Effective Development with GATE

and Reusable Code for Semantically Analysing Heterogeneous Documents

Adam Funk, Kalina Bontcheva

University of Sheffield

22 May 2010

Outline

1 Introduction

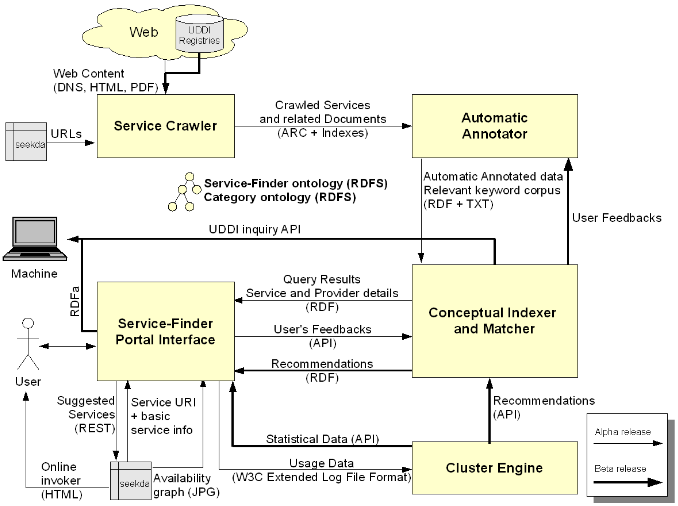
2 Annotation tasks

3 Implementation

4 Ontology population

5 Conclusions

The task: input



The task: input

- periodically (ideally monthly) analyse about 250 000 crawled documents (WSDL, HTML, PDF) relating to 25 000 web services from 8700 providers
- fortunately the services and providers are already instantiated and linked with each other and the relevant documents
- many duplicate documents with different URLs

The task: output

- carry out information extraction
- classify documents and services
- categorize services according to the ontology (59 subclasses of *Category* with multiple inheritance)
- express output as RDF according to the project's ontology

Typical batch

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

Annotation tasks

- analyse WSDLs to instantiate *Endpoint*, *Interface*, and *Operation* and properties associating them with each other and with *Service* instances—**not** an NLP task, but with borrowed code integrated
- classify documents by type (e.g., documentation, pricing, contact details) and rate them as low-, medium-, or high-interest
- carry out IE to identify providers' addresses, phone numbers, e-mail addresses, etc.
- carry out IE/classification over services to identify service level agreements, free trials, etc.
- categorize each service in one or more of the 59 subclasses of *ServiceCategory*

Basic principles

- *GATE Developer* for rapid development of the IE components and testing of most components; saving pipelines as files to be reloaded in the batch system
- GATE serial datastores for persistence; datastores and corpora for breaking the 250 000 documents into independently manageable chunks
- ANNIE as a starting point for IE
- JAPE for rapid development of PRs where appropriate
- *GATE Embedded* framework and libraries for custom PRs; persistence (serial datastores and pipelines); document/service/provider management (by corpora); headless batch jobs

Implementation: preprocessing

- Use serial datastores for persistence and manageable chunks.
- Use MD5 to merge duplicate documents; suppress HTTP error messages and empty documents: 31% reduction.
- Put each provider in its own corpus.
- 30 datastores
- 8700 corpora (about 290 per datastore)
- 173 000 documents (average 20 per corpus, but quite variable; 5770 per datastore)
- Carry out special WSDL analysis using seekda's code and store the RDF-XML (generated from templates)as a document feature on each WSDL.

Analysis of a datastore

Pipeline series to run over each corpus:

1. standard NLP components
2. ANNIE gazetteers and NER transducers
3. custom gazetteers
4. custom JAPE transducers for weighting keywords, marking documents as more or less interesting, etc.
5. GATE Batch Learning PR for service categorization
6. custom “voting” PRs for documents (types), services (category, free trials), and providers (e-mail and postal addresses); output to RDF-XML snippets (generated from templates) on the corpora and documents

Analyse all corpora then consolidate all the RDF-XML snippets into one output file.

Service categorization

- The final version of service categorization used machine learning (SVM document classification) as well as weighted keywords, run through a voting system for each service. ML categories were weighted to outvote keywords.
- See our “Ontology-Based Categorization of Web Services with Machine Learning” (also at LREC 2010) for full details and evaluation.

Approaches to ontology population

Ontology manipulation

- Load the ontology in memory and manipulate it with the GATE Ontology API
- Allows the program to query the ontology (and validate the data) and modify the class and property hierarchy
- Output can be saved as RDF-XML, N-Triples, N3 or Turtle
- Everything is in memory at the same time
- Used in CLOnE (SEKT), RoundTrip Ontology Authoring (NEPOMUK), SPRAT and SARDINE (NeOn) software

Approaches to ontology population

XML generation

- Write XML templates and fill them in using the values of specified annotation features (when the annotations match a template's requirements) or a `Map<String,String>`
- Suitable for generating instances and property values for a fixed class and property hierarchy
- No "live" data validation, but a good set of templates guarantees consistent output
- Generates XML snippets which can be saved in datastores as document and corpus features—very little in memory
- Used in semantic document annotation web services (MUSING and CLARIN) and in Service-Finder

Code re-use

from GATE	GUI, libraries, ANNIE, ML, the JAPE system
from TAO	source-code tokenizer
from MUSING	XML generator
to NeOn	headless batch control tools
to CLARIN	improved XML generator