easily and quickly integrate external softwares. The bundle includes also a XML-to-CAS analysis engine which parses any well formed XML data files and maps the XML structure, the elements and the attributs into generic annotation feature structures.
Some other components and tools are also planed depending on our project participations. In particular, in the context of the European TTC 2010–2012 project, we will develop UIMA wrappers for term extraction and alignement tools as well as UIMA collection management tools.
These resources and services have been set up by the Computer Sciences Laboratory of Nantes Atlantic (LINA), we invite anyone who wants to contribute to come and discuss in the `uima-fr.org` discussion list. Initially, due to some projects, the community was meant to be in Natural Language Processing and Speech Recognizing domains but it is widely open to any unstructured data management with UIMA issues.

[28] `maven.apache.org`
[29] `2009.rmll.info`
[30] `www.lina.univ-nantes.fr/-TALN-.html`
[31] `www.apache.org/licenses/LICENSE-2.0.html`
[32] GNU General Public License `www.gnu.org/licenses/gpl.html`
[33] Creative Commons Attribution-Noncommercial-Share Alike 2.0 France License `creativecommons.org/licenses/by-nc-sa/2.0/fr`
[34] GNU Free Documentation License, `www.gnu.org/licenses/fdl-1.2.html`

Acknowledgements

Currently, the research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement no 248005.

9. References

David Ferrucci and Adam Lally. 2004. Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348.

Iryna Gurevych, Max Mühlhäuser, Christof Müller, Jürgen Steimle, Markus Weimer, and Torsten Zesch and. 2007. Darmstadt knowledge processing repository based on uima. In *Proceedings of the First Workshop on Unstructured Information Management Architecture at Biannual Conference of the Society for Computational Linguistics and Language Technology*, Tübingen, Germany.

Udo Hahn, Ekaterina Buyko, Katrin Tomanek, Scott Piao, John McNaught, Yoshimasa Tsuruoka, and Sophia Ananiadou. 2007. An annotation type system for a data-driven nlp pipeline. In *The LAW at ACL 2007 – Proceedings of the Linguistic Annotation Workshop*, pages 33–40. Prague, Czech Republic, June 28-29, 2007. Stroudsburg, PA: Association for Computational Linguistics.

Yoshinobu Kano, Luke McCrohon, Sophia Ananiadou, and Jun'ichi Tsujii. 2009. Integrated NLP evaluation system for pluggable evaluation metrics with extensive interoperable toolkit. In *Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing (SETQA-NLP 2009)*, pages 22–30, Boulder, Colorado, June. Association for Computational Linguistics.

Peter Kluegl, Martin Atzmueller, and Frank Puppe. 2009. Textmarker: A tool for rule-based information extraction. In Christian Chiarcos, Richard Eckart de Castilho, and Manfred Stede, editors, *Proceedings of the Biennial GSCL Conference 2009, 2nd UIMA@GSCL Workshop*, pages 233–240. Gunter Narr Verlag.

Karin Verspoor, William Baumgartner Jr., Christophe Roeder, and Lawrence Hunter. 2009. Abstracting the types away from a uima type system. In *2nd UIMA Workshop at Gesellschaft für Sprachtechnologie und Computerlinguistik (GSCL)*, Tagung, Germany, October.

Generating an NLP Corpus from Java Source Code: The SSL Javadoc Doclet

Ninus Khamis, Juergen Rilling, and René Witte

Department of Computer Science and Software Engineering
Concordia University, Montréal, Canada

Abstract

Source code contains a large amount of natural language text, particularly in the form of comments, which makes it an emerging target of text analysis techniques. Due to the mix with program code, it is difficult to process source code comments directly within NLP frameworks such as GATE. Within this work we present an effective means for generating a corpus using information found in source code and in-line documentation, by developing a custom doclet for the *Javadoc* tool. The generated corpus uses a schema that is easily processed by NLP applications, which allows language engineers to focus their efforts on text analysis tasks, like automatic quality control of source code comments. The SSLDoclet is available as open source software.

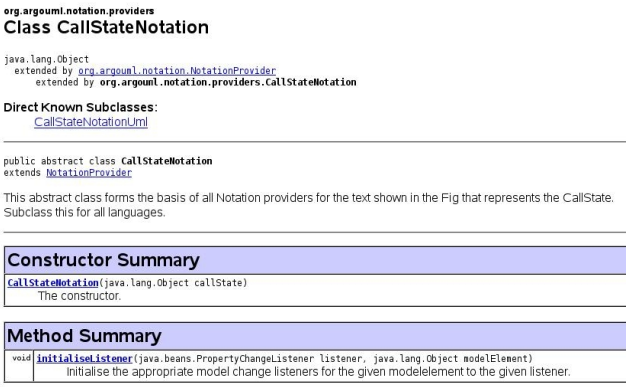
1. Introduction

One of the main challenges in Software Engineering is performing software maintenance tasks on an application that a developer is unfamiliar with. An important software engineering artefact used by developers and maintainers to assist in software comprehension and maintenance is source code documentation. Source code documents provide the insight needed by developers and maintainers to effectively perform their tasks, and therefore ensuring the quality of this documentation is extremely important. In-line documentation is at the forefront of explaining a programmer's original intentions for a given implementation. The blocks of text are inserted directly in source code, and are designed to efficiently assist others in understanding the source code. Since in-line documentation is written in natural language, they are a prime candidate for applying NLP and text mining methods, e.g., for quality assurance, ontology population, or traceability recovery. However, inline comments are mixed with source code in a way that makes them difficult to work with directly when using standard NLP tools such as GATE (Cunningham et al., 2002). *Javadoc* (Kramer, 1999) is a standard inline documentation tool that extracts and transform source code comments to HTML. This format is very well suited for human end users, but again not ideal for machine processing as many semantics are lost in this translation. For NLP processing, a meta-data language such as XML (Ray, 2003) is preferable.

In this paper, we introduce the *Semantic Software Lab Doclet (SSLDoclet)*, which is a custom Javadoc doclet that is able to generate a corpus using information found in source code and in-line documentation. The goal of the SSLDoclet is to covert the information found in source code by encoding it using an XML schema that is specifically designed for NLP applications. Our doclet is available under an open source license.[1]

2. Background

Javadoc (Kramer, 1999) is an automated tool that generates API documentation using Java source code and in-line documentation. In Figure 1 we show part of an API documentation generated using the Javadoc tool and the standard doclet.


org.argouml.notation.providers
Class CallStateNotation

java.lang.Object
  extended by org.argouml.notation.NotationProvider
    extended by org.argouml.notation.providers.CallStateNotation

Direct Known Subclasses:
CallStateNotationUml

public abstract class CallStateNotation
extends NotationProvider

This abstract class forms the basis of all Notation providers for the text shown in the Fig that represents the CallState. Subclass this for all languages.

Constructor Summary

CallStateNotation(java.lang.Object callState)
The constructor.

Method Summary

void initialiseListener(java.beans.PropertyChangeListener listener, java.lang.Object modelElement)
Initialise the appropriate model change listeners for the given modelelement to the given listener.

Figure 1: Excerpt of an API Documentation generated using the Standard Javadoc Doclet

Javadoc comments added to source code are distinguished from normal comments by a special comment syntax (`/**`). The `javadoc` tool extracts the source code and comments in order to transform the information into a variety of output formats, such as HTML, LaTeX, or PDF.

Javadoc provides an API that enables users to implement their own doclets[2] in order to generate documents using a desired output format.

3. Design & Implementation

Javadoc's standard doclet generates API documentation using the HTML format. While this is convenient for human consumption, automated NLP analysis applications require a more structured XML format.

The Javadoc library is in charge of parsing a source directory and providing an interface to a set of objects that is created as a result of the static source code analysis. Generation of the XML documents is then made possible by developing a custom doclet that uses the Javadoc library.

Implementing a custom doclet enabled us to (i) control what information from the source code will be included in the corpus, and (ii) mark up the information using a schema that NLP applications can easily process.

[1] SSLDoclet, http://www.semanticsoftware.info/javadoclet

[2] Java Doclet Overview, http://java.sun.com/j2se/1.4.2/docs/tooldocs/javadoc/overview.html

3.1. SSL Javadoc Doclet Design

In this section, we discuss in detail the level granularity and structure used when marking up Java source code using our SSLDoclet. As a running example, consider Figure 2, which shows an *Abstract Class* declaration taken from the open source project ArgoUML.

```
package org.argouml.notation.providers;

import java.beans.PropertyChangeListener;
import org.argouml.model.Model;
import org.argouml.notation.NotationProvider;

/**
 * This abstract class forms the basis of all Notation providers
 * for the text shown in the Fig that represents the CallState.
 * Subclass this for all languages.
 *
 * @author mvw@tigris.org
 */
public abstract class CallStateNotation extends NotationProvider
```

Figure 2: An Abstract Class Declaration taken from ArgoUML's Source Code

In Figure 3, we show the same declaration marked up using our XML meta-data tags, attributes, and elements.

```
<Abstract_Class_Block>
<Abstract_Class>CallStateNotation</Abstract_Class>
<Package>org.argouml.notation.providers</Package>
<Extends_Block>
        <Extends superclass="Object"
        qualifiedType="org.argouml.notation.NotationProvider"
                superclassFullType="java.lang.Object"
                        type="Abstract_Class">
        NotationProvider
        </Extends>
        <Extends_Comment>
                A class that implements this abstract class manages a
                text shown on a diagram. This means it is able to
                generate text that represents one or more UML objects.
                And when the user has edited this text, the model may be
                adapted by parsing the text.
                Additionally, a help text for the parsing is provided,
                so that the user knows the syntax.
        </Extends_Comment>
</Extends_Block>
<Class_Comment_Block>
        <Class_Comment>
                This abstract class forms the basis of all Notation
                providers for the text shown in the Fig that represents
                the CallState.
                Subclass this for all languages.
        </Class_Comment>
        <Author>mvw@tigris.org</Author>
</Class_Comment_Block>
```

Figure 3: A Section of the Corpus, generated using an Abstract Class Declaration

What makes XML superior over HTML for representing information that needs to be analysed by NLP applications, is that XML is much more versatile than HTML, and enables users to use custom *tags* and *attributes* to mark up the information of the XML elements (Ray, 2003), whereas with HTML we are limited to pre-defined tags such as `<p>` or `<head>`, and predefined attributes such as `font-size`. Such tags are designed to be rendered by a browser for human consumption (Antoniou and van Harmelen, 2008).

3.2. Marking Up Source Code

Our *SSLDoclet* is able to model both the syntactic and semantic information found in Java source code, such as:

- Parent/Child relationships between generalized and specialized *Classes*.
- The *Package* an *Interface* or *(Abstract) Class* belongs to.
- *Fields*, *Constructors* and *Methods* of a *Class*.
- The *types*, *modifiers (private, public, protected)*, and *constant* values of the fields.
- The *return types*, *parameter list*, and *thrown exceptions* of a *method*.

In Figure 3, we show how our doclet represents the information for the `CallStatNotation` abstract class, using the `<Package>` and `<Extends>` tags to model the package the abstract class belongs to, and the superclass that it extends. Figure 4 shows how the parameters of the `intialiseListener` method are modelled using the XML tag `<Parameter>`.

```
<Methods>
<Method_Block>
<Method modifier="public"
          visibility ="public" signature="()">
                enable
</Method>
<Method_Comment_Block>
        <Method_Comment>
                Method to enable the module.&lt;p&gt;
                If it cannot enable the module because some other
                module is not enabled it can return
                &lt;code&gt;false&lt;/code&gt;.
                In that case the module loader will defer this
                attempt until all other modules are loaded (or until
                some more of ArgoUML is loaded if at startup). Eventually
                it is only this and some other modules that is not loaded
                and they will then be listed as having problems.
        </Method_Comment>
</Method_Comment_Block>
<Return_Block>
        <Return>boolean</Return>
        <Return_Comment>true if all went well</Return_Comment>
</Return_Block>
</Method_Block>
</Methods>
```

Figure 4: A Section of the Corpus Generated using a Method Declaration

Both figures demonstrate how our doclet is able to represent more information effectively using XML attributes, compared to the standard HTML output. For example, we now also know that the parent of the `CallStatNotation`'s superclass is `Object`, and that the `listener` parameter of the `intialiseListener` method has the type *PropertyChangeListener*.

3.3. Marking Up Source Code Comments

Our SSLDoclet is also able to mark up the natural language information found in Javadoc comments, such as the *docComment*, *block*, and *in-line* tags.
Figure 2 shows an example of a Javadoc comment that includes a *docComment*, and uses the *@author* in-line tag. And in Figure 3, we show how Javadoc comments are

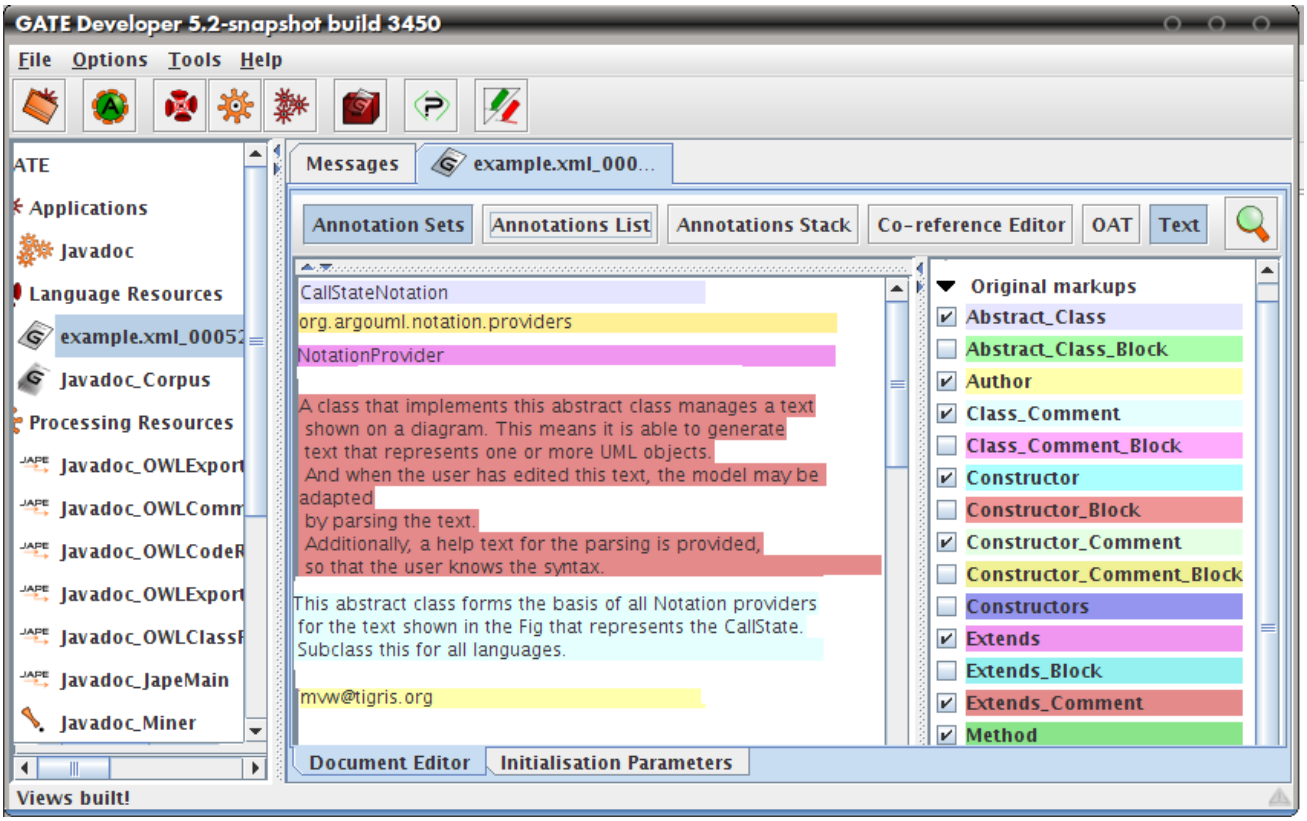


Figure 5: Corpus Generated using the SSLDoclet loaded within GATE

marked-up using the `<Extends_Comment>` tag, which contains the comment belonging to a super class. Additionally, the figure shows how the class comment belonging to `CallStatNotation` is represented using the `<Class_Comment>` and `<Author>` tags.

The SSLDoclet uses a schema that maintains the relationships found in source code, and represents the information using a combination of XML tags, attributes and elements. In Figure 6, we show how the relationships found in the sample source code is modelled. We eliminated the XML elements and attributes for readability purposes.

The information generated using the SSLDoclet could have been modelled in a number of different ways; however, it is important to keep in mind that when a corpus is loaded within an NLP framework such as GATE, the XML tags are interpreted as *annotations*, the XML elements are interpreted as *entities* of the annotation they belong to, and finally the XML attributes are interpreted as *features* of the annotation.

Our SSLDoclet is designed to generate a corpus using a schema that best utilizes how NLP frameworks interpret the information when initially loaded within the environment. In Figure 5, we show how *annotations*, *features*, and *entities* are created using only the original markups supplied by the corpus.

It is important that the information within a corpus be represented using a structure that enables the NLP environment to form an initial foundation that acts as the starting point for the automated NLP analysis.

```
<Abstract_Class_Block>
    <Abstract_Class/>
        <Package/>
            <Extends_Block>
                <Extends/>
                <Extends_Comment/>
            </Extends_Block>
            <Class_Comment_Block>
                <Class_Comment/>
                <Author/>
            </Class_Comment_Block>
            <Constructors>
            <Constructor_Block>
                <Constructor/>
                    <Constructor_Comment_Block>
                        <Constructor_Comment/>
                    </Constructor_Comment_Block>
                    <Constructor_Block>
                        <Constructor/>
                        <Parameter_Block>
                        <Parameter/>
                        <Parameter_Comment/>
                        </Parameter_Block>
                    </Constructor_Block>
            </Constructors>
            <Methods>
                <Method_Block>
                    <Method/>
                    <Parameter_Block>
                        <Parameter/>
                        <Parameter_Comment/>
                    </Parameter_Block>
                </Method_Block>
            </Methods>
</Abstract_Class_Block>
```

Figure 6: SSLDoclet Schema

4. Application and Evaluation

To execute the SSLDoclet, it is passed as a parameter to `javadoc` when processing a source directory. In Figure 7

Table 1: Open Source Project Versions, Lines of Code (LOC), Number of Comments and Identifiers, and Process Duration

Project	LOC	Number of Comments	Number of Identifiers	Duration (sec.)
ArgoUML v0.24	250,000	6,871	13,974	3.4
ArgoUML v0.26	600,000	6,875	14,262	8.9
ArgoUML v0.28.1	800,000	7,168	14,789	12.2
Eclipse v3.3.2	7,000,000	32,172	158,009	93.1
Eclipse v3.4.2	8,000,000	33,919	163,238	115.7
Eclipse v3.5.1	8,000,000	34,360	165,945	123.1

we show an example of a Javadoc `ant` task that indicates (i) the path and the name of the doclet, (ii) the path to the source directory, (iii) the name of the package in the source directory that needs to be processed, and finally (iv) any other additional parameters, for example, to increase the default Java heap space.

```
<target name="docs" depends="jar">
        <javadoc docletpath = "${doclet.dir}/
                               ${ant.project.name}.jar"
        doclet          = "${doclet}"
        sourcepath      = "${src.dir}"
        packagenames = "info.semanticsoftware.doclet"
        additionalparam = "-J-Xmx256m"
        />
</target>
```

Figure 7: Javadoc Ant Task that accepts the SSLDoclet as a Parameter

An NLP framework such as GATE can now process the generated XML meta-data as annotations, entities and features, which form the basis for the automated NLP analysis. In Figure 8, we show how GATE interprets the meta-data found within the corpus for the `intialiseListener` method.

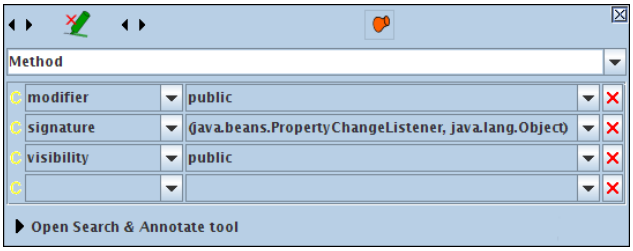


Figure 8: Annotations and Features created by GATE for a Method Declaration generated by the SSLDoclet

4.1. SSLDoclet Benchmarks

We performed a performance evaluation of our doclet to assess the time needed for creating a corpus from source code. Here, the SSLDoclet is passed as a parameter to the Javadoc parser. The parser is extremely efficient compared to other parsers when processing an entire source directory. In Table 1, we show the time required to process different versions of the ArgoUML and Eclipse open source projects.

4.2. Example Application: The JavadocMiner

We developed several NLP applications that currently use a corpus generated by the SSLDoclet as input. The *JavadocMiner* is a GATE application that assesses the quality of in-line documentation written in natural language found in source code. In Figure 9, we show an illustration of the processing resources that currently make up the JavadocMiner GATE pipeline. Each processing resource adds additional information in form of annotations to the corpus.

5. Related Work

A number of other doclets exist that can create XML files using `javadoc` and information found in source code, such as the `xml-doclet`,[3] Mavens's `XMLDoclet`,[4] and finally the `jeldoclet`.[5]

However, when looking at the schema generated by these doclets, we observed that the doclets were not necessarily designed for generating a corpus to be used within NLP applications.

For example, the `xml-doclet` marks up information using only XML tags and elements and does not make use of XML attributes to represent information. As mentioned earlier, XML attributes are interpreted by NLP frameworks as features of an annotation.

A doclet that generates a schema that closely resembles the SSLDoclet is the `jeldoclet`. The `jeldoclet` however does not attempt to differentiate between the different types of comments, which could minimize the descriptiveness of the corpus. The `jeldoclet` also does not capture the information provided by Javadoc when a certain class implements or extends another class, as shown in Figure 3. The source data being represented, and the output format is the same for all XML generating doclets, and the XML documents generated using the doclets mentioned herein can be loaded within an NLP framework. However, how the information is marked-up can drastically change the number of annotations, features and entities that are created, which can have a cascading effect on the rest of processing resource within the NLP application.

Having the most number of annotations, features or entities as result of how the information is marked up within an XML document is not necessarily beneficial. Providing a schema that enables NLP frameworks to differentiate between what is an annotation, feature, and entity is important when generating an XML document that is to be used as a corpus. None of the existing doclets that we examined were capable of doing so. For example, since the `xml-doclet` marks up all the information using XML tags only, no features are created when the document is loaded within an NLP framework and the number of annotations would exceed

[3] XML-Doclet, http://code.google.com/p/xml-doclet/
[4] Maven Doclet, http://maven.apache.org/maven-1.x/
[5] jeldoclet, http://jeldoclet.sourceforge.net/

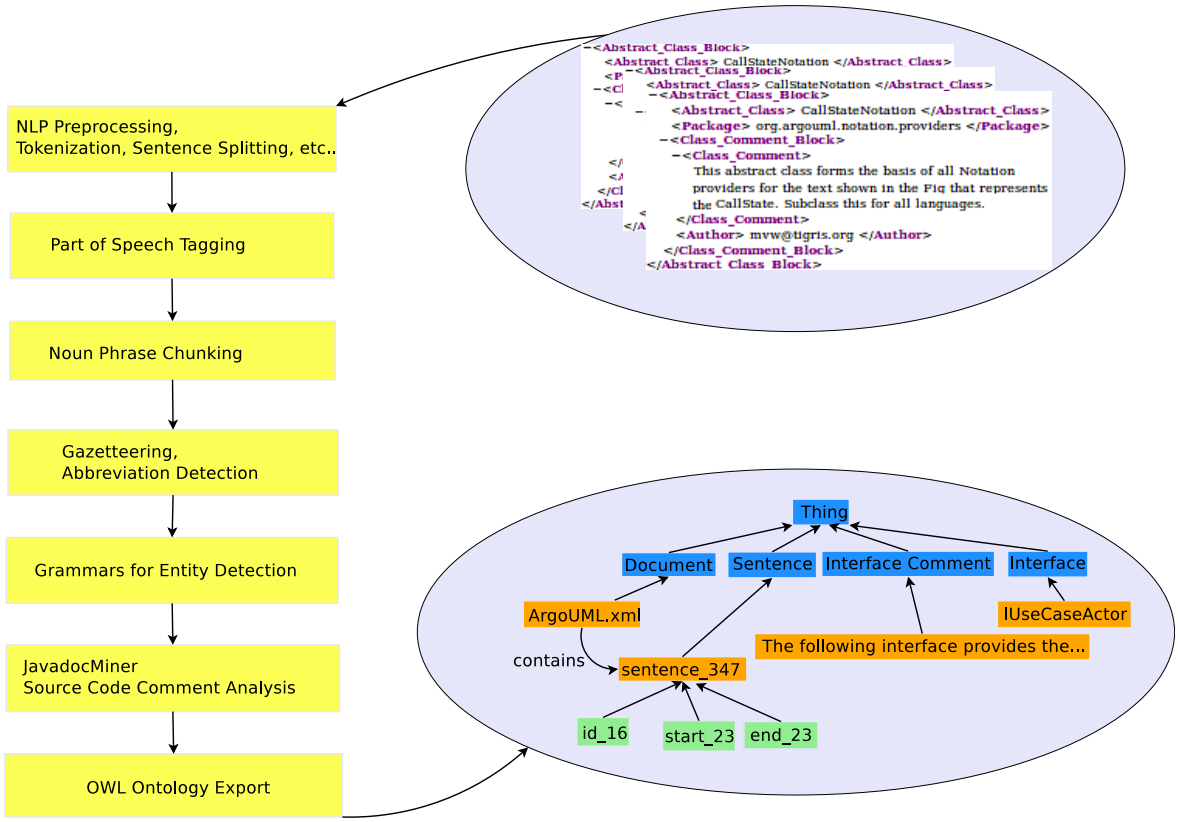


Figure 9: The JavadocMiner Pipeline for analysing the Quality of Source Code Comments

that of the SSLDoclet for the same amount of information. This will actually have a negative impact on the amount of work needed by the language engineers to make use of the generated corpus.

To conclude, even though there exists a number of XML generating doclets that can be downloaded from the net, we feel that our SSLDoclet differs from the rest due to its ability to generate XML output using a schema that is optimized for further NLP processing, which is an application scenario not targeted by existing efforts.

6. Conclusion & Future Work

In this paper, we presented a novel approach for using a custom doclet and Javadoc to generate a corpus that can be used as input to an NLP application. We emphasized the benefits of representing the information found in Java source code and in-line documentation using XML meta-data over HTML to facilitate automated NLP analyses.

We discussed how the SSLDoclet is able to generate a corpus using information found in both source code and in-line documentation. We also showed how the corpus can be used within existing text mining applications such as the JavadocMiner (Khamis et al., 2010).

And finally, we compared our SSLDoclet with other doclets that are currently published, and pointed out that it is the first to be explicitly designed for generating a corpus that is to be used within NLP applications. In particular, it is able to better differentiate between an annotation, feature, and entity, which the existing doclets are unable to do.

Future work is specifically needed in two areas: first, marking-up more of the information provided by Javadoc (for example, the information that exist in *enumerations*). This can be achieved by implementing more services using the Javadoc API. And second, enabling the user to generate multiple documents (a corpora) using a single source directory containing multiple files. The SSLDoclet will parse each source code file separately and generate an AST for each Java class. While this will add functionality to the SSLDoclet, we believe that it already provides significant functionality for language engineers targeting NLP analysis of source code and its inline comments.

7. References

Grigoris Antoniou and Frank van Harmelen. 2008. *A Semantic Web Primer*. The MIT Press, 2 edition, March.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002. GATE: an Architecture for Development of Robust HLT Applications. *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL)*.

Ninus Khamis, René Witte, and Juergen Rilling. 2010. Automatic Quality Assessment of Source Code Comments: The JavadocMiner. In *15th International Conference on Applications of Natural Language to Information Systems (NLDB 2010)*. Cardiff University, June 23-25.

Douglas Kramer. 1999. API documentation from source code comments: a case study of Javadoc. In *SIGDOC '99: Proceedings of the 17th annual international conference on Computer documentation*, pages 147–153, New York, NY, USA. ACM.

Erik T. Ray. 2003. *Learning XML*. O'Reilly & Associates, Sebastopol, California, 2nd edition, September.

Software Framework for Topic Modelling with Large Corpora

Radim Řehůřek and Petr Sojka

Natural Language Processing Laboratory
Masaryk University, Faculty of Informatics
Botanická 68a, Brno, Czech Republic
{xrehurek,sojka}@fi.muni.cz

Abstract

Large corpora are ubiquitous in today's world and memory quickly becomes the limiting factor in practical applications of the Vector Space Model (VSM). In this paper, we identify a gap in existing implementations of many of the popular algorithms, which is their scalability and ease of use. We describe a Natural Language Processing software framework which is based on the idea of *document streaming*, i.e. processing corpora document after document, in a memory independent fashion. Within this framework, we implement several popular algorithms for topical inference, including Latent Semantic Analysis and Latent Dirichlet Allocation, in a way that makes them completely independent of the training corpus size. Particular emphasis is placed on straightforward and intuitive framework design, so that modifications and extensions of the methods and/or their application by interested practitioners are effortless. We demonstrate the usefulness of our approach on a real-world scenario of computing document similarities within an existing digital library DML-CZ.

1. Introduction

"Controlling complexity is the essence of computer programming."
Brian Kernighan (Kernighan and Plauger, 1976)

The *Vector Space Model (VSM)* is a proven and powerful paradigm in NLP, in which documents are represented as vectors in a high-dimensional space. The idea of representing text documents as vectors dates back to early 1970's to the SMART system (Salton et al., 1975). The original concept has since then been criticised, revised and improved on by a multitude of authors (Wong and Raghavan, 1984; Deerwester et al., 1990; Papadimitriou et al., 2000) and became a research field of its own. These efforts seek to exploit both explicit and implicit document structure to answer queries about document similarity and textual relatedness. Connected to this goal is the field of topical modelling (see e.g. (Steyvers and Griffiths, 2007) for a recent review of this field). The idea behind topical modelling is that texts in natural languages can be expressed in terms of a limited number of underlying *concepts* (or *topics*), a process which both improves efficiency (new representation takes up less space) and eliminates noise (transformation into topics can be viewed as noise reduction). A topical search for related documents is orthogonal to the more well-known "fulltext" search, which would match particular words, possibly combined through boolean operators.

Research on topical models has recently picked up pace, especially in the field of generative topic models such as Latent Dirichlet Allocation (Blei et al., 2003), their hierarchical extensions (Teh et al., 2006), topic quality assessment and visualisation (Chang et al., 2009; Blei and Lafferty, 2009). In fact, it is our observation that the research has rather gotten ahead of applications—the interested public is only just catching up with Latent Semantic Analysis, a method which is now more than 20 years old (Deerwester et al., 1990). We attribute reasons for this gap between research and practice partly to inherent mathematical complexity of the inference algorithms, partly to high computational demands of most methods and partly to the lack of a "sandbox" environment, which would enable practitioners to apply the methods to their particular problem on real data, in an easy and hassle-free manner. The research community has recognised these challenges and a lot of work has been done in the area of accessible NLP toolkits in the past couple of years; our contribution here is one such step in the direction of closing the gap between academia and ready-to-use software packages[1].

Existing Systems

The goal of this paper is somewhat orthogonal to much of the previous work in this area. As an example of another possible direction of applied research, we cite (Elsayed et al., 2008). While their work focuses on how to compute pair-wise *document similarities* from individual document representations in a scalable way, using Apache Hadoop and clusters of computers, our work here is concerned with how to scalably compute these document representations in the first place. Although both steps are necessary for a complete document similarity pipeline, the scope of this paper is limited to constructing topical representations, not answering similarity queries.

There exist several mature toolkits which deal with Vector Space Modelling. These include NLTK (Bird and Loper, 2004), Apache's UIMA and ClearTK (Ogren et al., 2008), Weka (Frank et al., 2005), OpenNLP (Baldridge et al., 2002), Mallet (McCallum, 2002), MDP (Zito et al., 2008), Nieme (Maes, 2009), Gate (Cunningham, 2002), Orange (Demšar et al., 2004) and many others.

These packages generally do a very good job at their intended purpose; however, from our point of view, they also suffer from one or more of the following shortcomings:

[1] Interest in the field of document similarity can also be seen from the significant number of requests for a VSM software package which periodically crop up in various NLP mailing lists. Another indicator of interest are tutorials aimed at business applications; see web search results for "SEO myths and LSI" for an interesting treatment on Latent Semantic Indexing marketing.

No topical modelling. Packages commonly offer supervised learning functionality (i.e. classification); topic inference is an unsupervised task.

Models do not scale. Package requires that the whole corpus be present in memory before the inference of topics takes place, usually in the form of a sparse term-document matrix.

Target domain not NLP/IR. The package was created with physics, neuroscience, image processing, etc. in mind. This is reflected in the choice of terminology as well as emphasis on different parts of the processing pipeline.

The Grand Unified Framework. The package covers a broad range of algorithms, approaches and use case scenarios, resulting in complex interfaces and dependencies. From the user's perspective, this is very desirable and convenient. From the developer's perspective, this is often a nightmare—tracking code logic requires major effort and interface modifications quickly cascade into a large set of changes.

In fact, we suspect that the last point is also the reason why there are so many packages in the first place. For a developer (as opposed to a user), the entry level learning curve is so steep that it is often simpler to "roll your own" package rather than delve into intricacies of an existing, proven one.

2. System Design

"Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface."
Doug McIlroy (McIlroy et al., 1978)

Our choices in designing the proposed framework are a reflection of these perceived shortcomings. They can be explicitly summarised into:

Corpus size independence. We want the package to be able to detect topics based on corpora which are larger than the available RAM, in accordance with the current trends in NLP (see e.g. (Kilgarriff and Grefenstette, 2003)).

Intuitive API. We wish to minimise the number of method names and interfaces that need to be memorised in order to use the package. The terminology is NLP-centric.

Easy deployment. The package should work out-of-the-box on all major platforms, even without root privileges and without any system-wide installations.

Cover popular algorithms. We seek to provide novel, scalable implementations of algorithms such as TF-IDF, Latent Semantic Analysis, Random Projections or Latent Dirichlet Allocation.

We chose Python as the programming language, mainly because of its straightforward, compact syntax, multiplatform nature and ease of deployment. Python is also suitable for handling strings and boasts a fast, high quality library for numerical computing, *numpy*, which we use extensively.

Core interfaces

As mentioned earlier, the core concept of our framework is *document streaming*. A corpus is represented as a sequence of documents and at no point is there a need for the whole corpus to be stored in memory. This feature is not an afterthought on lazy evaluation, but rather a core requirement for our application and as such reflected in the package philosophy. To ensure transparent ease of use, we define *corpus* to be any iterable returning documents:

```
>>> for document in corpus:
>>>     pass
```

In turn, a document is a sparse vector representation of its constituent fields (such as terms or topics), again realised as a simple iterable:[2]

```
>>> for fieldId, fieldValue in document:
>>>     pass
```

This is a deceptively simple interface; while a corpus is allowed to be something as simple as

```
>>> corpus = [[(1, 0.8), (8, 0.6)]]
```

this streaming interface also subsumes loading/storing matrices from/to disk (e.g. in the Matrix Market (Boisvert et al., 1996) or SVMlight (Joachims, 1999) format), and allows for constructing more complex real-world IR scenarios, as we will show later. Note the lack of package-specific keywords, required method names, base class inheritance etc. This is in accordance with our main selling points: ease of use and data scalability.

Needless to say, both corpora and documents are not *restricted* to these interfaces; in addition to supporting iteration, they may (and usually do) contain additional methods and attributes, such as internal document ids, means of visualisation, document class tags and whatever else is needed for a particular application.

The second core interface are *transformations*. Where a corpus represents data, transformation represents the process of translating documents from one vector space into another (such as from a TF-IDF space into an LSA space). Realization in Python is through the dictionary [ ] mapping notation and is again quite intuitive:

```
>>> from gensim.models import LsiModel
>>> lsi = LsiModel(corpus, numTopics = 2)
>>> lsi[new_document]
[(0, 0.197), (1, -0.056)]

>>> from gensim.models import LdaModel
>>> lda = LdaModel(corpus, numTopics = 2)
>>> lda[new_document]
[(0, 1.0)]
```

[2] In terms of the underlying VSM, which is essentially a sparse field-document matrix, this interface effectively abstracts away from both the number of documents and the number of fields. We note, however, that the abstraction focus is on the number of *documents*, not fields. The number of terms and/or topics is usually carefully chosen, with unwanted token types removed via document frequency thresholds and stoplists. The hypothetical use case of introducing new fields in a streaming fashion does not come up as often in NLP.

2.1. Novel Implementations

While an intuitive interface is important for software adoption, it is of course rather trivial and useless in itself. We have therefore implemented some of the popular VSM methods, two of which we will describe here in greater detail.

Latent Semantic Analysis, LSA. Developed in late 80's in Bell Laboratories (Deerwester et al., 1990), this method gained popularity due to its solid theoretical background and efficient inference of topics. The method exploits co-occurrence between terms to project documents into a low-dimensional space. Inference is done using linear algebra routines for truncated Singular Value Decomposition (SVD) on the sparse term-document matrix, which is usually first weighted by some TF-IDF scheme. Once the SVD has been completed, it can be used to project new documents into the latent space, in a process called *folding-in*.

Since linear algebra routines have always been the front runner of numerical computing (see e.g. (Press et al., 1992)), some highly optimised packages for sparse SVD exist. For example, PROPACK and SVDPACK are both based on the Lanczos algorithm with smart reorthogonalizations, and both are written in FORTRAN (the latter also has a C-language port called SVDLIBC). Lightning fast as they are, adapting the FORTRAN code is rather tricky once we hit the memory limit for representing sparse matrices directly in memory. For this and other reasons, research has gradually turned to incremental algorithms for computing SVD, in which the matrix is presented sequentially—an approach equivalent to our *document streaming*. This problem reformulation is not trivial and only recently have there appeared practical algorithms for incremental SVD.

Within our framework, we have implemented Gorrell's Generalised Hebbian Algorithm (Gorrell, 2006), a stochastic method for incremental SVD. However, this algorithm proved much too slow in practice and we also found its internal parameters hard to tune, resulting in convergence issues. We have therefore also implemented Brand's algorithm for fast incremental SVD updates (Brand, 2006). This algorithm is much faster and contains no internal parameters to tune[3]. To the best of our knowledge, our pure Python (numpy) implementation is the only publicly available implementation of LSA that does not require the term-document matrix to be stored in memory and is therefore independent of the corpus size[4]. Together with our straightforward document streaming interface, this in itself is a powerful addition to the set of publicly available NLP tools.

Latent Dirichlet Allocation, LDA. LDA is another topic modelling technique based on the bag-of-words paradigm and word-document counts (Blei et al., 2003). Unlike Latent Semantic Analysis, LDA is a fully generative model, where documents are assumed to have been generated according to a per-document topic distribution (with a Dirichlet prior) and per-topic word distribution. In practice, the goal is of course not generating random documents through these distributions, but rather inferring the distributions from observed documents. This can be accomplished by variational Bayes approximations (Blei et al., 2003) or by Gibbs sampling (Griffiths and Steyvers, 2004). Both of these approaches are incremental in their spirit, so that our implementation (again, in pure Python with numpy, and again the only of its kind that we know of) "only" had to abstract away from the original notations and implicit corpus-size allocations to be made truly memory independent. Once the distributions have been obtained, it is possible to assign topics to new, unseen documents, through our transformation interface.

[3]This algorithm actually comes from the field of image processing rather than NLP. Singular Value Decomposition, which is at the heart of LSA, is a universal data compression/noise reduction technique and has been successfully applied to many application domains.

[4]This includes completely ignoring the right singular vectors during SVD computations, as the left vectors together with singular values are enough to determine the latent space projection for new documents.

2.2. Deployment

The framework is heavily documented and is available from `http://nlp.fi.muni.cz/projekty/gensim/`. This website contains sections which describe the framework and provide usage tutorials, as well as installation instructions.

The framework is open sourced and distributed under an OSI-approved LGPL license.

3. Application of the Framework

"An idea that is developed and put into action is more important than an idea that exists only as an idea."
Hindu Prince Gautama Siddharta, the founder of Buddhism, 563–483 B.C.

3.1. Motivation

Many digital libraries today start to offer browsing features based on pairwise document content similarity. For collections having hundreds of thousands documents, computation of similarity scores is a challenge (Elsayed et al., 2008). We have faced this task during the project of The Digital Mathematics Library DML-CZ (Sojka, 2009). The emphasis was not on developing new IR methods for this task, although some modifications were obviously necessary—such as answering the question of what constitutes a "token", which differs between mathematics and the more common English ASCII texts.

With the collection's growth and a steady feed of new papers, lack of scalability appeared to be the main issue. This drove us to develop our new document similarity framework.

3.2. Data

As of today, the corpus contains over 61,293 fulltext documents for a total of about 270 million tokens. There are mathematical papers from the Czech Digital Mathematics Library DML-CZ `http://dml.cz` (22,991 papers), from the NUMDAM repository `http://numdam.org` (17,636 papers) and from the math part of arXiv `http://arxiv.org/archive/math` (20,666 papers). After filtering out word types that either appear less than five times in the corpus (mostly OCR errors) or in more than one half of the documents (stop words), we are left with 315,167

distinct word types. Although this is by no means an exceptionally big corpus, it already prohibits storing the sparse term-document matrices in main memory, ruling out most available VSM software systems.

3.3. Results

We have tried several VSM approaches to representing documents as vectors: term weighting by TF-IDF, Latent Semantic Analysis, Random Projections and Latent Dirichlet Allocation. In all cases, we used the cosine measure to assess document similarity.

When evaluating data scalability, one of our two main design goals (together with ease of use), we note memory usage is now dominated by the transformation models themselves. These in turn depend on the vocabulary size and the number of topics (but not on the training corpus size). With 315,167 word types and 200 latent topics, both LSA and LDA models take up about 480 MB of RAM.

Although evaluation of the quality of the obtained similarities is not the subject of this paper, it is of course of utmost practical importance. Here we note that it is notoriously hard to evaluate the quality, as even the preferences of different types of similarity are subjective (match of main topic, or subdomain, or specific wording/plagiarism) and depends on the motivation of the reader. For this reason, we have decided to present all the computed similarities to our library users at once, see e.g. `http://dml.cz/handle/10338.dmlcz/100785/SimilarArticles`. At the present time, we are gathering feedback from mathematicians on these results and it is worth noting that the framework proposed in this paper makes such side-by-side comparison of methods straightforward and feasible.

4. Conclusion

We believe that our framework makes an important step in the direction of current trends in Natural Language Processing and fills a practical gap in existing software systems. We have argued that the common practice, where each novel topical algorithm gets implemented from scratch (often inventing, unfortunately, yet another I/O format for its data in the process) is undesirable. We have analysed the reasons for this practice and hypothesised that this partly due to the steep API learning curve of existing IR frameworks.

Our framework makes a conscious effort to make parsing, processing and transforming corpora into vector spaces as intuitive as possible. It is platform independent and requires no compilation or installations past Python+numpy. As an added bonus, the package provides ready implementations of some of the popular IR algorithms, such as Latent Semantic Analysis and Latent Dirichlet Allocation. These are novel, pure-Python implementations that make use of modern state-of-the-art iterative algorithms. This enables them to work over practically unlimited corpora, which no longer need to fit in RAM.

We believe this package is useful to topic modelling experts in implementing new algorithms as well as to the general NLP community, who is eager to try out these algorithms but who often finds the task of translating the original implementations (not to say the original articles!) to its needs quite daunting.

Future work will include comparison of the usefulness of different topical models to the users of our Digital Mathematical Library, as well as further improving the range, efficiency and scalability of popular topic modelling methods.

Acknowledgments

We acknowledge the support of grant MUNI/E/0084/2009 of the Rector of Masaryk University program for PhD students' research. Partial support of grants by EU #250503 CIP-ICT-PSP EuDML and by the Ministry of Education of CR within the Centre of basic research LC536 is acknowledged, too. We would also like to thank the anonymous reviewer for providing us with additional pointers and valuable comments.

5. References

J. Baldridge, T. Morton, and G. Bierner. 2002. The OpenNLP maximum entropy package. Technical report. `http://maxent.sourceforge.net/`.

Steven Bird and Edward Loper. 2004. NLTK: The Natural Language Toolkit. *Proceedings of the ACL demonstration session*, pages 214–217.

David M. Blei and John D. Lafferty. 2009. Visualizing Topics with Multi-Word Expressions. *Arxiv preprint `http://arxiv.org/abs/0907.1013`*.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *The Journal of Machine Learning Research*, 3:993–1022.

R. F. Boisvert, R. Pozo, and K.A. Remington. 1996. The matrix market formats: Initial design. Technical report, Applied and Computational Mathematics Division, NIST.

Matthew Brand. 2006. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 415(1):20–30, May. `http://dx.doi.org/10.1016/j.laa.2005.07.021`.

Jonathan Chang, Jordan Boyd-Graber, Chong Wang, Sean Gerrish, and David M. Blei. 2009. Reading Tea Leaves: How Humans Interpret Topic Models. volume 31, Vancouver, British Columbia, CA.

Hamish Cunningham. 2002. GATE, a General Architecture for Text Engineering. *Computers and the Humanities*, 36(2):223–254. `http://gate.ac.uk/`.

S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. 1990. Indexing by Latent Semantic Analysis. *Journal of the American society for Information science*, 41(6):391–407.

J. Demšar, B. Zupan, G. Leban, and T. Curk. 2004. Orange: From experimental machine learning to interactive data mining. *White Paper, Faculty of Computer and Information Science, University of Ljubljana*.

Tamer Elsayed, Jimmy Lin, and Douglas W. Oard. 2008. Pairwise Document Similarity in Large Collections with MapReduce. In *HLT '08: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies*, pages 265–268, Morristown, NJ, USA. Association for Computational Linguistics.

E. Frank, M. A. Hall, G. Holmes, R. Kirkby, B. Pfahringer, and I. H. Witten. 2005. Weka: A machine learning workbench for data mining. *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Practitioners and Researchers*, pages 1305–1314.

G. Gorrell. 2006. Generalized Hebbian algorithm for incremental Singular Value Decomposition in Natural Language Processing. In *Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL), Trento, Italy*, pages 97–104.

T. L. Griffiths and M. Steyvers. 2004. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(Suppl 1):5228.

Thorsten Joachims. 1999. SVMLight: Support Vector Machine. *SVM-Light Support Vector Machine* `http://svmlight.joachims.org/`, *University of Dortmund*.

Brian W. Kernighan and P. J. Plauger. 1976. *Software Tools*. Addison-Wesley Professional.

Adam Kilgarriff and Gregory Grefenstette. 2003. Introduction to the Special Issue on the Web as Corpus. *Computational Linguistics*, 29(3):333–347.

Francis Maes. 2009. Nieme: Large-Scale Energy-Based Models. *The Journal of Machine Learning Research*, 10:743–746. `http://jmlr.csail.mit.edu/papers/volume10/maes09a/maes09a.pdf`.

A. K. McCallum. 2002. MALLET: A Machine Learning for Language Toolkit. `http://mallet.cs.umass.edu`.

M. D. McIlroy, E. N. Pinson, and B. A. Tague. 1978. UNIX Time-Sharing System: Forward. *The Bell System Technical Journal*, 57(6 (part 2)), July/August.

P. V. Ogren, P. G. Wetzler, and S. J. Bethard. 2008. ClearTK: A UIMA toolkit for statistical natural language processing. *Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP*, page 32.

C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. 2000. Latent semantic indexing: A probabilistic analysis. *Journal of Computer and System Sciences*, 61(2):217–235.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. 1992. *Numerical recipes in C*. Cambridge Univ. Press, Cambridge MA, USA.

Gerard Salton, A. Wong, and C. S. Yang. 1975. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):620.

Petr Sojka. 2009. An Experience with Building Digital Open Access Repository DML-CZ. In *Proceedings of CASLIN 2009, Institutional Online Repositories and Open Access, 16th International Seminar*, pages 74–78, Teplá Monastery, Czech Republic. University of West Bohemia, Pilsen, CZ.

Mark Steyvers and Tom Griffiths, 2007. *Probabilistic Topic Models*, pages 427–446. Psychology Press, February.

Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. 2006. Hierarchical Dirichlet Processes. *Journal of the American Statistical Association*, 101(476):1566–1581.

S. K. M. Wong and V. V. Raghavan. 1984. Vector space model of information retrieval: a reevaluation. In *Proceedings of the 7th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 167–185. British Computer Society, Swinton, UK.

T. Zito, N. Wilbert, L. Wiskott, and P. Berkes. 2008. Modular toolkit for Data Processing (MDP): a Python data processing framework. *Frontiers in Neuroinformatics*, 2. `http://mdp-toolkit.sourceforge.net/`.

Predicate-Argument EXtractor (PAX)

Ralf Krestel,[1] René Witte,[2] and Sabine Bergler[2]

[1]L3S Research Center
Leibniz Universität Hannover, Germany

[2]Department of Computer Science and Software Engineering
Concordia University, Montréal, Canada

Abstract

In this paper, we describe the open source GATE component PAX for extracting predicate-argument structures (PASs). PASs are used in various contexts to represent relations within a sentence structure. Different "semantic" parsers extract relational information from sentences but there exists no common format to store this information. Our predicate-argument extractor component (PAX) takes the annotations generated by selected parsers and transforms the parsers' results to predicate-argument structures represented as triples (subject-verb-object). This allows downstream components in an analysis pipeline to process PAS triples independent of the deployed parser, as well as combine the results from several parsers within a single pipeline.

1. Introduction

Recent NLP applications have increasingly tackled semantic notions, such as textual entailment determination, incremental summary generation, or event extraction in BioNLP. The basic first issue in all these tasks is the scoping of predicative constructs as it is expressed in a predicate-argument structure (PAS). PASs can be extracted from the output of parsers; in particular dependency parsers output the component semantic relations of *subject* and *object* directly and assembling PAS structure amounts mainly to combining the related semantic relations for each verb.

Great progress has been achieved over the past 10 years with the increasing availability of different systems that tackle the same, matured tasks (POS tagging, parsing, IR) and with integration platforms that facilitate mixing and matching different application modules in a pipeline of greater and greater sophistication, reducing development time and increasing reuse of tested systems. One influential integration platform is GATE (Cunningham et al., 2002), which provides components for most steps in common NLP tasks, including several parsers. To manage these parsers' output within a pipeline in need of PAS annotations, a "normalized" output format is needed.

We present here a system that has been conceived as a GATE component that extracts PA structures from the output of several different parsers. It shields downstream components from the different output formats of the different parsers by providing a common result structure, thereby facilitating experiments that mix and match different parsers in an analysis pipeline. Our PAX component is available under an open source license.[1]

2. Predicate-Argument Structures (PASs)

Most verbs in English require a subject and an object to be specified in a grammatical sentence. For simple sentences, this subject-verb-object structure constitutes a complete analysis; for more complex sentences the task is to identify the PA structure, to assign the correct arguments to all verbs, and to identify adjuncts, i.e., PPs or NPs that are not in argument position (Merlo and Ferrer, 2006).

Dependency parsers have been addressing this as the major issue for some time and some prioritize correct dependencies over achieving a complete parse for a sentence. Even full-fledged constituent parsers have lately offered a conversion module that transforms a parse tree into dependency notation, because these notations have been most useful for different applications. Dependency relations are like severed components of predicate-argument structure or adjunct specifications, but they do not make the complete event structure explicit and it is surprisingly complex to extract the underlying PAS from dependency parser output.

As an example, consider the sentence:

> President Barack Obama will not meet the Dalai Lama during his five-day trip to the U.S. capital.

The outputs of SUPPLE, MiniPar, RASP, Stanford Parser, and the MuNPEx noun phrase chunker can be seen in Table 1.

SUPPLE
qlf=[meet(e26), adv(e26, not), time(e26, present), aspect(e26, simple), voice(e26, active), lobj(e26, e27), ne_tag(e27, offsets(165, 169)), name(e27, 'Lama'), ne_tag(e28, offsets(159, 164)), name(e28, 'Dalai'), realisation(e28, offsets(159, 164)), qual(e27, e28), det(e27, the), realisation(e27, offsets(155, 169)), realisation(e26, offsets(146, 169)), realisation(e26, offsets(146, 169))]
MiniPar
nn c_id=43, c_word=U.S., h_id=44, h_word=capital obj c_id=34, c_word=Lama, h_id=31, h_word=meet s c_id=1026, c_word=President, h_id=1031, h_word=meet
RASP
ncsubj meet:6 VV0 Obama:3 NP1 iobj meet:6 VV0 during:10 II dobj meet:6 VV0 Lama:9 NP1
Stanford Parser
args=[57, 63], kind=dobj, args=[57, 73], kind=prep args=[63, 59], kind=det, args=[63, 61], kind=nn
MuNPEx Noun Phrase Chunker
DET=his, HEAD=trip, HEAD_END=194, HEAD_START=190, MOD=five-day

Table 1: Excerpts from the output of the different parsers for the example sentence

Our PAX component normalises the different outputs into

[1]PAX, see http://www.semanticsoftware.info/pax

PAS as shown in Table 2.

RASP PAS	Obama – meet – trip
SUPPLE PAS	– meet – Lama
MiniPar PAS	Obama – meet – Lama
Stanford PAS	Obama – meet – Lama
Noun Phrase PAS	trip – be – five-day Dalai – be – Lama

Table 2: Output of three different parsers with extracted predicated-argument structures for the example sentence

As can be seen, the parsers have quite different opinions about the input sentence. This is not an exceptional, or special case, but typical for this task. Notice that we chose a rather simple sentence to demonstrate the different outputs. For more complex sentence structures, the difference in output is even greater and the extracted predicate-argument structures look quite different. Figure 1 gives an impression of the output of the MiniPar parser for a complete newspaper article.

3. Resource Description

Our PAX component is intended to be used as part of a larger processing pipeline, running after the individual parsers but before higher-level components that make use of PASs. It first collects the output of various parsers from the annotations added by them to a document. It then computes predicate-argument structures for each sentence as explained in Section 4. The predicate-argument structures for each sentence as extracted by PAX are then added as new annotations for this sentence and can be processed by other components in subsequent steps.

Supported Parsers. A variety of different parsers offer support for syntactic analysis of sentences. With this resource we try to extract predicate-argument structures using the output of different such parsers. Currently, we support MiniPar (Lin, 1998), RASP (Briscoe et al., 2006), SUPPLE (Gaizauskas et al., 2005), and the Stanford Parser (Klein and Manning, 2003a). In addition, we can extract PASs out of noun phrases, by making use of the output of a noun phrase chunker like MuNPEx.[2]

4. Design

We now describe in more detail how to extract predicate-argument structures from the output of different parsers as shown in Table 1.

Our PAS extractor is based on a set of rules for each of the three parsers. These rules determine which part of the parser output is considered the subject, verb, and object. Because of the different nomenclature and relations scheme of the parsers, this has to be done individually for each parser.

4.1. SUPPLE

For SUPPLE (Gaizauskas et al., 2005), the extraction process is quite straightforward. The parser outputs *semantic* relations, which comprise a logical subject and verb, and sometimes also a logical object. The PAS extractor therefore only has to filter out these elements from the output of SUPPLE. The coverage of SUPPLE is lower in comparison with other parsers. This is due to the philosophy of the parser (Gaizauskas et al., 2005): *"Rather than producing all possible analyses or using probabilities to generate the most likely analysis, the preference is not to offer a single analysis that spans the input sentence unless it can be relied on to be correct. This means that in many cases only partial analyses are produced, but the philosophy is that it is more useful to produce partial analyses that are correct than full analyses which may well be wrong or highly disjunctive."*

4.2. MiniPar

To obtain predicate-argument structures that represent the underlying sentence as closely as possible, we often have to choose between multiple candidates for the *object*. We employ a decision tree to select the grammatical structure to fill the *object* slot from the parser's output. If it exists and relates to the subject-verb pair we choose in this order: "obj," "obj1," "pred," and "pcomp-n."

Sometimes the *object* does not have a direct relation to the *verb* but an indirect link through another element in common like a "mod" construct. In this case we have to track down and identify this relation to find a representative object. A complex sentence can contain more than one subject and our extractor has to be able to handle them reasonably. Besides dealing with more than one "s" (subject) in one sentence, it can also handle conjunctions. Table 3 gives a general overview of the different conjunction types and how the extractor deals with them.

Sentence	**X and Y** buy a car.
Target PAS	X – buy – car Y – buy – car
Sentence	X **buys a car and sells his house**.
Target PAS	X – buy – car X – sell – house
Sentence	X buys **a car and a house**.
Target PAS	X – buy – car X – buy – house

Table 3: PAS extractor strategy for conjunctions

4.3. RASP

For RASP's version 3 (Briscoe et al., 2006) we developed a wrapper to be able to use it from within GATE. It calls the appropriate script and delivers the parser's output for further processing.

The strategy to find subject, verb, and object relations is to look for "ncsubj" occurrences in the parser output. They describe a subject together with the corresponding verb. To find a suitable object we often have to choose between different elements like "dobj," "iobj," "obj," or "xcomp." To obtain predicate-argument structures that accurately represent the underlying sentence, we use the following decision tree on what grammatical structure to use as *object*. If it exists and is related to the verb of the subject, we choose in this order: "obj," "dobj" if dependent of an "iobj," which itself relates to the relevant *verb*, "iobj," "dobj," and last "xcomp."

Besides dealing with more than one "ncsubj" in one sentence, we can also handle conjunctions. This has already

[2] Multi-lingual Noun Phrase Extractor (MuNPEx), http://www.semanticsoftware.info/munpex

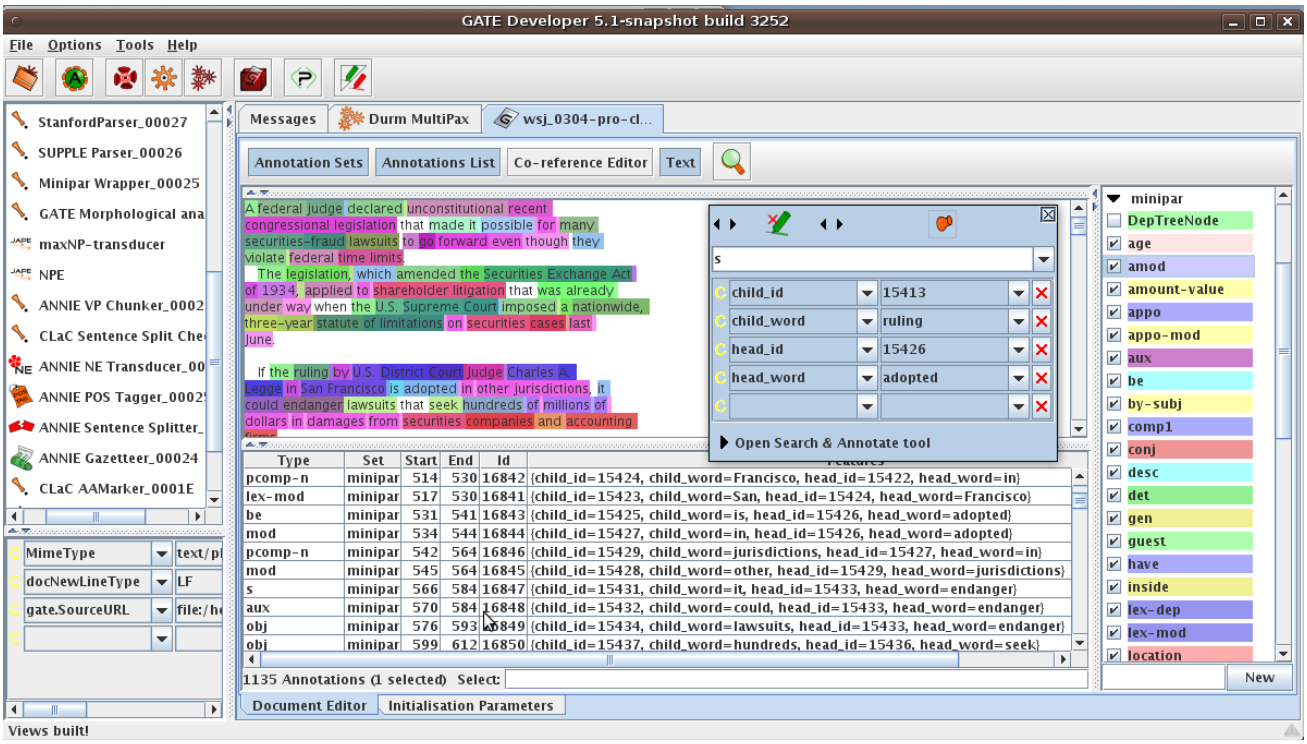


Figure 1: GATE screenshot of MiniPar results

been demonstrated for MiniPar in Table 3 and applies to RASP as well.

4.4. Stanford Parser

The Stanford Parser (Klein and Manning, 2003a), (Klein and Manning, 2003b) extracts dependency relations. We take all "nsubj" and "nsubjpass" elements for subjects and the associated predicates as verbs. For the object we take in this order: "dobj," "prepPobj," and "dep."
Conjunctions are already considered by the parser and there is no further processing needed from our side.

4.5. MuNPEx Noun Phrases

Each noun phrase that contains a modifier is a candidate for a predicate-argument structure. For example, the noun phrase "the rich king" contains the same information as the PAS "king – be – rich". Adding the noun phrase predications generates additional PASs that can be especially useful for certain task like comparing documents' content based on predicate-argument structures or for recognizing textual entailment (e.g., the RTE[3] tasks) between statements.

5. Implementation

Our resource is implemented as a component for the *General Architecture for Text Engineering* (GATE) (Cunningham et al., 2002). Figure 2 shows example output of the PAX component with the detected PASs for all supported parsers. For each parser x, a new annotation set of type xParserPaX is added to the document. If the component detects a predicate-argument structure in the output of the selected parser, the sentence containing the PAS is annotated and "sub", "obj", and "verb" properties are added to the annotation. In addition, we try to detect simple negations within the sentences like "not" or "never" (Figure 2).

[3]RTE, see http://www.nist.gov/tac/2009/RTE/

6. Evaluation

To evaluate our PAX component, we selected an article from the *Wall Street Journal* and annotated it manually with predicate-argument structures. The structure of the sentences was particularly complex with most of the time three or more PASs per sentence. For simple sentences of the shape "subject, verb, object" all parsers perform well and we can extract the predicate-argument structures reliably from the parsers' output. Therefore we are interested in the most difficult cases only. We excluded the noun phrase PAS extraction from this evaluation since it is a special case also yielding different types of errors. Table 4 gives an overview of the performance of the different parsers with correctly extracted PAS, wrong PAS, and partially correct PAS, where partially means for example that the object was not found or an indirect object instead of a direct one was found.
Some errors like unresolved pronouns, e.g. "that," "he," "who" or "myself" were not considered errors of the PAS extraction but need to be dealt with in the future, although for some parsers we are already able to resolve these constructs. Another possible source of errors are verb phrases like "declare unconstitutional," or "prevent s.o. from doing s.th." If we insist on having only one term as a predicate we need to decide which verb reflects the intended meaning of the PAS best.
Noun Phrases with modifiers can not always be converted to predicate-argument structures. For example, it works fine with "the elected President" $\rightarrow$ "President – be – elected"; but not for "last year's President" $\not\rightarrow$ "President – be – year".

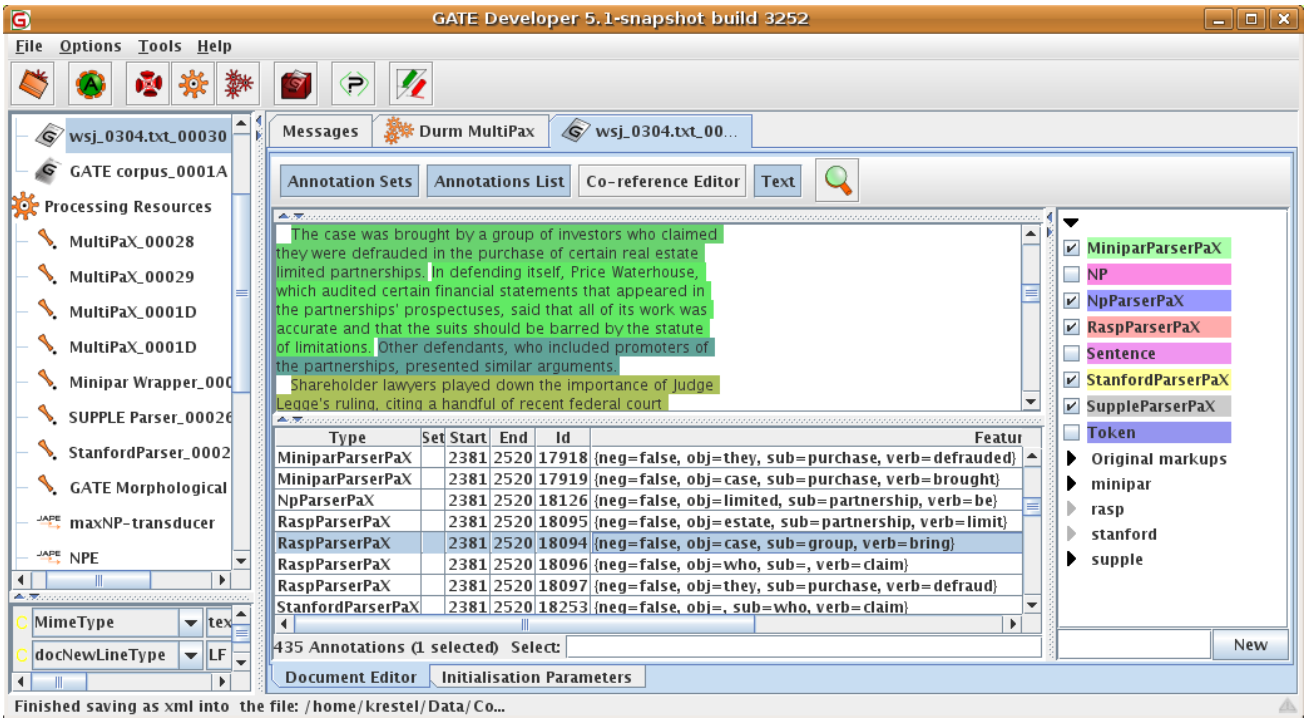


Figure 2: PAX results in GATE showing subject-verb-object triples extracted from MiniPar, RASP, Stanford, and MuNPEx

Sent	No of PAS	SUPPLE			MiniPar			RASP			Stanford		
		C	P	F	C	P	F	C	P	F	C	P	F
1	4	-	2	-	-	2	-	1	2	1	1	2	1
2	4	1	-	-	-	2	-	2	2	-	2	2	-
3	3	-	-	-	-	2	-	2	1	-	1	1	-
4	3	-	-	-	-	-	1	2	-	-	-	-	-
5	4	-	1	-	-	2	-	-	2	1	-	2	-
6	1	-	1	-	-	1	-	-	1	1	-	1	-
7	4	1	-	-	-	2	-	-	3	-	-	4	-
8	5	2	1	-	3	1	-	-	4	1	3	1	-
9	3	-	-	-	-	1	2	1	2	1	1	1	-
10	3	-	1	-	-	1	-	-	-	-	-	1	1
11	6	1	2	-	3	1	-	2	4	-	1	1	2
12	3	1	-	-	-	2	-	-	3	1	-	-	-
13	5	1	1	-	2	1	-	1	2	-	1	1	-
14	2	-	-	-	-	1	-	-	2	-	-	1	-
15	3	-	1	-	-	1	-	-	2	-	-	2	-
16	2	-	1	-	-	-	-	-	1	2	-	1	1
17	4	1	1	-	-	3	-	-	4	-	1	1	1
18	3	-	-	-	-	2	-	-	3	-	-	3	-
19	3	1	-	-	-	1	-	2	-	-	1	1	-
20	3	-	1	-	-	2	-	1	-	2	1	2	-
21	2	-	1	-	-	2	-	-	2	-	-	2	-
22	1	-	1	-	-	1	-	-	1	-	-	1	-
23	4	-	-	-	2	-	1	2	1	1	-	1	1
24	0	-	-	-	-	-	-	-	-	-	-	-	1
∑	75	9	15	-	8	31	3	16	42	11	13	32	8
Recall		0.32			0.52			0.77			0.60		
Precision		1.0			0.93			0.84			0.85		

Table 4: Results for the four parsers: C=correct, P=partially correct, F=false

7. Conclusion & Future Work

We described a strategy to extract predicate-argument structures from the output of different parsers and its implementation in the PAX component for GATE. PASs can be used to represent content and make it comparable. Finding similar content (e.g., for text summarization, paraphrase detection) or entailment (RTE, inferences) are some of the application areas of predicate-argument structures. Our component creates a common result structure for the different parsers and thereby allows downstream analysis components to work independently of a concrete parser's output. This simplifies the setup significantly of experiments where the impact of different parsers on the overall application performance needs to be measured.

In the future we want to introduce a voting algorithm to identify the best predicate-argument structures for each sentence based on the output of multiple parsers.

8. References

E. Briscoe, J. Carroll, and R. Watson. 2006. The Second Release of the RASP System. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proc. of the 40th Anniversary Meeting of the ACL*.

R. Gaizauskas, M. Hepple, H. Saggion, M. A. Greenwood, and K. Humphreys. 2005. SUPPLE: A practical parser for natural language engineering applications. In *Proc. of the 9th Intl. Workshop on Parsing Technologies (IWPT2005)*, Vancouver.

Dan Klein and Christopher D. Manning. 2003a. Accurate unlexicalized parsing. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 423–430, Morristown, NJ, USA. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2003b. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press.

Dekang Lin. 1998. Dependency Based Evaluation of MINIPAR. In *Proceedings of the Workshop on the Evaluation of Parsing Systems, First International Conference on Language Resources and Evaluation*.

Paola Merlo and Eva Esteve Ferrer. 2006. The notion of argument in prepositional phrase attachment. *Computational Linguistics*, 32(3):341–378.