Web Services and Processing Pipelines in HLT: Tool Evaluation, LR Production and Validation

WORKSHOP PROGRAMME

Monday May 17, 2010

14:30 – 14:45	Introduction
14:45 – 15:30	Invited talk: Graham Wilcock *Linguistic Processing Pipelines: Problems and Solutions*
15:30 – 16:00	Linguistic Processing Chains as Web Services: Initial Linguistic Considerations *Maciej Ogrodniczuk and Adam Przepiórkowski*
16:00 – 16:30	Coffee break
16:30 – 17:00	An Infrastructure for More Reliable Corpus Analysis *Kerstin Eckart, Kurt Eberle and Ulrich Heid*
17:00 – 17:30	The TANL Pipeline *Giuseppe Attardi, Stefano Dei Rossi and Maria Simi*

Tuesday May 18, 2010

09:30 – 10:00	Text Handling as a Web Service for the IULA Processing Pipeline *Hector Martinez, Jorge Vivaldi and Marta Villegas*
10:00 – 10:30	A Generic Chaining Algorithm for NLP Webservices *Volker Boehlke*
10:30 – 11:00	Coffee break
11:00 – 11:30	A Graphical Interface for Computing and Distributing NLP Flows *Ionut Cristian Pistol, Andrei Arusoaie, Andrei Vasiliu and Adrian Iftene*
11:30 – 12:00	Corpora by Web Services *Adam Kilgarriff*
12:00 – 12:30	An Open Service Framework for Next Generation Localisation *David Lewis, Stephen Curran, Dominic Jones, John Moran, Kevin Feeney*
12:30 – 13:00	Discussion
13:00 – 14:30	Lunch break
14:30 – 15:00	Invited talk: Nancy Ide *A Web Service for Customized Corpus Delivery*
15:00 – 15:30	Web Communication Protocols for Coordinating the Modules of AnHitz, a Basque-Speaking Virtual 3D Expert on Science and Technology *Igor Leturia, Arantza del Pozo, David Oyarzun, Urtza Iturraspe, Xabier Arregi, Kepa Sarasola, Arantza Diaz de Ilarraza, Eva Navas, Igor Odriozola and Iñaki Sainz*
15:30 – 16:00	Utilizing Web Service Technology to Create Danish Arabic Language Resources *Mossab Al-Hunaity*
16:00 – 16:30	Coffee break
16:30 – 17:00	Technology-Neutral Machine Translation with an Abstracted Technology Stack *Joachim Van den Bogaert*
17:00 – 17:30	Wrap-up & discussion

Table of Contents

Author Index

PREFACE

With the emergence of large e-infrastructures and the widespread adoption of the Service Oriented Architecture (SOA) paradigm, more and more language technology is being made available through web services. Extending such services to linguistic processing pipelines, tool evaluation or LR production and validation involves considering both the methodologies and technical aspects specific to the application domains.

Distributed architectures such as web services allow communication and data exchange between applications. They are a suitable instrument for automatic, less often semi-automatic, tool evaluation as well as resource production processes both for practical and conceptual reasons. At a practical level, web services support quick results, centralised data storage, remote access etc.; at a conceptual level, they allow for the combination of more than one processing components that may be located on different sites. Such processing pipelines are set up to tackle a particular analysis task. To support these, new techniques have to be developed that organise well-established practices into workflows and support the exchange of data by standards and open tool architectures.

The workshop focuses on current uses and best practices for the deployment of web services and web interfaces in the HLT domain, including processing pipelines, LR production and validation, and evaluation of tools. It highlights relevant aspects for the integration of linguistic or evaluation web services within infrastructures (e.g. authorisation and authentication, service registries) and infrastructural requirements (e.g. interface harmonisation, metadata generation). The workshop also aims at demonstrating different approaches on how to combine linguistic web services into a composite web service.

The expected outcome of the workshop is a comparison of the practices in architectures and processing pipelines that people build and discussion of the issues involved. Topics of interest include, but are not limited to:

– *Technical aspects*: approaches, protocols, management of huge amounts of data, data structures and formats, performance, manual components (e.g. annotation or evaluation), composition and configuration, interoperability, security, monitoring and recovery strategies, standardisation of APIs, tools and frameworks supporting HLT services deployment, architectures.

– *Scientific aspects*: influence of web services on evaluation or resource production, meta-evaluation / validation of architectures, annotation agreements, needs for tools evaluation and resource production, status of the data produced.

– *Commercial aspects*: licensing, privacy, advertising, brokering, business possibilities, challenges, exploitation of the resulting data.

The papers presented at the workshop range from the basic principles of linguistic processing pipelines to implementations of web services for building such pipelines as well as language resources.

Linguistic Processing Chains as Web Services: Initial Linguistic Considerations

Maciej Ogrodniczuk, Adam Przepiórkowski

Institute of Computer Science
Polish Academy of Sciences
ul. Ordona 21, Warsaw, Poland
maciej.ogrodniczuk@ipipan.waw.pl, adamp@ipipan.waw.pl

Abstract

At the end of 2009 the review of a number of available Web services implementing linguistic processing chains (CLARIN deliverable D5R-3a, 2009) was prepared as part of Common Language Resources and Technology Infrastructure (CLARIN) Working Group 5.6 (LRT integration) activities. Basing on the showcases contributed by WG members, the summary of features of both chained and individual Web Services was compiled, preparing the ground for comparisons between selected linguistic properties of registered frameworks. The article aims at presenting preliminary generalizations regarding functionalities, communication standards and representation of linguistic resources being adopted as web services, which were initially put forward in the CLARIN paper. The major features of the tools are summarized to provide starting point for discussion over interchange formats and tagsets, standards of encoding of linguistic resources and linguistic data categories. Apart from concentrating on representation of linguistic annotation, very preliminary conclusions concern technical, formal and semantic interoperability of language resources.

1. Introduction

Working Group 5.6 fulfils CLARIN mission of *creating, coordinating and making language resources and technology available and readily useable for scholars in the humanities and social sciences*[1] by concentrating on interoperability issues, mainly at the linguistic level (e.g., the problem of mapping between tagsets).

Within the Work Package 5 (Language Resources and Technologies Exploration) the group intended to provide the consortium with a broad overview of the LRTs available as web service chains and get an understanding of their status. This has been achieved by studying examples of the LRTs obtained as showcases from contributing partners (CLARIN consortium members) and compiled into initial summary of their status, properties, adopted standards and individual qualities.

2. Web service showcases

The call for contribution resulted in gathering descriptions of 8 frameworks, summarized according to the template delivered in the beginning of the process. On account of potential grave differences among submissions, the questions asked allowed some latitude in providing the general information on described solutions while remaining strict about their linguistic properties (languages covered, implemented NLP services, web service protocols, language resource standards and linguistic data encoding). The obtained materials were characterized by good quality and all partners showed advanced responsiveness while presenting and clarifying their solutions.

The next subsections attempt to summarize the showcases in a concise form, providing brief information on linguistic properties, performed functions, available web services (in form of WSDL[2] references, wherever available) and organizations involved in their preparation.

2.1. WebLicht

WebLicht (Web Based Linguistic Chaining Tool) is a SOA[3] framework of 25 web services performing specialized NLP[4] tasks for German, English, Italian, French and Finnish, such as sentence border detection, tokenization, POS[5] tagging, named entity recognition, lemmatization, constituent parsing, co-ocurrence annotation and semantic annotation. The open architecture allows for stacking existing services into processing chains as well as incorporating external tools and web services into existing solution.

The common representation of texts and annotations within the WebLicht processing chain is TCF (Text Corpus Format), an XML-based format supporting stand-off annotation and compatible with ISO LAF[6]. Converters for Negra[7], Paula (Dipper, 2005), MAF[8] and TüBa-D/Z[9] are available; the constituent parser output is TIGER-XML[10] (Mengel and Lezius, 2000), also TCF-encoded. Linguistic data is represented by means of language-dependent tagsets

[1]See the CLARIN Web page, `http://www.clarin.eu/`.

[2]Web Service Definition Language

[3]Service-Oriented Architecture

[4]Natural Language Processing

[5]Part-of-Speech

[6]Linguistic Annotation Framework, ISO/DIS 24612, see `http://www.tc37sc4.org/`.

[7]See `http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/negra-corpus.html`.

[8]Morpho-Syntactic Annotation Framework

[9]Tübinger Baumbank des Deutschen / Zeitungskorpus (Tübingen Treebank of Written German), see `http://www.sfs.uni-tuebingen.de/tuebadz.shtml`.

[10]See `http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERCorpus/`.

such as STTS[11] for German or the Penn Treebank tagset (UPenn)[12] for English.

WebLicht results from cooperation of linguistic departments of major German research institutions (Berlin Brandenburgische Akademie der Wissenschaften, University of Leipzig, University of Stuttgart and University of Tübingen).

2.2. GATE Web Services

GATE (General Architecture for Text Engineering) is open source software offering a wide range of language processing functionalities to be organized in maintainable workflows. Initially offered as plugins for the downloadable architecture, GATE subsystems are being gradually transformed into web services with information extraction (tokenizer, sentence splitter, POS tagger, named entity recogniser and classifier), phrase chunking, lemmatization and POS tagging tools leading the way.

Input data for the services may be encoded in a variety of text formats (plain text, HTML, SGML/XML, RTF/MS Word, PDF). The output is SynAF[13] (for noun/verb phrase chunker) and MAF-compliant XML (for lemmatizer and English/Bulgarian/Dutch POS taggers). Linguistic data are categorized by means of Penn Treebank tags.

GATE Web Services have been developed by the GATE group[14] at the University of Sheffield, UK.

2.3. IULA Web Services

The IULA Web Services family (Vivaldi Palatresi, 2009; Bel et al., 2006; Atserias et al., 2006; Villegas et al., 2009) allows for uploading and indexing text corpora to perform statistical queries (such as calculation of several lexicometric measures, word co-occurrences, relevance, distribution, extract and group concordances etc.) and various NLP tasks (e.g., tokenization, sentence splitting, morphological analysis, named entity detection and classification, POS tagging, chart-based shallow parsing, rule-based dependency parsing, nominal correference resolution or WordNet-based sense annotation and disambiguation), also in a chained manner. All services are available for English and Spanish, some of them (Freeling[16]) also for Catalan, Galician, Italian, Welsh, Portuguese and Asturian.

Input format for statistical processing is plain text while corpus analysis of annotated text requires EAGLES[17]/PAROLE[18] compliance. AAILE web service (Automatic Acquisition of Lexical Information by extracting syntactic patterns and contexts of concordances in a corpus) employs IULA tagsets for Spanish[19] and English[20].

The Web Services are maintained by Institut Universitari de Lingüística Aplicada at University Pompeu Fabra (IULA-UPF) in Barcelona, Spain.

2.4. ILSP Text Processing Chain

The main tools integrated by ILSP TPC are tokenizer and sentence splitter, POS tagger, lemmatizer, chunker and dependency parser.

All processing tools from the chain generate annotations compatible with UIMA annotation type system, an extension of JULIE Lab annotation scheme[21]. The services can also export results to other structured formats, e.g., GATE XML or XCES[22] (Ide et al., 2000). POS information is represented using PAROLE-compatible tagset, while dependency relations are described using Prague Dependency Treebank syntax.

The tools are provided by Institute for Language and Speech Processing (ILSP) from Athens, Greece. For more information see (Papageorgiou et al., 2002; Prokopidis and Georgantopoulos, 2010).

2.5. RACAI Services

The RACAI framework offers multiple linguistic tools for language identification (all EU languages), tokenization, tagging and lemmatization (TTL service, also containing remote procedures for sentence splitting and chunking), dependency parsing or wordnet browsing (remaining tools for Romanian and English).

Along with several proprietary formats, the tools encode results in XCES format. Lexical tagsets used is MULTEXT-EAST[23]-compliant (Erjavec, 2004; Tufiş, 2000).

The services are maintained by Research Institute for Artificial Intelligence, Romanian Academy of Sciences (RACAI), Bucharest, Romania.

2.6. WS-LexicalPlatform

The platform provides web service interface to the Italian SIMPLE lexicon, assisting in retrieving information concerning phonology, morphology, syntax and semantics.

The interchange data format is LMF[24] with ISO DCR[25]-mappable data categories basing on EAGLES-ISLE[26] (to be promoted to the future ISO standardization of data categories and, therefore, ISOCat).

[11]Stuttgart-Tübingen Tagset, see http://www.ims.uni-stuttgart.de/projekte/corplex/TagSets/stts-table.html.

[12]See http://www.cis.upenn.edu/~treebank/.

[13]Syntactic Annotation Framework, see http://www.tc37sc4.org/new_doc/ISO_TC37_4_N244_SynAF_WD_draft.pdf.

[14]See http://www.gate.ac.uk/.

[16]See http://www.lsi.upc.edu/~nlp/freeling/.

[17]Expert Advisory Group on Language Engineering Standards, see http://www.ilc.cnr.it/EAGLES96/home.html.

[18]See http://www.elda.org/catalogue/en/text/doc/parole.html.

[19]See http://www.iula.upf.edu/corpus/etqfrmes.htm.

[20]See http://www.iula.upf.edu/corpus/etquk.htm.

[21]See http://www.julielab.de/JULIE_Lab.html.

[22]XML Corpus Encoding Standard

[23]Multilingual Text Tools and Corpora for Central and Eastern European Languages, see http://nl.ijs.si/ME/.

[24]Lexical Markup Framework

[25]ISO 12620, Data Category Registry, see http://www.isocat.org/.

[26]International Standard for Language Engineering, see http://www.mpi.nl/ISLE/.

	Language identification	Sentence border detection	Tokenization	POS tagging / MSD[15]	Named Entity recognition	Lemmatization	Parsing	TreeBank browsing	Co-ocurrence annotation	Collocation extraction	Frequency analysis	Association measures	Semantic annotation	WordNet –related functionality	Thesaurus-related functionality	Lexicon access	Machine translation
WebLicht		●	●	●	●	●	●	●	●	●	●	●	●	●	●		
GATE		●	●	●	●	●							●				
IULA		●	●	●	●	●	●		●	●	●	●	●	●			
ILSP		●	●	●		●	●										
RACAI	●	●	●	●		●	●							●			●
WS-LexPl																●	
LXService		●	●	●													
WROCUT/ICS PAS		●	●	●		●	●		●	●	●	●		●			

Table 1: LRT functionality available in reviewed frameworks

The services are provided by Consiglio Nazionale delle Ricerche, Istituto di Linguistica Computazionale (CNR-ILC), Pisa, Italy.

2.7. LXService

The web service offers chunking, tokenization (Branco and Silva, 2003) and tagging (Branco and Silva, 2004; Silva, 2007) functionality for Portuguese. More tools, such as morphological analyser (Branco and Silva, 2006; Nunes, 2007; Martins, 2008) or parser (Silva et al., 2010) are being currently integrated. Proprietary formats are used both for encoding resources and linguistic data categories.

The body responsible for the services is University of Lisbon, Department of Informatics, Natural Language and Speech Group (NLX), Lisbon, Portugal. For more information see (Branco et al., 2008).

2.8. WROCUT/ICS PAS services

The tool set (language independent, although currently used with a grammar and tagset for Polish) comprise a tagger (Piasecki and Godlewski, 2006), a lemmatizer, tokenizer and morphologic analyser (Woliński, 2006), a shallow parser and disambiguation tool (Buczyński and Przepiórkowski, 2009), as well as an automatic harvester of lexical semantic relations from corpora for Polish and English (Broda and Piasecki, 2008; Piasecki et al., 2009).

Resources are represented is XCES and Wordnet-LMF (Aliprandi et al., 2009), while linguistic data is encoded using proprietary (currently de facto standard for Polish) ICS PAS tagset (Przepiórkowski and Woliński, 2003)[27] and CLAWS5 (British National Corpus tagset).

The services are the result of co-operation between Institute of Informatics, Wrocław University of Technology (WROCUT) and Institute of Computer Science, Polish Academy of Sciences (ICS PAS), Warsaw, Poland.

3. Summary of linguistic properties

3.1. NLP-specific functions

Table 1 presents the scope of LRT functionalities offered by the reviewed frameworks. The most complex web service-enabled processing chains seem to provide the widest linguistic coverage which obviously results from their background — due to increasing popularity of the remote service approach, existing tools are often being converted into web services. This tendency should be considered a good sign for small-size providers of linguistic material and services since their individual tools may effectively compete in the global network with their large-scale equivalents.

3.2. Encoding of linguistic resources

Table 2 presents the encoding formats of reviewed services. The first observation is that no common input/output format can be distinguished, neither any format is clearly standing out. The lowest common denominator for all reviewed formats seems to be XML — even the tools using text proprietary formats are, to some extent, XML-compatible or use XML as a variant representation (e.g., RACAI Services use internal Tab-separated SGML format along with XCES-encoded output).

Another dimension while evaluating formats is „standard or proprietary”, with similar findings: proprietary formats tend to exist along with established standards or even gradually become standards, on local or multinational level.

[27]A slightly modified version of the tagset (Przepiórkowski, 2009) is used in the National Corpus of Polish (`http://nkjp.pl/`) and defined in ISOcat as a public data category set "NKJP" (cf. `http://www.isocat.org/interface/`).

	Acknowledged standards						Proprietary formats		
	XML-based formats								
	LMF-XML	LMF-WordNet	MAF	SynAF	TIGER-XML	XCES	XCES proprietary extension	XML proprietary format	Plain text proprietary format
WebLicht			●		●			●	
GATE			●	●				●	
IULA						●			
ILSP						●		●	
RACAI						●	●	●	●
WS-LexPl	●								
LXService								●	●
WROCUT/ICS PAS		●					●		

Table 2: Output formats of reviewed services

	Standard tagsets					Proprietary tagsets				
	CLAWS5	EAGLES/ PAROLE	MULTEXT-EAST	Prague Dependency Treebank	UPenn	ICS PAS (PL)	LX tagset (PT)	RACAI tagset (EN, RO)	SIMPLE-based tagset (IT)	STTS (DE)
WebLicht					●					●
GATE					●					
IULA		●								
ILSP		●		●						
RACAI			●							
WS-LexPl									●	
LXService							●			
WROCUT/ICS PAS	●					●				

Table 3: Tagsets used to encode linguistic annotation

WebLicht TCF is a good example here: being proprietary, it retains compatibility with ISO LAF/LMF/MAF standards.

In many cases proprietary extensions of recognized formats can supplement them with project-specific properties which makes the border between standard and non-standard even more vague. The need for compatibility is (and should be) in such cases satisfied by providing converters between internal and widely accepted formats (such as TCF-to-PAULA and MAF formats for WebLicht).

3.3. Linguistic data categories

Table 3 presents tagsets used by reviewed services for representing linguistic data categories. Similarly to the previous section, the border between standard and proprietary seems flexible. Some tagsets (such as STTS for German or ICS PAS tagset for Polish), while being non-standard, i.e., not recognized worldwide or approved by official standards development organization, are universally used for certain languages or constitute regional norms. Regardless of the process of emerging new standards-to-be, the tendency to normalize is noticeable since most frameworks tend to adopt well-known tagsets, either exclusively or along with their private formats.

4. Preliminary findings

Before making any generalizations it is worth to point out that neither the overview of text processing chains and web services in the LRT area, nor the initial findings were planned as an exhaustive summary, rather a study of usage scenarios including chains of operation.

Firstly, the presence of such a broad spectrum of different standards, both for encoding of linguistic resources and annotation categories, shows that the unification process is still in its beginnings. The reasons behind such condition do not seem to be the underestimation of the necessity of using widely-accepted standards by NLP community, but rather high costs of conversion of proprietary formats and preparation of mapping tools or, probably, the lack of linguistically mature interchange models. The role of such projects as CLARIN and FLaReNet[28] to create and endorse standards is therefore highly significant. In the long run, the concept of data conversion to impose formats and data categories loses the contest with a vision of ensuring compliance of current representation with some, preferably ISO-related, encoding standard. This scenario is universally adopted by most reviewed environments and remains compatible with CLARIN goals.

4.1. Interoperability issues

In general, interoperability of language resources can be discussed on three major levels: technical, syntactic and semantic. Technical interoperability, regarding e.g., web service protocols, is hardly of any concern here and has been adressed in (CLARIN deliverable D2R-6b, 2009). Formal interoperability, obtained by standardizing data exchange format and common language resource data model is already attainable with XML-based interchange formats following official representation standards. Semantic interoperability issue is still open, but appears to be solvable by providing formal mapping of proprietary categories to standard classes (such as those of ISOCat).

4.2. Linguistic standards

As stated above, the use of different representation standards is not discouraged and therefore the adoption of general metamodels seems the most appropriate solution for accommodating many encoding conventions. However, unambiguous unifying procedures (such as examples and best practices of how to convert, for instance, Penn Treebank-style representation into LAF) are necessary to ensure real interoperability between standards.

Practical assessment of methods and formats seems also necessary to strike a balance between permissiveness and constriction to enable accurate, yet flexible representation. Until then, a wider range of standards may be used to achieve better precision of linguistic description.

5. Closing notes

More and more linguistic processing chains are being available as web services and, however it will still be a long time before the new interfaces reach the quality of separate tools, the need of making their advanced functionalities available according to popular web service protocols is clearly visible and several renowned frameworks (such as the one of DFKI) are currently being amended with or ported to web service frameworks (as for DFKI, it is planned to be completed before the end of 2010).

The investigation of a growing network of linguistic tools available as services is therefore being continually underway, along with research and development in the closely related area of linguistic data interchange. As a result, the initial CLARIN document will be followed by an extended version containing final conclusions on the subject of harmonized access to resources via published interfaces to enable the interoperable domain. This deliverable will be available in the beginning of 2011.

6. References

Carlo Aliprandi, Federico Neri, Andrea Marchetti, Francesco Ronzano, Maurizio Tesconi, Claudia Soria, Monica Monachini, Piek Vossen, Wauter Bosma, Eneko Agirre, Xabier Artola, Arantza Diaz de Ilarraza, German Rigau, and Aitor Soroa. 2009. Database models and data formats. KYOTO Deliverable NR 1/WP NR 2, Version 3.1, 2009-01-31. *http://www2.let.vu.nl/twiki/pub/Kyoto/WP02:SystemDesignD2.1Database_Models_and_Data_Formats_v3.1.pdf*.

Jordi Atserias, Bernardino Casas, Elisabet Comelles, Meritxell González, Lluís Padró, and Muntsa Padró. 2006. FreeLing 1.3: Syntactic and semantic services in an open-source NLP library. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, Genoa, Italy. European Language Resources Association (ELRA).

Núria Bel, Sergio Espeja, and Montserrat Marimon. 2006. New tools for the encoding of lexical data extracted from corpus. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, pages 1362–1367, Genoa, Italy. European Language Resources Association (ELRA).

António Branco and Joao Silva. 2003. Contractions: breaking the tokenization-tagging circularity. *Lecture Notes in Artificial Intelligence 2721*, pages 167–170.

[28] *Fostering Language Resources Network*; see `http://www.flarenet.eu/`.

António Branco and Joao Silva. 2004. Evaluating Solutions for the Rapid Development of State-of-the-Art POS Taggers for Portuguese. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal. European Language Resources Association (ELRA).

António Branco and Joao Silva. 2006. Dedicated Nominal Featurization of Portuguese. *Lecture Notes in Artificial Intelligence 3960.*

António Branco, Francisco Costa, Pedro Martins, Filipe Nunes, Joao Silva, and Sara Silveira. 2008. LXService: Web Services of Language Technology for Portuguese. In Nicoletta Calzolari, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, and Daniel Tapias, editors, *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008)*, Paris. European Language Resources Association (ELRA).

Bartosz Broda and Maciej Piasecki. 2008. SuperMatrix: a General Tool for Lexical Semantic Knowledge Acquisition. *Speech and Language Technology. Vol. 11*, pages 239–254.

Aleksander Buczyński and Adam Przepiórkowski. 2009. Spejd: A Shallow Processing and Morphological Disambiguation Tool. *Human Language Technology: Challenges of the Information Society. Vol. 5603*, pages 131–141.

CLARIN deliverable D2R-6b. 2009. Requirement Specification Web Services and Workflow Systems. `http://www-sk.let.uu.nl/u/D2R-6b.pdf`.

CLARIN deliverable D5R-3a. 2009. Linguistic processing chains as Web Services: Initial linguistic considerations. `http://www-sk.let.uu.nl/u/D5R-3a.pdf`.

Stefanie Dipper. 2005. Stand-off representation and exploitation of multi-level linguistic annotation. In *Proceedings of Berliner XML Tage 2005 (BXML 2005)*, Berlin.

Tomaz Erjavec. 2004. MULTEXT-East Version 3: Multilingual Morphosyntactic Specifications, Lexicons and Corpora. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal. European Language Resources Association (ELRA).

Gil Francopoulo, Monte George, Nicoletta Calzolari, Monica Monachini, Nuria Bel, Mandy Pet, and Claudia Soria. 2006. Lexical Markup Framework (LMF). In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, pages 233–236, Genoa, Italy. European Language Resources Association (ELRA).

Gil Francopoulo, Nuria Bel, Monte George, Nicoletta Calzolari, Monica Monachini, Mandy Pet, and Claudia Soria. 2009. Multilingual Resources for NLP in the Lexical Markup Framework (LMF). *Language Resources and Evaluation Journal. Vol. 43:1*, pages 57–70.

Karypis George. 2002. CLUTO — a clustering toolkit. Technical Report Technical Report 02-017, Department of Computer Science, University of Minnesota.

Nancy Ide, Patrice Bonhomme, and Laurent Romary. 2000. XCES: An XML-based standard for linguistic corpora. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC 2000)*, pages 825—-830, Athens, Greece. European Language Resources Association (ELRA).

Violetta Koseska-Toszewa, Ludmila Dimitrova, and Roman Roszko, editors. 2009. *Representing Semantics in Digital Lexicography: Proceedings of MONDILEX Fourth Open Workshop*, Warsaw.

Mieczysław A. Kłopotek, Sławomir T. Wierzchoń, and Krzysztof Trojanowski, editors. 2006. *Intelligent Information Processing and Web Mining*. Advances in Soft Computing. Springer-Verlag, Berlin.

Alessandro Lenci, Nuria Bel, Federica Busa, Nicoletta Calzolari, Elisabetta Gola, Monica Monachini, Antoine Ogonowski, Ivonne Peters, Wim Peters, Nilda Ruimy, Marta Villegas, and Antonio Zampolli. 2000. SIMPLE: A General Framework for the Development of Multilingual Lexicons. *International Journal of Lexicography XIII (4)*, pages 249–263.

Dekang Lin. 1993. Principle-based parsing without overgeneration. In *Proceedings of the 31st Meeting of the ACL*, pages 112–120.

Pedro Martins. 2008. Desambiguaçao Automática da Flexao Verbal em Contexto. Master's thesis, University of Lisbon.

Andreas Mengel and Wolfgang Lezius. 2000. An XML-based encoding format for syntactically annotated corpora. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC 2000)*, pages 121–126, Athens, Greece. European Language Resources Association (ELRA).

Filipe Nunes. 2007. Verbal Lemmatization and Featurization of Portuguese with Ambiguity Resolution in Context. Master's thesis, University of Lisbon.

Harris Papageorgiou, Prokopis Prokopidis, Iason Demiros, Voula Giouli, Alexis Konstantinidis, and Stelios Piperidis. 2002. Multi-level XML-based Corpus Annotation. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC 2002)*, Las Palmas, Canary Islands, Spain. European Language Resources Association (ELRA).

Maciej Piasecki and Grzegorz Godlewski. 2006. Reductionistic, tree and rule based tagger for Polish. In Kłopotek et al. (Kłopotek et al., 2006), pages 531–540.

Maciej Piasecki and Adam Radziszewski. 2009. Morphosyntactic Constraints in Acquisition of Linguistic Knowledge for Polish. Aspects of Natural Language Processing (a festschrift for Professor Leonard Bolc). *Lecture Notes in Computer Science, 5070*, pages 163–190.

Maciej Piasecki, Stanisław Szpakowicz, and Bartosz Broda. 2009. *A Wordnet from the Ground Up*. Oficyna Wydawnicza Politechniki Wrocławskiej.

Prokopis Prokopidis and Byron Georgantopoulos. 2010. Extending a Text Processing Pipeline for Greek. Submitted in LREC 2010.

Adam Przepiórkowski and Marcin Woliński. 2003. The unbearable lightness of tagging: A case study in morphosyntactic tagging of Polish. In *Proceedings of the*

4th International Workshop on Linguistically Interpreted Corpora (LINC-03), EACL 2003, pages 109–116.

Adam Przepiórkowski. 2004. *The IPI PAN Corpus: Preliminary version*. Institute of Computer Science, Polish Academy of Sciences, Warsaw.

Adam Przepiórkowski. 2009. A comparison of two morphosyntactic tagsets of Polish. In Koseska-Toszewa et al. (Koseska-Toszewa et al., 2009), pages 138–144.

Nilda Ruimy, Monica Monachini, Raffaella Distante, Elisabetta Guazzini, Stefano Molino, Marisa Ulivieri, Nicoletta Calzolari, and Antonio Zampolli. 2002. Clips, a multi-level Italian computational lexicon: A glimpse to data. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC 2002)*, Las Palmas, Canary Islands, Spain. European Language Resources Association (ELRA).

Joao Silva, António Branco, Sérgio Castro, and Ruben Reis. 2010. Out-of-the-box Robust Parsing for Portuguese. In *Proceedings of the 9th International Conference on the Computational Processing of Portuguese (PROPOR 2010)*, Porto Alegre. Pontifícia Universidade do Rio Grande do Sul.

Joao Silva. 2007. Shallow Processing of Portuguese: From Sentence Chunking to Nominal Lemmatization. Master's thesis, University of Lisbon.

Antonio Toral and Monica Monachini. 2007. SIMPLE-OWL: a Generative Lexicon Ontology for NLP and the Semantic Web. In *Workshop on Cooperative Construction of Linguistic Knowledge Bases (AIIA 2007)*.

Dan Tufiş. 2000. Using a Large Set of EAGLES-compliant Morpho-Syntactic Descriptors as a Tagset for Probabilistic Tagging. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC 2000)*, pages 1105–1112, Athens, Greece. European Language Resources Association (ELRA).

Marta Villegas, Núria Bel, Santiago Bel, Francesca Alemany, and Hector Martínez. 2009. Lexicography in the grid environment. In *Proceedings of e-lex 2009*, Louvain: Cahiers du Cental.

Jorge Vivaldi Palatresi. 2009. Corpus and exploitation tool: IULACT and bwanaNet. A survey on corpus-based research = Panorama de investigaciones basadas en corpus. In Pascual Cantos Gómez and Aquilino Sánchez Pérez, editors, *Actas del I Congreso Internacional de Lingüística de Corpus (CICL-09)*, pages 224–239. Universidad de Murcia, Asociación Espanola de Lingüística del Corpus.

Marcin Woliński. 2006. Morfeusz — a practical tool for the morphological analysis of Polish. In Mieczysław A. Kłopotek, Sławomir T. Wierzchoń, and Krzysztof Trojanowski, editors, *Proceedings of the International IIS: IIPWM'06 Conference*, pages 511–520, Wisła, Poland.

Sue Ellen Wright. 2004. A global data category registry for interoperable language resources. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal. European Language Resources Association (ELRA).

An Infrastructure for More Reliable Corpus Analysis

Kerstin Eckart, Kurt Eberle, Ulrich Heid

Universität Stuttgart, Institut für Maschinelle Sprachverarbeitung
Azenbergstr. 12, 70174 Stuttgart
{eckartkn,eberle,heid}@ims.uni-stuttgart.de

Abstract

We present an infrastructure supporting different pipelines in an approach for more reliable corpus analysis. Two interrelated pipelines build a bootstrapping approach for task specific disambiguation. The first one extracts potentially ambiguous items and proposes a reading, if enough information is available. The second one extracts potential reading indicators, which denote relevant context factors for disambiguation. These indicators are manually classified and then inserted into the lexicon of the disambiguation tool, which is in turn utilized in the next iteration step of the first pipeline. A third pipeline includes a comparison of analyses e.g. from different tools. As each tool has its own focus, adding information from one tool to the analysis of another one improves the analysis. Furthermore, where independent tools produce the same analysis, it may more likely be correct. We designed a database which supports the development of the disambiguation tool and the versions of its knowledge sources. As the analyses change along with the tool, the database provides for representation of this temporal aspect.

1. Introduction and context

This contribution deals with elements of a corpus processing infrastructure targeted at improving the reliability of corpus analyses. The context of our work[1] are two interrelated pipelines of corpus processing. We conceive the reliability tools as a third one.

The first one is aimed at the extraction of readings of potentially ambiguous items from text: it comprises tokenizing, pos-tagging and parsing, as well as an interpretation step which performs task-specific disambiguation. In our application, the targeted items are sortally ambiguous nominalizations of German verbs, such as *Abdeckung*, which can have an event reading ('the act of covering sth.'), a state reading ('being covered') and an object reading ('the cover'). Disambiguation is carried out on parsed text: the potentially ambiguous nominalizations can be disambiguated, if they appear with modifiers or selectors that have specific sortal selection restrictions (called 'indicators' in our approach). Examples are *gestrige Absperrung* ('yesterday's blocking of...', event) vs. *hölzerne Absperrung* ('wooden barrier', object) vs. *die Absperrung dauerte 3 Tage* ('the road block lasted for 3 days', state), where the adjectives *gestrig* and *hölzern* or the verb *dauern* select events, objects or states, respectively.

The second pipeline is intended to support data provision for the first one. The disambiguation tool must be provided with data about the sortal selection properties of modifiers and/or selectors of nominalizations. To this end, we can use our tool to extract example sentences from corpora which illustrate a particular syntactic construction (e.g. *Absperrung* as a direct object of verbs) or which may even receive an underspecified representation. Queries of this kind produce indicator candidates, which are then (manually) classified and inserted into the lexicon of the disambiguation tool. Thus, this second pipeline consists of parsing, data extraction, sorting and automatic pre-classification, as well as of manual classification and storage in the dictionary of the disambiguator.

The two pipelines together instantiate the typical bootstrapping spiral of corpus linguistics: we start from an initial hypothesis (in our case about sortal restrictions supporting disambiguation), extract data from the corpus, inspect these, improve the tool ('s lexicon) and reiterate data extraction, with an enhanced tool, on the basis of a refined hypothesis.

The infrastructure we discuss in this paper is meant to support reliability in this bootstrapping process. We designed a database which stores the different versions of our disambiguator and of its linguistic knowledge sources, as well as the analyses produced by these: within a homogeneous framework, we can relate individual (sets of) sentences and (sets of) analyses with the tool (stage)s by which the sentences were processed. This is crucial for efficient iterative lexicon improvement.

As a third pipeline, we extend the functionality to a comparison of analyses, possibly from different tools, on the following assumption: if we find two (or more) analyses produced independently by different tools which show the same structure, these analyses may more likely be correct than if different tools produce (significantly) diverging analyses. Therefore in the third pipeline, analyses from different tools should be merged according to rules depending on the reliability of the different analyses. The merged analysis is assumed to be more reliable than the best single analysis.

The remainder of this paper is structured as follows: In section two we give an overview of related work. In sections three and four we describe the database as a basis for our pipelines in detail. In section five, we give examples for the bootstrapping pipeline and the reliability approach, and in section six, we conclude with an outlook to future work.

2. Related Work

On the topic of disambiguation with bootstrapping approaches there is a certain similarity to approaches in word sense disambiguation, e.g. (Yarowsky, 1995). However in-

[1]Project B3, SFB 732, `http://www.uni-stuttgart.de/linguistik/sfb732/`

stead of purely statistical methods applied to syntactically unanalyzed texts we use a combination of symbolic sentence representations and frequency information, where the focus is on flat complete analyses of the context (sentence) and the comparison, and if applicable the combination, of those analyses.
Computational linguistic work on the combination of analysis results from different tools in turn is mostly aimed at robust semantic processing. A prominent example is the WHITEBOARD architecture and the pertaining middleware 'Heart of Gold' (HoG, cf. Schäfer (2007)). It combines deep semantic processing components, based on Minimal Recursion Semantics, with more surface-oriented components. The mapping is based on stand-off XML representations.
HoG has so far mainly been used for the integration of flat and deep analyses, and for multilingual comparisons; contrary to our ongoing work, HoG has however not been used for comparing several deep analyses, and in particular not for underspecified ones.
Another strand of work relevant for our objectives is the evaluation of parsers and the related comparison of their output. Mostly such evaluation work is based on a given gold standard, as e.g. in the CoNLL-X Shared Task on Multi-Lingual Dependency Parsing[2]. Our database approach is closer, however, to that of the French PASSAGE project[3]: different analysis results are compared among each other, and improvements of the analyses are achieved by means of a feedback loop; thereby, a treebank is bootstrapped, by combination of the best results (cf. the predecessor project EASy[4]). As they aim at a treebank, PASSAGE and EASy don't take underspecification into account.

3. Requirements

The database should manage different types of data which may be needed at a given point in time for the analysis of linguistic phenomena. The following are examples of such data (cf. Eckart (2009)):

- Primary data: (partial) corpora, texts or single sentences used for analyses;
- Analysis results produced by the tools, when applied to the primary data;
- The findings of the inspection of analysis results, possibly produced semi-automatically;
- Graph-based representations of individual analyses or inspections;
- Tools (or: tool versions) which produce analyses of the primary data and which may be further developed in the course of the use of the database;
- Metadata:
  - Contents-related metadata, such as indications of the author, the language, and the source of primary data;
  - Technical metadata, such as character encoding, the size of individual portions of data, etc.;
  - Metadata indicating, for a given analysis, which version of a given tool, which parameters and knowledge sources have been used in its creation.

Contrary to most database systems which are created for completely fixed processes, our database, B3DB, has been designed explicitly with a view to its infrastructural function in linguistic research. Not only has the database to manage a steady flow of new data, but also the tools used to analyse textual data, their combination, their knowledge sources, etc. are in constant evolution. This implies that the database has to cater to the temporal aspect of tools, analyses, and knowledge sources; moreover, it has to be *extensible*, *theory-independent*, and it has to support the *reproducibility* of analyses and inspections.
The requirement of *extensibility* should make it possible to introduce types of data objects which are new to the database, without need for massive changes of the database schema. The requirement of *theory-independence* concerns the intended use of the database as an infrastructure for comparing data from different tools and theories. Thus, the design of the database should not be inspired by one particular tool, but rather be generic.
The requirement of *reproducibility* implies that the database should allow us to reproduce analyses with the same tools and knowledge sources as before. This is important when tools evolve, as intermediate stages of the development need to be represented in the database.
The overall objective of the database development is to represent analyses in such a way that they can be queried, compared, and possibly integrated into more reliable analyses. In particular, different stages of the development of a single tool or a tool suite should be documentable and reproducible. This is very important for a bootstrapping approach to the development of knowledge sources for linguistic analyses.

4. Design of the database

4.1. Basic approach

It would in principle be possible to define a new database table for every type of object described by the database and for every type of relation between such objects. In our database, we use, however, another approach: B3DB foresees only few tables and expresses the properties of objects and relations in the form of different values of a specific attribute which is used for characterizing groups of objects or relations (cf. Eberle and Eckart (2009)).
This approach supports extensibility, as adding new types of data objects only requires the definition of a new attribute value (a new group), without modification of the existing schema. In our view, this approach is most economical for the specific objective of representing a changing tool

[2]In conjunction with *Tenth Conference on Computational Natural Language Learning*, `http://nextens.uvt.nl/~conll/`

[3]`http://atoll.inria.fr/passage/`

[4]EASy French parsing evaluation campaign, `http://www.technolangue.net/article198.html`

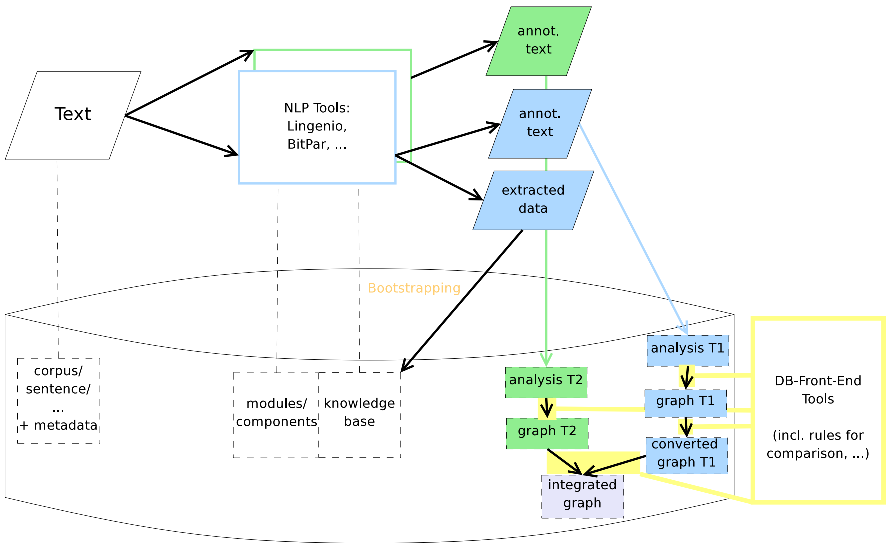


Figure 1: The database as an infrastructure to support research workflow

and knowledge source environment, both conceptually and technically.
The results of analyses or inspections can lead to rather complex representations which are not easy to search, if represented as texts. As we do not want to implement specific functions, such as those provided, for example, by `tgrep`, for tree structures, we opted for another representation of linguistic analyses. To provide for detailed queries, we therefore convert the analysis results into graph structures in the front end. These graphs (nodes and edges) are not meant to be linguistic representations, but only a general representation which can be queried more easily within the database. The individual analyses are not reinterpreted, when mapped onto the graphs, but just reformatted for internal processing purposes.

4.2. Layers of the database

The database is conceptually divided into two layers, a macroscopic one and a microscopic one. A given analysis may be an atomic object at the macro-level, and its graph-based representation is a structured object of the micro-level. Consequently, primary data and tools are always and only objects of the macro-level, whereas analyses and inspections are represented on both levels. On both layers we consistently separate objects (and their descriptions) from the relations between these objects (separate database tables).
The distinction between macro- and micro-level is implemented via a type system utilizing atomic types and Boolean combinations of types. The details of the type system are represented by a type lattice (cf. Eberle and Eckart (2009)).

4.3. Front-end routines and versioning

Above, we mentioned the overall objective (and requirement) of the database: to support the comparison between different analyses and inspections and their integration. To support such processes, front end routines can be used to abstract, modify, or translate graph representations of analyses or inspections and to reinsert the results of these operations back into the database. Such conversions may lead to the re-representation of a given analysis in a more abstract format, in an exchange format or simply in a format which is close to that of another tool, for ease of comparison. In cooperation with the Potsdam-based research group SFB 632/D1[5], we have experimented with a translation of the output of our linguistic analysis tools into the PAULA format (cf. Dipper (2005)). Ongoing work aims at the translation towards the generic format GrAF, a part of the upcoming ISO Standard LAF, the Linguistic Annotation Framework, ISO/DIS 24612 (2009).
As mentioned above, the database should support the management of different versions of tools evolving over time. To achieve this, the respective data has version labels, and a given item can become invalid, when, for example, a newer version of the same item is available in the database. The newer version may be differently classified, or corrected because errors may have been removed. It may however be necessary to keep 'outdated' data in the database as long as there exist other data related with the erroneous ones by means of explicit relations.

4.4. Implementation

The schema was implemented as a PostgreSQL[6] relational database system. It was also enhanced with a schema part based on the GrAF-Serialisation of the Linguistic Annotation Framework (LAF, ISO/DIS 24612 (2009)) and the constraint based model for representing ambiguities by Kountz et al. (2008). In this enhanced schema part, analyses can be stored according to the LAF-format which presents the basis for the merging of different analysis, because different types of analysis can be equally represented.

5. Examples: analysis reliability and bootstrapping

In the following, we give two examples of workflows which can be supported by the B3DB database, one for reliability-

[5]`http://www.sfb632.uni-potsdam.de/~d1/`
[6]`http://www.postgresql.org/`

oriented comparison of analyses, and a second one for resource bootstrapping.
Figure 1 depicts the database, as well as a text stored in it, and two parsers, BitPar (cf. Schmid (2004)) and Lingenio (cf. McCord (1991) and Eberle (2002)). BitPar's output are constituent structures, whereas the Lingenio research prototype[7] produces dependency structures. In figure 1, the analyses are called "analysis T1", produced by Lingenio and "analysis T2", produced by BitPar. For each analysis, there is a graph representation (graph T1, graph T2) at the micro-level.

5.1. Comparing and merging analyses for improving reliability

When it comes to comparing the two analyses, we use rules that insert nodes and projections into the dependency structure, to convert it into a BitPar-like format (cf. Eberle (2002)). For the following discussion, we use an example from our corpus: *Auch bei den CO-Werten liegen die Messungen weit unter dem zulässigen Grenzwert von 250ppm (parts [per million, Bestandteile in einer Million Teile)] . . .* (Also when it comes to the values for carbon monoxide, the measured data are much below the allowed threshold of 250 ppm.)
Figure 2 shows an example of a conversion rule which inserts constituency structure for the preposition *unter* (below) in the prepositional phrase *unter dem Grenzwert* (below the threshold).
One could think of the object named "converted graph T1" (in figure 1) as a result of such a conversion from Lingenio's dependency structure to BitPar format (cf. figure 3).

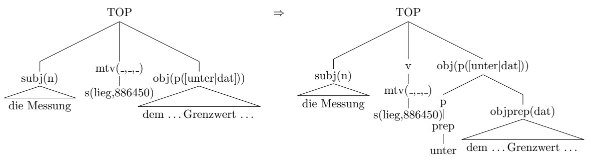


Figure 2: Example for inserting constituency structure nodes

As a result, both "graph T2" (e.g. the constituency tree in figure 4) and "converted graph T1" are now represented in a constituent-structure-like format and thus comparable.

Comparison: Similarities and differences between Lingenio and BitPar The above example shows a few differences between the two types of analyses, which can be classified as follows, according to their frequency and relevance[8]:

- Interpretation of the item *ppm*: contained in the Lingenio dictionary, but not in BitPar.
  If a tool has lexical information about a given item, we assume this tool is more reliable (on that particular item) than others which don't; cf. the correction rule in figure 7 which adjusts in this case a major structural difference;
- Interpretation of the item *weit ('much')*: difference in attachment: adverbial in the Lingenio analysis, preposition modifier in BitPar;
  This difference shows up frequently, but is of relatively little importance for the comparison, as in both analyses there is the comparable verbal phrase, which includes the prepositional phrase; cf. the mapping rule in figure 6;
- Interpretation of the *von*-PP: underspecified attachment in Lingenio; modifier of *Grenzwert* ('threshold') in BitPar;
  Structural difference without impact on the analysis of the substructure, which stays the same in both cases (*ppm* itself is treated with a separate rule, see above and figure 7);
- Interpretation of the item *CO-Wert* ('values for carbon monoxide'): decomposed compound in Lingenio, unanalyzed in BitPar;
  Difference without impact on the analysis;
- Representation of the structure of PPs: flat in BitPar (e.g. `p det adj n`), structured in Lingenio (e.g. `p objprep (det adj n)`);
  Relatable by means of a rule (see figure 5, below).

After the identification and classification of the differences, merging rules can be applied. We will give some examples of rules that are relevant for the differences mentioned above.
(Graph-)equivalence rules match representation conventions like for the structure of PP-attachments (cf. figure 5).

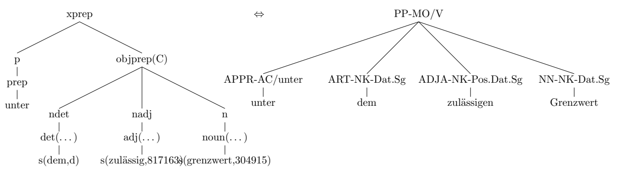


Figure 5: Graph equivalence rule for PP-representation-conventions

Another type of rules are *rules for local interpretation differences* to cope with frequent but minor differences like the representation of an adverb as adverbial-modifier versus as scalar-modifier.
The most relevant class of rules are *rules for (resource-based) correction*, that enhance the reliability of the overall analysis result, by compensation of erroneous structures. An example is the correction of a (e.g. lexically) uninformed interpretation according to the more informed representation (cf. figure 7).
Figure 8 displays the common subgraph after the merging process; it abstracts away from equivalent representations (cf. the case covered by the rule in figure 5) and from local interpretation differences.

[7]`http://www.lingenio.de/English/Research/Cooperations/unis-ims-sfb732-b3.htm`
[8]In this case, relevance refers to the impact a differing detail has on the global analysis structure.

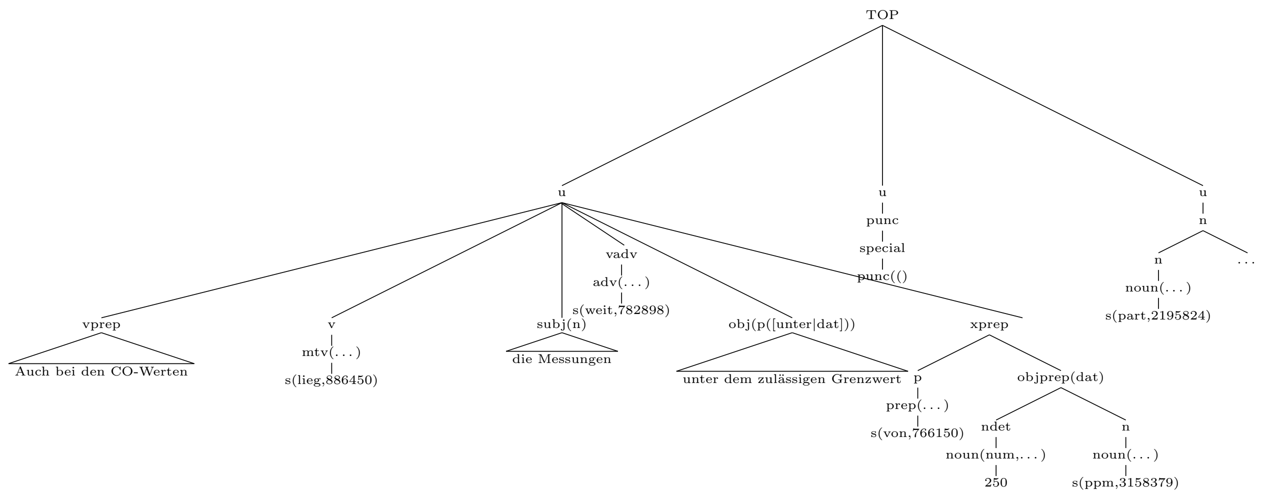


Figure 3: Constituency graph of the dependency analysis (= converted graph T1)

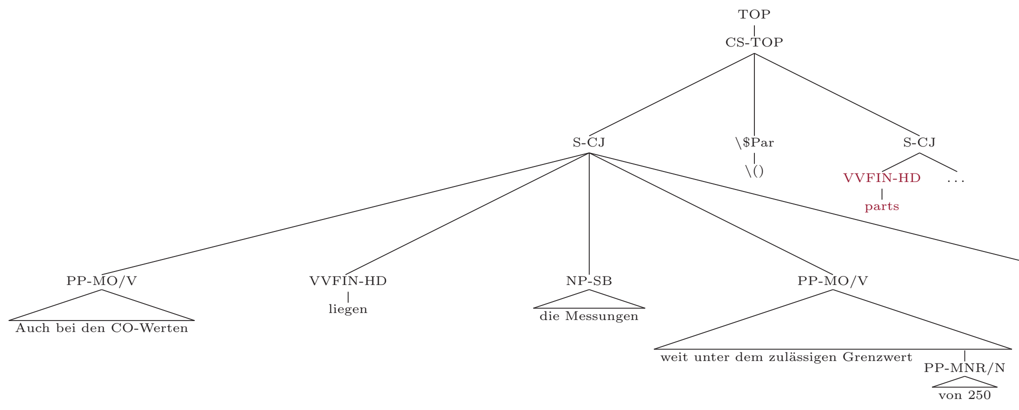


Figure 4: BitPar constituency graph (= T2)

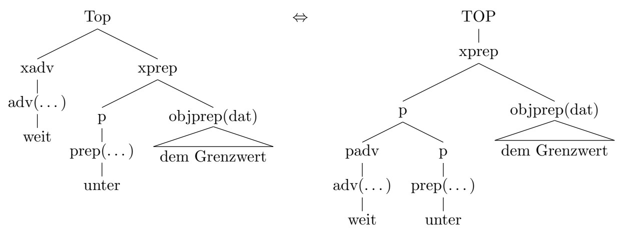


Figure 6: Rule for local interpretation differences

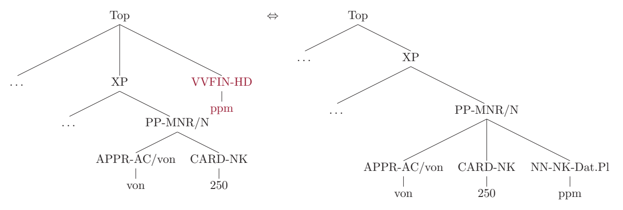


Figure 7: Correction according to informed representation

In figure 1, the object named "integrated graph" can be seen as the result of the merging of two different analyses. As we intend to combine merging procedures with procedures for identifying the reliability of analyses (and the confidence of a given tool with respect to a given analysis), a voting approach will be helpful. The rules allow to integrate those parts of the individual analyses which are individually seen as reliable, or which are equivalent. The result of the merging is a new analysis graph which is typically expected to be more reliable than the individual analyses.

5.2. Bootstrapping of resources

Another application of the database is the realization of a bootstrapping approach in tool development. Here, we make use of the temporal aspect of the database and of versioning of tools and resources. For example, the Lingenio-based disambiguator is a research prototype which is in constant evolution, and its knowledge source is constantly updated, e.g. with new indicator candidates for sortal readings of *-ung*-nouns.

Whenever such indicators are classified manually and reinserted into Lingenio's dictionary, the respective new version of the tool will be held in the database. This new tool

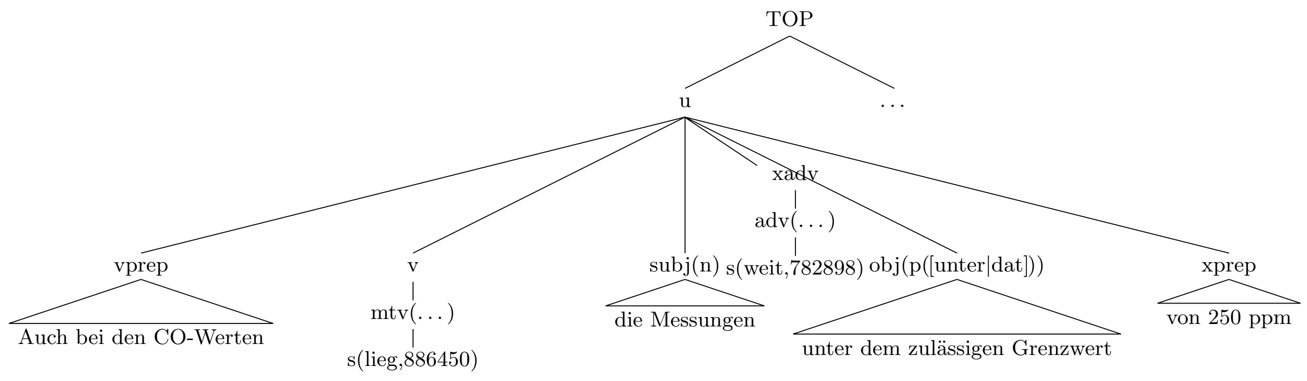


Figure 8: Merged analysis modulo equivalence and local differences in interpretation

version can then be applied on a test suite or on corpus data, to determine the improvement gained by the update. In this sense, the database supports a bootstrapping approach to tool development.

6. Future Work

Obviously, there can be many more differences between analyzers or between analysis results than shown in our example above; thus the comparison has to take into account further aspects; among these are the formalism used, its expressivity, and the levels of linguistic description covered.
Thus the "meaning-preserving translation" shown in the above example is rather a special case; in a more general fashion, a reinterpretation of the output produced by different tools is needed before a mapping can be attempted (semantic interoperability).
In future work, we intend to address this issue by analyzing in more detail the following two aspects of such a comparison, as well as their interaction:

- comparing syntactic representations with flat semantic representations: for this comparison, a mapping has to be defined which uses information about the syntax/semantics interface, similar to (flat) semantics construction tools;
- comparing fully specified and underspecified representations from a given level of description: a starting point for this task are the specific readings subsumed by the underspecified representation.

To integrate several analyses into a common, maximally reliable analysis (possibly an underspecified one), quality criteria for the individual analyses have to be defined. As we do not intend to evaluate against a gold standard, but to compare different tool outputs among each other, these need to be weighted beforehand. For example, a tool which has lexical information for a given text portion under analysis will receive more weight than an uninformed one. We will use a voting approach to combine those partial analyses on which most tools agree. We will also need to include effectual methods for comparing the analyses as such and so be eventually able to evaluate the quality of the combined analyses.

7. Summary

In a research project where different types of data are acquired and tools are developed by means of bootstrapping, an infrastructure must provide for a representation of this temporal aspect. The B3DB explicitly supports the bootstrapping approach in tool development regarding the tool modules, knowledge bases and analysis results. Moreover the database also supports a voting-like approach to reliability where analyses from different NLP tools can be combined.

8. References

Stefanie Dipper. 2005. XML-based Stand-off Representation and Exploitation of Multi-Level Linguistic Annotation. In *Berliner XML Tage*, pages 39–50.

Kurt Eberle and Kerstin Eckart, 2009. *Eine Datenbank für Textanalysen - Design und Beispiele*. Institut für maschinelle Sprachverarbeitung, Universität Stuttgart. Unpublished manual.

Kurt Eberle. 2002. Tense and aspect information in a FUDR-based German French Machine Translation System. In Hans Kamp and Uwe Reyle, editors, *How we say WHEN it happens. Contributions to the theory of temporal reference in natural language*, pages 97–148. Niemeyer, Tübingen. Ling. Arbeiten, Band 455.

Kerstin Eckart. 2009. Repräsentation von Unterspezifikation in relationalen Datenbanksystemen. Diploma thesis, Universität Stuttgart.

ISO/DIS 24612. 2009. Language resource management - Linguistic annotation framework (LAF).

Manuel Kountz, Ulrich Heid, and Kerstin Eckart. 2008. A LAF/GrAF-based encoding scheme for underspecified representations of dependency structures. In *Proceedings of the 6th Conference on Language Resources and Evaluation (LREC 2008)*, Marrakech, Morocco, May. [CD-ROM].

Michael McCord. 1991. The slot grammar system. In Jürgen Wedekind and Christian Rohrer, editors, *Unification in Grammar*. MIT-Press.

Ulrich Schäfer. 2007. *Integrating Deep and Shallow Natural Language Processing Components - Representations* *and Hybrid Architectures*. Ph.D. thesis, Saarland University.

Helmut Schmid. 2004. Efficient Parsing of Highly Ambiguous Context-Free Grammars with Bit Vectors. In *Proceedings of the 20th International Conference on Computational Linguistics, Coling'04*, volume 1, pages 162–168, Geneva, Switzerland.

David Yarowsky. 1995. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, Cambridge, MA.

The Tanl Pipeline

Giuseppe Attardi, Stefano Dei Rossi, Maria Simi
Dipartimento di Informatica, Università di Pisa
Largo B. Pontecorvo 3, I-56127 Pisa, Italy
E-mail: attardi@di.unipi.it, deirossi@di.unipi.it, simi@di.unipi.it

Abstract

Tanl (Natural Language Text Analytics) is a suite of tools for text analytics based on the software architecture paradigm of data pipelines. Tanl pipelines are data driven, i.e. each stage pulls data from the preceding stage and transforms them for use by the next stage. Since data is processed as soon as it becomes available, processing delay is minimized improving data throughput. The processing modules can be written in C++ or in Python and can be combined using few lines of Python scripts to produce full NLP applications. Tanl provides a set of modules, ranging from tokenization to POS tagging, from parsing to NE recognition. A Tanl pipeline can be processed in parallel on a cluster of computers by means of a modified version of Hadoop streaming. As a case study for the Tanl suite we annotated the English and Italian subsets of Wikipedia. We present the architecture, its modules and some sample applications.

Introduction

Text analytics involves many tasks ranging from simple text collection, extraction, and preparation to linguistic syntactic and semantic analysis, cross reference analysis, intent mining and finally indexing and search. A complete system must be able to process textual data of any size and structure, to extract words, to classify documents into categories (taxonomies or ontologies), and to identify semantic relationships.

A full analytics application requires coordinating and combining several tools designed to handle specific subtasks. This may be challenging since many of the existing tools have been developed independently with different requirements and assumptions on how to process the data.

Several suites for NLP (Natural Language Processing) are available for performing syntactic and semantic data analysis, some as open source and other as commercial products. These toolsets can be grouped into two broad software architecture categories:

- *Integrated Toolkits*: these provide a set of classes and methods for each task, and are typically bound to a programming language. Applications are programmed using compilers and standard programming environments. Examples in this category are: LingPipe (LingPipe), OpenNlp (OpenNLP), NLTK (NLTK).
- *Component Frameworks*: these use generic data structures, described in a language independent formalism, and each tool consumes/produces such data; a special compiler transforms the data descriptions into types for the target programming language. Applications are built using specific framework tools. Examples in this category are: GATE (GATE), UIMA (UIMA).

Both GATE and UIMA are based on a *workflow software architecture*, where the framework handles the workflow among the processing stages of the application, by means of a controller that passes data among the components invoking their methods. Each tool accepts and returns the same type of data and extends the data it receives by adding its own information, as shown using different colors in Figure 1: the Tokenizer adds annotations to represent the start and end of each token, the PosTagger adds annotations representing the POS for each token.

Since the controller handles the whole processing in a single flow, each processing component receives the whole collection and returns the whole collection. If the collection is big, this might require large amounts of memory.

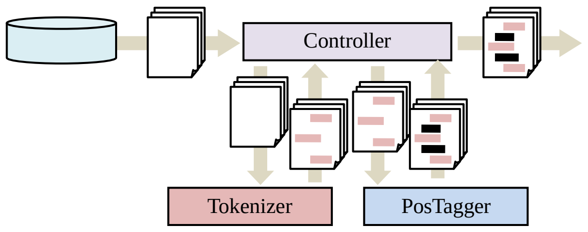


Figure 1: Workflow Software Architecture.

In this paper we present an alternative architecture based on the notion of *data pipeline*. The Tanl pipeline (Natural Language Text Analytics) uses both generic and specific data structures, and components communicate directly exchanging data through pipes, as shown in Figure 2. Since each tool pulls the data it needs from the previous stage of the pipeline, only the minimum amount of data passes through the pipeline, therefore reducing the memory footprint and improving the throughput. The figure shows single documents being passed along, but the granularity can be even smaller: for instance a module might just require single tokens or single sentences. This would be hard to handle with a workflow architecture, since the controller does not know which amount of data each tool requires.

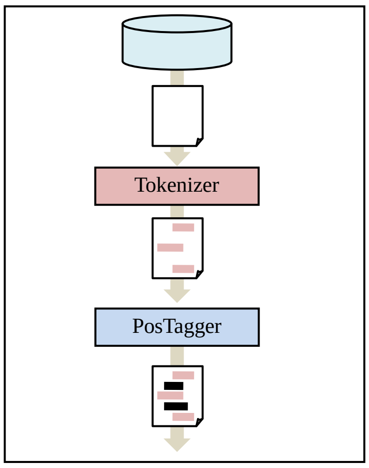


Figure 2: Tanl data pipeline.

Related work

We present an overview of some representative NLP toolsets and highlight the differences with the approach adopted for the Tanl pipeline.

Integrated Toolkits

NLTK (Natural Language Toolkit) is a suite of libraries and programs written in Python for symbolic and statistical natural language processing (Steven et al., 2009). For each task NLTK provides a specific API, implemented by several alternative modules. For example there are several chunker modules providing the `ChunkParserI` interface, classifier modules providing the `ClassifierI` interface, etc. Each interface specifies different data types, for instance the `ChunkParserI` interface operates on tokens represented as tuples (word, tag), the `ParserI` interface accepts a string and returns a `Tree`. Since many modules were developed independently, sometimes they provide their own API that extends the generic one. For instance one implementation of a dependency parser requires as input two lists, a list of tokens and a list of tags, another implementation operates on files, hence it creates an intermediate temporary file.

Workflow Frameworks

GATE (General Architecture for Text Engineering) is a Java framework organized according to three concepts: language resources, processing resources and the controller. A GATE application handles the following types of data:

- `Features`: a set of name/values pairs;
- `Annotation`: consists of a tuple (`start`, `end`, `type`, `features`), the start and end character positions in the text, the type of the annotation and the features associated to the annotation;
- `Document`: consists of a triple (`content`, `annotations`, `features`), where the content is the text of the document, annotations are the annotations in the document and features are those associated to the document.
- `Corpus`: a list of `Documents`.

In the following example, two processing resources are created (a `Tokenizer` and a `PosTagger`), a language resource is opened (a `Corpus`) and a controller is created of type `SerialAnalyserController`. The language and processing resources are supplied to the controller which supervises and coordinates the overall workflow: at each analysis step it passes data to a processing resource, gets back the enriched results and passes them along to the next step.

```
SerialAnalyserController sac =
   Factory.createResource(
                "SerialAnalyserController", …);
FeatureMap params = Factory.newFeatureMap();
sac.add(Factory.createResource(“Tokeniser”,
        params));
sac.add(Factory.createResource(“PosTagger”,
        params));
Corpus corpus = …;
sac.setCorpus(corpus);
sac.execute();
```

UIMA (Unstructured Information Management Architecture) is a general framework for the analysis of text and other media. The fundamental UIMA data model is called Common Analysis Structure (CAS): it provides data modeling, definition and retrieval facilities for the annotations stored in it. Annotations are defined in a hierarchically organized type system rooted in a basic type that contains the start and end position in the document as well as a set of features. Processing is performed by Analysis Engines (AE) according to a simple I/O logical interface model: each AE gets a CAS as input and produces a CAS as output. Typically each AE analyzes a CAS containing a document and adds more metadata to the CAS structure.

Each UIMA component, written in Java or C++, implements interfaces defined by the framework and provides self-describing metadata via XML descriptor files. An application can be created by joining together various components as shown in the following example.

```
AnalysisEngine tokAnnotator =
                    produceAnalysisEngine(…);
AnalysisEngine posAnnotator =
                    produceAnalysisEngine(…);

ArrayList<AnalysisEngineMetaData> mdl =
                      new ArrayList<…>();
mdl.add(tokAnnot.getAnalysisEngineMetaData());
mdl.add(posAnnot.getAnalysisEngineMetaData());

CAS aCAS = createCAS(mdl);
aCAS.setDocumentText(getTextFromFile(…));

tokenAnnotator.process(aCAS);
posAnnotator.process(aCAS);
```

Two AEs are created, a tokenizer and a POS tagger. Then a CAS is created that contains both metadata from the tokenizer and the POS tagger. The CAS is given to both the AEs in sequence and each adds its own annotations. The framework manages the AEs and the data flow between them.

A CAS Consumer processes the CAS produced by an AE.

For example one can collect all annotations of type `Entity` with the following code:

```
ArrayList<String> entities = new …;
JFSIndexRepository idx =
    CAS…getJFIndexRepository();
Iterator<Entity> it =
idx.getAnnotationIndex(Entity.type).iterator()
;

while (it.hasNext()) {
  Entity en = it.next();
  entities.add(en.getCoveredText());
}
```

UIMA additionally provides capabilities to wrap components as network services.
The JULIES NLP Toolsuite consists in a collection of UIMA components (JULIES NLP).

Tanl design

The Tanl architecture is based on a data pipeline paradigm (Figure 2) and allows integrating modules written in different languages.
The approach has the advantage that each component is directly connected to the other ones through pipes, so it is not necessary to wait until the end of one processing phase before starting the next one. As a tool produces the first result it is immediately passed to the next one through the pipeline without producing intermediate data structures.
Another advantage of this approach is that it is possible to compose a Tanl pipeline using a general scripting language, for example Python or Perl, instead of introducing ad-hoc tools.
Most of the Tanl tools exploit quantitative and statistical machine learning methods and they require an annotated training corpus for creating statistical models of the data.

Software architecture

In this section we describe the basic components of the Tanl pipeline.

Pipeline

The pipeline components can be distinguished into three basic types:

- *Source*: creates an initial pipe (e.g. a document reader, reading from a text file and creating a stream of tokens to be sent through the pipeline);
- *Transform*: receives data from one pipe and produces output on another pipe;
- *Sink*: consumes the output of a pipe.

For example a source can be created as an instance of `SentenceSplitter` and connected in pipe to an input stream:

```
ss = SentenceSplitter('ita.punkt').pipe(stdin)
```

The pipe can then be connected to other tools performing various tasks such as tokenization, POS tagging and parsing as follows:

```
wt = Tokenizer().pipe(ss)
pt = PosTagger('italian.pos').pipe(wt)
pa = Parser.create('italian.MLP').pipe(pt)
```

Each line in the above example represents a transformation stage in the pipeline. No processing of data happens while the pipeline is being built.
Processing in the assembled pipeline only starts when a sink is connected to the pipeline and starts drawing items from it, in a fully data-driven process. Each stage in the pipeline, when requested for the next item, requests items from its preceding stage in order to produce the next output.
A sink can just be defined through a standard Python iterator pulling data from the last stage of the pipeline:

```
ret = ""
for s in pa:
   ret += string(s) + "\n"
return ret
```

Using a general purpose scripting language for composing the pipeline avoids the need for compilers and other development tools. Special processing can be added at any stage of the pipeline for whatever need with a few lines of code and exploiting the facilities of Python.
For example, if one needs to monitor what is happening at a certain stage in the pipeline, a tee component can be added for analyzing the items passing through that stage. Here is an example of a tee used for printing all items after the POS tagging stage:

```
pt = PosTagger('italian.pos').pipe(wt)
tee = Tee(printSink, pt)
pa = Parser.create('italian.MLP').pipe(tee)
```

The first argument to the `Tee` constructor is a sink through which items have to be pushed. However, since sinks behave in a pull mode, the `Tee` has to create an inversion of control, turning a sink into a pipe, by exploiting the functional mechanisms of Python.

```
class Tee:
    def __init__(self, func, arg):
        self.func = func
        self.arg = arg

    def next(self):
        aux = self.arg.next()
        func(aux)
        return aux
```

The `next` method of the `Tee` applies the function to the item (in this case the printing function) before passing the item itself to the next module in the pipeline.
An alternative solution would be to run sinks within separate threads and adopting an asynchronous streaming model, where each consumer processes data at its own pace. This however involves providing buffering in the components, partly defeating the purpose of a data driven pipeline, where data is produced only when requested and processed immediately.
A disadvantage of an integrated multi-language pipeline is dealing with debugging: since at the scripting language level the pipeline components are only visible as black boxes, it is hard to step into their execution for debugging code. An instrumented version of the Python runtime is required in order to start a process in debugging mode.

Metadata handling

As data passes through a pipeline, global or specific metadata might need to be collected. For example, Wikipedia articles contain metadata information such as hypertext links, internal references as well as internal document structure such as titles and sections. These data are useful for certain modules of the pipeline, but unnecessary or unmanageable for others.

Workflow systems like GATE or UIMA store these as annotations in a global structure like the CAS. Since in Tanl items are passed along the pipe, there is no place to store global data.

Our solution consists in storing metadata in the tokens, using a field called context. A context contains a set of key/value pairs representing metadata. Contexts can be nested, referring to a parent context, for representing nested structures such as sections within documents or XML trees. Tokens that belong to the same context share the same context object, so that memory overhead is reduced, even using a distributed representation rather than a global structure.

Implementation

Enumerators and Tokens

Each module of the Tanl pipeline consumes a stream produced by a previous module and produces a stream. Streams consist of tokens or combinations of tokens, e.g. sentences which are sequence of tokens.

Tokens are the basic data structure type that all components manipulate. The token data structure was designed to be extensible, so that each tool can add to it its own annotations, which are passed along to later stages.

A token contains a string that represents its form and an arbitrary number of attributes and links. Attributes are simple key/value pairs, while links are labeled arcs referring to other tokens through their id:

```
struct Token {
  string     form;    ///< word form
  Attributes attributes;
  Links      links;
  Context*   context; ///< context
};

struct Link {
  int    target; ///< the ID of the target
  string label;  ///< the label for the link
};
```

In the implementation of attributes though, we avoid the naive solution of using a hash table, since this would entail a significant cost for each token: instead the token only contains the attribute values and an index of attribute names is used to retrieve an attribute by name. The index is shared among all the tokens in a `Corpus`.

A `Corpus` represents the common aspects of a collection of documents, including the tongue, the list of token attributes, the attribute name index, the file format and it also provides methods for writing and reading sentences from corpus documents.

A stream is defined through a generic class `Enumerator` that provides methods to advance to the next item and to access it:

```
template <class T>
class Enumerator {
public:
  virtual bool    MoveNext() = 0;

  virtual T       Current() = 0;

  virtual void    Reset() {}
};
```

Listing 1: Generic Enumerator interface

Each module provides an interface for connecting to a pipeline:

```
template <class Tin, Tout>
struct IPipe {
   Enumerator<Tout>* pipe(Enumerator<Tin>&);
};
```

Listing 2: Pipeline interface

Language Integration

C++ modules can be invoked from scripting languages by means of wrappers created with SWIG (SWIG), an automated tool for generating wrappers directly from code. In particular Tanl provides predefined wrappers for Python.

C++ Enumerators as Python iterators

SWIG allows exposing C++ objects and methods to Python, but an even tighter integration is provided that allows operating on C++ objects in a more convenient way. In particular the standard Python iterator constructs, for instance `for x in pipe:` …x…, can be used to process pipeline streams. Since the Python iterator protocol consists of a single method `next()` and termination is obtained by raising an exception, a magic trick is required in the SWIG code in order to conform to this protocol:

```
%exception Tanl::Enumerator<Item>::next() {
  $action
  if (!result) {
    PyErr_SetObject(PyExc_StopIteration,
                    Py_None);
    return NULL;
  }
};
```

This SWIG notation is used to add a few lines to the wrapper for method `next()` that will raise the required exception.

Memory management

Since objects are passed between C++ and Python, stored within wrappers, memory must be managed properly so that objects are released when no longer in use. This is normally handled by telling to SWIG which objects must remain under control by Python. Python uses reference counting, and when an object is no longer accessible, it automatically calls its C++ destructor.

However there are cases where this mechanism is not sufficient, for example when a pipe is created like this:

```
pp = Parser.create("model").pipe(sr)
```

both a parser object and a parser proxy that wraps it are created. Then a pipe is created which refers to the parser object and assigned to the variable `pp`. The parser should survive as long as pp exists. However Python destroys the parser proxy, since it has no references to it, and calls the parser C++ destructor. In order to avoid this, a reference count is added to the C++ parser object, reflecting the number of Python objects referring to it. The C++ destructor is only invoked when this count goes to 0. In order to maintain this counter, it must be incremented when the pipe is created from Python. This can be done in SWIG with the following rule:

```
%exception Parser::pipe {
  $action
  arg1->incRef();    // arg1 is the parser object
}
```

When the count of the pipe proxy reaches zero, Python calls the pipe destructor.
Similarly, the parser counter must be decremented when the pipe proxy that embeds the parser gets destroyed. This is done with:

```
%feature("unRef")
Tanl::Enumerator<Tanl::Sentence*>
"$this->Dispose();"
```

which will call the following method on the pipe:

```
void ParserPipe::Dispose() { parser.decRef();
                             delete this; }
```

that will decrement/release the parser before deleting the pipe. Finally, in order to keep in synch the reference count of the parser proxy, it must be updated whenever Python creates a reference to it, by using these SWIG rules:

```
%feature("ref") Parse "$this->incRef();"
%feature("unref") Parse "$this->decRef();"
```

A reference count mechanism is also required to manage `Context` objects used in tokens, since their lifetime duration is independent from that of the token where they appear.

Map Reduce

A Tanl pipeline can be processed in parallel using the Map/Reduce pattern, for instance using Apache Hadoop (Hadoop). The data to be processed is partitioned into subsets, each of which is assigned for processing to a node in the cluster.
The mapper and reducer functions are normally written in Java, but the framework also provides a facility called Hadoop streaming that allows running any executable as a mapper or reducer.
Unfortunately the standard implementation of Hadoop streaming does not ensure that the outputs of each mapper are combined by the reducer preserving the original order. To overcome this problem we modified the implementation (Tanl Hadoop Streaming) by adding a sequence number to each document passed to the mapper and introducing a reducer that uses these numbers for recombining the documents in the original order.

Pipeline Modules

The following modules are currently available as part of the Tanl pipeline:

- Sentence Splitter: splits the text into sentences, producing an enumerator of strings, each representing a sentence. The module is written in Python and is based on the Punkt Tokenizer from the NLTK suite, which implements the technique by Kiss and Strunk (Kiss & Strunk, 2006).
- Word Tokenizer: deals with the segmentation of a sentence into tokens, producing a stream of vector of tokens. The module consists of C++ code produced using Quex (Quex), a generator of lexical analyzers, capable of handling Unicode characters.
- Word Aggregator: combines polyrematic expressions of common use into a single token (e.g. “a meno che” becomes “a_meno_che”).
- POS Tagger: enriches the structure `Token` representing a token within a sentence with attributes representing the PoS and lemma. Two alternative taggers are available: one based on TreeTagger (Schmid, 1994) and one based on Hunpos (Halácsy et al., 2007), an open source reimplementation of TnT (Brants, 2000).
- Morph Splitter: splits the POS of each token into separate POS and morphology attributes and also splits clitic forms into two or more tokens (e.g. the verb “avercelo” becomes “aver- ce- lo”).
- Parser: parses sentences producing dependency parse trees. The module takes as input a stream of vectors of tokens, and produces a stream of sentences. It uses DeSR, a state-of-the-art multilingual dependency parser based on the Shift/Reduce paradigm (Attardi, 2006; Attardi et al., 2007; Attardi et al., 2009).

A few semantic analysis modules are also available; as the previous modules, they consume and produce a stream of vectors of tokens, adding specific semantic attributes to the structure `Token`:

- Named Entity Tagger: identifies and classifies atomic elements such as person names, organizations, locations, temporal expressions, quantities, percentages etc.
- SuperSense Tagger: assigns a semantic category to nouns, adjectives and verbs, corresponding to the WordNet lexicographer class labels (Fellbaum, 1998).

Both tools use a Maximum Entropy classifier provided in the Tanl library.
The Tanl Indexer produces a special full-text search index enriched with syntactic and semantic information. The index is organized in multiple layers, so that at each document position a stack of values is present. Each layer represents a different token attribute, e.g. form, lemma, POS, NE, SuperSense and dependency relations. The index also maintains information on sentence boundaries so that the search can return sentences matching queries rather than documents. An additional inverted index is also present that allows searching for pairs of word in a given syntactic relation. A special query language allows expressing queries involving not just words, but any attributes of tokens and in particular dependency paths in

the parse trees. Typical boolean, proximity and phrase operators allow forming even more complex queries.

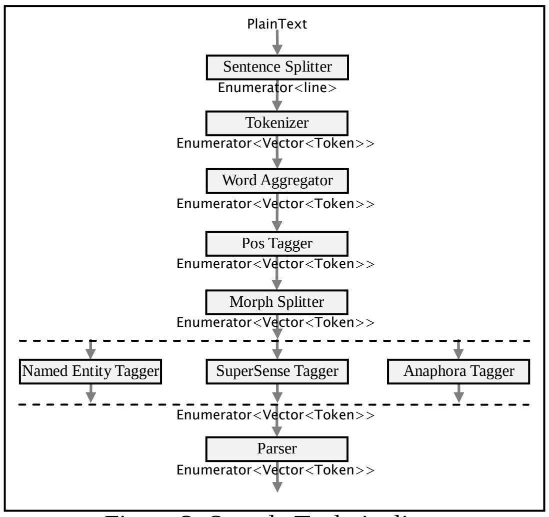

PlainText
Sentence Splitter
Enumerator<line>
Tokenizer
Enumerator<Vector<Token>>
Word Aggregator
Enumerator<Vector<Token>>
Pos Tagger
Enumerator<Vector<Token>>
Morph Splitter
Enumerator<Vector<Token>>
Named Entity Tagger
SuperSense Tagger
Anaphora Tagger
Enumerator<Vector<Token>>
Parser
Enumerator<Vector<Token>>
Figure 3: Sample Tanl pipeline.

Figure 3 shows an example of a full Tanl pipeline built with some of the available modules.

Applications

As a case study for the Tanl suite we annotated two significant subset of Wikipedia: the English Wikipedia, consisting in over 3 million articles for a total of 29.320.747 sentences and the Italian Wikipedia, consisting of over 660.000 articles for a total of 5.507.225 sentences. The Wikipedia is challenging both in terms of size and in terms of the variety of material and topics covered. DeepSearch and Yahoo! Correlator are two applications that use the annotated Wikipedia.

DeepSearch

DeepSearch (DeepSearch) is a Wikipedia search engine that exploits syntactic and semantic annotations extracted from Wikipedia articles. The extended query language allows expressing queries that involve various attributes in the annotation.
For example “Who killed Caesar?”, can be answered by sentences where Caesar is the object of the verb ‘to kill’: this can be expressed in our special query language as a query for the word ‘Cesar’ occurring as the dependent of a dependency labeled as ‘OBJ’ and whose head is a word with lemma ‘kill’.
Similarly “What Edison did not invent?” can be answered retrieving sentences where ‘Edison’ is the subject of a verb of category ‘Creation’ (one of the Super Senses), with a negation as a modifier of the verb.

Yahoo! Correlator

Yahoo! Correlator (Yahoo! Correlator) is a search engine and content aggregator that extracts and organizes information from text, collects and displays related names, concepts, places, and events correlated to user queries.
The online demo is based on an annotated version of the English Wikipedia processed with earlier versions of the Tanl pipeline tools.
The main result page shows a synthetic page assembled from several Wikipedia entries matching the search, grouped using the Wikipedia category structure. Additional pages display names of people, places on a map, concepts or events in a timeline related to those found in answers to the query.

Dependency Parser

A dependency parser can be built with a few lines of scripting similar to those presented in Section 0. This can be turned into a Web service for processing multiple requests by creating the transforms just once:

```
ss = SentenceSplitter('italian.punkt')
tk = Tokenizer()
pt = PosTagger('italian.ttagger')
ms = MorphSplitter()
pa = Parser.create('italian.MLP')
```

A pipe is created connecting these modules each time a request is received to parse a string `text`:

```
p1 = [text]
p2 = ss.pipe(p1)
p3 = tk.pipe(p2)
p4 = pt.pipe(p3)
p5 = ms.pipe(p4)
p6 = pa.pipe(p5)
ret = ""
for s in p6:
   ret += c.toString(s) + "\n"
return ret
```

A Web service actually running this code is available at http://paleo.di.unipi.it/parse (Tanl Parser). The parser used is the DeSR dependency parser, which uses a MultiLayer Perceptron model and produces parse trees annotated using the Tanl Dependency Notation (Tanl Dependency Notation). The output parse trees are displayed graphically in HTML or can be obtained in the CoNLL X format (CoNLL X Format).
TornadoWeb (TornadoWeb) is used as an application server for Python.

Performance

The parse service described above is capable of parsing sentences of a dozen tokens in 10-20 milliseconds.
A batch pipeline from pure text to parse trees can process typically four Wikipedia articles per second. As a consequence, by parallelizing the process on a dozen of nodes, the whole Italian Wikipedia can be processed in about 4 hours.

Conclusions

We presented the software architecture underlying the Tanl suite. The benefits of the pipeline can be summarized as follows:

- Data pipeline: modules share a common data model based on a flexible and extensible representation of tokens which are passed along the pipe;
- Processing on demand: processing is data-driven and each stage pulls data as needed from the previous stages;
- Data granularity: the blocks of data traversing the pipeline are smaller than in the other toolsets. This reduces memory requirements and improves latency.
- Efficiency: core algorithms are written in C++;
- Flexibility: Python wrappers allow configuring

pipelines using simple scripts and activating or monitoring the pipelines by inserting intermediate stages;
- Parallelism: collections can be partitioned and several pipes can be run in parallel on a cluster using a modified version of Hadoop Streaming (Tanl Hadoop).

Acknowledgments

Francesco Tamberi and Antonio Fuschetto participated in the development of the Tanl pipeline; Felice Dell'Orletta contributed to the DeSR parser.
Partial support was provided by Yahoo! Research and by Fondazione Cassa di Risparmio di Pisa.

References

G. Attardi. Experiments with a Multilanguage non-projective dependency parser. In *Proc. of the Tenth CoNLL*. 2006.
G. Attardi, A. Chanev, M. Ciaramita, F. Dell'Orletta and M. Simi. Multilingual Dependency Parsing and Domain Adaptation using DeSR. *Proc. the CoNLL Shared Task Session of EMNLP-CoNLL 2007*. Prague, CZ. 2007.
G. Attardi, F. Dell'Orletta, M. Simi, J. Turian. Accurate Dependency Parsing with a Stacked Multilayer Perceptron. *Proc. of Workshop Evalita 2009*. 2009.
T. Brants. TnT–A Statistical Part-of-Speech Tagger, *Proc. of ANLP-NAACL Conf.* 2000.
C. Fellbaum, WordNet An Electronic Lexical Database. MIT Press, 1998.
P. Halácsy, A. Kornai, C. Oravecz. HunPos – an open source trigram tagger, In *Proc. of the Demo and Poster Sessions of the 45th Annual Meeting of the ACL*, Prague, Czech Republic 209–212 (2007)
T. Kiss and J. Strunk. Unsupervised multilingual sentence boundary detection, *Computational Linguistics*. Cambridge, USA: MIT Press, 2006, vol. 3-(4).
H. Schmid. Probabilistic Part-of-Speech Tagging Using Decision Trees. In *Proc. of the International Conference on New Methods in Language Processing*, 44-49 (1994)
B. Steven, E. Klein and E. Loper. *Natural Language Processing with Python*. O'Reilly Media Inc. 2009.
CoNLL X Format: http://depparse.uvt.nl/depparse-wiki/DataFormat
DeepSearch: http://semawiki.di.unipi.it/demo.html
GATE: http://gate.ac.uk/
Hadoop: http://hadoop.apache.org/
JULIE NLP: http://www.julielab.de/Resources/Software/NLP_Tools.html
LingPipe: http://alias-i.com/lingpipe/
NLTK: http://www.nltk.org/
OpenNLP: http://opennlp.sourceforge.net/
Quex: http://quex.sourceforge.net/
SWIG: http://www.swig.org/
Tanl Dependency Notation: http://medialab.di.unipi.it/wiki/Tanl_Tagsets
Tanl Hadoop Streaming: http://medialab.di.unipi.it/wiki/Hadoop
Tanl Parser: http://paleo.di.unipi.it/parse
TornadoWeb: http://www.tornadoweb.org/
UIMA: http://incubator.apache.org/uima/
Yahoo! Correlator: http://sandbox.yahoo.com/Correlator

Text handling as a Web Service for the IULA processing pipeline

Héctor Martínez, Jorge Vivaldi, Marta Villegas
Institut Universitari de Lingüística Aplicada (IULA)
Universitat Pompeu Fabra, Barcelona, Spain
E-mail: hector.martinez@upf.edu, jorge.vivaldi@upf.edu

Abstract

A text handler that analyzes free text and outputs sentence boundaries, among other basic text patterns, is a necessary tool for most NLP tasks. This basic tool is not equally covered for every language. Therefore the IULA has decided to develop its own text handling system focused on Catalan and Spanish, but also expanded for English. The universal need of a tool like this has led us to publish the text handling tool as a Web Service, so that users that want to include this type of process in their HLT processing pipelines can do so with minimal effort. This paper describes the design decisions and functionalities of the tool, and offers a first proposal for a common WSDL interface for text handling tools. The system presents a general set of language-independent rules, but makes use of lexicon-based heuristics to determine its tagging decisions.

1. Introduction

Text handling or preprocessing is a basic need for any Natural Language Processing (NLP) system. As it is well known, text processing implies coping with a number of practical issues, which not only derive from the inherent difficulties of NLP, but also include: misspelling or unknown words, a myriad of punctuation signs, numbers, labels, dates in various formats, multi-word units, proper nouns, foreign words, etc. Some of these items have specific conventions for every language (decimal signs, dates or proper nouns, among others), but all of them have to be taken into account to produce material that is good enough to be useful for linguistic research.

The preprocessing stage is very often overlooked or grossly approximated. Sentence boundaries, for instance, are often established by hand or placed after every period followed by a whitespace character and a capital letter, without further context evaluation, even though the role of punctuation signs is often ambiguous. This ambiguity may give bizarre results, if not carefully treated. For instance, preprocessing might fail in strings as simple as "*Dr. Smith*", where no sentence boundary is expected but could be falsely assigned.

Manual tagging has the downside of being very slow (and, like any manual task, not error-free), when compared to an automatic process, which becomes necessary for the quick loading of large amounts of data (dozens of documents, each with several thousand words) into a corpus.

The text handler is a crucial part of the process of loading a text from its original file into the IULACT[1] corpus. The IULACT is a corpus compiled at IULA[2] to support research and teaching activities. This corpus and its corresponding tools provide the computational basis for a number of tasks in both monolingual and multi-lingual frameworks, such as concordances based on morphosyntactic information, term detection and extraction, text alignment, automatic summarization, syntactic analysis, etc. (Vivaldi, 2009)

In corpus compilation, as in most NLP tasks, the text undergoes a series of steps referred to as processing chain (at least normalization, verticalization and pos-tagging).. The first step in our case is the handling, where a text is analyzed and tagged to make subsequent NLP tasks easier and less error-prone.

Given the importance of this basic tool for all NLP applications and the lack of resources for the Catalan language at the time of the initial compilation of the IULACT (early nineties), the IULA (*Institut Universitari de Lingüística Aplicada*) developed its own text-handling system. At that time, the corpus was built in compliance with the Corpus Encoding Standard, mixing manual and automatic procedures. After gaining some experience in compiling/using the corpus, it became clear that some of the implemented features increased the processing difficulty and also the necessary resources (manual intervention was higher than desirable or expected) in order to compile the corpus, without any major benefit for corpus users.

Consequently, it was decided to develop a new text handling software, with the main objective of minimizing human intervention in corpus compiling, keeping the key characteristics of CES standard. This new tool has been deployed as a Web Service for internal and external usage. If a tool is made available for usage on the Web, the interface has to be streamlined to make sure that it can be interoperable and easy to include into other process pipelines. The decision to deploy the handler as a web service has also been fostered by the CLARIN initiative, which aims to facilitate access to NLP resources for

[1] IULA Corpus Tècnic (http://www.iula.upf.edu/corpus/corpus.htm)
[2] Institut Universitari de Lingüística Aplicada (http://www.iula.upf.edu/)

professionals in the fields of humanities and social sciences[3].

The paper offers an overview of the state of the art of current text handling applications (2) and describes some significant implementation details (3 and 4). After covering language-specific aspects of the implementation (5), evaluation (6) and benchmarking (7), the deployed Web Service for the text handler is presented (8). The WS data interface is explained, followed by the rationale for the parameters (8.1, 8.2 and 8.3) and a proposal for an interoperability-oriented common interface for general text handlers (8.4). Lastly, the conclusions for both the Web Service and the tool itself are exposed (9) along with the further work (10).

2. State of the art of text handling

Simple solutions are proposed through Perl modules or gawk scripts, usually focused on English requirements (cf. the *Lingua-EN* modules available at CPAN, among others). A fully trainable, stochastic system for text segmentation is presented in Reynar et al. (1997). Although the authors claim that it may be trained for any Roman-alphabet language, it has apparently only been tested for English. The only resource used for training is a corpus of about 40 thousand sentences from the Wall Street Journal manually corrected for punctuation and sentence boundaries. It claims an accuracy of about 98%.
General purpose packages provide some modules for text handling. *Freeling*, a well known tool for multilingual text processing, provides rule-based tokenizer and splitter modules. Capitalization is the basic clue for NE detection, although, recently, a machine learning algorithm has been added; it requires training. Other known packages like *JULIE NLP Toolsuite* or *NLTK* also include modules for these kinds of tasks. In the first case, there is an intensive use of machine learning techniques, while, in the second one, more linguistics-based techniques are used.
Also *GATE*, a well known architecture for text processing based on language processing modules, includes resources for creating most of the functionalities described in this paper, mostly for English but adaptable to other languages. The focus –or restriction– on English is a major issue for most text handlers, like, for instance, the commercial software *NLProcessor*.

3. Project overview

The two main concerns of the project are:

- Providing reliable text handling for Catalan, English and Spanish, paying special attention to the fact that Catalan is an under-resourced language and tools that are specialized or general enough are not available.
- Reducing the tagging errors of the subsequent POS-tagger by recognizing multiword Named Entities, non-analyzable items like URLs, etc.

[3] CLARIN (Common Language Resources and Technologies) : http://www.clarin.eu/external/

Conceptually, the text-handling system is rule-based, i.e., it neither takes any decisions based on stochastic methods nor depends on a previous machine-learning process, which would be cumbersome to implement, given the lack of pre-existing, structurally tagged text. The program is organized in fully independent modules that allow interchangeability when necessary.

In order to improve the results and increase configurability, the system depends on a series of resources for any of the analysis languages:

- A grammatical phrase list (*vid*. 4.4)
- A foreign expression list (*vid*. 4.4)
- A follow-up abbreviation list (*vid* 4.1)
- A word-form lexical database, which is the same lexicon employed by the POS-tagger (*vid*. 4.1)
- A stoplist, which is used to increase efficiency by significantly reducing the required time to indicate whether a given token is a word in the analysis language or not (*vid*. 4.1).

The rule of thumb of the developing process has primed the generality of rules, trying to leave the solution to language-dependent phenomena to lexicon checking. This has not always been possible (vid. 5.2), but none of the aforementioned resources is a necessary requisite for the system, although each of them contributes to the quality of the results. A processing system for Italian, for instance, could be started by merely providing a stoplist, which is the most useful resource, as well as the easiest to obtain.

4. Functionalities

As seen in the introduction, the text handler is a rule-based system that uses a lexicon to complement its set of rules to get around the general and language-specific problems that a system like this can attempt to solve without using deeper linguistic analysis.

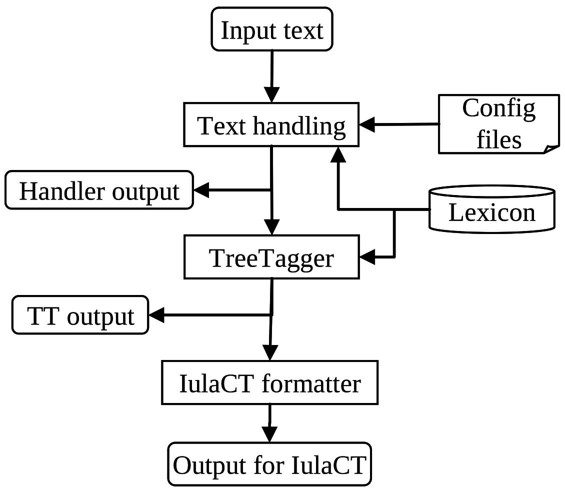


Figure 1: IULA processing pipeline.

The figure above shows the IULA processing pipeline, in which the text handler is inserted. *InputText* is the text to be processed, which can have been manually revised. The other rounded boxes describe the provided output formats for the text handler, TreeTagger and format adapter for the IULACT, although this paper only covers the details of the first stage of the process, namely the text handler.

4.1 Sentence-boundary detection

Sentence-boundary detection is arguably the most important task for any text handler that processes free text. When sentences are detected, they are marked as <s>…</s>. Periods are the most ambiguous typographic element, since they can be indicators of abbreviations, sentence boundaries, ordinal numbers, etc. A canonical sentence boundary is a period, followed by blank space (or a newline) plus a capitalized word. For instance, "*...by Europe. After...*" must be tagged as "*...by Europe.</s><s>After...*". Although a sequence like "*...Dr. Smith...*" looks superficially the same, it does not have a sentence boundary within.

To address this problem and others of similar nature, a number of strategies have been implemented. A follow-up abbreviation list is used to avoid false positives in sentence splitting. After a token like "*Dr.*" or "*Mrs.*", the system will accumulate the next capitalized word, instead of generating a new sentence tag.

A similar problem appears with abbreviated names like "*Francis S. Fitzgerald*", which have no surface difference from "*Vitamin A. However...*". The word after the period ("*Fitzgerald*" or "*However*") is searched in the word-form database. If the word appears on the list, it is not considered a possible part of a name, and the previous period is treated as a sentence boundary. The system would detect the following sentence boundary: "*Vitamin A.</s><s> However...*" but would maintain "*Francis S. Fitzgerald*" as a full proper name. This heuristics fails for last names that are actual words of the analysis language (e.g. "*J. R. Black*"), but has nonetheless given good results.

Since this text handling is conceived as a general tool and not as corpus-specific, certain heuristics like average-sentence length are not used.

4.2 General structure-marking

Paragraphs are tagged as <p>…</p> and formed by groups of sentences, each group ending with at least one blank line. Lists are marked as <list> and are groups of items (<item>), each item beginning with a label like "-" or "a)" or "1.2.3.b", which appears as an attribute of the <item> tag. Items share the hierarchy of a paragraph and contain sentences.

Titles are marked as <head> and cannot contain more than one sentence. They never end in a period, but might end in punctuation marks like "?" or "!".

4.3 Non-analyzable element recognition

Strings that match the regular expressions for URLs, email addresses and IP addresses are tagged as non-analyzable elements (<na>…</na>).

4.4 Phrase and loanword recognition

Given a list of grammaticalized expressions such as adverbial phrases or complex prepositions of the chosen language, strings that match them are tagged as <loc pos="X">...</loc>. This way, a segment like "*I found them in front of the house*" would be marked as "*I found them <loc pos="P">in front of </loc> the house*".

Assigning a single POS to a multiword expression spares us from a more complex analysis in the following NLP steps and significantly reduces the noise of our POS-tagger in the following linguistic analysis stages.

Using a different phrase list as input for the same algorithm allows the system to tag foreign expressions with a <foreign lang="XX">...</foreign> tag. The most extensive foreign expression list is for Latin idioms found in technical literature such as "*ad libitum*" or "*ex nihilo*", or abbreviated Latin expressions like "*vid.*" or "*ibid.*", although other language lists are also available.

4.5 Date recognition

Full or partial dates are tagged and translated into their ISO8601 representation. A date like "May 2nd, 1872" is tagged as "*<date ISO8601='1872-05-02'> May 2nd, de 1872</date>*". Several formats have been foreseen, from strictly numerical to partial dates like "*second of June*".

4.6 Number recognition

Both Arabic and Roman numerals are recognized. An Arabic representation for each Roman numeral is provided as a tag attribute. To minimize overlapping with Named Entities, one-digit Roman numerals are not marked to avoid overtagging structures like "*X rays*". Numbers expressed as words (or expected combinations of numbers and words) are also tagged, like "*one hundred twenty-two*" or "*78 millions*".

4.7 Named Entity (NE) recognition

Capitalization is the basic clue for NE detection, as a named entity is generally defined as a chain of one or more capitalized tokens, possibly connected by a small set of joining elements such as ampersands or prepositions, like "*Piero della Francesca*" or "*Jameson & Johnson*". The major issue in this module is the tagging of a possible NE at the beginning of a sentence, which will always be capitalized, and is therefore ambiguous. In order not to overtag, a series of heuristics deal with this phenomenon.

NE type recognition

Any multiword NE can be headed or tailed by a type-defining word, like "*Anson County Hospital*" or "*Ministry of Foreign Affairs*", which are tagged as <name

type="organization">…</name>. Abbreviations can also precede named entities and pinpoint their type, like "*Dr. Smith*" (<name type="person">…</name>). The system uses a list of those elements that are considered unmistakable triggers for these types of named entities, particularly persons (generally triggered by abbreviated titles of address), organizations and locations.

Named Entity	Type
Saint Matthew	person
<abbr>St.</abbr> Matthew	person
Saint Matthew's Hospital	organization
Ministry of Agriculture	organization
Sunset Boulevard	location

Table 1: Cases of NE type detection

Acronym expansion

A technical text very often presents segments of text which contain definitions for acronyms, "*Computed tomography (CT) is a medical imaging method...*". When a parenthetical acronym is found right after its full name, its value is stored and added as an attribute to all the <name> tags containing the acronym. A segment like "CT was useful" would be tagged as "*<name expansion="computed tomography">CT</name> was useful"*, provided that a definition context like the previously mentioned one is found within the input text.

This is used to provide finer information to the automated summary or terminology extraction software, enabling the relation of occurrences of a term in both complete and abbreviated form.

4.8 Pre-tagged input

The text handler has been conceived to minimize or avoid end-user annotation or segmentation, but sometimes it may happen that the text obtained is already segmented or that the user wishes to segment the text by himself to take care of text peculiarities. Moreover, a user can also tag the input texts to indicate some information that the system would not be able to provide, e.g. tagging a large citation in a foreign language with a <foreing lang=XX>...</foreign> to make sure that it does not interfere with the handler. The input parameters *SegmentedInput* and *Keeptags*, respectively, allow the system to process the input in the desired manner, and are further explained in 8.2.

5. Language-specific issues

Each language of analysis presents a series of typographical phenomena that can be problematic and had to be addressed. The main problem that any typographical phenomenon poses for our system is the risk that, by not recognizing a token as a word (or combination thereof) of the language, a false sentence boundary or named entity might be set.

5.1 Catalan

The particular phenomena for Catalan are:

- Hyphenated verbal clitics: Verb forms in Catalan can appear with a large list of pronominal clitics, which means that verbal forms are very numerous. These clitics, however, are written hyphenated, as in "*Trobant-la*" (En. "*Finding her*"), which means that they can be easily decomposed and analyzed to determine whether their root (in our case "*Trobant*") appears in the lexicon or not.
- Apostrophes: Catalan also shows grammatical word contraction, like preposition contraction in "*D'avui*" (En. "*of today*"), which is the contraction of "*De avui*". The strategy to separate clitics from the verbal root is also employed to determine whether the analyzed token is a word of the language, since "*d'avui*" should be considered as such and, therefore, separated, but "*D'Alembert"* should not, and must be kept together as a single token.

5.2 English

The particular phenomena for English are:

- Saxon genitive: Without any particular treatment for Saxon genitives, a string like "*Karl Marx's birthday*" would be tagged as "*<name>Karl Marx's</name> birthday*", which is not a desired output. Saxon genitives (and contracted forms of the verb forms "*is*" or "*has*", as well) are separated from the names they are attached to, for results like"*<name>Karl Marx</name>'s birthday*".
- Hyphenated nouns: Most hyphenated words will not be present in the lexicon and can be partially mistagged as <name>. Hyphenated words are split or kept together depending on the capitalization of their second element.
- Capitalized denonyms: In English, denonyms, language names and some denominal adjectives are written capitalized ("*Englishman, Spanish, Shakespearian*"), although they are not named entities. To solve that, an exclusion list is kept to avoid mistagging tokens like "*Texan*" or "*Oxonians*" with a <name> tag. The list currently ranks 12,000 entries, both singular and plural. The system only excludes perfect matches on the lists, which means that multiword named entities that include a token from the exclusion list will still be tagged, like "*Spanish Foreign Office*".

5.3 Spanish

The particular phenomenon for Spanish is the problem posed by non-hyphenated verbal clitics. For an infinitive verb like "*dar*" (En. "*to give*"), there are 32 forms with pronominal clitics (*"darme", "dármelo", "dármela", "dármelas"*, etc.), but none of them are hyphenated and they cannot be separated without making mistakes. The appearance of an acute accent in some forms makes a brute-force stemming difficult and, as per now, Spanish verbal forms with clitics must be kept in the lexicon to determine if they belong to the language.

6. Evaluation

The output results of the current text handler have been tested against a previous hand-tagged collection of texts used as a gold standard. The DTDs of the gold standard and the handling output are slightly different, and the comparisons are an estimate — e.g. in the gold standard, the text inside <item></item> does not need to be enclosed between <s></s> sentence markers, whereas they are mandatory in the handling output, although both DTDs are CES-compatible. The first set of texts is a 60,000-word press corpus, and the second one is a collection of eleven specialized articles on genomics, ranking 38,000 words.

Tag	Press	Genomics
Sentence	99.39%	91.55%
Head	82.00%	92.20%
Paragraph	97.60%	97.11%
Name	95.43%	99.76%
Name_0	76.85%	85.00%

Table 2: Accuracy rates

The *Sentence, Head, Paragraph* and *Name* rows show the accuracy for the corresponding <s>, <head>, <p> and <name> tags, whereas *$Name_0$* shows the performance of the system when tagging a possible NE that appears in the first-token position of a sentence, thus showing ambiguous capitalization.

The sentence boundary detection performs significantly better for the press corpus, which is understandable because scientific literature shows formulae and other non-expected elements that the system does not cope with. The downside of press text is its name richness, which becomes apparent on the lower accuracy for *$Name_0$* in the press corpus. The variety of proper names is much larger in press than in natural sciences.

7. Benchmarking

A representative value of number of words for the three most common document types of corpus input – press note, article, and full newspaper – has been used to benchmark the throughput of the handler.

As seen in the Table 3, the duration of execution for the three longest files is very different depending on the language. The main cause is that the phrase and loanword list for Spanish (*vid*. 4.4) is the most complete of the three, whereas the list for English is the shortest one. The phrase recognition module has an execution time which is proportional to the length of its phrase list times the size of the input file.

Language	Words	Time in seconds
Catalan	300	4
	3,000	36
	30,000	181
English	300	1
	3,000	4
	30,000	76
Spanish	300	2
	3,000	14
	3,0000	291

Table 3: Execution times

8. Web Service

The need to publish the text handler as a Web Service arises when the following issues are considered:

- The deploying of the tool on a computer may be not trivial for some users, as certain versions of Perl or DB clients will not work.
- Keeping a single deployed Web Service up-to-date guarantees the users that the code being executed will always be the last available version.
- Access control to the DBs can be regulated in an easier manner, since local users will not have to store any configuration files with users and passwords.

Once the handler has been published for internal network access, allowing external access seems the best idea, given the current emphasis on common language resources and distributed processing.

The most common pipelines in our system are:

- Handling + POS-Tagging + Loading into the IULA Corpus
- Handling + POS-Tagging + Loading into SketchEngine
- Handling + Terminology extraction (Cabré et al., 2001)
- Handling + Automatic Summarization (Da Cunha et al., 2009)

When deploying the Text Handler as a web service, there are a number of decisions that will affect its potential interoperability. Typically, a text handler is a link in a longer chain or pipeline. Most input or output values can be represented as strings, but a string is a basic type without any service semantics to it. Therefore, we need to specify our WSDL with a higher-level typing than simply a string, to enhance interoperability and make integration easier.

A common interface approach has been designed to guarantee interoperability and service usability. Inputs and outputs have to be typed so that they can be understood by potential users. Whenever possible, *ISOcat* standards have been used for parameter naming and typing.

The parameters that have been deemed implementation-specific have been made optional in the WSDL, which leaves the small set of *Language* and *InputText* as shared, general parameters that must be provided for a WS invocation to work.

Parameter Name	Description
Language	Input language. Currently supports text in Catalan, English or Spanish.
InputText	The text to be processed.
[AnnotationFormat]	Sets output format. Can be "Verticalized", "TreeTagger", "IULACT" or "XmlTag" (default).
[InputMIMEType]	Input text encoding, defaults to UTF-8.
[OutputMIMEType]	Input text encoding, defaults to UTF-8.
[Keeptags]	Keeps tags in previously tagged input text. Defaults to "false".
[Tagset]	Chooses tagset from the available list. Defaults to the IULA tagset.
[HtmlEntities]	Encodes entities into characters. Defaults to "false".
[Filename]	Filename for output header. Defaults to a Timestamp.
[SegmentedInput]	Indicates the system that the input is already segmented with a sentence in each line.

Table 4: Web Service input. Bracketed parameters are optional.

8.1 Mandatory parameters

The only mandatory parameters are *Language,* which can be Catalan ("ca"), English ("en") or Spanish ("es"), and *InputText,* which contains the text that must be processed.

8.2 Optional parameters

Optional parameters allow more precise use of the system, which is not assumed in the core functionalities.

- The *InputMIMEType* and *OutputMIMEType* parameters are fairly straightforward and allow the user to determine the encoding that the *InputText* or *OutputText* have.
- *AnnotationFormat* determines the type of output provided and is further explained in the next section.
- *Keeptags* is a boolean flag that allows the system to keep any Xml tag that the input might contain, or to remove them if set to "false". It can be used to clean Html text or, on the contrary, to preserve any other annotation that the text previously had.
- *HtmlEntities* set to "true" takes Html character entities like "*&agrave;*" and converts them to their corresponding character value ("*à*", in this case). This parameter is useful to automatically clean Html files, along with *Keeptags* set to "false".
- *SegmentedInput* prevents the system from running the sentence boundary detector and respects the sentence-per-line input indicated by the user.

8.3 Output

For interoperability reasons, we have devoted special efforts to accommodate the Text Handler to the requirements of current and potential providers. All of the used formats or the processing stages in studied pipelines have been listed and grouped as possible values for the optional parameter *AnnotationFormat.*

Again, our approach has tried to abide by standards as much as possible, and the CES-compliant format has been considered the default output option. Current practices, however, require that we include additional output formats such as the *de facto* standard verticalized format, in order to cover a wider array links between our handler and other text-consuming applications.

AnnotationFormat	Description
XmlTag	SGML CES-compliant Xml Tagging.
SentencePerLine	No Xml Tagging, every sentence or head takes one line in the output format.
Verticalized	Xml tagging, one token per line.
TreeTagger	As above, with specific format for TreeTagger input.

Table 5: Values for the *AnnotationFormat* parameter.

XmlTag:
This value for *AnnotationFormat* generates an Xml string which is SGML CES-compliant.

OneSentencePerLine:
This value for *AnnotationFormat* outputs plain texts without tags, each line of the text being what the system considers a sentence or head.

Verticalized
This value for *AnnotationFormat* is the general, all-purpose format for tokenized –also known as verticalized– text, in which each token or tag takes a single line in the text. This is the typical input for most POS-Taggers.

TreeTagger
This value for *AnnotationFormat* provides as output the results of the POS-tagger *TreeTagger*. POS-tagging is the next step after tokenization, and this output value includes the previous step (*Verticalized*).The results are tokenized, with each token or tag taking one single line in the output text.

IULACT

This value for *AnnotationFormat* gives the POS-Tagged input text (see previous step) that has been transformed into the input format of the IULACT.

8.4 WSDL

Interoperability and reusability best practices require that we define services in a modular fashion. Thus, we distinguish between type definitions, message definitions and binding, as set by the WSDL consensus. Type definitions are designed to enable type sharing and service level semantics above basic types. Separating Messages from Binding will allow multiple service bindings to the same message.

Table 6 shows the operation in the WSDL, in which input and output messages are assigned the corresponding type.

Operation
<operation name="TextHandler"> <input message="typens:InputHandler"/> <output message=" typens:OutputHandler "/> </operation>

Table 6: WSDL excerpt

The *InputHandler* type defined in Table 7 distinguishes between base input parameters and optional parameters. The first ones are grouped into the *BaseHandlerIO* type and are compulsory. The second ones are collapsed into the *OptionalHandlerIO*.

Input
<complexType name=" InputHandler "> <all> <element name=”base” type=”**xsd:BaseHandlerIO**”> <element name=”optional” type**=”xsd:OptionalHandlerIO**”> </all> </complexType>
xsd: BaseHandlerIO
<element name="Language" type="xsd:Language-ISO639-1"/> <element name="TextFile" type="xsi:TextFile" />
xsd: OptionalHandlerIO
<element name="OutMode" type="string" minOccurs="0" /> <element name="InputEncoding" type="string" minOccurs="0"/> <element name="OutputEncoding" type="string" minOccurs="0"/> <element name="Filename" type="string" minOccurs="0"/> <element name="Keeptags" type="boolean" minOccurs="0"/> <element name="HtmlEntities" type="boolean" minOccurs="0"/>

Table 7: WSDL excerpt

The shown typing reflects the attempt to provide comprehensive, standardised and interoperable descriptions for text handler-like service. This first approach proposes a simple way of encapsulating implementation-dependent parameters and separating them from base and (hopefully) uncontroversial input parameters.

Higher service-semantic typing for base parameters will allow common type descriptors and type sharing between services. It is reasonable to expect that these parameters will be assigned some type from a central Namespace registry.

The type for *Language* is also a general trait that can be standardized. The system uses the two-letter standard ISO 639-1 (“ca”, “en” and “es”), but it could be modified to accept the three-letter versions after ISO 639-3. Language format specification is also a significant aspect for the setting of a common interface.

The type for language is *xsi:TextFile*, which can either contain its data in a string value or point to an URI. This enables the system to process an input string or read the text from a source in the Web.

Output
<complexType name="OutputHandler "> <all> <element name=”base” type=”**xsd: BaseHandlerIO**”> <element name=”optional” type=”**xsd: OptionalHandlerIO**”> </all> </complexType>
xsd: BaseHandlerIO
<element name="Lang" type="xsd:Language-ISO639-1"/> <element name="Text" type=" xsi:TextFile " />
xsd: OptionalHandlerIO
<element name="OutMode" type="string" minOccurs="0" /> <element name="InputEncoding" type="string" minOccurs="0"/> <element name="OutputEncoding" type="string" minOccurs="0"/> <element name="Filename" type="string" minOccurs="0"/> <element name="Keeptags" type="boolean" minOccurs="0"/> <element name="HtmlEntities" type="boolean" minOccurs="0"/>

Table 8: WSDL excerpt for output parameters

Optional parameters are no further typed, but rather they are collected under a ‘local’ type (very often, a string) and can be assigned a default value.

Note that this strategy allows decoupling complex command line parameters from WSDL interfaces. We can change or add new parameters and never impact the full WSDL description.

The type defined in the tables above as *BaseHandlerIO* is a straightforward tuple of language and text, which can be either input or output (ideally both, when seen as a part of a process pipeline).

Any Web Service that does any form of text handling or preprocessing will have this tuple as input. This minimal data structure can be proposed as the base Input/Output unit for any application of this kind.

The basic data types are shared between input and output operations, which allow us to have a single type (*BaseHandlerIO*) for both. This is desirable, due to the fact that an HTL process pipeline chains its outputs as inputs for the subsequent modules.

9. Conclusions

Text Handler

We consider that the basic aim has been fully reached, as any text in the target languages may be successfully processed with a minimal effort. Also some additional information may be manually added if the users consider it is necessary.

The benefit of each of the lexical resources is not the same, the stoplist being both the most useful (as it avoids the largest amount of false positives on $Name_0$ recognition) and the easiest to obtain. Phrase and loanword recognition can become a processing bottleneck. Therefore, an optional parameter could be added to switch this module off, if only sentence boundaries are desired.

The system has been developed with a general-purpose scope in mind, but the nature of the input texts naturally biases the quality of the possible output. More canonical (e.g. press) texts will provide cleaner segmentations than texts with more typographical variety, like scientific writing.

Web Service

The importance and usefulness of minimal, common interfaces has been assessed. Providing Web Service data interfaces that separate their general, domain-specific input/output values from those that are set by the implementation allows us to build processing pipelines in an easier manner.

10. Further work

The further work consists in the subsequent deployment of the other stages of the IULA's processing pipeline as Web Services, as well as expanding the number of languages that the text handler can accept.
The work on separable clitics and the like has already been implemented, which means that including new languages in the processing system would require the addition of the lexical resources described in 3 and little or no coding for languages like Italian, French or Dutch. German, however, would require a completely different Named Entity recognition strategy, since all nouns are capitalized.

11. Acknowledgements

This project has been made possible by the funding of the CLARIN project. The CLARIN project in Spain is co-funded by the 7FP of the EU (FP7-INFRASTRUCTURES-2007-1-212230) and the Spanish *Ministerio de Educación y Ciencia* (CAC-2007-23) and *Ministerio de Ciencia e Innovación* (ICTS-2008-11). Furthermore, the *Departament d'Innovació, Universitats i Empresa* of the *Generalitat de Catalunya* has funded the development of a demonstrator that guarantees the integration of the Catalan language in CLARIN.

12. References

Cabré, M.T.; Estopà, R.; Vivaldi, J. (2001) Automatic term detection: A review of current systems. In *Recent Advances in Computational Terminology,* John Benjamins, Amsterdam :

Carreras X, Màrquez L. and Padró L. (2002). Named Entity Extraction Using AdaBoost. In *Proccedings of CoNLL 2002. Shared Task Contribution.* Taipei, Taiwan

Clough, P. D. (2001). A Perl program for sentence splitting using rules. University of Sheffield:

Corpus Encoding Standard : *http://www.cs.vassar.edu/CES/*

DaCunha, Iria; Torres-Moreno, Juan-Manuel; Velazquez-Morales, Patricia; Vivaldi, Jorge (2009). "Un algoritmo lingüístico-estadístico para resumen automático de textos especializados" en Linguamática 2. Pàg. 67-79. ISSN 1647-0818 http://www.linguamatica.com/index.php/linguamatica/issue/view/2

Freeling*: http://www.lsi.upc.edu/~nlp/freeling/*

GATE: *http://gate.ac.uk/*

IULA - Institut de Lingüística Aplicada : *http://www.iula.upf.edu/*

IULACT*: http://www.iula.upf.edu/corpus/corpuses.htm*

ISO-639-*1 :* http://www.infoterm.info/standardization/iso_639_1_2002.php

ISOcat *:* http://www.isocat.org/

JULIE NLP Toolsuite: *http://www.julielab.de/Resources/Software/NLP_Tools.html*

NLTK: *http://www.nltk.org*

Padró M. and Padró L. A Named Entity Recognition system based on a finite Automata, Acquisition Algorithm, *Universidad Politécnica de Catalunya*.

Reynar J. C. and Ratnaparkhi A.(1997). A Maximum Entropy Approach to Identifying Sentence Boundaries. In *Proceedings of the fifth conference on Applied natural language processing*, NY. USA : 16–19

Sketch Engine: http://www.sketchengine.co.uk/

TreeTagger: http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

Vivaldi, J. (2009). "Corpus and exploitation tool: IULACT and bwanaNet*", CILC-2009*, Murcia, Spain.

A Generic Chaining Algorithm for NLP Webservices

Volker Boehlke
NLP Group – Department of Computer Science – University of Leipzig
Universität Leipzig, Institut für Informatik, PF 100920, 04009 Leipzig, Germany
E-mail: BoehlkeV@informatik.uni-leipzig.de

Abstract

This paper will point out why chaining of webservices can be compared to the type checking process used in most compilers for common programming languages. Based on that point of view, it will be discussed, how the specification of input parameters and output values of textual NLP webservices should look like, in order to allow the implementation of a generic chaining algorithm. The concept and implementation of an automatic chain builder based on this algorithm is discussed. Furtheron the general design of a simple infrastructure is described. This infrastructure allows users unaware of the underlying specifications and algorithms to build custom process chains out of elements implemented as webservices. The DSpin[1] prototype, one of the german contributions to the Clarin[2] initiative, is a basic proof of concept implementation for the most important parts of this infrastructure. The paper will give a quick overview of the current state of this prototype. Some of the available services, tools and applications will be described and additionally the limits of the current implementation will be discussed.

1. Introduction

1.1 Motivation

More than ever the scientific world today is a globalized world. People not only travel into distant countries to give lectures or to study, but they work together every day by using the communication services provided by the world wide web. Technology closes geographical gaps and allows highly specialized scientists to work in teams distributed all over the world. Communication technology has become a driving force for collaboration and competition not only in the scientific community.

But sharing information, thoughts and results is only the first step towards a web based science community. Along comes the need to share data and algorithms, resources and tools. In a world where communication is cheap and information is available on a world wide scale, the need to reuse the work of others, to base work on resources provided by an institution situated in a distant country, to creatively combine tools implemented for a different purpose in order to solve problems becomes more and more important.

In the IT world, more exactly in software technology, this problem is known since several decades. Reusing once implemented code again and again was one of the first challenges being tackled. Starting from small, module based systems the development quickly went on to libraries providing highly optimized functionality accessible through easy to use API's and further on to component based systems especially designed to support the dynamic composition of complex applications out of reusable, versioned and loosely coupled components.

When it comes to the construction of systems based on access to distributed components, giving access to resources and tools, SOA's, service oriented architectures, are currently the common technology to use. The question on how the different services and components available in such an infrastructure can be orchestrated by unexperienced users is a very difficult one. This is because it involves the knowledge on how these single services work, which data formats are used to represent the data being computed and how the dependencies of these different services to each other look like from a semantical point of view. If it is intended to not concern users with these topics, an automatic system assisting the user while orchestrating available services in order to solve the problem at hand is needed.

In this paper we will concentrate on a generic chaining algorithm and its ramifications on a simple infrastructure utilizing this algorithm in a basic workflow system. It is not intended to build a fully fleshed SOA or to describe the wiring of this algorithm to established, professional workflow tools. But the experiences and facts described in this paper will hopefully help in the implementation of such an infrastructure and to evaluate possible candidate components and solutions in the future.

1.2 Relation to previous work

When it comes to the creation of workflows and the chaining of webservices the focus very often is put on

[1]Deutsche Sprachressourcen Infrastruktur; The German complement to Clarin. See *http://www.d-spin.org/*
[2]Common Language Resources and Technology Infrastructure. See *http://www.clarin.eu/*

tools allowing to define business processes in a graphical user interface and standards designed to describe those workflows. But the problem not only is to define the order and dependencies of services in a workflow, but to close the structural and semantic gaps between those possibly heterogeneous services forming the workflow. In [6] Cardoso and Sheth point out, that not only the structure of the data being computed by those services, but also the understanding on what a certain piece of information really means can differ. As a solution they propose a system that uses ontology based schema integration in order to discover services and to overcome the semantic and syntactic differences when being integrated into a workflow. The same issue is addressed by Shiyong, Bernstein and Lewis in [7] from a more theoretical point of view. Their approach on chaining and automatic workflow generation is in several aspects, especially concerning the concept of preconditions and postconditions, similar to the one described here.

In this paper the generic chaining of NLP webservices will be discussed from a practical point of view. A basic chaining algorithm suitable for most of the tasks at hand is introduced and its usage as part of an automatic chain builder will be discussed. Additionally a limited infrastructure supporting the functionality and making use of both algorithms is described.

2. Chaining

2.1 Chaining of NLP webservices

Chaining of webservices can be compared to the nesting of functions in programming languages. The result of function f_a is used as a parameter of function f_b, its result is consumed by a function f_c and so on: $f_c(f_b(f_a))$.

If such a “workflow” is defined by code in a programming language, a type checker is used to determine whether the return type of f_a matches the input parameter definition of f_b. Note that this check usually is done during compile-time, but is possible and sometimes also done on run-time.

If we look at chaining of webservices from this point of view, chains can be build based on formal specifications of “input parameters” and “return types” of the webservices intended to be chained. This has to be done on build-time of the chain, because the amount of data being computed and the amount of time and resources needed to do that can be very high. In order to make sure a specified chain is valid, we need to use a type checker on the parameters and return types of those webservices forming the chain. But what are these types?

In programming languages some basic predefined types are usually present, while new types can be built out of these basic building blocks. Common basic building blocks are strings, representations of numbers and a set type. In NLP possible candidates for these basic building blocks are text, tokens or POS-tags. Just like a number may be represented in different ways (low/big endian, integer/floating point), this kind of data can be encoded using different standards too. For text this may be iso-latin1 or utf8 and for POS-tags these are many different not necessarily comparable tag sets.

Another aspect that needs to be addressed is the format issue. If we plan on interchanging and annotating text using webservices, the data we send to these services and the data that is produced should be based on established standards, like those provided by the Text Encoding Initiative, TEI[3]. On an abstract level this means an NLP webservice for text produces and possibly consumes:

1) A document formatted according to a defined standard, for example TEI P5 or DSpin-TextCorpus[4]

2) A document containing a certain set of types of information, for example POS-tags,...

3) … which are either just present or also encoded using a specified standard, for example STTS[5] for POS-tagging in german.

By definition this also means that our services are working document based. The data send to a service consists of only one document which holds all the information this service will work on. Of course additional data will sometimes be used in the background in order to implement the functionality of the service. Additionally we specify, that a service just adds, but does not remove any information from the input document it is invoked with.

Summing this up, a complete service description from the point of view of a chaining algorithm consists of two identically structured specifications. An input and an output specification. These specifications consist of a format that is used to represent the data and a set of pairs of parameter-types and standards definitions for these types. The input specification represents the information that needs to be present in the input document, while the output specifications defines which kind of information is produced by the service and is additionally present in the output document. Table 1 and 2 show examples for the input- and output specification of a POS-tagger webservice, that consumes documents according to the TextCorpus-Format, which contains german text that was split into tokens. It produces a TextCorpus document additionally containing POS-tags based on the STTS-tagset. A graphical representation of the invocation

[3]An organization which maintains a format for digital text representation. See *http://www.tei-c.org/index.xml*
[4]A corpus representation format for linguistic webservices used in DSpin. See [5] for details.
[5]Stuttgart Tübingen Tagset. See *http://www.sfb441.uni-tuebingen.de/a5/codii/info-stts-en.xhtml*

of this service is shown in Figure 1.

Format	TextCorpus

Input	text = utf8
	language = german
	tokens = present

Table 1: Example input specification for a POS-tagger webservice.

Format	TextCorpus
Output	POS-tags = STTS

Table 2: Example output specification for a POS-tagger webservice

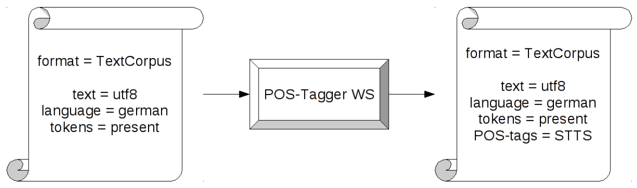


Figure 1: invocation of a POS-tagger webservice

2.2 A generic chaining algorithm

If the input- and output specification of webservices is structured as described above, a generic chaining algorithm can be implemented. Based on the information extracted from these specifications the chaining algorithm is able to determine whether a service WS_n can be executed after a chain[6] of other services $\{WS_1,...,WS_{n-1}\}$, by checking the following constraints:

1) The format specified in the output specification of the previously run service WS_{n-1} has to be equal (which means the exactly same format is used) to the format specified in the input specification of service WS_n.

2) All parameter-type/standard pairs out of the input-specification of WS_n need to be present in the set of pairs produced by the previously executed services in the chain $\{WS_1,...,WS_{n-1}\}$.

If both checks come up with a positive result, WS_n is compatible and can be added to the end of the chain $\{WS_1,...,WS_{n-1}\}$. While the first constraint appears rather simple, the second in detail proves to be the more complex one. In simple cases the algorithm just has to accumulate the parameter-type/standard pairs of all services in the chain in order to match those against the set of needed inputs. Figure 2 demonstrates the construction of a simple chain based on this algorithm.

[6]this also includes chains consisting of only one service, for example a data service at the begin of a chain about to be build

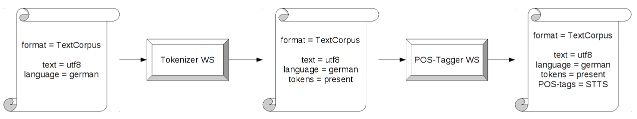


Figure 2: simple chain

But there is one exception to this simple process. If a service converts from one format to another, it might not be possible to represent a certain type of information in the output format, that was present in the input. One cause may be, that the output format simply does not support this kind of information. Another possible cause is, that in the input format the information is expressed in a more or in a less detailed way, than it is possible in the output format. Therefore we would need to remove some parameter-type/standard pairs and add other ones to the set of available pairs. This breaks the boundaries of what can be expressed based on the currently simple input and output specifications.

Therefore we may define that in case the input format of a service differs from the format specified in the output, all produced parameter-type/standard pairs have to be specified. A better and more complex solution is, to allow the definition of If-clauses like "*If this set of parameter-type/standard pairs is present in the input, the following set of pairs will be present in the output*" in the output specification. All needed information is still available during "build time" of a chain. If-clauses should be allowed not only for converter services, but in all service specifications because this drastically widens the expressiveness of service specifications. In the prototype implementation of the chaining algorithm we choose to go for the first basic solution without If-clauses in order to keep things simple.

The chaining algorithm is limited to the knowledge available at "build time" of the chain and therefore to the input-/output specifications of the services. There is no way to foresee run time errors that might occur when executing the chain. The algorithm as it is described above works independent from the data that is computed. It is also independent from the way formats and the input-/output parameter-type/standard pairs are specified. But there has to be an agreement on how to reference a certain format, parameter-types and standards. Section 3.1 will give more information on how this may be done in the NLP domain in a standardized way.

2.3 An automatic chain builder

The chaining algorithm discussed in the previous section may be used as one of the core components of an automatic chain builder. Given a repository of services we want to build chains from a starting point to an end point. The starting- and end-points are specified by services. Given a list L_{all} of services and a start-point service W_{start} and an end-point service W_{end}, both contained in L_{all}, a basic automatic chain builder algorithm looks like this:

```
1) Create an empty list of service
   chains $L_c$
2) Create a one element chain
   $C = \{W_{start}\}$
3) Copy all elements of $L_{all}$ to an
   previously empty list of
   services $L_s$
4) Remove all services from $L_s$ that
   are not compatible to the last
   entry in $C$, according to the
   chaining algorithm.
5) Remove all services from $L_s$
   that are already present in $C$
6) For each entry $S$ in $L_s$ :
    Create a copy $C_2$ of $C$ and add
   $S$ to the end of $C_2$
   if $S = W_{end}$ :
     yes: add $C_2$ to $L_c$
     no: recursively go back to 3)
     and set $C = C_2$
```

Once this recursive algorithm stops, all possible chains consisting of services from L_{all}, from W_{start} to W_{end} will be present in L_c. If no such chain exists, the result list is empty. Because of the constraint in step 5 circles are not possible and therefore the recursive algorithm will definitely stop. If we look closely on this basic implementation of the algorithm, there are a few problems we may have to solve.

A first problem is, that there will be many chains present which are possible doublets. We define a chain C to be a doublet of chain D, if its services $\{D_1,\dots,D_n\}$ are just a reordering of the services $\{C_1,\dots,C_n\}$ in C. A simple check for doublets ignores the order of the services in chains and just evaluates, if the two sets of services being present in the according chain, do consist out of the same members. If this check is performed for the chain C_2, created in step 6) of the algorithm, against all already present chains in the List of results L_c, the performance of the algorithm can be improved significantly, because many recursions don't need to be executed. Please note that the way doublets were defined is motivated only on a syntactical but not on a semantical level. It is possible to define doublet chains that, given the same input, will result in different documents in the end. These documents might even be equally structured, which means they do contain the same type of content. But the information stored in them can still be different.

Another problem is that, given even small numbers of services, the amount of results can be very high. Even though sorting out all doublets decreases this number significantly, the number of results may be too big in order for a human user to explore. Therefore a ranking system for the chains should be introduced. A basic implementation of a ranking system can be based on the length of the chain. This simple solution proved to be helpful in the prototype infrastructure. But a shorter chain not necessarily needs to be "better" than a longer one. Aggregator services, services aggregating the functionality of several other services usually available independently from each other, will in many cases be part of shorter chains. Let's assume an aggregator service A aggregates 5 services $\{W_1,\dots,W_5\}$. A user may only need to make use of W_1 and W_2, but because A appears as a single service to the ranking system, a chain containing A, that is otherwise identical, will be higher ranked than the longer alternative containing W_1 and W_2. But A will produce information the user does not need and will potentially consume a high amount of resources while doing so.

A better implementation of the ranking algorithm should make use of user feedback or the fact that certain chains are often built and executed, probably because the users felt the given combination of services works well. This data needs to be obtained by other components of the infrastructure.

3. Infrastructure

3.1 Building a basic infrastructure that utilizes the chaining algorithm

In the previous sections we described a chaining algorithm that works on parameter-type/standard pairs, but we didn't explain how these values are defined or where they are stored and managed. We also described an automatic chain builder algorithm that in one of its steps is based on the knowledge of all available services. But it wasn't defined where this knowledge should come from. In order to clarify these questions, the basic building blocks of an infrastructure that can be build around these two algorithms and allows the maintenance of all necessary data will now be sketched out.

In a SOA the exposed functionality of all infrastructure elements is available through services. Complex operations are orchestrated out of available, reusable services. Therefore a findAllAvailableServices-function that can be used in the automatic chain builder is one of those basic services offered by at least one component of the infrastructure. The chaining algorithm which determines whether a certain services can be run after a chain of other services is another basic service. These two basic services are "orchestrated" in order to implement the complex "automatic chain builder"-process. Figure 3 shows a sketch of this simplified infrastructure.

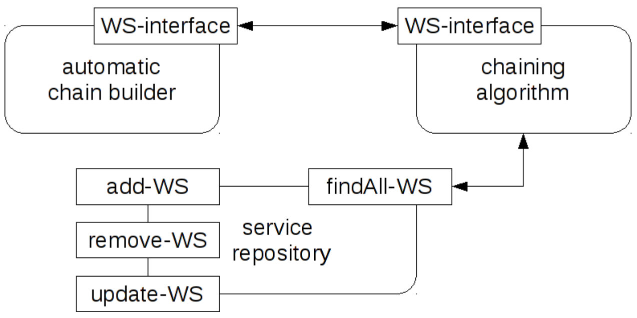


Figure 3: simplified infrastructure

All available services descriptions are stored in a service repository. This repository offers methods, which allow to add, update and remove services that may be used as part of a chain. It also exposes a findAll-method which returns the service description of all available services in the repository. This method is used by the "automatic chain builder"-component which itself is based on another service implementing the chaining algorithm.

In this first sketch the infrastructure lacks an application layer which allows human users to comfortably interact with infrastructure services. One important application in this respect is a basic workflow builder and handler, that is used to build the process chains according to the needs and specifications of an external user. In order to fulfill this task, the workflow tool uses the repository, the chaining algorithm and the automatic chain building services and offers a graphical user interface that reflects information obtained by these services. A user may search for or browse through the available services. Depending on the problem to be solved, the user is able to select a service out of the available ones and, by using the chaining algorithm, to manually build a chain step by step. Additionally the automatic chain builder may be used after each step in order to make proposals of possible chains according to a user defined end point. The workflow tool also has the responsibility of invoking chains and handling runtime errors that might occur. The result of a chain invocation can either be passed on to an external tool or be presented directly to the user.

Another important application can be named "repository manager". A graphical user interface simplifies the access to basic repository functions in order to allow human users to add, update or remove services from the service repository. The service descriptions stored in the repository are structured according to section 2.1. Therefore an additional infrastructure component which allows to define, store and manage the basic building blocks of these service descriptions is needed. These basic building blocks are input- and output formats and parameter-type/standard pairs. A specification on which parameter-type/standard pairs are valid combinations and which of these pairs are valid in which formats has to be done in order to support the user when registering new or managing the attributes of existing services.

Since the chaining algorithm and the whole infrastructure is not based on deeper knowledge of the inner structure of the data being computed, the same rule has to apply to this component too. This new component allows the following actions:

- creation of format-, parameter-type- and standard identifiers, such as TextCorpus, POS-tags and STTS
- definition of valid parameter-type/standard pairs: for example POS-tags = STTS is valid, but tokens = STTS is not
- description of a format by referencing valid standard identifiers to it: POS-tags are a member of TextCorpus, but it is not possible to add audio data to a TextCorpus document, because the format simply doesn't support it

The task of defining a correct set of formats and parameter-type/standard pairs that holds no conceptual doublettes and reflects the actual definition of the used format as well as possible, is a very diffcult one. Therefore the data stored in this component should be based on profound work on this matter. In NLP the ISOcat DR[7] can be used in order to reference uniquely identified concepts.

Figure 4 shows a sketch of this more complete infrastructure. For simplification the task of storing format definitions was added to the already existing repository component. Additionally an application layer consisting of a worklow builder tool and a repository management tool was added. The repository management tool links to the ISOcat DR in order to allow the usage of concepts and unique identifiers when specifying a format and its members consisting of parameter-type/standard pairs.

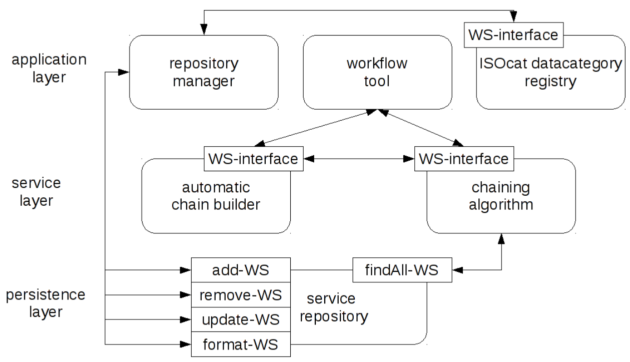


Figure 4: advanced infrastructure

3.2 The DSpin prototype infrastructure

The DSpin prototype is a partial implementation of the infrastructure that was just described. It consists of a service repository which stores its data in a MySQL

[7]ISOcat DR: ISOcat datacategory registry. A registry for data categories, which stores a unique identifier along with other metadata. See [3] and *http://www.isocat.org/*

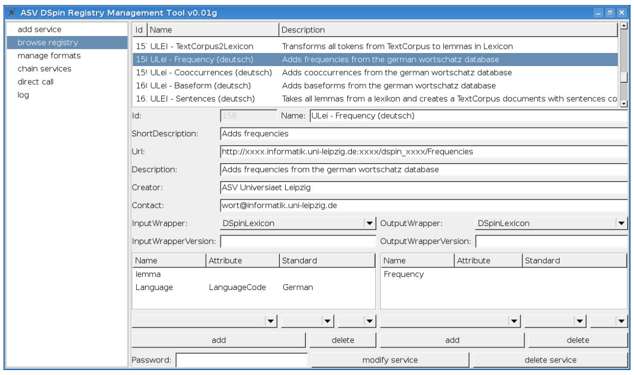
database. The chaining algorithm and the automatic chain builder webservices are currently running on the same machine. This decision was made because of performance issues due to the amount of requests the automatic chain builder currently sends to the chaining algorithm. Especially once the number of available services grows significantly, this might otherwise result in the automatic chain builder service running too slow. Although it is possible to reduce the amount of calls to the chaining algorithm service significantly by adding a cache, the automatic chain builder needs further optimization if the chaining algorithm service should run remotely in this scenario.

All services intended for chaining currently need to be implemented as REST webservices and rely on HTTP-POST in order to receive the request/input document. Infrastructure services, such as the findAllAvailableServices-function of the repository or the chaining algorithm, are implemented based on HTTP-GET using Java Servlets and the Apache Tomcat webserver. The responses of infrastructure services consist of simple XML-files containing the requested information, for example a list of services including technical metadata like service-urls, input-/output specifications and others.

On the application layer there are currently two applications available. The first one is the repository management tool which allows to manage the services available in the repository. When the DSpin prototype project started, it was unclear if the currently available data categories in the ISOcat DR were sufficent in order to describe all services intented to be added to the prototype infrastructure. Instead a proprietary set of unique identifiers, used for identification of formats, parameter types and standards, was specified and hardwired into the management application. There is no usage of the ISOcat DR or some other registry up to now. The management tool also includes a basic workflow system that makes use of the chaining service and is designed to be used for testing purposes.

The second application available is the WebLicht web interface[8], which is a web-based workflow tool. It allows human users to easily build and invoke process chains by making use of the chaining algorithm. The chaining webservice is invoked based on the services already present in the chain currently beeing specified. The results of this webservice call are offered as compatible services to the user. Step by step the user is able to build a valid process chain according to the task at hand. Currently there is a wide range of services available. These services are offered by several partners from different countries. Some of those provide resource access, such as coccurrences, frequencies, example sentences and several others provided by the corpus portal of the “Deutscher Wortschatz”-project[9] or access to GermaNet, TüBa-D/Z[10] and others. Some other services provide access to tools. Beside many other tool-services several tokenizers and pos-taggers for different languages, a semantic annotator, a constituent parser and a morphological analyzer for german are available. The huge majority of these services is currently based on the TextCorpus-format, which was introduced at an early stage of the DSpin prototype initiative.

Figures 5 and 6 show screenshots of the DSpin repository management tool and weblicht.



Id	Name	Description
15'	ULEI - TextCorpus2Lexicon	Transforms all tokens from TextCorpus to lemmas in Lexicon
15(	ULei - Frequency (deutsch)	Adds frequencies from the german wortschatz database
15(	ULei - Cooccurrences (deutsch)	Adds cooccurrences from the german wortschatz database
16(	ULei - Baseform (deutsch)	Adds baseforms from the german wortschatz database
16:	ULEI - Sentences (deutsch)	Takes all lemmas from a lexikon and creates a TextCorpus documents with sentences co

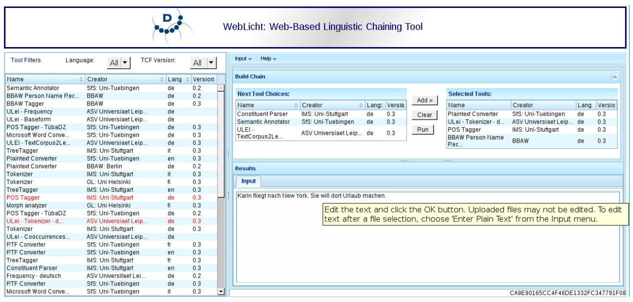
Id: 158 Name: ULei - Frequency (deutsch)
ShortDescription: Adds frequencies
Url: http://xxxx.informatik.uni-leipzig.de:xxxx/dspin_xxxx/Frequencies
Description: Adds frequencies from the german wortschatz database
Creator: ASV Universiaet Leipzig
Contact: wort@informatik.uni-leipzig.de
InputWrapper: DSpinLexicon OutputWrapper: DSpinLexicon
InputWrapperVersion: OutputWrapperVersion:

Name	Attribute	Standard
lemma		
Language	LanguageCode	German

Name	Attribute	Standard
Frequency		

Password:

Figure 5: DSpin repository management tool



Language: All TCF Version: All

Name	Creator	Lang	Version
Semantic Annotator	SfS: Uni-Tuebingen	de	0.2
BBAW Person Name Rec...	BBAW	de	0.2
BBAW Tagger	BBAW	de	0.3
ULei - Frequency	ASV Universiaet Leip...	de	
ULei - Baseform	ASV Universiaet Leip...	de	
POS Tagger - TübaDZ	SfS: Uni-Tuebingen	de	0.3
Microsoft Word Conve...	SfS: Uni-Tuebingen	de	0.3
ULEI - TextCorpus2Le...	ASV Universiaet Leip...	de	0.3
TreeTagger	IMS: Uni-Stuttgart	it	0.3
Plaintext Converter	SfS: Uni-Tuebingen	en	0.3
Plaintext Converter	BBAW: Berlin	de	0.2
Tokenizer	IMS: Uni-Stuttgart	it	0.3
Tokenizer	GL: Uni Helsinki	fi	0.3
TreeTagger	IMS: Uni-Stuttgart	en	0.3
POS Tagger	IMS: Uni-Stuttgart	de	0.3
Morph analyzer	GL: Uni Helsinki	fi	0.3
POS Tagger - TübaDZ	SfS: Uni-Tuebingen	de	0.2
ULei - Tokenizer - d...	ASV Universiaet Leip...	de	0.3
Tokenizer	IMS: Uni-Stuttgart	de	0.3
ULei - Cooccurrences...	ASV Universiaet Leip...	de	
RTF Converter	SfS: Uni-Tuebingen	fr	0.3
RTF Converter	SfS: Uni-Tuebingen	en	0.3
TreeTagger	IMS: Uni-Stuttgart	fr	0.3
Constituent Parser	IMS: Uni-Stuttgart	en	0.3
Frequency - deutsch	ASV Universitaet Lei...	de	0.2
RTF Converter	SfS: Uni-Tuebingen	de	0.3
Microsoft Word Conve...	SfS: Uni-Tuebingen	it	0.3

Next Tool Choices:

Name	Creator	Lang	Versio
Constituent Parser	IMS: Uni-Stuttgart	de	0.3
Semantic Annotator	SfS: Uni-Tuebingen	de	0.3
ULEI - TextCorpus2Le...	ASV Universiaet Leip...	de	0.3

Selected Tools:

Name	Creator	Lang	Versio
Plaintext Converter	SfS: Uni-Tuebingen	de	0.3
ULei - Tokenizer - d...	ASV Universiaet Leip...	de	0.3
POS Tagger	IMS: Uni-Stuttgart	de	0.3
BBAW Person Name Rec...	BBAW	de	0.3

Karin fliegt nach New York. Sie will dort Urlaub machen.

Edit the text and click the OK button. Uploaded files may not be edited. To edit text after a file selection, choose 'Enter Plain Text' from the Input menu.

CA9E90165CC4F46DE1332FC347791F08

Figure 6: WebLicht

4. Conclusion

The DSpin prototype proves the capabilities of the basic implementation of the chaining algorithm that is currently used. It allows experienced users to build custom process chains out of services available in the prototype infrastructure. A future implementation should set its focus on the usage of existing and well established standards and techniques. For example, it should be checked whether SOAP-webservices and services descriptions based on WSDL are compatible to the currently used approach. Security aspects are also a big concern. In the current infrastructure security functionality can only be enforced on the application layer but not on the level of services. The proposal of defining valid combinations of parameter-type/standard pairs by

[8]See *http://clarin.sfs.uni-tuebingen.de:8080/WebLicht1/*

[9]A webportal, which provides access to over 50 corpora based monolingual dictionaries. See [2] and *http://corpora.informatik.uni-leipzig.de/* for details.

[10]TüBa-D/Z: A german treebank. See [1] for details.

referencing to ISOcat data categories in order to describe the content of documents and to define input-/output specifications of webservices also needs further practical evaluation. These and other aspects will be addressed by the upcoming Clarin infrastructure in the future, which may make use of a more advanced implementation of the chaining algorithm and the experiences gathered on the implementation and maintenance of the DSpin prototype.

Future experiments should focus on the performance and scalability of the chaining- and the automatic process chain builder algorithms. Especially the automatic chain builder algorithm needs to be improved in order to work over several thousand or even more services that will be available in the upcoming infrastructures. The possibility of using user generated feedback on process chains invoked in the past, either directly entered by the users or indirectly gained by the usage of services, in order to improve the quality and ranking of results and the speed of the automatic chain builder should be explored and evaluated in a real application.

5. References

[1] Telljohann, H.; Hinrichs E.; Kübler, S. (2004). *The TüBa-D/Z tree-bank: Annotating German with a context-free backbone*. Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004), Lisbon, Portugal, 2004.

[2] Quasthoff, U.; M. Richter; C. Biemann (2006). *Corpus Portal for Search in Monolingual Corpora*. Proceedings of the fifth international conference on Language Resources and Evaluation, LREC 2006, Genoa, pp. 1799-1802.

[3] Kemps-Snijders, M.; Windhouwer, M.A.; Wittenburg, P.; Wright, S.E. (2008). *ISOcat: Corralling Data Categories in the Wild*. European Language Resources Association (ELRA) (ed), Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC 2008), Marrakech, Morocco, May 28-30, 2008.

[4] Hinrichs, E.; Hinrichs, M.; Zastrow, T.; Heyer G.; Boehlke, V.; Quasthoff, U.; Schmid, H.; Heid, U.; Fritzinger, F.; Siebert, A.; Didakowski, J. (2009). *Weblicht: Web-based LRT services for German*. Workshop on linguistic processing pipelines, Potsdam, GSCL Jahrestagung 2009.

[5] Heid, U.; Schmid, H.; Eckart, K.; Hinrich E. (2010). *A corpus representation format for linguistic web services: the D-SPIN Text Corpus Format and its relationship with ISO standards*. LREC 2010.

[6] Cardoso, J.; Sheth, A. P.; 2003. *Semantic e-workflow composition.* Netherlands: Journal of Intelligent Information Systems 21:3, pp. 191-224.

[7] Shiyong, L.; Bernstein A.; Lewis, P. (2006). *Automatic Workflow Verification and Generation*. Theoretical Computer Science (TCS), 353(1-3), pp. 71-92.

A Graphical Interface for Computing and Distributing NLP Flows

Ionuţ Cristian Pistol, Andrei Arusoaie, Andrei Vasiliu, Adrian Iftene

General Berthelot Street, 16, Iasi 700483, Romania
E-mail: {ipistol, andrei.arusoaie, andrei.vasiliu, adiftene}@infoiasi.ro

Abstract

Large scale linguistic processing flows are more common now than ever. Virtually all well-known NLP meta-systems (such as GATE and UIMA) are developing functionalities to allow users to build and run processing flows using web-services as components or using networked computers as a cluster or Grid in order to speed up the execution of complex applications on large corpora. ALPE (Automated Linguistic Processing Environment) is a newly emerging NLP meta-system similar to GATE and UIMA but aiming at improved usability by users less familiar with programming and NLP, such as humanities and social studies specialists. GECC (General Environment for Cluster Computing) is also a newly developed distributed computing environment managing computer clusters capable of running flows. This paper describes our proposed merging of ALPE and GECC to produce a system which has the potential to solve many of modern day NLP challenges as defined in projects such as CLARIN and FLaReNet.

1. Introduction

Making sure that corpora, resources and tools are reusable in different contexts than that of the originating project is one of the recent main topics of interest in the Natural Language Processing community. Re-using a resource initially developed for a specific project usually fails for one of two reasons: either the resource is not properly documented (the requirements are not known to the re-user), or the resource is not directly accessible (the location or the availability are not known to the re-user). Making sure a project's results are well organized and accessible ensures a better impact and a longer lasting significance, as more people will be able to use the developed resources and tools. Projects such as CLARIN[1] and FLaReNet[2], among others, intend to offer both developers and users of language resources and tools a management solution for the growing set of resources available. The primary objectives of these projects are to provide reusability in new contexts for existing resources and to guarantee maximum visibility and reusability for newly developed resources. An easy widening of the original setting of usage means a multiplication of the visibility of a tool and, finally, of the productivity of the research activity.

One of the latest developments in NLP, and one which promises to have a significant impact for future linguistic processing systems, is the emerging of linguistic annotation meta-systems, which make use of existing processing tools and implement some sort of processing architecture, pipelined or otherwise. Systems such as GATE – General Architecture for Text Engineering (Cunningham et. al. 2002) and UIMA – Unstructured Information Management Application (Ferrucci and Lally, 2004) allow users to combine linguistic processing modules (previously integrated in the system) in processing flows, which can then be saved and executed on any number of documents. Recently, both GATE and UIMA allow the execution of modules and even flows on computer clouds. GATE is developing GATE Cloud[3], promising to offer distributed computing of flows over a cloud of dedicated computers. UIMA is offering a user-developed tool called Simple Server[4] which converts UIMA flows to REST (Fielding and Taylor, 2002) descriptions allowing online discovery, deployment and execution of those flows.

ALPE (Automated Linguistic Processing Environment) is a system offering a new perspective to the task of exploiting NLP meta-systems by helping a community of users to have an integrated look at a whole range of tools that are able to communicate on the basis of common formats. ALPE allows a user, even with very limited programming capabilities, to automatically exploit already walked-on processing paths or to configure new ones on-the-spot, by exploiting the annotation schemas at intermediate steps. The configuration of processing flows is done simply by selecting an input and output format and selecting one of the automatically computed flows, which might differ by financial cost, estimated duration and precision, or number of intermediate formats produced. This is a process allowing non-specialists access to NLP technologies, which is the main goal of projects such as CLARIN and FlaReNET.

In the last years the computational Grids (Gannon and Grimshaw, 1998, Gannon et. al. 2002) have become an important research area in large-scale scientific and engineering research. The computational Grids offer a set of services that allow a widely distributed collection of resources to be tied together into a relatively seamless computing framework, teams of researchers can

[1] CLARIN: http://www.clarin.eu/
[2] FLaReNet: http://www.flarenet.eu/
[3] GATE Cloud: http://gatecloud.net/g8/contact/
[4] UIMA Simple Server: http://incubator.apache.org/uima/sandbox.html#simple-server

collaborate to solve problems that they could not have attempted before. Unfortunately, after years of experience in this area, the task of building Grid applications still remains extremely difficult, mainly because there are few tools available to support developers.
GECC (General Environment for Cluster Computing) is a project aiming to implement distributed computing for NLP tasks described as graphs, similar to those built by ALPE. The model implemented offers similar capabilities to a Grid network, but also brings new functionalities making it more suitable to NLP tasks such as XML configuration of files, modules and processing flows, streamlined graphical description of flows and compatibility with ALPE built flows and modules. GECC adopts a cluster model (Baker et. Al. 1999) as opposed to a strict Grid, which allows ease of deployment and execution of applications on any available computers.
Section two of this paper briefly presents the theoretical base and the general functionalities of ALPE. Section three describes GECC and shows the potential benefits of integrating the two systems. The conclusions, as well as the further planned developments are described in section four.

2. ALPE

The description of the ALPE system is beyond the scope of this paper, however detailed descriptions can be found in (Cristea et. al. 2008, Pistol and Cristea 2009). ALPE builds and runs linguistic processing flows originating in a hierarchy (directed acyclic graph) whose nodes identify annotation formats and on whose edges processing modules can be attached.
If a user wants to process an XML file from one input format to some output format he uploads the file and a correspondence will be made between the file and a node in the graph. Then the user selects another node in the hierarchy corresponding to the desired output format and a processing flow (a sequence of one or more sequential or parallel processing tasks) will be computed between the two nodes. Generally, a processing task involves a transformation by some module capable to receive the input format and to output the required final format. The ALPE philosophy details such a processing task in relation with the pair of input-output schemas by establishing the way these schemas interrelate from the point of view of the subsumption relation. Two cases can be evidenced: either the two schemas do observe a subsumption relation or not. When they do, then the node corresponding to the input file can be connected through a direct descending or ascending edge to the one corresponding to the output file. It will be descending if the output schema results from the input schema through some adds, and it will be ascending if in order to obtain the output, simplifications applied to the input are required. When the two schemas are not in a subsumption relation, then there should be a node such that either both are subsumed by it, or both subsume it.
ALPE comes with a core hierarchy whose nodes act as a grid of fixed bench-marks with respect to which the locations of the input and output schemas are set out. When the pair of users' schemas matches two nodes of the core hierarchy, then processing can be drawn in terms of known (built-in) interconnected modules. When a match (modulo, as noticed above, the XML elements name space and/or differences in configurations of attributes still conveying the same information) of one or even both of user's schemas against nodes of the hierarchy is not possible, then the non-matching schemas should be seen as new nodes of the hierarchy.
ALPE offers the following functionalities:

- the user can generate a new hierarchy;
- the user can input an annotated file and ALPE will classify it in the existing hierarchy;
- the user can input a linguistic processing tool and some required data (see next section) and it will be added to an existing hierarchy, and will be usable in

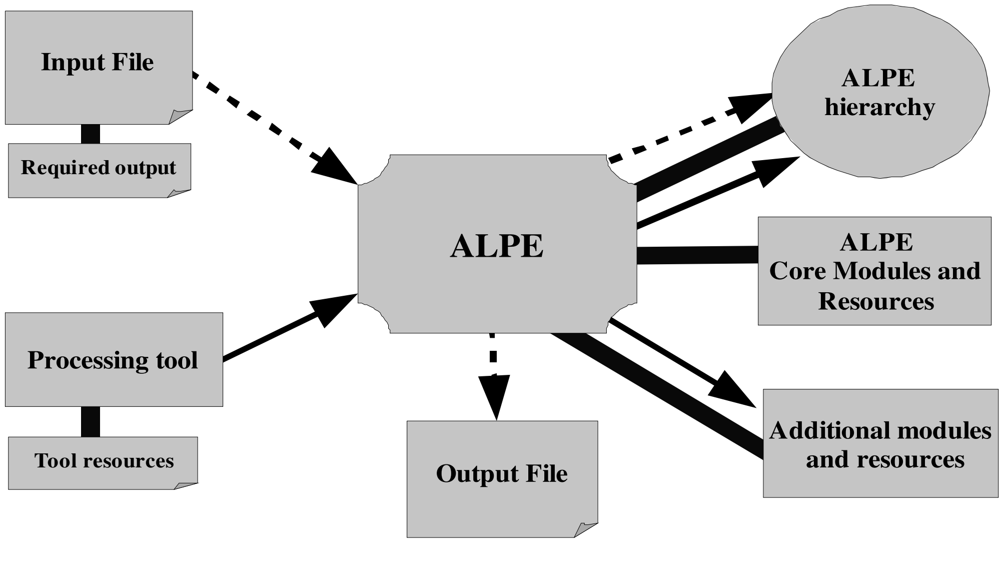


Figure 1: The ALPE architecture and general functionalities

later computed flows;

- the user can input an annotated file and specify a required format (either selecting from the existing hierarchy, or inputting a new schema specification) and ALPE will compute processing flows between the two formats. The user then has the choice as to which of the computed flows to be executed by ALPE, which will output the file with the required format.

In figure 1, the existence of a thick line between two components denotes the obligatory presence of both connected components. Basically, ALPE requires an ALPE hierarchy, the core modules and resources and the additional modules and resources. If the user inputs a file to be processed, he has to specify the required output and ALPE will possibly input some changes in the available hierarchy, as well as produce the output file. If the user inputs a new processing tool, ALPE will input the changes implied in the hierarchy, will add the tool to the existing additional modules, as well as add the tool's resources to those available.

ALPE includes 11 core modules, used in any ALPE hierarchy (the hierarchy augmented with processing power, as described) but not attached to any edge. The core ALPE modules perform:

- language identification for input documents;
- format identification and classification for an annotated document;
- simplification of an annotated document to a format in the hierarchy;
- merging of multiple annotated versions of the same text;
- creation/development of an ALPE hierarchy;
- integration of a new tool in the hierarchy.

These are the ALPE core modules required in the current state of the system; further developments may add additional modules. These core modules are used in any ALPE hierarchy and are not replaceable by user tools. They ensure that any ALPE hierarchy is able to perform according to the specified features.

Since the flow computation process may produce two or more flows for a single user task, a selection can be made. Each computed flow is characterized by a set of features. These features include properties such as flow length (defined as number of processing steps involved) and flow weight (number of intermediate formats produced if computing the flow). Other features are the cost of the flow (the actual financial cost, if one or more modules involved require payment), the estimated precision of the flow (computed using the performance measure specified when adding a new tool to the hierarchy) and the estimated time of computation. The user can then select and run the flow most suitable to his or her needs. The user will be able to specify some default value for the selection, so flow computation, selection and execution can be performed automatically.

ALPE has been extended recently allowing the inclusion of non-XML annotation formats in ALPE hierarchies. This is done by manually establishing a semantic identity between a node in the hierarchy and a non-XML annotation format. If the user also provides wrappers between that format and one or more nodes in the hierarchy then that format can serve as either input or output node for future computed processing flows.

3. GECC

The need to develop distributed systems capable of performing complex calculations is very common nowadays. There are many distributed systems specially built to perform calculations in different scientific fields. Folding@Home[5] (intensive simulations of protein folding) or MilkyWay@Home[6] (uses data from the Sloan Digital Sky Survey to deduce the structure of the Milky Way galaxy) are just two examples of the fastest distributed systems. A more general approach for distributed computations is BOINC – Berkeley Open Infrastructure for Network Computing[7] (Anderson, 2004). The intent of BOINC is to make it possible for researchers to tap into the enormous processing power of personal computers around the world. It was originally developed to support the SETI@home[8] project before it became useful as a platform for other distributed applications in areas as diverse as mathematics, medicine, molecular biology, climatology, and astrophysics. As we can observe, all these projects are designed to solve a specific problem, such as protein folding, analyzing Milky Way structure or perform complex calculations. To create a project using BOINC means to implement a specific API and create some XML configuration files. These are used by the BOINC platform to identify the regions that can be separately executed and to distribute them to available computers. BPEL – Business Process Execution Language is an OASIS[9] standard executable language intended to specify interactions with Web Services. It is an orchestration language which can specify a process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the orchestration designer. Setting up a process with BPEL involves describing automata, writing the XML configuration files and finding and using available and suitable web services.

The GECC intent is to offer a generalized and simple way to run applications on a Cluster. It provides an easy to use graphical interface which allows creating and controlling an execution scheme. The execution scheme represents the way the tasks will be assigned to computers from the Cluster, thus achieving a separation between the way the distributed system is executing an application and the application's purpose. GECC allows the execution of any correctly configured scheme which can contain different applications and files. The Cluster can execute any type of

[5] Folding: http://folding.stanford.edu/
[6] MilkWay: http://milkyway.cs.rpi.edu/milkyway/
[7] BOINC: http://boinc.berkeley.edu/
[8] SETI: http://setiathome.berkeley.edu/
[9] OASIS: http://www.oasis-open.org/home/index.php

application without being limited to a scientific field performing special calculation. Our project offers the possibility to manage and build an execution flow over a distributed system in an interactive manner, without asking the user for advanced knowledge to use the system. The user is not supposed to write configuration files or to monitor the execution over the cluster, but rather only to specify the execution flow on a graph, using the graphical interface.

One of the goals of this project is to use a graphical model to organize the execution of a sequence of steps. For example, consider the annotation of text or XML files. These will be annotated using some executables, also called annotators, receiving text files as arguments and returning another annotated XML file. There are cases where certain executables can annotate text only if it has already been annotated by one or more annotators previously. Our approach for modeling the sequence of annotations is to use a directed graph with two types of nodes: *pack nodes* and *transformer nodes*. An edge can only connect nodes of different types: there may be a connection from a *pack* to a *transformer* or from *transformer* to a *pack*. A *pack node* represents a container of files. In the annotation use case, these nodes will contain text files or XML files. A *transformer node* collects files from *packs* connected to its inputs and produces a transformation, with the resulting files then being distributed to its outputs. On a specific session, a transformation is specified by a command, such as running an executable over the input files.

GECC's main objective is distributed execution of tasks from the review of a graph. From the graph described above for sequence annotation, GECC can cause a splitting into smaller units of execution, which can be run separately on different computers in the cluster. The command associated to the transformer, the required programs (such as the annotator) and the file to be annotated, which are stored in the *pack nodes*, are sent to a computer in the cluster. That computer executes the command and sends the results to a central repository, an approach that gives an advantage in terms of execution time.

The user can control dependencies with the flow graph, and can also directly specify if certain transformations can be executed in parallel for each of its input files (for example, the annotation of each file), or not (building an archive with all results, a process that can only take place if the other tasks were already finished). The way the application manages tasks is closely related to the model built, thus considering execution dependencies like tasks that cannot be executed unless other tasks have been executed.

3.1 GECC Architecture

In terms of implementation, the project can be divided into four major components: a graphical user interface, a web server, a main server and a cluster. There is an extra component that ensures the detection of system components by recording the IP addresses. Figure 2 shows the connections between the main components. Some components, like MainServer, WebServer and Cluster Computer, are using the "fire and forget" principle. This means that components must self-configure themselves, must find each other automatically (only possible by using the IPTracker, with other discovery methods currently in development), and must be remotely controllable. One of our main ideas was to facilitate access to the system; for this reason, the UI Component will be accessible from any computer connected to the Internet, directly from the browser. The WebServer must serve the UI Component with all the information the user needs to configure and use the system. The communication between these components is done via HTTP (the classical request-response scheme). The WebServer must also offer a way to use the distributed system composed by Main Server and Cluster components. Once a workflow has been created it must be executed by the cluster area of the system. The WebServer sends the workflow to a manager capable to split it into pieces(independent execution units), send them to computers for execution and join the results. This manager is the MainServer component, whose role is to organize the execution by assigning tasks to available computers from the cluster and handle the errors that might appear during the execution. Combined, the MainServer, WebServer and Cluster components form GECC's distributed system. IPTracker is the component which facilitates the auto-configuration of components,

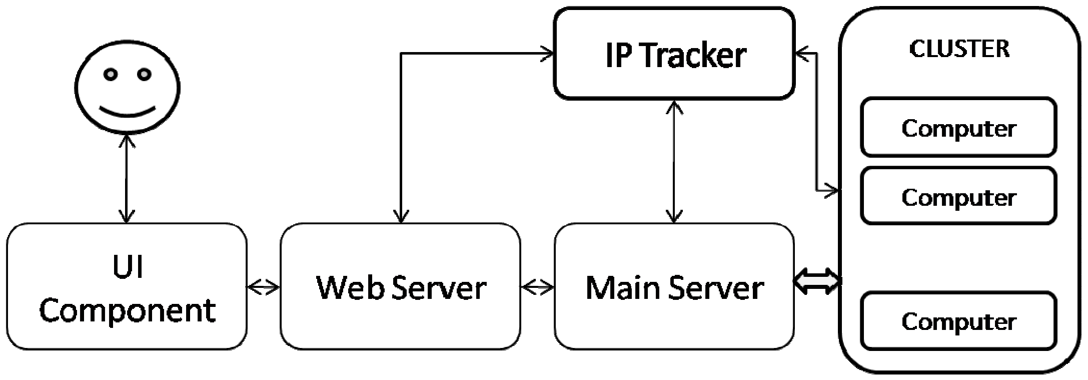
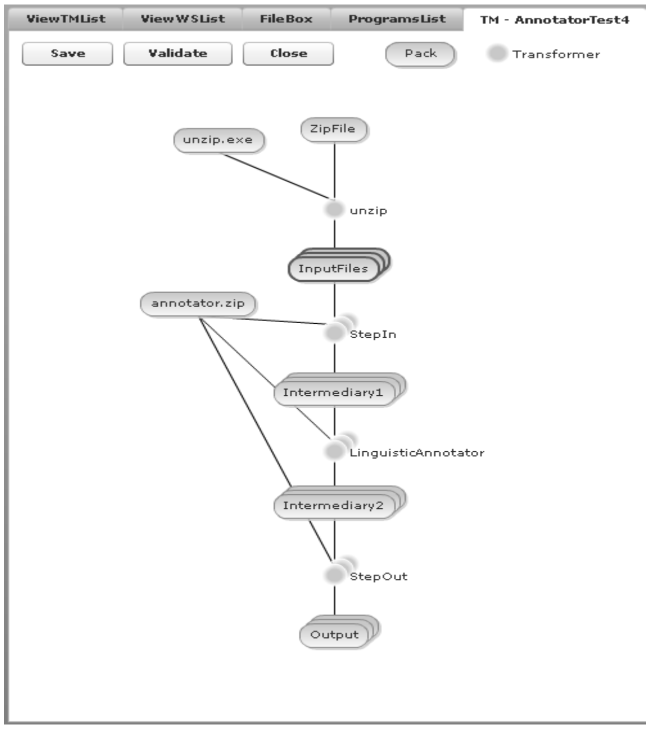


Figure 2: General architecture of GECC

by providing them IP addresses of other nearby components; a component scans these addresses, and establishes connections with the components it can reach. Once an instance of MainSever, WebServer or Cluster Computer is created it must announce its presence to IPTracker so that any other instance can have knowledge of the newly created component. Suppose that the WebServer sent a workflow to a MainServer, which is keeping it busy. In order to send another workflow, it must wait the MainServer to finish. But, by requesting other IPs from the IPTracker, the WebServer can find another instance of a MainSever which is idle and can accept the workflow; the same situation is met when a new computer joins the Cluster and the MainServer can start sending any pending tasks to it. Next we will describe the components and interactions among them.

User Interface

The User Interface component represents the interaction between system and user, by which he shall coordinate the entire activity. Once it is started, the user has everything he needs to easily configure an execution workflow, and also has access to previously saved workflow models and uploaded files. The main idea for the interface is to offer a simple way to create and save the workflows. GECC saves a workflow as a *transformation model*, which is nothing less than a digraph describing a model in an XML file. The user is allowed to create a new model from scratch and save it for further use. Also, the user can create new nodes over the created *transformation model*, link them to each other, and set various properties. Before it is saved, the *transformation model* must be validated first. The validation means that model can be interpreted by a MainServer so that it can organize the execution flow as described in the model. It is also possible to import a *transformation model* from an external XML file, both to help with the sharing of models, and to ease integration with other applications capable of generating models usable by GECC (such as, for example, ALPE).



```
…
<packNode name="InputFiles" isSplitter="true"
id="4">
      <input node="3"/>
      <output node="5"/>
      <pattern regex=".*"/>
</packNode>
<packTransformerNode name="StepIn" id="5">
      <input node="4,7"/>
      <output node="6"/>
      <command exec="java -jar StepIn.jar
$InputFiles ./">
          <requires program="java" />
          <requires program="StepIn" url=""
lastUpdated=""/>
      </command>
    </packTransformerNode>
…
```

Figure 3: A processing flow in GECC

Having a workflow saved as a *transformation model*, a user can create a *work session* based on it. A *work session* is represented by a *transformation model* in which every node has been configured. For a *pack node*, the user can upload files or specify what kind of files it may contain (a *pack node* may also contain output files generated by a transformation, and may accept these files only if they meet certain criterea, such as matching their file name against a regular expression). For a *transformer node* a command is set, which specifies exactly what the transformation is; most often, a command will execute a specific application, with specific arguments. The newly created *work session* is executed by the cluster and the user is able to check the execution progress. The results are available in *packs* and can be downloaded easily.

WebServer

The WebServer component represents the link between the User Interface and the MainServer components. Its purpose is to mediate communication between them and to provide support for file management involved in the session. This component is a collection of Servlets and JSP pages, each of them having a well defined scope. The WebServer acts as a file and information repository, and can offer lists of available *transformation models*, uploaded files, etc. There are also a few specific tasks which involve more that sending back a response; for example, the validation process implies that the WebServer needs to find an available MainServer, which has already announced its presence to the IPTracker, and send to it the XML representation. It receives the MainServer's response and encapsulates it in XML format, which will be sent to the client. On success, the WebServer saves or overwrites the *transformation model* XML in a well-known location; on error, the XML response will contain the reason for that error (the error message, the node that caused the error, etc).

Another complex action is to handle the creation of a *work session*. When the WebServer receives a request with the name of the model, it must find an available MainServer to assign the execution of the received model. If no MainServer is explicitly specified, the WebServer will try each one from its list (retrieved from IPTracker) until one responds. If a suitable MainServer is found, the WebServer will then act as a relay between the browser-based client and the MainServer.

MainServer

The MainServer is the component responsible for managing tasks, and the cluster computers that will run those tasks. When it is asked to validate a model, the server parses the model XML, attempts to construct a representation, and return whether it succeeded. Analyzing the graph, the MainServer builds a list of tasks which will be assigned to cluster computers. This list is dynamic, as some of its contents will depend on the results of other tasks; as such, the list is incomplete at the beginning. When a MainServer wants a cluster computer to execute a task, it will send a call with details about the task. These details will include at least a task ID, an IP to send the result to, a list of files to be downloaded, an URL to use for downloads, an URL to use for uploads, and a command to execute. Once the execution of the task is finished, the MainServer receives a notification, and mark the task as finished from the tasks list.

The communication with computers from the cluster is done via a communicator component; our implementation is currently based on Java RMI calls, with plans to move to a more general protocol, such as TCP or UDP.

One of the more common problems of processing flows is that some tasks might fail. For this reason, the MainServer must also include failover mechanisms: If a task fails due to problems with the network connection, the MainServer will send it to a different computer in the Cluster, or if it fails due to an error reported by a specific tool, it will notify the user, so that the problem can be diagnosed properly.

Cluster

The Cluster component consists of a set of computers connected to the Internet, each of them capable of communicating, in one way or another, with the IPTracker, the MainServer and the WebServer. Due to the zero-configuration requirement, as soon as a Cluster Computer process is started, it must retrieve a list of MainServers from IPTracker and notify each one of its presence. When notifying its presence to the MainServer, some state information must also be sent, such as its IP and its number of CPUs or CPU cores. The comunication to WebServer is restricted to upload/download of files. Because the MainServer knows only the locations of files to avoid managing them (which is not the MainServer's purpose), the cluster computer must get the files from the repository (the WebServer), a process which is done with the standard HTTP operations. To add a new computer to a GECC cluster is a simple task: just execute an automatic configuration module on that computer and it would be available to run future processing tasks.

The main concern of a Cluster computer is to execute a task, which implies a download of necessary files, execution of a command and an upload of the results. This means that when downloading/uploading, CPU is not used, and when executing a command, network bandwidth is not used. To use both resources simultaneously the Cluster computer uses blocking queue structures to store the tasks in three different states: the state before download, the state before execution and the state before upload. Each operation is handled in a separate thread to ensure a parallel execution of it; this approach is also especially useful on multi-core (or multi-processor) machine: if a machine has multiple cores, it can execute tasks in parallel on each core, thus achieving better time results even if GECC is used on a single computer.

A cluster computer must also offer some useful mechanisms. For example, to avoid the download of a file already downloaded for a previous task, the cluster computer uses a local cache with a specified maximum local size. Another example is the possibility to upload zip files. The cluster computer detects and unzips them in the task's folder.

Once the execution of a task is successfully finished, the cluster computer uploads only the new or modified files to the WebServer and notifies its corresponding MainServer about it.

3.2 From ALPE flows to GECC

ALPE offers the user the possibility to save computed flows for further reference or execution on corpora. This is the step in which GECC fits in, thus the ALPE flows can now be saved as GECC configuration files. A GECC configuration file is an XML resource describing the actual sequence of modules needed to be run according to the ALPE flow. A GECC configuration file specifies only the minimum required information required to run the flow (module name and required software) and is independent from the original graph the flow was computed on. An ALPE flow actually links the sequence to a graph of annotations, thus it is complementary to the

GECC format and cannot be replaced by it.
A simple flow, involving three sequential processing steps, loaded in GECC and ready to process new input files, can be seen in figure 3. In figure 3 are also included fragments from the generated XML, produced by ALPE, describing the flow.
The flow involves three modules (two implemented in Java, one in Python), accepts as input raw text files and produces annotated XML documents (tokenization, POS-tagging, lemmatization and sentence-level segmentation is performed). The GECC flow, as shown in figure 3, includes also an unpacking step required as it assumes inputted files as zip archives. After the files are extracted automatically, the three linguistic processing modules (*StepIn*, *LinguisticAnnotator*, *StepOut*) are executed in sequence. The actual execution is done on the distributed network available to GECC who serves as a server for the processing network. All intermediate files and processing modules are stored on the main server and are made available to the processing units on the network when needed.

3.3 Evaluation

Comparing GECC with similar systems is generally a qualitative evaluation and has more to do with ease of deployment and flexibility of the used clusters. A new computer, regardless of general configuration, can be easily included in an existing cluster.
Deployment of linguistic processing flows in a distributed environment is still a new development; a direct comparison is speed of execution and overhead for execution in a cluster is not yet possible. In order to provide a quantitative evaluation of the benefits of running an ALPE flow in GECC, we used a set of 454 xml files as input, each file having a size smaller or equal to 4Kb. We used the flow described above for all tests.
At first run, 454 XML input files are annotated by the annotator running on a single computer, without using GECC system. The measurement of time starts when the annotator is started and stops in the same time with it, when all files were successfully annotated. Each computer CPU has 2 cores, meaning that each of them can execute 2 tasks in the same time. Then, GECC is used with different configurations regarding the number of computers from Cluster. Running a processing task with GECC includes now a small overhead mainly due to packing and unpacking temporary annotated files in order to reduce network transfers. First, we registered the overhead time when the Cluster contains only one computer (just 1 core): 283 seconds. The time is then recorded for a Cluster with sizes: 2, 4, 8 and 16. Each XML file and the annotator are packed and send as a task to be executed by a computer from the cluster. The measurement of time starts when the first task is sent and stops when the last task results are uploaded.
The time obtained for each test case is shown table 1.

Conditions		**Time elapsed (seconds)**	**Saved time (seconds)**
Test annotator without GECC		3523	-
Test annotator with GECC The Cluster contains	2 computers	1358	-
	4 computers	689	669
	8 computers	362	327
	16 computers	210	152

Table 1: Running times for the evaluation experiment

Using a Cluster to execute a sequence of annotations is an advantage in terms of execution time. Analyzing the results we can observe the execution time is decreasing when more computers are added to the Cluster. Doubling the number of computers from the Cluster the execution time is reduced by about 50%. From the initial baseline running time of 3523 seconds the maximum improvement measured is for the maximum number of 16 available networked computers, for which the same flow on the same input took just 210 seconds (about 6% of the original time).
Another less measurable advantage of adopting the ALPE/GECC way of building and running complex processing flows is the ease with which these flows can be computed and executed, especially for users not familiar with the modules employed. Both ALPE and GECC offer visual interfaces in which annotation formats and processing modules are shown as nodes and edges respectively on a graph. In ALPE, the user can input files which are classified on a processing graph, and then he can select a desired output node. After selecting one of the computed flows, the user can run that flow in GECC automatically. The process of building and running the flow manually can require the user to collect and select available processing modules, then to decide which modules to run and in which order, after making sure he has the hardware and software requirements to do so. All this makes building and running large linguistic processing flow prohibitive for a large scale of humanities and social studies researchers, often requiring linguistic technologies. Even computer scientists working in NLP can benefit from the streamlined flow management by easily designing test cases and observing the impact of a different module in a complex flow.

4. Conclusions

ALPE will be used as a management tool for Grid services, in itself being adapted as a Grid service. Due to its particular functionalities, it will offer improved usability and access to linguistic tools and resources, factors especially important to large scale and multilingual research projects. Using ALPE as a Grid services management environment allows the creation of a global linguistic hierarchy, integrating a multitude of services targeting linguists and students alike.

GECC can serve as a central hub for any linguistic annotation flows as computed by ALPE. The significant increase in speed will increase the appeal of complex processing flows making them available for large scale annotation efforts and comparisons between processing modules on large scale corpora. Also, since Grid type networks already are employed in several NLP research projects, the developed tools and resources of those projects can be easily integrated into ALPE. Both GATE and UIMA plan to offer the integrated modules as web services in the near future, which will make them easily usable as ALPE modules.

One important further development of ALPE will be a web-service allowing users to build, configure and use ALPE hierarchies on the web, either as a limited password-protected resource or a global linguistic resources collection. This type of hierarchy is able to manage multilingual resources and resources which require a fee to be paid before usage. Each user will be able to contribute its own tools and annotated resources, as well as using processing chains adapted to its specifications, both in terms of input and output formats and cost and performance issues. GECC will be made available via the GNU General Public License in the near future.

5. Acknowledgements

The authors would like to thank MsC. students Mereuță Radu-Adrian and Frona Ioana, for their help and support at different stages of system development.

6. References

Cristea, D., Forăscu, C., Pistol, I.C. (2006) "Requirements-Driven Automatic Configuration of Natural Language Applications". In Bernadette Sharp (Ed.): *Proceedings of the 3rd International Workshop on Natural Language Understanding and Cognitive Science - NLUCS 2006*, in conjunction with ICEIS 2006, Cyprus, Paphos. INSTICC Press, Portugal. ISBN: 972-8865-50-3.

Cristea, D., Pistol, I. (2008): "Managing Language Resources and Tools Using a Hierarchy of Annotation Schemas". Proceedings of the Workshop on Sustainability of Language Resources, LREC-2008, Marakesh.

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 ACL (ACL'02)*. Philadelphia, US.

Ferrucci D., Lally, A. (2004) "UIMA: an architectural approach to unstructured information processing in the corporate research environment", *Natural Language Engineering* 10, No. 3-4, 327-348.

Gannon, D., Bramley, R., Fox, G., Smallen, S., Rossi, A., Ananthakrishnan, R., Bertrand, F., Chiu, K., Farrellee, M., Govindaraju, M., Krishnan, S., Ramakrishnan, L., Simmhan, Y., Slominski, A., Ma, Y., Olariu, C., Rey-Cenvaz, N., (2002) "Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications", *In Cluster Computing journal*, Volume 5, Number 3, Pp. 325-336.

Gannon, D., Grimshaw, A., (1998) "Object-Based Approaches", *The Grid: Blueprint for a New Computing Infrastructure*, Ian Foster and Carl Kesselman (Eds.), pp. 205-236, Morgan-Kaufman.

Pistol I. C., Cristea D. (2009) "Managing Metadata Variability within a Hierarchy of Annotation Schemas", Proceedings of the 6th International Workshop on Natural Language Processing and Cognitive Science - NLPCS 2009, Milan, Italy - May 2009, pp. 111-116, ISBN: 978-989-8111-92-0.

Fielding R., Taylor R. (2002) "Principled design of the modern Web architecture", *ACM Transactions on Internet Technology,* volume 2, issue 2, pp. 115-150, ISSN: 1533-5399

Baker, M., Buyya, R., Hyde, D. (1999) "Cluster Computing: A High-Performance Contender", IEEE Computer 32(7): pp. 79-80

Corpora by Web Services

Adam Kilgarriff
Lexical Computing Ltd
Brighton, UK
E-mail: adam@lexmasterclass.com

Abstract

Corpora are large objects and querying them efficiently is non-trivial. There are substantial costs to building them, storing them, maintaining them, and building and maintaining software to access them. We propose a model where this work is done by a corpus specialist and NLP systems then use corpora via web services. Our corpus tool is fast, even for billion-word corpora, and offers a wide range of queries via its web API. We have large corpora available for twenty-six languages, and are experts in preparing large corpora from the web, with particular expertise in web text cleaning and de-duplication. We regularly increase our coverage of the world's languages via our 'corpus factory' programme. For English, we are building corpora that are both bigger and more richly marked up than others available. We present a case study of a current project using the Sketch Engine, via its web API, to automatically draft 'fill-the-gap' test items for language testing. The combination of the web services model, the corpora, and the tools, will allow many NLP researchers to use bigger and better corpora in more sophisticated ways than would otherwise be possible.

1. Barriers to entry

In the days of rule-based NLP, starting a PhD was easy. The student could write a few grammar rules, lexical entries and example sentences, and all the technology required was a prolog system.

Since the advent of empirical methods, it is harder. Now the student needs a corpus and tools to access it. Before embarking on their research question - perhaps about syntax, or parsing, or anaphora, or discourse structure - they must first review the different resources they might use, or work out if they must build their own, and then cross the technical and administrative hurdles to building it or acquiring it. They must then either write their own code for accessing it or install and become expert on somebody else's tool. Any output for the first few months is likely to be dominated by aspects of the data or tool that they had not anticipated rather than linguistic ones, and it is all too likely that they start feeling their thesis is being sidetracked into corpora and corpus tools. If they do not have the programming skills or technical support to clear these hurdles, they are likely to become dispirited or to shy away from the question that first motivated them and to switch to one which makes use of corpora in simpler ways, though they may then forever be dogged by the anxiety that their research will not stand up to scrutiny by the researcher, otherwise like them, but who did have the support or computational skill to `do everything properly'.

Might it be possible to use corpora without all this overhead, like a driver collecting a hire car?

We believe not only that it is possible (and that we already have a service offering what is required), but that it is likely to improve the quality of research as energies are not wasted on non-specialist, mediocre, corpus-preparation and corpus-accessing, but are directed at the topic that motivated the researcher.

2. The Sketch Engine

The Sketch Engine is a corpus query tool. It has been widely used for lexicography, by clients including Oxford University Press, Cambridge University Press, Collins, Macmillan and FrameNet, and for linguistic and language technology teaching and research at universities. Corpora for many languages have been installed. It is fast, responding promptly for most queries for billion-word corpora. It offers all standard corpus query functions: concordancing, sorting and sampling of concordances, wordlists and collocates according to a range of parameters, full regular-expression searching, subcorpus specification and searching on subcorpora. It also offers some non-standard ones:

- word sketches: one-page summaries of a word's grammatical and collocational behaviour, see Figure 1
- a distributional thesaurus
- keyword lists which identify the distinctive words of a subcorpus: see Figure 2.

The basic input is a corpus, preferably lemmatised and part-of-speech tagged. For the word sketches and thesaurus, either the corpus must already be parsed, or another input is required: a shallow grammar, written as a regular expression over words and POS-tags, in which each grammatical relation to appear in the word sketch is defined. For a computational linguist with a knowledge of the language in question, preparing a basic grammar is not a large task.

2.1 The Sketch Engine Web Service and API

Lexical Computing Ltd., the owner of the Sketch Engine, provides a web service which gives easy access to corpora. Users can start using the corpus for their question directly: the user interface is simple and there is no software to install.

For four years now there has been a Web API for the Sketch Engine. It is written in JSON and is designed for easy integration into tools written in Java, Python, Perl etc. It covers the core functionality of the Sketch

web **BiWeC freq = 787440**

object_of	33396		and/or	18695		pp_of-i	10574		modifies	649632	
surf	1199	9.79	clipart	503	9.48	deceit	198	7.86	site	264048	11.04
browse	851	8.33	software	2083	6.16	intrigue	176	7.54	page	97315	10.12
weave	629	8.21	correu	36	5.95	spider	95	5.79	browser	19649	9.36
host	1487	7.86	spider	68	5.11	lie	247	5.78	server	14586	8.73
spin	523	7.8	print	372	5.05	interconnection	31	5.66	design	14395	7.96
base	4884	7.71	desktop	115	4.98	deception	67	5.56	cam	4617	7.77
search	1386	7.62	email	509	4.97	interrelationship	21	5.38	designer	5430	7.68
crawl	166	6.66	transience	19	4.97	quill	20	5.34	standard	7949	7.3
scour	116	6.47	designer	213	4.87	interdependence	23	5.26	developer	4218	7.29
chat	256	6.3	gopher	19	4.64	datum	2007	5.0	application	9633	7.11
untangle	83	6.25	telnet	19	4.62	corruption	71	3.99	address	5487	6.99
interconnect	75	5.91	multimedia	63	4.62	trust	126	3.98	interface	3077	6.64

Figure 1: Word sketch for the English noun *web*, drawn from the 5.5b BiWeC corpus, based on 787,440 occurrences (truncated to fit.) The first figure for each collocation is the frequency count, the second is the salience score (Logdice, see help pages at http://www.sketchengine.co.uk). One can sort by either. Other options include 'more data', 'less data' and clustering of collocates. Clicking on the frequency count gives a concordance of the instances.

	new_model_corpus:speech			*new_model_corpus*			
lemma	**Freq**	**ARF**	**ARF/mill**	**Freq**	**ARF**	**ARF/mill**	**Score**
sir	6559	537.5	560.0	8641	1365.5	24.5	16.5
Yeah	12839	1015.8	1058.3	17703	3342.1	60.0	15.3
hey	9294	781.6	814.3	12371	2709.5	48.7	14.1
Hello	4623	411.1	428.3	6574	1392.0	25.0	12.5
okay	9512	709.7	739.4	14043	2997.1	53.8	11.7
fuck	7720	456.6	475.7	11540	1750.3	31.4	11.7
hi	3309	291.0	303.1	4449	961.2	17.3	11.5
shit	4607	370.6	386.1	7266	1554.7	27.9	10.4
No	12340	1097.5	1143.4	20342	5795.3	104.1	10.1
Huh	2667	234.5	244.4	3708	903.8	16.2	9.7
uh	3106	219.2	228.4	4439	828.3	14.9	9.6
oh	17684	1498.8	1561.5	31159	9048.8	162.5	9.1
Bye	1145	102.6	106.9	1284	196.9	3.5	8.6
bye	1233	110.8	115.4	1529	320.0	5.7	8.0
bitch	1643	146.3	152.4	2495	629.8	11.3	7.6
sorry	8127	726.5	756.8	15558	5234.7	94.0	7.4
yes	18823	1667.0	1736.8	37498	12833.4	230.4	7.3
darling	1364	119.9	124.9	1971	495.4	8.9	7.1
honey	1588	140.9	146.8	2689	681.6	12.2	7.1
you	336328	30020.7	31276.7	759009	251027.5	4507.6	6.9

Fig. 2. Top keywords of spoken component of New Model Corpus, as computed and presented in the Sketch Engine, with simple-maths parameter of 10. Component parts (*won, don)* of contracted forms removed.

Engine: one can submit queries which return concordances, word sketches, word lists and thesaurus entries.[1]

2.2 Corpora available in the Sketch Engine

We specialise in large general-language corpora (as required for lexicography). We have publicly-accessible corpora of over 5m words for twenty-six languages (including all major world languages), with over 1 billion words for three: see Table 1.[2] We have a 'Corpus Factory' (Kilgarriff et al 2010) programme for adding to the list of languages in our repertoire by preparing 100m word corpora from web sources, using BootCat methods (Baroni and Bernardini 2004).

Arabic (MSA)	174	Persian	6
Chinese (simp and trad)	456	Portuguese	66
Czech	800	Romanian	53
Dutch	128	Russian	188
English	5,508	Slovak	536
French	126	Slovene	738
German	1,627	Spanish	117
Greek	149	Swedish	114
Hindi	31	Telugu	5
Indonesian	102	Thai	108
Irish	34	Vietnamese	174
Italian	1,910	Welsh	63
Japanese	409		
Norwegian	95		

Table 1: Languages, and the largest corpus available for that language in the Sketch Engine (April 2010, figures in millions of words+punctuation)

3 The Merits of Big, High-Quality Corpora

Since Banko and Brill (2004), it is entirely clear that corpus-based NLP methods tend to perform better, the bigger the corpus. This is one reason for wanting a big corpus. Another is simply to have ample data even for rare phenomena. A third is that a very large corpus will have many large subcorpora. If, for example, we wish to look at Business English, or medical English, or informal English, we can build a classifier to distinguish text of this type from others, and then apply the classifier to a very big corpus, which will then give a subcorpus large enough to support research and model-building for the specific variety.

Corpus quality is less discussed than corpus size. It is harder to define and measure. Also, if people become aware of bad data in their corpus, they are more likely to remove it than announce it. Data cleaning is not high-status work, and papers are likely to pass over it lightly at best, either ignoring the failings of the dataset or presenting results after obvious anomalies have been excluded. Thus a paper which describes work with a vast web corpus of 31 million web pages devotes just one paragraph to the corpus development process, and mentions de-duplication and language-filtering but no other cleaning (Ravichandran, Pantel, and Hovy 2005, section 4). Another paper using the same corpus notes, in a footnote, "as a preprocessing step we hand-edit the clusters to remove those containing non-English words, terms related to adult content, and other webpage-specific clusters" (Snow, Jurafsky, and Ng 2006).

Academic papers do not often present results which compare performance on 'better' and 'worse' corpora. Nonetheless, few would dispute the near-tautology that better corpora are likely to give better results. There are many forms that bad data in corpora can take. They include duplicates, navigation bars and other web material, long lists, logfiles, code, texts in the wrong language, and language-like computer-generated spam. (There are other issues about texts in the correct language but which introduce unwanted biases because there are so many of them. Almost all general-language corpora have this problem, at least from some users' perspective.)

We are corpus specialists. We have explored in depth the issues of web data cleaning (Baroni et al 2008), character encoding (Kilgarriff et al 2010) and de-duplication of large datasets (Pomikalek et al 2009). People accessing our corpora will very often be accessing bigger and better corpora than would otherwise be possible.

3.1 The Google/Yahoo/Bing option

A number of researchers have followed the lead of Grefenstette (1999) and gathered data through extensive querying of one of the main search engines (in Grefenstette's case, Altavista, now usually Google, Yahoo or Bing); see for example Keller and Lapata (2003), Nakov and Hearst (2005), Nakov (2008). The search engines access far more data than we do even in our largest corpora: as against our 5.5 billion, Google indexes at least a trillion words of English. Search engines can be used as a corpus query tool, and if size of data is the overriding consideration, we can offer no alternative. However there are numerous disadvantages to using Google, Yahoo or Bing in this way:

- they are not linguistically aware so do not permit e.g., searches for lemmas
- the query syntax is limited (and subject to change without notice)
- they limit the number of queries one can make
- they limit the number of results per query
- results are sorted according to a scheme which bears no relation to a linguist's wish to see a random sample
- results are not replicable.

(For a full critique, see Kilgarriff 2007.) Using the search engines is a solution with many downsides: if a corpus of 5.5 billion words (for English) is big enough (and for very many, though by no means all, kinds of

[1] Full documentation at http://trac.sketchengine.co.uk/wiki/SkE/Methods/index

[2] We collaborate with numerous groups, and some corpora were built by others, in particular Serge Sharoff at the University of Leeds, UK, and Marco Baroni, Silvia Bernardini, Adriano Ferraresi and colleagues at the Universities of Bologna and Trento, Italy.

research it will be) then there are many advantages to using a specialised service for linguists, such as the Sketch Engine, rather than a search engine.

3.2 New English Corpora 1: BiWeC

BiWeC (Big Web Corpus, Pomikalek et al 2009) is a response to the ongoing need for bigger corpora, and to bridging the gap between corpora that are available in corpus query tools and the web as available via search engine indexes. Our target is 20b words, perhaps 1% of the non-duplicate textual data indexed by Google (see Kilgarriff 2007 for more on relative sizes of large corpora and Google indexes). Our work here has focused on, first, efficient crawling, and then, high-accuracy data cleaning and de-duplication. At time of writing, 5.5 b words have been fully cleaned, de-duplicated, lemmatised, POS-tagged, and loaded into the Sketch Engine.

3.3 New English corpora 2: New Model Corpus

The British National Corpus[3] has been very widely used across linguistics and language technology, and has often been held up as a model for how to design a corpus. However it was designed in the 1980s, before the web existed, and the model, as well as the data, is out of date (for the case in full see Kilgarriff et al 2007).

The next question is: what does a contemporary model corpus look like? The New Model Corpus is a response, comprising 100m words gathered entirely from the web but with proportions of different text types not unlike those of the BNC. It is available for research, and we plan to annotate it as a community-wide exercise, with all NLP researchers invited to download the data, process it with their tools, and return their annotations to us. We shall then integrate the annotations to give a multi-annotated corpus which will also be available for research.

4 A Case Study: TEDDCLOG

TEDDCLOG (Taiwan English Data Driven CLOze test Generation) is a system which drafts fill-the-gap exercises (sometimes known as cloze tests) for learners: the learner is given a sentence in which one word (the *key*) has been replaced with a gap, and a choice of four or five words (the key plus three or four *distractors*) to fill the gap. Exercises of this kind are popular with teachers of English and also with language testing organisations. However the tests are usually based on invented sentences, created by human ‘test item writers’. There is a now well-chronicled tendency for there to be a mismatch between the language of invented sentences and that found in corpora of naturally-occurring English.

4.1 The Algorithm

TEDDCLOG uses the following algorithm:

1. User inputs the key
2. Look up the key in the thesaurus to find distractors
3. Find collocates for the key in its word sketch
4. Find a collocate which is used with the key but not with any distractors (the ***koc***, key-only collocate)
5. Find a short simple sentence containing key+koc
6. Prepare output: blank out key from sentence, present key and distractors in random order.

Steps 2-5 each use the web API. They are described in detail below.

4.2 The Corpus

We currently use UKWaC (Ferraresi et al 2008, 1.5 billion words) and may switch to BiWeC.

Size is important for two reasons:

1. A corpus has to be very large to provide more than a handful of sentences for most key-collocate pairings. With more to choose from, there is a better chance that there will be one which is short and simple.
2. It is critical that the distractors are not acceptable alternatives to the key, in the context provided by the sentence. If the corpus is big enough, then the absence of any occurrences of the koc with the distractors is evidence that they are not acceptable.

It would be possible to use a far larger corpus than UKWaC, by using the web as indexed by Google or Yahoo directly. This could give stronger evidence of the non-acceptability of distractors with the koc. (Sumita et al (2005) use a method of this kind.) However the use of the web in this way raises other difficulties as discussed above.

4.3 Worked Example

We want to test the use of the verb *react*. The writer enters *react* into the system.

Finding distractors: the Thesaurus Module

The API call to the thesaurus returns words which typically occur in the same context as the search term. Table 2 shows the SkE Thesaurus for *react*. (The table reveals that most of the words with similar distribution to *react* relate to the human-interaction uses of the word, probably because this is the most frequent kind of use of *react.*) The three top-ranking list members, *respond, interact* and *behave*, are noted and retained for use as PDs (potential distractors).

Finding the Key-only Collocate (Koc): Sketch Differences Module

The Sketch Engine also provides a “Sketch Difference” or *sketchdiff* display, showing which collocates are shared (and “how shared they are”) and which are not, between two similar words. Figure 4 shows the sketch differences for *react* and *respond*. We see that *react* occurs 232 times with *positively* as a MODIFIER, and *respond*, 1624 times. The user can click on the number to see the 232, or 1624, concordance lines.

[3] http://natcorp.ox.ac.uk

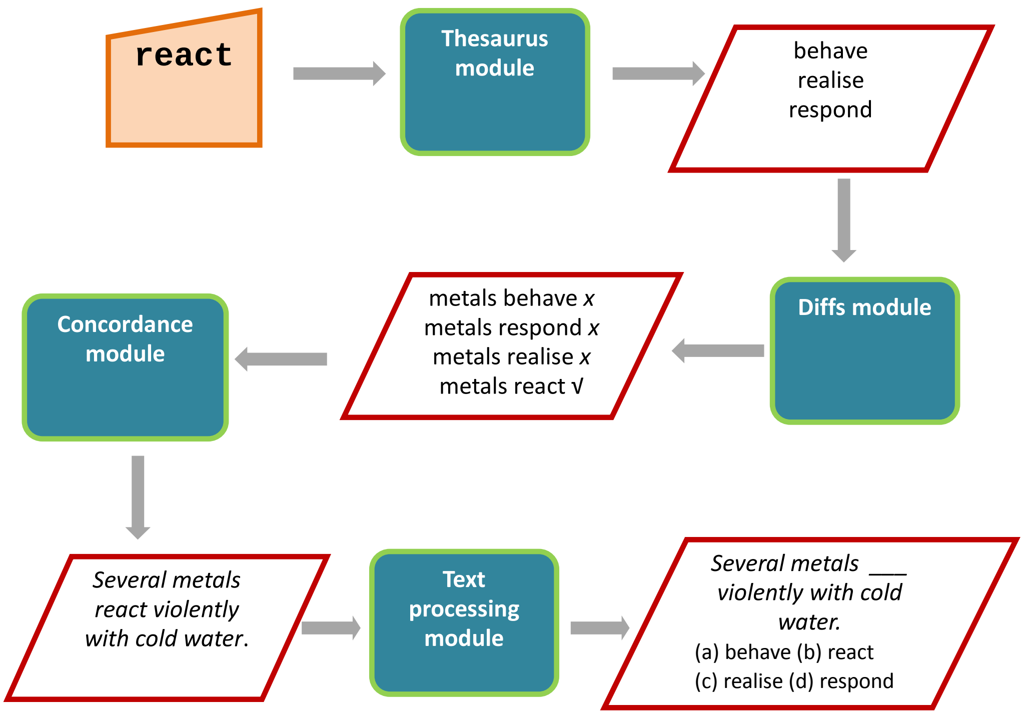
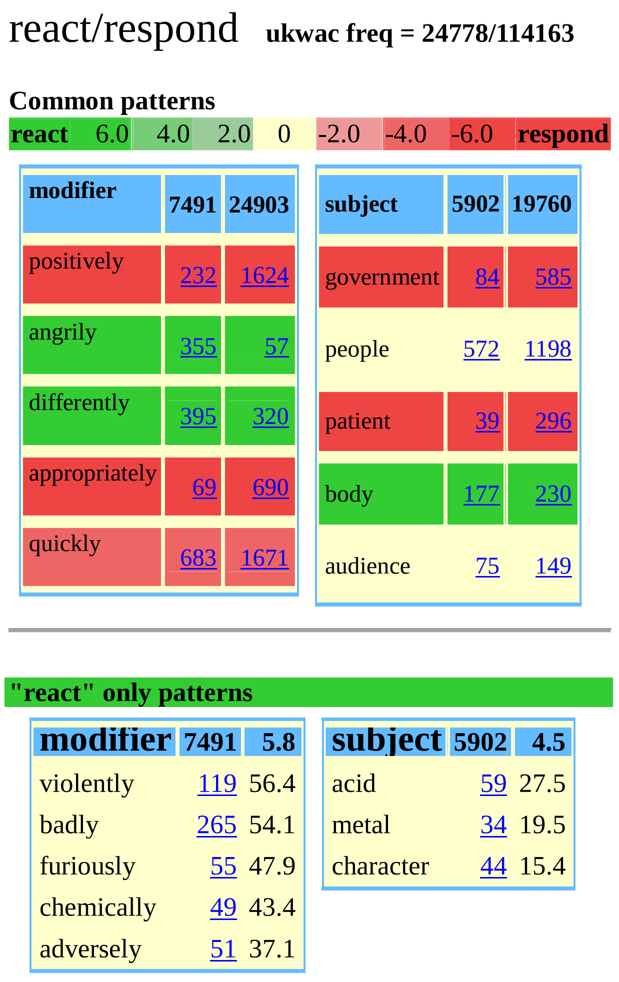


Figure 3. TEDDCLOG System architecture

react **ukWaC freq = 24778**

Lemma	Score	Freq
respond	0.417	114163
interact	0.305	25685
behave	0.296	24508
realise	0.25	110985
cope	0.247	48313
adapt	0.245	50930
listen	0.238	127002
answer	0.237	105714
intervene	0.237	14898
contribute	0.235	137428

Table 2: Distributional thesaurus entry for *react*.



react/respond **ukwac freq = 24778/114163**

Common patterns

react	6.0	4.0	2.0	0	-2.0	-4.0	-6.0	**respond**

modifier	**7491**	**24903**
positively	232	1624
angrily	355	57
differently	395	320
appropriately	69	690
quickly	683	1671

subject	**5902**	**19760**
government	84	585
people	572	1198
patient	39	296
body	177	230
audience	75	149

"react" only patterns

modifier	**7491**	**5.8**
violently	119	56.4
badly	265	54.1
furiously	55	47.9
chemically	49	43.4
adversely	51	37.1

subject	**5902**	**4.5**
acid	59	27.5
metal	34	19.5
character	44	15.4

Figure 4: Sketch Diff for *react* and *respond* (truncated).

TEDDCLOG needs collocates that are not shared with distractors (kocs). Candidate kocs can be seen under the "*react* only" patterns. TEDDCLOG takes the high-salience collocates that do not occur with the first distractor, applying the condition that the collocate must be a correctly spelled English word and not a proper name.
In the simplest case, the first candidate koc does not co-occur with any of the three distractors. In other cases, TEDDCLOG either finds new candidate kocs until one is found that does not occur with any of the distractors, or, depending on parameter settings, finds new distractors from the thesaurus that do not occur with a potential koc. The process is continued until we have a koc, and set of distractors that do not occur with it.
At this point in the algorithm, we have decided on the key and three distractors. We have also established that we wish our carrier sentence to include the collocation: in our example, *metals react.* The next step is to determine what the carrier sentence will be.

Selection of Carrier Sentence
The carrier sentence needs to contain *metal* as subject of *react.* There are 34 such sentences in UKWaC. The next task is to choose the most suitable for a language-teaching, cloze exercise context.
Many sentences are unsuitable, for a range of reasons. For example:

> 2H 2 O 2(aq) == 2H 2 O (l) + O 2(g) or a metal **reacting** with acids, and you can study the effects of a catalyst e.g. adding Cu 2+ (aq) ions to a zinc-acid mixture, though I 'm not sure easy it is to get good quantitative results for advanced level coursework?

Firstly, the sentence is too long, giving the learner work to do which is not directly related to the task that the exercise assesses. Secondly, it contains formulae which will be incomprehensible to non-chemists. Another example is:

> It uses these reactions to explore the trend in reactivity in Group 1. The Facts General All of these metals **react** vigorously or even explosively with cold water .

Here, the problem is that we have not one sentence but two, and a heading and subheading in between. The corpus processing has been led astray by the period following the 1, interpreting it as part of the token "1." rather than as an end-of-sentence marker, and has also failed to mark off the heading ("The facts") and subheading ("General") as not being part of the following sentence.
Atkins and Rundell (2008) discuss the criteria for good examples in dictionary definitions, concluding that such examples must be intelligible to learners, avoiding difficult lexis and structures, puzzling or distracting names, and anaphoric references which cannot be understood without access to the wider context. These lexicographical desiderata are equally applicable to the selection of carrier sentences for cloze exercises. The SkE concordancing software is equipped with a feature called GDEX (Good Dictionary Example Extraction: Kilgarriff et al, 2008), which ranks sentences extracted from corpora according to the following criteria:

- Sentence length: a sentence between 10 and 25 words long is preferred, with longer and shorter ones penalized. (Overshort sentences may not provide enough context to show the user the intended meaning of constituent words.)
- Word frequencies: a sentence is penalized for each word that is not amongst the commonest 17,000 words in the language, with a further penalty applied for rare words.
- Sentences containing pronouns and anaphors like *this that it* or *one* often fail to present a self-contained piece of language which makes sense without further context, so sentences containing these words are penalized.
- Sentences where the target collocation is in the main clause are preferred (using heuristics to guess where the main clause begins and ends, as we do not yet use a parser).
- Whole sentences – identified as beginning with a capital letter and ending with a full stop, exclamation mark, or question mark, are preferred.
- Sentences with 'third collocates', that is, words that occurred with high salience in sentences containing the key and koc, are preferred. This will increase the chances that the context in which the collocation is shown is typical for the collocation.
- Sentences with more than two or three capital letters, and more than two or three punctuation marks and other non-alphanumeric characters, are penalized. This turns out to be a simple way of setting aside most aberrant and junk-filled 'sentences'.

GDEX sorts the concordance lines for any SkE search so that the 'best' sentences are presented first. The sentences which are most likely to be selected for dictionary examples or cloze exercises appear at the beginning of the concordance display. Unwanted sentences, including web noise, are relegated to the end of the concordance so a human user need not waste time looking at them.
TEDDCLOG uses the API with GDEX switched on to find the best sentence containing the key+koc collocation, here *metal* as subject of *react.*

Current status is that we have a prototype system (Smith et al 2009) and are developing a proposal in collaboration with a testing organisation, to turn it into an industrial-strength system.

5 Relation to CLARIN

The EU Project CLARIN[4] aims to establish research infrastructure for language technology, based on web services, and we have approached CLARIN regarding the role that the services discussed here might play. However it seems that CLARIN's perspective is on a longer term and more ambitious plane, with emphasis

4 http://www.clarin.eu

on standards and community-wide integration, rather than currently-available modest services as here.

6 Summary

We have made the case for 'corpora by web services' with NLP researchers using corpora without needing to store them on their local machines or expend effort on building or maintaining them or associated software. In this way researchers will be able to make use of larger and better corpora than is otherwise possible. The Sketch Engine is a very fast and flexible corpus query tool, into which many large corpora for many languages are already loaded, with a web API, so we are already set for 'Corpora by Web Services' and indeed we already have some users developing NLP applications in this way.

For English, we are developing two new resources with 'Corpora by Web Services' in mind: firstly BiWeC, which moves the scale of resource we offer up by a scale of magnitude, and second, the New Model Corpus, with which we hope to update the BNC as a reference corpus for English. All being well, these two projects will come together in a very large, very well marked up corpus for English which is fully accessible by web API. Using a corpus will not merely be like picking up a hire car, it will be like picking up a Ferrari.

7 References

Banko, Michele, and Eric Brill 2001. Scaling to Very Very Large Corpora for Natural Language Disambiguation. *Proc ACL.* Toulouse, France.

Baroni, Marco and Silvia Bernardini 2004. BootCaT: Bootstrapping Corpora and Terms from the Web. *Proc LREC*, Gran Canaria.

Baroni, Marco, Francis Chantree, Adam Kilgarriff and Serge Sharoff 2008. CleanEval: a competition for cleaning web pages. *Proc LREC.* Marrakech, Morocco.

Ferraresi, Adriano, Eros Zanchetta, Silvia Bernardini and Marco Baroni 2008. Introducing and evaluating UKWaC, a very large web-derived corpus of English . *Proc. 4th WAC workshop*, LREC, Marrakech, Morocco.

Grefenstette, Gregory. 1999. The WWW as a resource for example-based MT tasks. In *ASLIB Translating and the Computer Conference*, London.

Keller, Frank and Mirella Lapata. 2003. Using the web to obtain frequencies for unseen bigrams. *Computational Linguistics*, 29(3):459–484.

Kilgarriff, Adam 2007. Googleology is Bad Science. *Computational Linguistics* 33 (1): 147-151.

Kilgarriff, Adam, Sue Atkins and Michael Rundell 2007. BNC Design Model Past its Sell-by. *Proc. Corpus Linguistics*, Birmingham, UK.

Kilgarriff, Adam, Milos Husák, Katy McAdam, Michael Rundell, Pavel Rychlý 2008. GDEX: Automatically finding good dictionary examples in a corpus. *Proc EURALEX*, Barcelona, Spain.

Kilgarriff, Adam, Siva Reddy, Jan Pomikalek 2010. A Corpus Factory for many languages. *Proc LREC*, Malta.

Nakov, P. 2008. Noun compound interpretation using paraphrasing verbs: Feasibility study. *Proc. Artificial Intelligence: Methodology, Systems, Applications* (AIMSA'08).

Nakov, Preslav and Marti Hearst. 2005. Search engine statistics beyond the n-gram: Application to noun compound bracketing. *Proc. Computational Natural Language Learning (CoNLL-2005)*, pages 17–24, Ann Arbor, Michigan.

Pomikalek, Jan, Pavel Rychlý and Adam Kilgarriff 2009. Scaling to Billion-plus Word Corpora. Advances in Computational Linguistics. Special Issue of *Research in Computing Science* Vol 41, Mexico City.

Ravichandran, Deepak, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and NLP: Using locality sensitive hash functions for high speed noun clustering. In *Proc. ACL*, Ann Arbour, Michigan, USA.

Smith, Simon, Adam Kilgarriff, Scott Sommers, Gong Wen-liang, Wu Guang-zhong Automatic Cloze Generation for English Proficiency Testing *Proc. LTTC International Conference on Language Teaching and Testing*, Taipei, Taiwan.

Snow, Rion, Daniel Jurafsky, and Andrew Ng. 2006. Semantic taxonomy induction from heterogeneous evidence. In *Proceedings of ACL*, Sydney

Sumita, E., Sugaya, F. and Yamamoto, S. 2005. Measuring Non-native Speakers' Proficiency of English by Using a Test with Automatically-Generated Fill-in-the-Blank Questions. *Proc. 2nd Workshop on Building Educational Applications using NLP*, Ann Arbor.

An Open Service Framework for Next Generation Localisation

David Lewis, Stephen Curran, Dominic Jones, John Moran, Kevin Feeney
Centre for Next Generation Localisation
Knowledge and Data Engineering Group
School of Computer Science and Statistics
Trinity College Dublin
College Green, Dublin, Ireland
E-mail: {Dave.Lewis|Stephen.Curran|Dominic.Jones|John.Moran|Kevin.Feeney}@scss.tcd.ie

Abstract

The localisation industry makes strong use of language processing pipelines at the core of its bulk localisation workflows, where software text and technical manuals are translated into the languages of target markets. Natural language technologies such as machine translation and text analytics are now maturing to a stage where they are being adopted as components in these workflows. However, they also offer the opportunity to broaden the localisation business into domains where the source content is less predictable and produced and consumed more rapidly and in higher volumes by a wider range of users. To exploit the business opportunities of such Next Generation Localisation, the localisation industry must adopt a more flexible, extensible and lower cost mechanism for the integration of language processing workflows across many, increasingly specialised players. This paper outlines an open services framework that is being developed by the Centre for Next Generation Localisation that will allow industry to react rapidly to changing business models and new opportunities by exploiting service oriented architectures for service reuse and (re)composition, extensible meta-data driven interoperability and flexible service and workflow management capabilities.

1. Introduction

Localisation is the industrial process of adapting digital content to culture, locale and linguistic environment (Johnson 2007). It is a key enabling, value adding, multiplier component of global manufacturing, services, software and content distribution industry so as a business process it must be conducted at high quality, speed, volume and low cost. The localisation industry makes strong use of language processing pipelines at the core of its bulk localisation workflows, where software text and technical manuals are translated into the languages of target markets. These language processing workflows have been well tuned to this domain by the various players in the value chain, such as the multinationals that are high volume generators of content requiring localisation and the Language Service Providers (LSPs) that provide outsourced localisation services, including the management of the translation of textual content. The business drivers in this industry produce workflows that are driven by the cost reduction needs of bulk publishers, resulting in little innovation into new business areas or applications.

Natural language technologies such as machine translation and text analytics are now maturing to a stage where they are being adopted as components in these workflows. However, they also offer the opportunity to broaden the localisation business into domains where the source content is less predicable and produced and consumed more rapidly in higher volumes by a wider range of users. The potential for innovation for the localisation industry exists in several directions;

- Outwards: addressing language as the next big barrier to be overcome in the use of the Internet for global communication and value generation
- Inward to focus on the need of the individual consumer through personalisation, i.e. the tailoring of the delivery of content, not only to the users locale but also to their personal content consumption preferences and their current physical, social and task context.
- Sideways into other corporate activities of existing knowledge- and service-intensive localisation clients, e.g. customer care and customer relations management or leveraging Web 2.0 technologies to engage with crowd-sourcing or open innovation value networks.

We identify such a shift and broadening in the localisation industry as *Next Generation Localisation*. This will involve making the workflows for linguistic processing and translation much more customer driven, rather than product driven as currently. It will require dealing with a much wider range of content sources, including user generated content and highly transient content that provides much of the value found in Web 2.0. It will also involve leveraging a wider range of linguistic human skills and value exchange models, beyond the scope of today’s professional translators.

This presents a major challenge in systematically integrating fine-grained, on-demand quality into web content and web application localisation. This requires integrating mechanisms to determine and deliver quality, reliability and speed that match immediate user requirements into such web offerings. Though linguistic technologies allow us to automate some tasks, such as machine translation or entity recognition, the bounds in the confidence of the quality of outcomes needs to be understood and carefully managed. Key to this is empowering the user to assess that quality and demand more if required and indicate the level of quality they are willing to pay for in a given context. Content owners must then be able to adaptively tailor allocation of localisation resources (whether human or automated) to a wider and more dynamic range of quality targets.

To exploit the business opportunities of such Next Generation Localisation, industry must adopt a more flexible, extensible and lower cost mechanism for the integration of language processing workflows across many, increasingly specialised players. Service Oriented Architectures (SOA) offer a viable route to addressing this challenge. This paper outlines an Open Service Framework that is being developed by the Centre for Next Generation Localisation (www.cngl.ie) that harnesses the power of SOA to enable industry to react rapidly to changing business models and opportunities through service reuse and composition, extensible meta-data interoperability and flexible service and workflow management capabilities.

2. Background

The localization industry has already undertaken a number of separate document focussed standardization activities to support interoperability between different localisation applications. The Localisation Industry Standards Association (LISA – www.lisa.org) has developed various localisation standards:

- *Translation Memory Exchange (TMX)* for exchanging Translation Memory (TM) database content (TMX 2005). Such content is key in eliminating the re-translation of content segments that have previously been translated. TMs also support fuzzy matches, where translations of similar source segments can be considered by translators. Many TM tool providers have implemented support for TMX in their products.
- *Term Base eXchange (TBX):* XML Terminology Exchange Standard, to allow terminology to be exchanged between content author and translator tools (TBX 2008). An XML linking standard for terms, called Term Link, is also being investigated.
- *Segmentation Rules eXchange (SRX),* for exchanging the rule by which content is originally segmented. There has been very little support to date for SRX because segmentation is the main component that distinguished TM tools. Segmentation has direct consequences for the level of reuse of a TM. A TM's value is significantly reduced without the segmentation rules that were used to build it.
- *Global information management Metrics eXchange (GMX)*: A partially populated family of standards of globalization and localization-related metrics

The Organization for the Advancement of Structured Information Standards (OASIS – www.oasis-open.org), which produces e-business standards has had a number of initiatives, the most notable being XML Localisation Interchange File Format (XLIFF 2008). XLIFF is the most common open standard for the exchange of localisable content and localisation process information between tools in a workflow. Many tool providers have implemented support for XLIFF in their products.

The W3C, which develops many core web standards, has an Internationalisation Activity (www.w3.org/International) working on enabling the use Web technologies with different languages, scripts, and cultures. Specific standardisation includes the Internationalisation Tag Set to support internationalisation of XML Schema/DTDs (ITS 2007).

To date, therefore, though file interoperability is supported in places, standard localisation processes and workflows and associated open interfaces addressing common interoperability issues have not yet been widely adopted. Outside of proprietary scenarios, digital publishers and service providers cannot easily integrate their processes and technologies and monitoring end to end process performance is extremely difficult. This implies lost business opportunities for many and missed opportunities for significant performance improvement for most of the stakeholders.

SOA coupled with workflow technologies are therefore well placed to address this lack of interoperability and end-to-end process management. Anecdotal evidence suggests that elements of the localisation industry were quick to consider the use of Web Service technology. In 2003 Bowne Global Services presented a case study (Reynolds 2003) showing how they connected Interwoven's TeamSite Content Management System (CMS) to their in-house workflow engine (then named Elcano) using Web Services. IBM also presented a white paper discussing how web services and workflow management feature such as supported by their WebSphere product range could stream line the localisation process (Flinter 2003). However, these were focussed on integration within the enterprise, and end to end web service solutions have been slow to emerge, though several tools now make internal APIs available via Web Services for enterprise integration and support of custom client applications, e.g. for accessing TM content. Two examples of interfaces provided to human translation services are those provided by Translated.net and by Lionbridge to their Freeway system. Web Service interfaces to Machine Translation systems are more straightforward due to less branching logic and, as such, more common. Examples include the WebSphere Translation Server and the Google Translate API.

In 2007 the OASIS Translation Web Services (TWS) 1.0.3 draft specification (Reynolds 2007) was released with the aim of standardising the communication between translation providers and their clients (Reynolds 2003)(Bargary 2006). TWS remains the only real attempt to define web-services to support the end to end localization process. However, TWS has a limited scope. Rather than aiming to support the dynamic composition of language services into flexible localization workflows, it concentrates on supporting the negotiation of “jobs” between service providers. It is primarily intended to support the efficient out-sourcing of localization and translation jobs and it does not address the composition of language-services to form automated workflows. It is not clear to what extent this draft specification has found traction in industry to date.

3. Open Service Framework

Therefore, in order to deploy web-services to support such composition, there is little standardisation to rely on. Thus, a first step in addressing the problem is to design a set of web-services and their interfaces suitable for the task. In designing these services, it is worthwhile to recall the general goals of service-oriented architectures; the services should be designed to be as flexible and general as possible and they should neither be tightly coupled to one another, nor to the overall system which they are part of. Furthermore, in keeping with the general trends in service designs (Foster 2008), variability in service behaviour should generally be supported through the passed data-structures rather than through different function signatures.

Our ultimate aim is to establish a Unified Localisation Factory (ULF) that will enable future web content and service providers of all sizes instantiate localisation processes tailored to their needs and those of their customers. The ULF will allow future localisation-focussed applications that leverage advanced language and digital content management technologies to be rapidly integrated at low cost. This requires an Open Service Framework for presenting and assessing individual technologies, applications, evaluation techniques, design patterns, interoperability standards and workflows is a SOA.

This framework will consist of the following;

- *Core Principles and Concepts:* The core shared domain knowledge that characterise the vision of Next Generation Localisation.
- *Process Map:* A Business Level Reference Framework expressed using business process modelling concepts.
- *Methods and Techniques:* The procedural guidance needed to apply the framework, to evaluate that application and to contribute to the refinement of the framework in an open manner.
- *System Services Architecture:* The software system architecture needed to ground the application of the framework in operational software systems.
- *Reusable Elements:* Specifications, models, service definitions, APIs, software components and various forms of design patterns (e.g. for workflow, software integration, SOA etc) that can be used in a specific applications of the Open Service Framework.

4. Next Generation Localisation Process Map

As the Open Services Framework aims to support interoperability across next generation localisation workflows consisting of multiple parties and their various services and applications, then ultimately it must support this through the definition of common meta-data. The benefits from the identification of common meta-data models in a particular domain are to provide the foundation for interoperability standards. Such standards thereby:

- Reduce cost of system integration
- Support multi-vendor system architectures, increasing the benefits of vendor competition by reducing lock-in for the different process actors
- Maximise the reuse of data and processes and the software services that underpin them.

However, localisation, in common with many application domains, possesses multiple stakeholders operating multiple systems in multiple interlinked business processes. These factors complicate efforts towards convergence and agreement on common industry-wide meta-data. Attempting a programme of meta-data modelling for interoperability standards therefore requires a common business-level reference framework in order to understand and discuss the different data and meta-data requirements at different interoperability points.

Other industries have successfully used a Process Map as a business level reference framework within which detailed business process definitions, and thereby specific interoperability models, can be worked upon within a shared set of terms and associated meanings. This requires an abstract process map that is not a reflection of any one company's model and therefore provides neutral means for discussing shared interoperability concerns. Examples of the use of such process maps in other industries are: the electronic Telecommunication Operations Map (eTOM) used by the TeleManagement Forum to support industry interoperation agreements between vendors of telecommunications management packages (Reilly 2009). Another example is the Smart Building Process map used to enable standardization of the exchange of data between CAD tools and building operations tools in the construction and facilities management industry (SmartBuilding).

Currently the Localisation Industry does not possess such an agreed process map, so as part of the Framework we propose a novel 'Next Generation Localisation' Process Map. The scope of this should be business processes covered by our broad vision of Next Generation Localisation, beyond conventional localisation workflows into areas of: crowd-sourcing, integrated language technologies such as machine translation, speech processing and text analytics that use statistical approaches; information retrieval; digital content management and personalization; web service development and governance. The NGL Process Map provides a top down common analytical frame within which specific business scenarios can be modelled. By overlaying specific business process flows of the process map we can start to identify where: existing standards such as XLIFF, TMX can be applied and if necessary extended and where new meta-data agreements are needed, the proposal of which is an activity within CNGL. The NGL Process Map therefore acts as a stakeholder-neutral medium to communicate requirements, seek solutions and contextualise the design and agreement of interoperability standards. Of course the structure of the Process Map itself will therefore influence the direction of such deliberation, so we remain open to proposals to refine this structure.

The structure currently proposed for the Process Map consists of an orthogonal grid of abstract Stakeholder Types ranged against shared Business Process areas. The Business Process areas represent recognisable collections of activities that span the localisation business process lifecycle and includes the processes related to process improvement. The areas can be individually populated with more specific processes, both for abstract business modelling and for capturing specific business scenarios. The Stakeholders differentiate the broader range of actor types involved in Next Generation Localisation, ranging beyond those just concerned with the generation and translation of content to include directly the consumer, online communities and software developers. These can in turn be specialised as niche stakeholders are identified or when applying the grid to a specific concrete business scenario. In this the process map can be used at both an abstract industry-wide level and for the analysis of specific business relationships and their interoperability requirements. The orthogonal structure breaks the domain down into a set of regions, the boundaries between which become the primary foci for capturing requirements. As the map is used as a frame for specific business scenarios and their associated process flows, solutions to interoperability issues at these boundaries can be collated and after review combined into broader interoperability models at the top levels from which future concrete scenarios spanning the same boundaries can select appropriate solutions.

In the current Process Map the two orthogonal axes are defined as follows:

Stakeholder Types:

- *Corporate:* This contains processes performed by organisations employing workers in a professional capacity. It encompasses any processes that are performed for monetary exchange, thereby encompassing public bodies and NGOs. Several specific sub-categories of this pool have been identified for the NGL domain: Content Generator; Language Service Provider; Translation Agency; Translation Sole Trader; Web Search Service Provider and Content Service Provider.
- *Consumer:* This contains processes conducted by the ultimate consumer of content. It is distinguished from other stakeholders in that it does not consume content for the purpose of providing it to other processes. Process for this stakeholder may annotate content to provide feedback to other processes, but only as a secondary activity to the consumption of that content.
- *Community:* This stakeholder represents processes that are subject to collective decision-making and content processing work performed by volunteers. It therefore excludes any activities performed for monetary reward directed to those performing it. The processes are distinct from those of the consumer in that they are indented to produce results of value to some other party and they are knowingly performed as part of a collaborative effort.
- *Service Developer*: This contains processes related to the development of new software services that can subsequently used by processes elsewhere in the process map. It excludes processes related to the localisation of that software, in such cases the processes should be conducted as part of the corporate stakeholder.

Business Process Areas:

- *Content Generation:* This includes authoring, internationalization and the development of terminology, domain models and content style guidelines.
- *Content Localisation:* Translating content from a source language to one or more target languages and making other locale specific changes to content.
- *Content Consumption:* The user driven consumption of content including search and personalisation of content.
- *Content/Asset Management:* The collection, storage, refinement and general husbanding of reusable digital assets, e.g. TM, term-bases, guidelines, user models, annotations, quality assessments etc.
- *Process Management:* The process involved in monitoring, analysing and modifying business processes with the view to improving performance metrics.

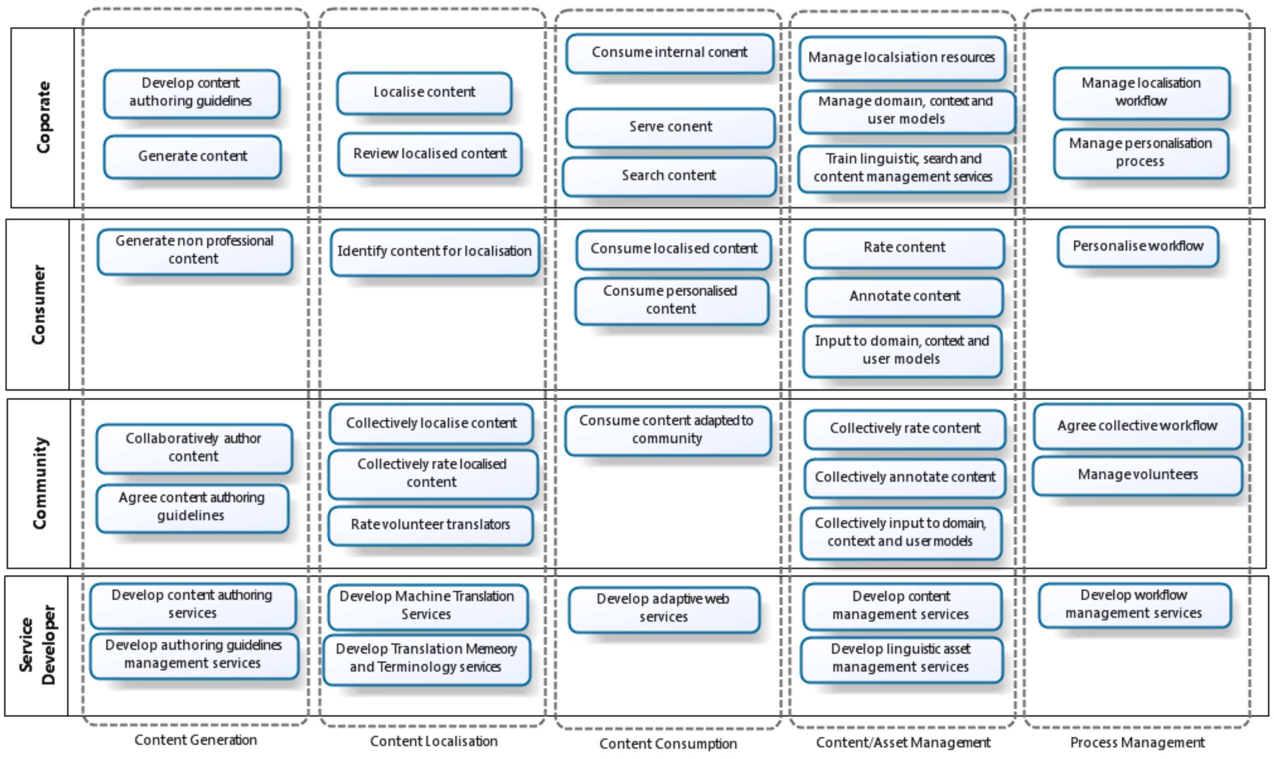

Coporate
Develop content authoring guidelines
Generate content
Localise content
Review localised content
Consume internal conent
Serve conent
Search content
Manage localsiation resources
Manage domain, context and user models
Train linguistic, search and content management services
Manage localisation workflow
Manage personalisation process
Consumer
Generate non professional content
Identify content for localisation
Consume localised content
Consume personalised content
Rate content
Annotate content
Input to domain, context and user models
Personalise workflow
Community
Collaboratively author content
Agree content authoring guidelines
Collectively localise content
Collectively rate localised content
Rate volunteer translators
Consume content adapted to community
Collectively rate content
Collectively annotate content
Collectively input to domain, context and user models
Agree collective workflow
Manage volunteers
Service Developer
Develop content authoring services
Develop authoring guidelines management services
Develop Machine Translation Services
Develop Translation Memeory and Terminology services
Develop adaptive web services
Develop content management services
Develop linguistic asset management services
Develop workflow management services
Content Generation
Content Localisation
Content Consumption
Content/Asset Management
Process Management

Figure 1: Next Generation Localisation Process Map high level processes in each region

Figure 1 outlines a set of processes that we have identified as populating the process map. The significance of this model is the emphasis given to activities beyond the traditional corporate work-flow, highlighting the important of managing the dynamic relationship with the consumer of content; of leveraging the collective intelligence of online communities and integrating the software service developer into SOA-based process improvement.

5. Language Processing Services Architecture

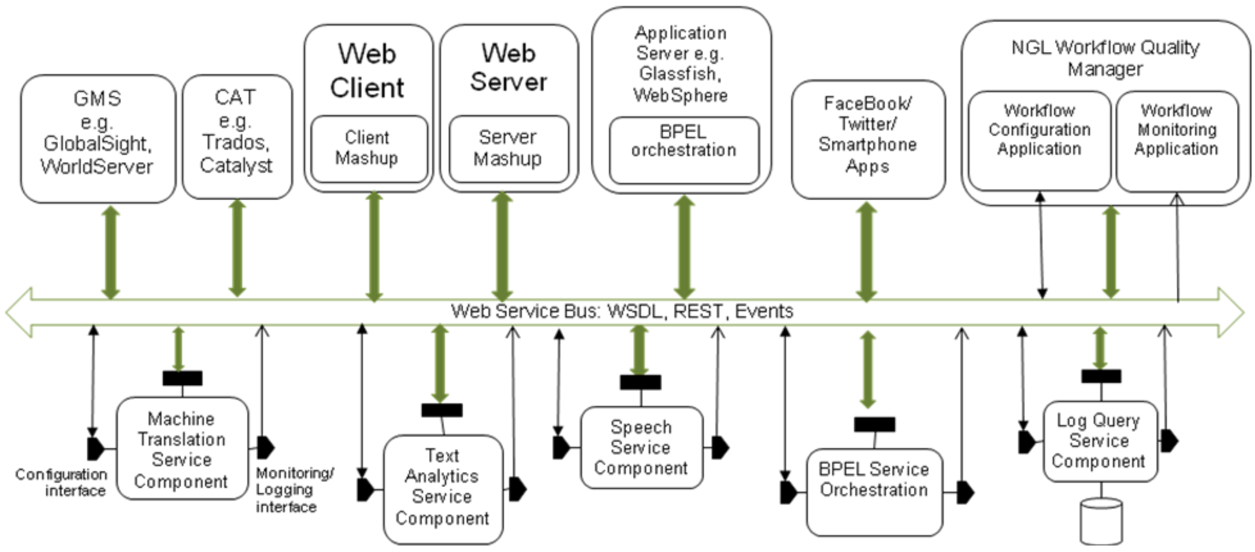
Adoption of an SOA advocates software integration through well defined functional interfaces that can be invoked remotely, typically using the Web's HTTP protocol with input and output parameters encoded in XML. The W3C have standardized an XML format, The Web Service Description Language (WSDL), for describing and exchanging such service definitions. Web services can be composed into more complicated applications using explicit control and data flow models that can be directly executed by workflow engines. This allows new workflow applications to be defined declaratively and immediately executed, thus greatly reducing the integration costs of developing new workflows and increasing the flexibility to modify existing ones. Such web-service based service composition is known as Web Service Orchestration. OASIS has a standardized web service orchestration language called the Business Process Execution Language (BPEL), which has resulted in the development of several commercial execution platform and BPEL workflow definition tools, which support workflow definition through drag-and drop interfaces. This approach has already been by the LanguageGrid project (Inaba 2007) for the rapid development of linguistic applications by defining BPEL orchestration of web services. These services offer access to language resources provided in a mutual manner by different academic and research organisations. Resources include parallel text, cross-lingual dictionaries, machine translators and morphological analysers. In our prior work in this area (Lewis 2008) we used BPEL composition of a machine translation service and a language identification service to integrate more flexible content handling into WorldServer, a well established localisation workflow management product from SDL Inc (www.idiominc.com). This work highlighted how linguistic processing services for localisation workflow can be readily abstracted into services that take a source language segment and either adds target language segment, sort target segments or annotate segment pairs. From these a wide range of specialised linguistic services can be derived and composed to address linguistic processing needs for localisation. However, the simplicity of this linguistic processing service taxonomy does not reflect the need to configure and train the systems underlying these services, this being the processes where core value is derived. Further, as modern linguistic processing is increasingly statistical, the monitoring of statistical performance analysis over the various source language content flow being localised becomes a vital part of the processes. For this reason we add to the core linguistic processing service interface taxonomy, two parallel abstract interface types:

- *A Service Configuration interface*: via which the component can be configured to operate in the desired manner, e.g. by providing a domain trained statistical model to a Machine Translation service component
- *A Service Monitoring and Logging interface* via which operational data about the performance of the component can be remotely monitored or locally logged. This interface has generic operations for the configuration of the behaviour of monitoring and logging, e.g. producing event on a threshold being reached.

These parallel interfaces are seen a essential to developing web services for Next Generation Localisation due to the key role played by ongoing process monitoring in the design and deployment of new, improved processes, tailored to a wider range of business, social and individual consumer needs. The definition of these interfaces may be implemented as simply as dedicated operations on a web service, though ongoing work of the Service Component Architecture group at OASIS (www.oasis-opencsa.org) and the web service activities at W3C (www.w3.org/standards/webofservices/) promise a more standardised mechanism for assembling web service components with multiple interfaces.

Figure 2 gives an indication of the different combinations in which systems can be assembled in accordance with the service architecture as a range of service components deployed and accessed from a variety of client applications. These could range from service invocations made by existing Globalisation Management Systems (GMS) and Computer Assisted Translation (CAT) software. Platforms such as WorldServer and Trados from SDL and Catalyst from Alchemy already have extension APIs that allow invocation of third party services. A communication bus between the service components and the client applications based on WSDL/SOAP providing the best operational support for workflow-based clients, such as BPEL web service orchestration engines, which can then explicitly define fault and compensation handling workflow branches. However it is recognized that many applications may be better suited to RESTful service invocation models typical of web mash-ups, e.g. in JavaScript web browser clients or PHP web server modules. This mode of operation can also fit naturally with invocations from web application platforms such FaceBook and Twitter or for invocation from third party mobile software clients such as iPhone Apps, however in these cases WSDL client access is also commonly available.

The service components currently under development in the CNGL divide broadly into those that provide linguistic processing and those providing on-demand personalised access to multilingual content. The linguistic processing services such as Machine Translation, Speech Synthesis and Recognition and Text Analytics provide value that depends on statistical training over large volumes of data. The personalisation services adapt content to particular user preferences and context, based on rich meta-data assembled about the user, the context and the setting of the interaction. In both cases, the value of the service depends on appropriate configuration, which must therefore become a major element of any progressively improving workflow or application. At the same time, it must be acknowledged that statistical language processing and meta-data driven adaptation will never deliver complete accuracy in all cases, so integrated operational monitoring is needed to support ongoing reconfiguration of individual components through statistical retraining or improved meta-data modelling (which itself may rely on statistical techniques for semantic annotation of content or social network analysis of user activity). Therefore to support operational monitoring and process improvement for end-to-end workflows that span our NGL stakeholder types, shared data-model for monitoring and configuring the operation of the various server components, via the parallel interfaces identified above, will be essential.



GMS e.g. GlobalSight, WorldServer
CAT e.g. Trados, Catalyst
Web Client
Client Mashup
Web Server
Server Mashup
Application Server e.g. Glassfish, WebSphere
BPEL orchestration
FaceBook/ Twitter/ Smartphone Apps
NGL Workflow Quality Manager
Workflow Configuration Application
Workflow Monitoring Application
Web Service Bus: WSDL, REST, Events
Configuration interface
Machine Translation Service Component
Monitoring/ Logging interface
Text Analytics Service Component
Speech Service Component
BPEL Service Orchestration
Log Query Service Component

Figure 2: Potential configurations of the NGL Open Services Architecture

6. Example Next Generation Localisation Processes

In the CNGL project, we are applying this Framework as we develop a series of demonstrator systems that bring together advanced technologies from the Centre's academic partners and the current real world needs of its industrial partners or of broader real-world applications.

In relation to advancing the current localisation workflow with statistical machine translation and text analytics service, figure 3 shows an idealised reference workflow in the Business Process Modelling Notation that we are using to explore this area. It highlights how any instance of this workflow can exploit multiple instances of component sub-workflows or activities. The importance of intelligently selecting between these instances at various points in the workflow are then highlighted (the diamond, star-filled BPNM icon for a complex gateway decision). So for example a content publisher may select different instances of service providers who can perform the Translation Job sub-workflow. Within this, the service provider may divide the job and select different Translate Section sub-workflows, which employ different Translation Memories and then themselves make intelligent decisions about selecting between different Machine instances and post-editors on a segment by segment basis. After reassembly, the content provider may again select between different providers to perform the Review Job sub-workflow. Key to the aim of developing mature, optimising and transparent processes, as well as publishing the result, the quality checked translations and other process monitoring data are fed back passed to a process. One feedback path already in common practice assembles Translation Memories for future use. Another uses them to train future Statistical Machine Translation (SMT) instances, potentially by grouping content by domain or style to get more accurate results with more computationally efficient SMT engines. Also, feedback may be provided on problems encountered with terminology and content consistency. By supporting these steps with web services and using web service orchestration, configuration and monitoring, such decision making can become highly dynamic, itself being driven by statistical analysis of the content against domain categorisations. This accompanying reduction in process management overhead means this process can be conducted in a wider range of scenarios, beyond transitional bulk translation scenarios. For instance, cheap, or zero cost SMT coupled with suitably motivated crowd-sourcing for post-editing and reviewing tasks may move the value of the process away from translation and towards the intelligent husbanding of STMs and training of MTs, allowing niche operators who combine linguistic and natural language technology skills to emerge.

By way of example consider a new application developed in CNGL to demonstrate the integration of MT and crowd-sourcing in support of multilingual Twitter consumption. This takes Twitter feeds followed by a user and passes them through, first a language identification service, which then routes this to a language specific MT engine.

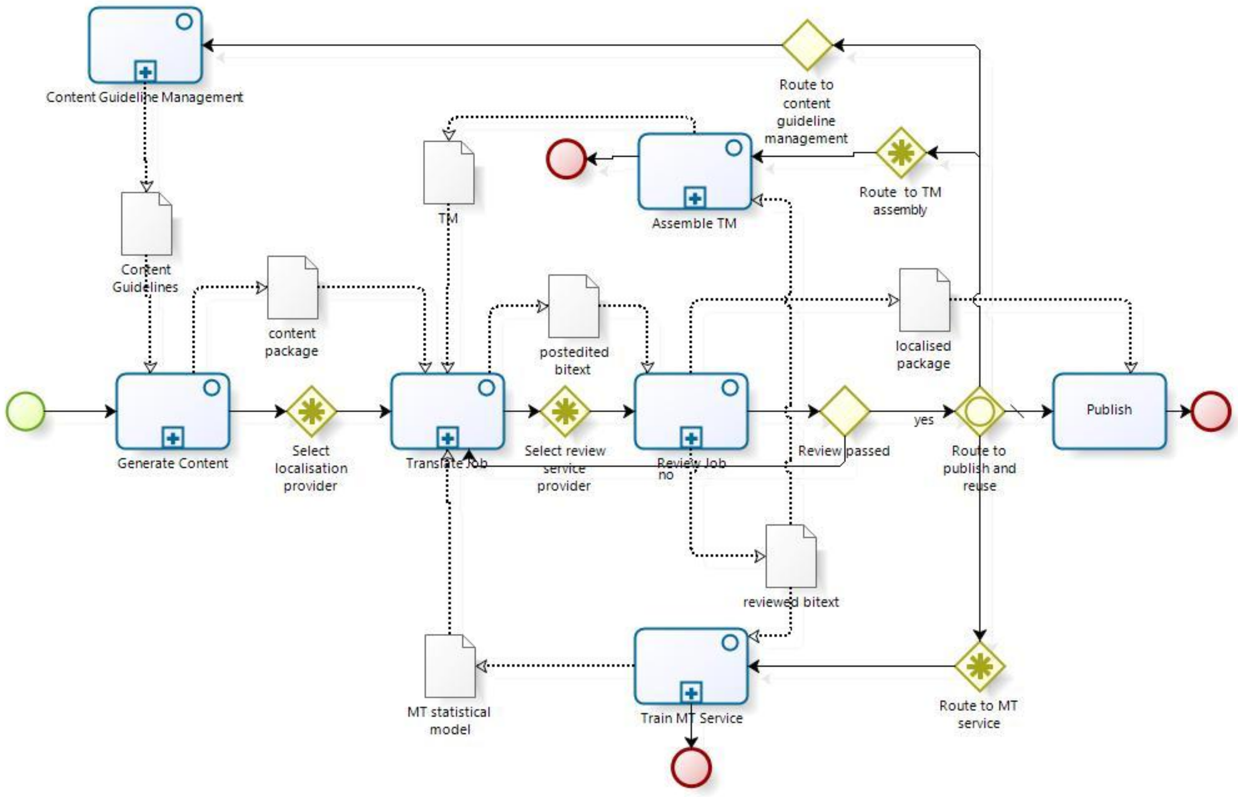
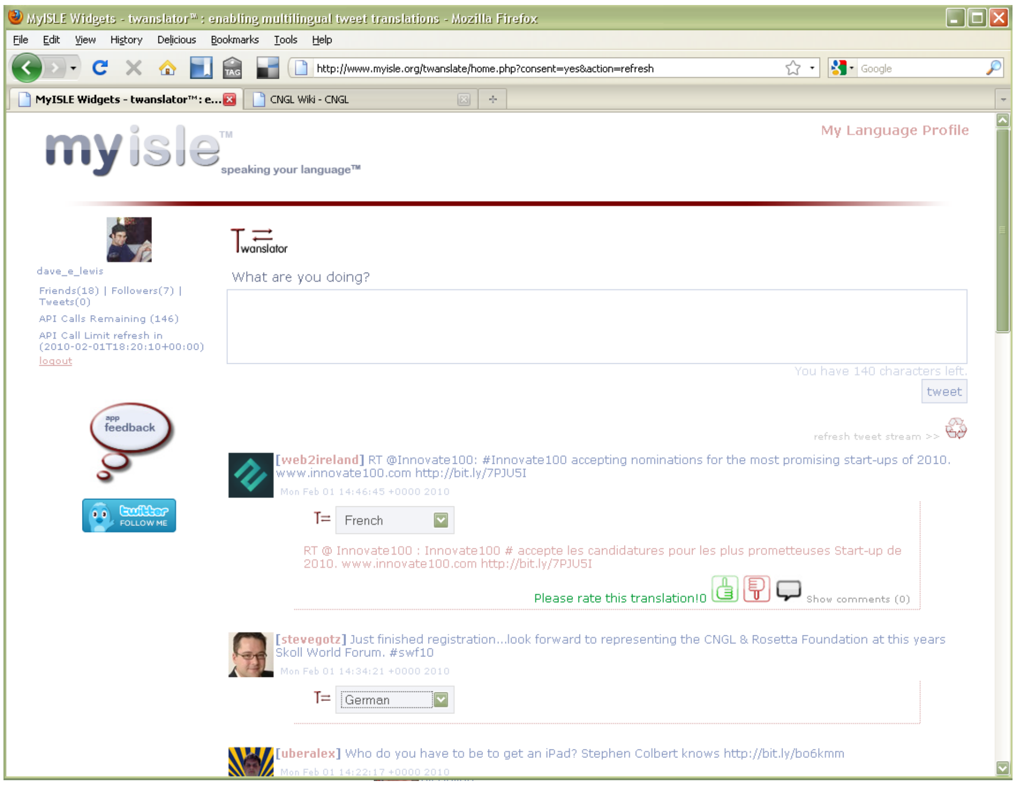


Figure 3: Generic Bulk Localisation Process Flow showing process feedback loops


MyISLE Widgets - twanslator™ : enabling multilingual tweet translations - Mozilla Firefox

File Edit View History Delicious Bookmarks Tools Help

http://www.myisle.org/twanslate/home.php?consent=yes&action=refresh

MyISLE Widgets - twanslator™ : e... | CNGL Wiki - CNGL

myisle™ speaking your language™

My Language Profile

dave_e_lewis

Friends(18) | Followers(7) | Tweets(0)

API Calls Remaining (146)

API Call Limit refresh in (2010-02-01T18:20:10+00:00)

logout

Twanslator

What are you doing?

You have 140 characters left.

tweet

refresh tweet stream >>

[web2ireland] RT @Innovate100: #Innovate100 accepting nominations for the most promising start-ups of 2010. www.innovate100.com http://bit.ly/7PJU5I

Mon Feb 01 14:46:45 +0000 2010

French

RT @ Innovate100 : Innovate100 # accepte les candidatures pour les plus prometteuses Start-up de 2010. www.innovate100.com http://bit.ly/7PJU5I

Please rate this translation!0 Show comments (0)

[stevegotz] Just finished registration...look forward to representing the CNGL & Rosetta Foundation at this years Skoll World Forum. #swf10

Mon Feb 01 14:34:21 +0000 2010

German

[uberalex] Who do you have to be to get an iPad? Stephen Colbert knows http://bit.ly/bo6kmm

Mon Feb 01 14:22:17 +0000 2010

Figure 4: Screen shot of MyIsle Twanslator App at http://www.myisle.org/twanslate

This reuses a BPEL sequence that combines a text analytics web service for language identification and a web service wrapper for the Matrex MT service, trained in specific language pairs. Though this orchestration was originally developed to demonstration their support for conventional localisation workflows in the WorldServer GMS (Lewis 2009), this application allows the same technology to be integrated into a crowdsourced post-editing setting. Here users are encouraged to rate and if willing to post-edit the machine translated Tweets, providing input to further training of the MT (see screenshot on figure 4). A simple initial gauging of the user's language competence allows this rating and post-editing dialogue to be personalised to the user and also to their willingness to participate in these steps. Though simply assembled in a few days as a mash up of the Twitter API and invocation of the BPEL web service orchestration, this application can now be easily scaled to a fully managed service that can be revised and tuned over time at low cost. Such Twitter applications therefore quickly enable the study of a number of issues key to NGL, i.e. collective content annotation, rating and translation (crowd-sourcing) across a social network; machine translation of perishable content and short text form content; social network informed personalisation of content querying and content translation and automated semantic annotation based on domain personalisation and text analytics.

7. Conclusion

We have described an Open Services Framework that we are developing to enable a broad range of Next Generation Localisation. We have emphasised the need for common meta-data to support web service interoperability as well as for the configuration and monitoring of systems via web services interfaces. The proposed NGL Process Map provides a semantic, process-oriented frame for discussing such meta-data agreements as new forms of NGL processes encompassing the broader set of stakeholders is explored. Our further work will involve expanding the range of example processes, including application in personalised multilingual customer care and social networking. Web service interfaces and associated meta-data will be harvested and common models proposed. Web service definition structures will be aligned with emerging service component architectures, including support for access control and business rules. Where meta-data requirements are volatile, we will adopt triple-base models based on the W3C's Resources Description Framework (www.w3.org/RDF) to allow rapid refinement of data models. We will also explore the deployment of compute-heavy processes such as MT training onto cloud computing environments.

8. Acknowledgements

This research is supported by the Science Foundation Ireland (Grant 07/CE/I1142) as part of the Centre for Next Generation Localisation at Trinity College Dublin.

9. References

Bargary, K. (2006) "Ignite implements Web Services", Localisation Focus, vol. 5, Issue 1, Pages 22-23, Ireland, 2006

Lewis, D., Curran, S., Feeney, K., Etzioni, Z., Keeney, J., Way, A., and Schäler, R. (2009). Web service integration for next generation localisation. In Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance For Natural Language Processing (Boulder, Colorado, June 05 - 05, 2009). ACL Workshops. Association for Computational Linguistics, Morristown, NJ, 47-55.

Flinter, S. (2003) "A J2EE based Localization Services Architecture", IBM Whitepaper, Jan 2003, accessed from ftp://ftp.software.ibm.com/software/globalization/documents/j2ee_lsa.pdf on 25 Feb 2010

Foster, I., Parastatidis, S., Watson, P., and Mckeown, M. 2008. How do I model state?: Let me count the ways. Commun. ACM 51, 9 (Sep. 2008), 34-41.

SmartBuilding "Information Delivery Manual Guide to Components and Development Methods", SmartBuilding Norway, accessed from http://idm.buildingsmart.no/confluence/download/attachments/446/IDM2_Methodology.pdf?version=1 on 25 Feb 2010

Inaba, R., Murakami, Y., Nadamoto, A., Ishida T. (2007) "Multilingual Communication Support Using the Language Grid" in Intercultural Collaboration, LNCS 4568, Springer, 13 Aug 2007, pp118-132

Internationalization Tag Set (ITS) (2007) Version 1.0 W3C Recommendation 03 April 2007, accessed from http://www.w3.org/TR/its/ on 25 Feb 2010

Johnson, D. (2007) "Getting Started in Localization", Multilingual, Oct/Nov 2007, pp 3-5

Reilly, J., Kelly, M. (2009) "The eTOM - A Business Process Framework Implementer's Guide", TM Forum

Reynolds, P. (2003) "Web Services for Translation", accessed from http://www.translationdirectory.com/article395.htm on 25 Feb 2010

Reynolds, P. et al (2007) "Translation Web Services - draft committee specification" OASIS Translation Web Services TC, Proposed draft specification 16 May 2007, access from http://www.oasis-open.org/committees/download.php/24350/trans-ws-spec-1.0.3.html on 25 Feb 2010

Systems to manage terminology, knowledge, and content – TermBase eXchange (TBX), Localization Industry Standards Association, 29 Oct 2008, access from http://www.lisa.org/TBX-Specification.33.0.html on 25 Feb 2010

TMX 1.4b Specification OSCAR Recommendation, Localisation Industry Standards Association, 26 April 2005, accessed from http://www.lisa.org/fileadmin/standards/tmx1.4/tmx.htm on 25 Feb 2010

XLIFF "A white paper on version 1.2 of the XML Localisation Interchange File Format (XLIFF)", Revision: 1.0, 17 Oct 2007 accessed from http://xml.coverpages.org/XLIFF-Core-WhitePaper200710-CSv12.pdf on 25 Feb 2010

XLIFF Version 1.2, OASIS Standard, 1 February 2008, accessed from http://docs.oasis-open.org/xliff/xliff-core/xliff-core.html on 25 Feb 2010

Web Communication Protocols for Coordinating the Modules of AnHitz, a Basque-Speaking Virtual 3D Expert on Science and Technology

Igor Leturia
Elhuyar Foundation
Zelai Haundi kalea 3
Osinalde Industrialdea
20170 Usurbil, Spain
i.leturia@elhuyar.com

Arantza del Pozo, David Oyarzun
Vicomtech
Mikeletegi pasealekua, 57
Miramon Teknologia Parkea
20009 Donostia-San Sebastian, Spain
{adelpozo, doyarzun}@vicomtech.org

Urtza Iturraspe
Robotiker
202. eraikina
Zamudioko Teknologia Parkea,
48170 Zamudio, Spain
uiturraspe@robotiker.es

Xabier Arregi, Kepa Sarasola, Arantza Diaz de Ilarraza
IXA Group, University of the Basque Country
Informatika Fakultatea
649 posta-kutxa
20080 Donostia-San Sebastian, Spain
{xabier.arregi,kepa.sarasola,a.diazdeilarraza}@ehu.es

Eva Navas, Igor Odriozola, Iñaki Sainz
Aholab Group, Basque Country University
Ingeniaritza Goi Eskola Teknikoa
Urkijo Zumardia, z.g.
48013 Bilbao, Spain
{eva.navas,igor.odriozola}@ehu.es, inaki@aholab.ehu.es

Abstract

AnHitz is a prototype of a virtual Basque-speaking 3D expert that can answer questions or perform cross-lingual searches on science and technology, and show the search results in Basque by means of machine translation. It has been named after the 3-year strategic research project on language, speech and visual technologies for Basque carried out by several organizations, and built as a demonstrator of the technologies developed. Because the six modules comprising AnHitz have been implemented by different organizations using different operating systems, programming languages or libraries, it was impossible to build the demonstrator in a single executable or machine. As a result, the prototype has been constructed using separate programs in various machines that interact using network communication and web services protocols. Despite the employed approach having some drawbacks –mainly higher time delays when audio rendering is done via the web–, the outcome is indeed satisfactory, as it has allowed us to build a fully functional demonstrator that has showed good performance and acceptance in an evaluation made with 50 users, and has made a great impact on the Basque media.

1. Background

1.1 The AnHitz project

AnHitz is a prototype demonstrator that sets out to show the potential of the integration of language, speech and visual technologies. It is the outcome of a 3-year strategic research project on language, speech and visual technologies for Basque, also called AnHitz. This project has been promoted by the Basque Government in its Science and Technology Plan for 2006-2008 to develop language technologies for Basque.

1.2 The AnHitz consortium

AnHitz is a collaborative project between five participants, each of them with expertise in a different area:

- Vicomtech (http://www.vicomtech.org/): An applied research centre working in the area of interactive computer graphics and digital multimedia. It was founded jointly by the INI-GraphicsNet Foundation and by the EiTB, the Basque Radio and Television broadcasting corporation.
- Robotiker (http://www.robotiker.com): A technology centre specialized in information and telecommunication technologies, part of the Tecnalia Technology Corporation.
- Elhuyar Foundation (http://www.elhuyar.org): A non-for-profit organization that aims to promote the normalization and standardization of Basque, with activities in the fields of lexicography and terminology, dictionary publishing, language planning, science and technology communication, textbooks and multimedia products and services, alongside R&D in language technologies for Basque.
- The IXA Group of the University of the Basque Country (http://ixa.si.ehu.es): Specialized in the processing of written texts at different levels (morphology, syntax, semantics; corpora, machine translation, IE-IR…).
- The Aholab Signal Processing Laboratory Group of the University of the Basque Country (http://aholab.ehu.es): Specialized in speech technologies (speech synthesis and recognition, speaker identification…).

1.3 Vision

Basque is an agglutinative language with a very rich morphology. There are around 700,000 Basque speakers, about 25% of the total population of the Basque Country, but they are not evenly distributed. There are six dialects, but since 1968 the Academy of the Basque Language (Euskaltzaindia) has been involved in a standardization process. At present, the morphology is completely standardized, but the lexical standardization process is still under way.

Language technology development for Basque differs in several aspects from the development of similar technologies for widely used and standardized languages such as French (Chaudiron & Mariani, 2006), Norwegian (Maegaard et al., 2006) or Dutch-Flemish (D'hallewey et al., 2006). This is mainly due to two reasons:

- The size of the speakers' community is small. As a result, there are not enough specialized human resources, they lack financial support, and commercial profitability is, in almost all cases, a very difficult goal to reach.
- Due to its rich inflectional morphology, Basque requires specific procedures for language analysis and generation. Thus, it is not always possible to reuse language technologies developed for other languages. This is relevant in both rule-based and corpus-based approaches, since this applicability (or portability) depends largely on language similarity.

For these reasons, we believe that research and development for Basque should be (and, in the case of the members of AnHitz, usually is) approached following these guidelines:

- High standardization of resources to be useful in different lines of research, tools and applications.
- Reuse of language resources, tools, and applications.
- Incremental design and development of language resources, tools, and applications in a parallel and coordinated way in order to get the maximum benefit from them. Language resources and research are essential to create any tool or application; but, by the same token, tools and applications will be very helpful in the research and improvement of language resources.
- Use of open source tools.

1.4 Resources, tools and applications developed

Some of the organizations that are part of AnHitz have been working in Natural Language Processing and Language Engineering for Basque since 1990. The most basic tools and resources (lemmatizers, POS taggers, lexical databases, speech databases, electronic dictionaries, etc.) had been developed before AnHitz, but most of them have been further improved within it, and many others have been created in this project:

- Textual resources: ZT Corpusa (Areta et al., 2007), a corpus of science and technology texts (http://www.ztcorpusa.net); EPEC, a corpus tagged and disambiguated at the morphological, syntactic and semantic levels.
- Speech resources: SpeechDat FDB1060-EU, a SpeechDat-like database for Basque that contains recordings obtained over the fixed telephone network; SpeechDat MDB600-EU, another SpeechDat-like database for Basque that contains recordings obtained over the mobile telephone network; EMODB (Navas et al., 2004), emotional speech database recorded by a female speaker in the six MPEG4 emotions and neutral style; Amaia and Aitor (Saratxaga et al., 2006), emotional speech database phonetically balanced for female and male voices; BIZKAIFON (Castelruiz et al., 2004), multimodal (speech and video) database for the Western dialects of the Basque language (http://bizkaifon.ehu.es).
- Textual tools: Erauzterm (Gurrutxaga et al., 2004), tool for automatic term extraction from Basque texts and corpora; ElexBI (Alegria et al., 2006a), tool for the extraction of pairs of equivalent terms from Spanish-Basque translation memories (http://itzulterm.elhuyar.org/); Corpusgile and Eulia (Areta et al., 2007), advanced tools to create, linguistically annotate and query corpora; CorpEus (Leturia et al., 2007a), a web-as-corpus tool for Basque that allows the querying of the Internet as if it were a Basque corpus (http://www.corpeus.org); Dokusare (Saralegi & Alegria, 2007), a system to identify science news of similar content in a multilingual environment by using cross-lingual document similarity techniques; Co3 (Leturia et al., 2009), a system to automatically build multilingual comparable corpora (Spanish-English-Basque) using the Internet as a source; AzerHitz (Saralegi et al., 2008), a system to automatically extract pairs of equivalent terms from Spanish-Basque comparable corpora; Elezkari (Saralegi & López de Lacalle, 2009), a cross-lingual information retrieval system focused on Basque, Spanish and English; Eulibeltz (Díaz de Ilarraza et al., 2007), a tool to create and linguistically annotate bilingual aligned corpora; Eihera (Alegria et al., 2006b), named entity recognizer for Basque.
- Speech tools: AhoT2P, a letter to allophone transcriber for standard Basque; AhoTTS_Mod1, a linguistic processor for speech synthesis.
- Text applications: Xuxen (Aduriz et al., 1997), spell-checker suited to the agglutinative nature of Basque that combines dictionaries and morphological analysis; Lemmatization based dictionaries for text-processors; Elebila (Leturia et al., 2007b), a public search engine for content in Basque (http://www.elebila.eu); Opentrad-Matxin (Alegria et al., 2007; Alegria et al., 2008), open-source machine translation system for Spanish-Basque (http://www.opentrad.org); English-Basque statistical MT (Stroppa et al., 2006); Ihardetsi (Ansa et al., 2008), a Question Answering system for Basque.
- Speech applications: AhoTTS (Hernáez et al., 2001), a modular Text-To-Speech conversion system for Basque, Spanish and English (http://aholab.ehu.es/tts/tts_en.html); AhoTTS for

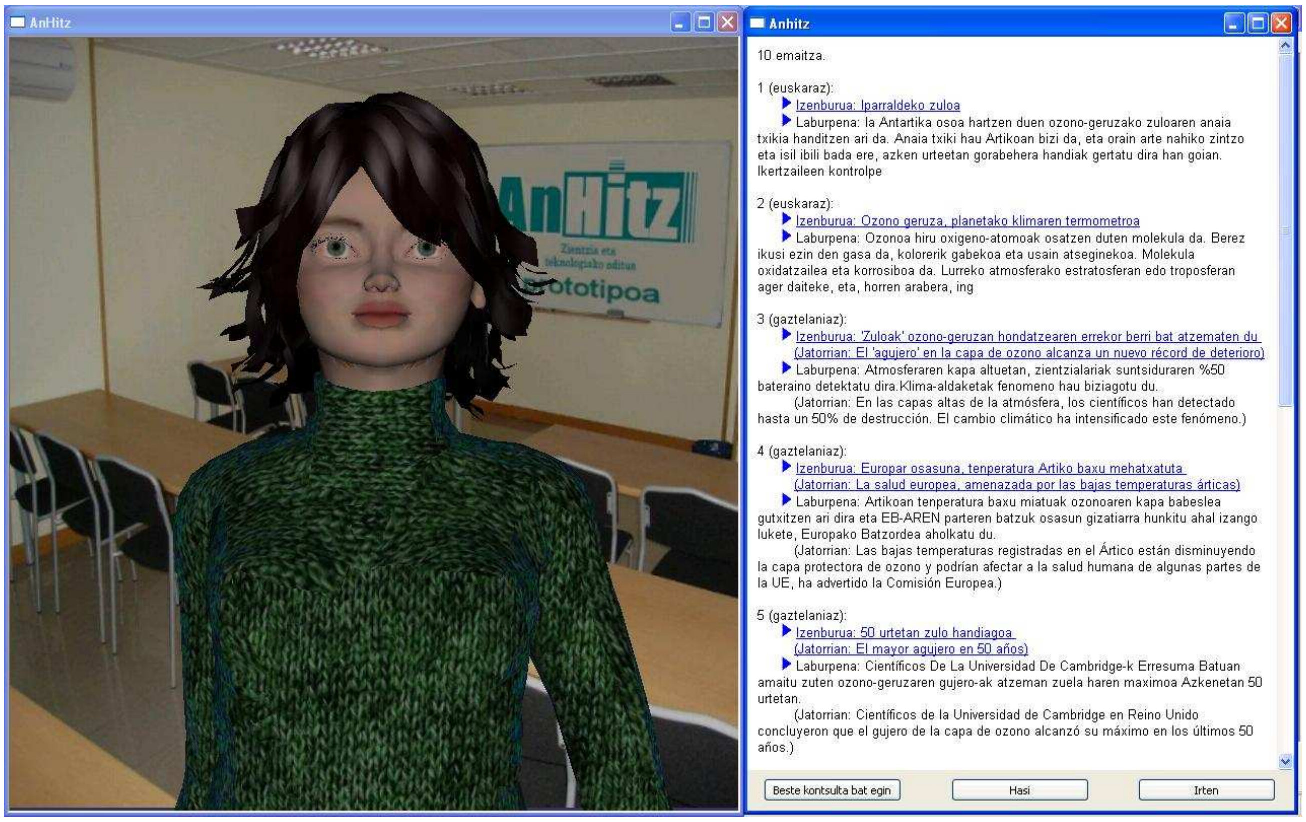
PDA (Sanchez et al., 2006); ZTRec, a Basque speech recognizer of science and technology terms and questions.
- Visual applications: AnHitzDlg, Basque speaking avatar.

2. The AnHitz demo

Apart from the various resources, tools and applications developed in the AnHitz project, we have also built a demonstrator that sets out to show the potential of all these different technologies working together. It is a 3D virtual expert on science and technology that includes Question Answering and Cross Lingual Information Retrieval systems in those areas. The interaction with the system is carried out in Basque (the results of the CLIR module that are not in Basque are translated through Machine Translation) and is speech-based (using Speech Synthesis and Automatic Speech Recognition).
This is a typical conversation with AnHitz when performing a cross-lingual search:

- Kaixo, nola duzu izena? (*Hello, what's your name?*)
- Kepa (*Kepa*)
- Kepa esan al duzu? (*Did you say Kepa?*)
- Bai (*Yes*)
- Kaixo, Kepa, ni Anhitz naiz (*Hello, Kepa, I am Anhitz*). Zientzia eta Teknologiari buruzko galderak erantzun ditzaket (*I can answer questions about Science and Technology*), edo hitzak bilatu dokumentu biltegi eleaniztun batean (*or look for documents containing keywords in a multilingual repository*). Kepa, zer egin nahi duzu egitea, galdera bati erantzutea edo edo hitzak bilatzea? (*Kepa, what do you want me to do, answer a question or look for keywords?*)
- Hitzak bilatzea (*Look for keywords*)
- Esan bilatu nahi dituzun hitzak, Kepa (*Tell me which keywords you want me to look for, Kepa*)
- Ozono geruza (*Ozone layer*)
- "ozono geruza" esan duzu? (*Did you say "ozone layer"?*)
- Bai (*Yes*)
- "ozono geruza" bilatzen… (*Looking for "ozone layer"…*) Espainierazko dokumentuak itzultzen… (*Translating documents from Spanish…*) Hauek dira aurkitu ditudan emaitzak: (*These are the results I have found:*)

And next AnHitz will show on the screen to its right a list with clickable titles and snippets of the results found. The results that were not originally in Basque are translated by means of machine translation (see Figure 1). And the user can tell the system to read the titles or snippets he/she wants aloud, using TTS.



10 emaitza.

1 (euskaraz):
▶ Izenburua: Iparraldeko zuloa
▶ Laburpena: la Antartika osoa hartzen duen ozono-geruzako zuloaren anaia txikia handitzen ari da. Anaia txiki hau Artikoan bizi da, eta orain arte nahiko zintzo eta isil ibili bada ere, azken urteetan gorabehera handiak gertatu dira han goian. Ikertzaileen kontrolpe

2 (euskaraz):
▶ Izenburua: Ozono geruza, planetako klimaren termometroa
▶ Laburpena: Ozonoa hiru oxigeno-atomoak osatzen duten molekula da. Berez ikusi ezin den gasa da, kolorerik gabekoa eta usain atseginekoa. Molekula oxidatzailea eta korrosiboa da. Lurreko atmosferako estratosferan edo troposferan ager daiteke, eta, horren arabera, ing

3 (gaztelaniaz):
▶ Izenburua: 'Zuloak' ozono-geruzan hondatzearen errekor berri bat atzematen du (Jatorrian: El 'agujero' en la capa de ozono alcanza un nuevo récord de deterioro)
▶ Laburpena: Atmosferaren kapa altuetan, zientzialariak suntsiduraren %50 bateraino detektatu dira.Klima-aldaketak fenomeno hau biziagotu du.
(Jatorrian: En las capas altas de la atmósfera, los científicos han detectado hasta un 50% de destrucción. El cambio climático ha intensificado este fenómeno.)

4 (gaztelaniaz):
▶ Izenburua: Europar osasuna, tenperatura Artiko baxu mehatxatuta (Jatorrian: La salud europea, amenazada por las bajas temperaturas árticas)
▶ Laburpena: Artikoan tenperatura baxu miatuak ozonoaren kapa babeslea gutxitzen ari dira eta EB-AREN parteren batzuk osasun gizatiarra hunkitu ahal izango lukete, Europako Batzordea aholkatu du.
(Jatorrian: Las bajas temperaturas registradas en el Ártico están disminuyendo la capa protectora de ozono y podrían afectar a la salud humana de algunas partes de la UE, ha advertido la Comisión Europea.)

5 (gaztelaniaz):
▶ Izenburua: 50 urtetan zulo handiagoa (Jatorrian: El mayor agujero en 50 años)
▶ Laburpena: Científicos De La Universidad De Cambridge-k Erresuma Batuan amaitu zuten ozono-geruzaren gujero-ak atzeman zuela haren maximoa Azkenetan 50 urtetan.
(Jatorrian: Científicos de la Universidad de Cambridge en Reino Unido concluyeron que el gujero de la capa de ozono alcanzó su máximo en los últimos 50 años.)

Figure 1: Screen capture of the AnHitz demo

An example of a conversation involving a question would be like this (with the same introductory part):

- Kepa, zer egin nahi duzu egitea, galdera bati erantzutea edo edo hitzak bilatzea? (*Kepa, what do you want me to do, answer a question or look for keywords?*)
- Galdera bati erantzutea (*Answer a question*)
- Esan egin nahi duzun galdera, Kepa (*Put the*

question you want answered, Kepa)
- Nork asmatu zuen telefonoa? (*Who invented the telephone?*)
- "Nork asmatu zuen telefonoa?" esan duzu? (*Did you say "who invented the telephone?"*)
- Bai (*Yes*)
- Galderaren erantzuna bilatzen… (*Looking for the answer to your question…*)
- Erantzuna Graham Bell izan daitekeela uste dut. (*I think the answer is Graham Bell.*) Nahi al dituzu ikusi aukera guztiak euren probabilitatearekin? (*Do you want to see all the possible answers I have found and their probabilities?*)
- Bai (*Yes*)
- Hor ondoan dituzu aurkitu ditudan aukera guztiak euren probabilitatearekin: (*These are the possible answers I have found and their probabilities:*)

And then AnHitz shows the list of possible answers on the text screen, together with each one's probability and the paragraph from which each has been inferred.

3. Description of the modules

The AnHitz prototype, as we have already pointed out, comprises various modules, which will be described in more detail in the following subsections. The architecture of the overall system is shown in Figure 2.

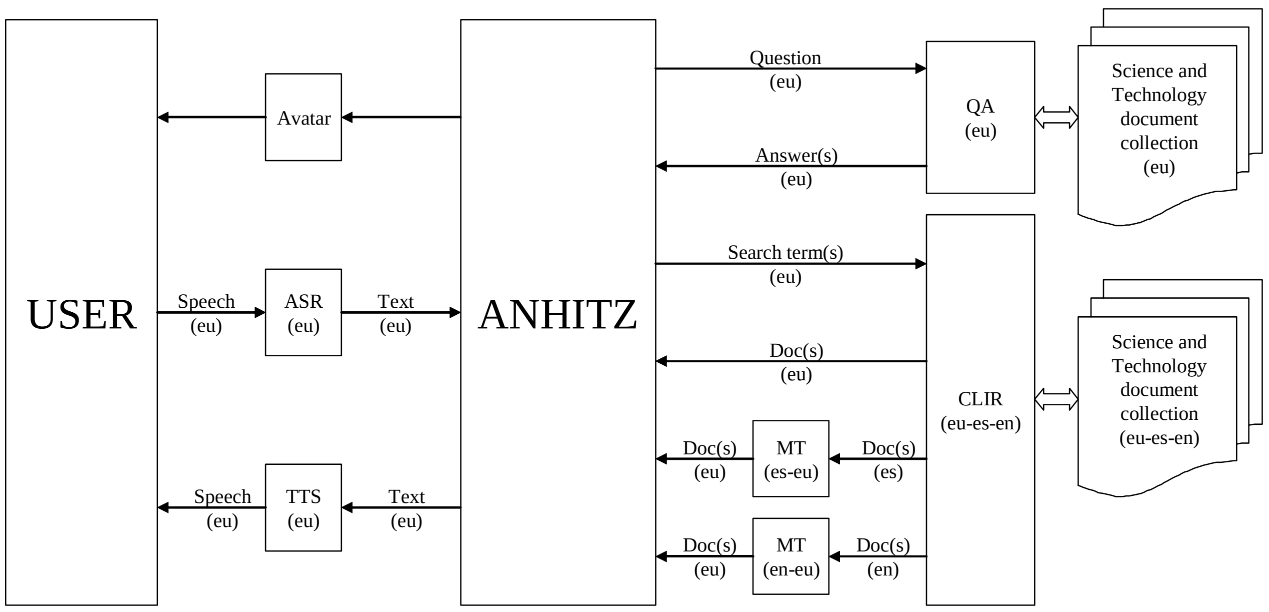


Figure 2: Architecture of the modules of the AnHitz demo

3.1 Avatar

The avatar module developed by Vicomtech, AnHitzDlg, includes all the necessary functionalities for showing and animating the 3D character that acts as the front-end of the AnHitz demonstrator. Its lip animation is synchronised with the audio synthesised by the multilingual TTS module in several languages and it can also show facial emotions when required. In addition, the module generates blinking and head movement animations through a set of behaviour rules in order to increase the illusion that the 3D character is alive.

It has been developed in C++, using OpenSceneGraph (http://www.openscenegraph.org) as its graphic library. Although it has only been tested in the Windows XP and Windows Vista Operative Systems, the multiplatform features of both the avatar module and the OpenSceneGraph library allow its easy migration to Linux systems too.

3.2 ASR

The only Automatic Speech Recognition (ASR) system for Basque available at the time of AnHitz's development was Nuance's Vocon3200. This ASR system is grammar-based. Elhuyar and Robotiker had to design various grammars in Backus-Naur Form or BNF (http://en.wikipedia.org/wiki/Backus-Naur_Form) in order to adapt the system to the AnHitz demo: a grammar for names, a grammar for yes/no answers, a grammar for search terms, a grammar for common science and technology questions, etc. The search terms and questions were extracted out of the most frequent searches of the logs of Zientzia.net (http://www.zientzia.net), a Basque popular science website owned by Elhuyar.

Vocon3200 is written in C and can be used under Windows.

3.3 TTS

The Text-To-Speech system used in AnHitz is AhoTTS (Hernáez et al., 2001), a multilingual system developed by the Aholab Signal Processing Group of the University of the Basque Country for commercial and research

purposes. The system has a modular architecture because it has been specially designed to develop all the modules that integrate a TTS system independently. The system uses three main processing modules: the text processor, the linguistic processor and the synthesis engine. In addition, three databases are used: one dictionary which includes morphological and phonetic information about the words, a database for prosody prediction, and the synthesis database containing the recordings that will be manipulated to generate the synthetic speech (Sainz et al., 2009). The system currently works in Basque, Spanish and English, using Festival (Taylor et al., 1998) for the English text processing module.
AhoTTS is written in C/C++ and is fully functional both in the Unix and Windows operating systems.

3.4 CLIR

The Cross-Lingual Information Retrieval module used is Elezkari, which has been developed by Elhuyar (Saralegi, & López de Lacalle, 2009). The search terms are entered in Basque and the information retrieval is done in various popular science corpora in Basque, English and Spanish. In order to achieve this, the search terms have to be properly translated into the other languages (dealing with ambiguous translations, Out-Of-Vocabulary words, etc.).
The system works under Linux. It is programmed in C and makes use of the Indri search engine (http://www.lemurproject.org/indri/).

3.5 MT

Matxin, developed by the IXA group of the University of the Basque Country (Alegria et al., 2007; Mayor et al., 2009), is the system used for Machine Translation in AnHitz. It translates text from Spanish into Basque using a transfer rule-based approach. The first version of an English-Basque rule based MT system is being developed at the moment.
Its modules have been programmed using C and C++ programming languages, it works under Linux and its free code is publicly available in Sourceforge (http://matxin.sourceforge.net).

3.6 QA

The Question Answering system used in AnHitz is Ihardetsi, developed by IXA (Ansa et al., 2008; Alegria et al., 2009). It works over a Science and Technology corpus compiled by Elhuyar and IXA, the ZTCorpus. As is common in question answering systems, Ihardetsi is based on three main modules: the question analysis module, the passage retrieval module and the answer extraction module. These modules have been implemented as autonomous web services by reusing some linguistic tools previously developed in the IXA group, and the QA system becomes a client that calls these services in a pipeline when it needs them by using the SOAP (Simple Object Access Protocol) communication protocol.
The Ihardetsi QA system runs under Linux.

3.7 AnHitz main program

The AnHitz main program controls the conversation flow and responds to the user's queries by making use of the other modules.
The control of the conversation flow includes introducing itself, prompting the user for his/her name, the action to perform, the terms to search or the question to answer, showing different emotions depending on the certainty of the answer, etc. To improve the performance of the ASR system when it did not understand correctly, we used the confidence level returned by the ASR system, and empirically found reasonably good thresholds of this confidence level for correct recognition, doubtful recognition and incorrect recognition. Thus, the system asks for confirmation in the case of doubtful recognition and repeats the question in the case of incorrect recognition.
The main program has been developed in Python and, since it uses no operating system-specific libraries, it can run indistinctly under Linux or Windows.

4. Communication among the modules

Bearing in mind what has been said in the previous section, we can observe that the AnHitz modules, since they have been built by different organizations, run on different operating systems and use different programming languages and/or libraries. This made it extremely difficult, if not impossible, to integrate the demonstrator into a single machine, let alone into an executable file.
As a result, the AnHitz prototype demonstrator has been constructed as a distributed system running in different machines over the Internet and communicating via net protocols and web services, even among the modules running in the same machine. The main program of the demo, the avatar module and the ASR system run on a Windows laptop. The TTS system runs on a Linux machine at Aholab, the CLIR system on a Linux machine at Elhuyar and the MT and QA systems on different Linux machines at IXA. The modules have been implemented as servers using network communication protocols, and the main program is the client that makes requests to them.
Different protocols have been used for communication. There has been no particular reason to choose one or the other: in some cases, the module was already on the web implemented as a web service with a certain protocol; in others, the programmer just used the protocol he/she was more familiar with.
The avatar module receives its speaking orders via simple sockets. It then asks the TTS module for the audio file and phoneme information file using an HTTP request. The ASR system activates itself and the microphone when it receives a request via sockets. The CLIR module is called using the SOAP protocol. The MT system for translating the texts from Spanish into Basque is also called using SOAP. The QA module is queried via sockets. An illustration showing the communication among the modules is shown in Figure 3.

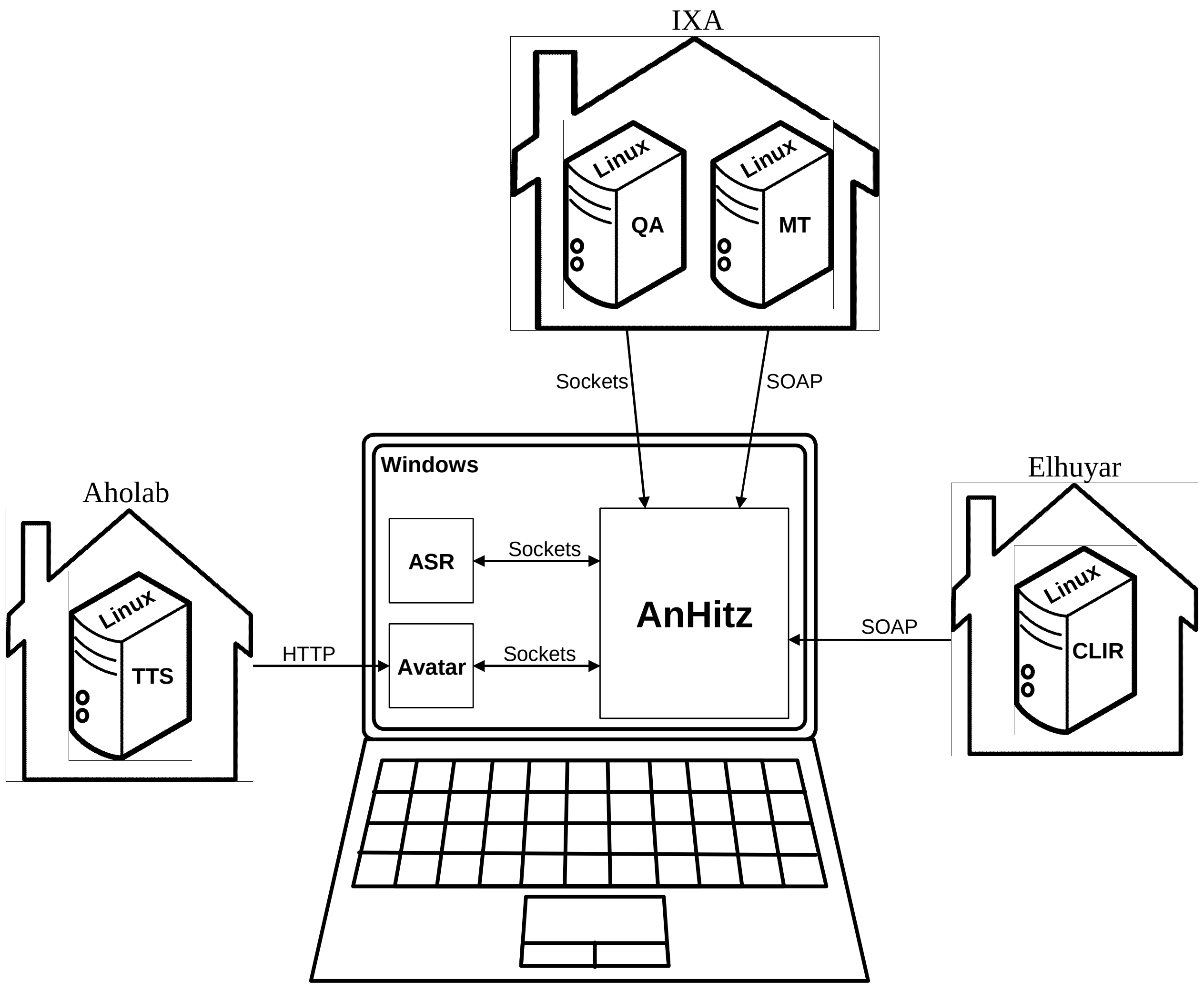


Figure 3: Communication among AnHitz’s modules

5. Evaluation

The demo prototype developed in AnHitz was evaluated in order to measure its performance and weigh up the impression of potential users about it. 50 users formulated 3 questions and 3 cross-lingual search queries each, testing the system 300 times in total. During the trials, several objective observations, such as the number of failures and successes of the ASR or QA systems, were noted down. At the end of each interaction, the testers filled in a questionnaire regarding more subjective matters (quality of the TTS, CLIR or MT systems, general impression, etc.).

The aim of the evaluation was twofold: on the one hand, the performance of the individual modules was evaluated in real world uses; on the other, we were able to weigh up the AnHitz demo’s performance and the users’ impressions about it. The results of the evaluation are detailed in the following paragraphs.

The ASR system, together with the confirmation/repetition system implemented in the AnHitz demonstrator, understood correctly 63.19% of the times. Another 12.59% of the times it understood correctly but was not sure and asked for confirmation. 13.43% of the times the system did not understand correctly and asked for confirmation, so the user could repeat the sentence. Only in 10.79% of the cases did the system understand wrongly without giving the option to correct. When users were asked whether AnHitz had understood what they said overall, 55.11% of the testers answered “almost always” or “most of the times”, 34.69% “sometimes” and 10.20% “a few times”. No one chose “hardly ever”.

Regarding the intelligibility of AnHitz's speech, 85.42% thought it was “very good” or “good” and 14.58% “quite good”. No one chose “bad” or “very bad”. 43.75% of the testers judged the speech as “very natural” or “natural”, 31.25% “quite natural” and 25.00% “artificial” or “very artificial”.

The question answering system returned the correct answer 30.61% of the times, and in another 15.30% the correct answer was among the given first five possible answers. 54.08% of the times the system did not return a correct solution or did not answer at all. However, some

of these incorrect outcomes might be due to the correct answer not being in the corpus, and so the results could have been better.
The users judged the CLIR results to be “very good” or “good” 68.35% of the times; found them to be “quite bad” in 22.30% of the cases, and thought they were “completely unrelated” 9.35% of the time.
30.00% of the times the users found the translations of the MT system to be “very good”, “good” or “quite good”; “comprehensible” in another 38.89%; and “quite bad”, “bad” or “very bad” in the remaining 31.11%.
Regarding usefulness, 62.50% of the users thought the system was “very useful” or “useful” and 37.50% thought it was “quite useful”. No one said it was “quite useless” or “completely useless”. When asked about the suitability of extending the AnHitz approach to other application domains, 20.83% said “it should always be like this with machines”, 39.58% that they would like to see it “in many cases” and another 39.58% “in some cases”. No one chose “maybe in a few cases” or “never”.

6. Dissemination

At the end of the AnHitz project, its participants and some members of the Basque Government gave a press conference, which was very well attended by the media. Practically every radio, TV or newspaper covered the news the same day or the next. Furthermore, the demo prototype aroused great interest, and many media devoted a video, interview or article to it. Some of these appearances of AnHitz in the media can be seen at http://www.elhuyar.org/hizkuntza-zerbitzuak/EN/Anhitz-project.
We also showed the prototype to the general public during the Week of Science and Technology 2008, in two stands in Donostia-San Sebastian and Bilbao. Students from schools and members of the public in general had the chance to try it out and play with it, and they were generally surprised and interested.

7. Conclusions

The AnHitz project has proved to be very effective for improving the already existing language and speech resources for Basque and for creating new ones. The AnHitz prototype demonstrator implemented to integrate the tools and resources developed has shown that collaboration between agents working in the different language, speech and visual research fields is crucial for exploiting the potential of the technologies and build applications useful for the end user. The evaluation of the AnHitz demonstrator has shown that despite being based on systems still in the research stage, its performance is acceptable.
In order to build the completely functional AnHitz demo integrating different language, speech and visual technologies, a modular remote architecture with network communications has been used. The benefits of building the demo using this networked approach have been enormous, and we doubt we could have built it otherwise. However, this approach also presents some drawbacks. The main disadvantage is the time delay that originates when AnHitz has to speak, due to the transmission over the Internet of the audio files generated by the TTS module. Other delays exist too, because some processes, mainly MT and QA, need their time; but these are inevitable and not due to the modular architecture. However, we reduced the effect of these delays to some extent by locally caching the already processed audios, queries and translations, so that the most frequent and repetitive sentences and queries can be executed instantly.
Another drawback is the lack of control over the remote modules and servers. If something goes wrong in one of them, the whole AnHitz demo is affected and it is not trivial to put things back up.
Nevertheless, we consider that the results have been very satisfactory overall, since both the responses obtained from the users in the evaluation and the media coverage have been very positive.
One future improvement of the AnHitz demo could be the use of virtualization to run two operating systems at a time and thus allow the installation of all the modules in the same machine. However, we are not sure this approach would work out due to the complexity, libraries, versions, etc. of the modules. But even with all the modules in one machine, the network communications approach would still be used.
Another possible improvement to explore in the future is the implementation of the AnHitz demonstrator as a web application, which would allow the general public to experiment with the potential of the combination of language, speech and visual technologies.

8. Acknowledgements

This work has been partially founded by the Industry Department of the Basque Government under grant IE06-185 (AnHitz project).

9. References

Aduriz, I., Alegria, I., Artola, X., Ezeiza, N., Sarasola, K. (1997). A spelling corrector for Basque based on morphology. *Literary & Linguistic Computing*, 12(1), pp. 31--38.
Alegria, I., Gurrutxaga, A., Saralegi, X., Ugartetxea, S. (2006a). Elexbi, a basic tool for bilingual term extraction from Spanish-Basque parallel corpora. In *Proceedings of Euralex 2006*. Torino: Euralex, pp. 159--165.
Alegria, I., Arregi, O., Ezeiza, N., Fernandez, I. (2006b). Lessons from the development of a named entity recognizer for Basque. *Procesamiento del Lenguaje Natural*, 36, pp. 25--37.
Alegria, I, Díaz de Ilarraza, A., Labaka, G., Lersundi, M., Mayor, A., Sarasola, K. (2007). Transfer-based MT from Spanish into Basque: reusability, standardization and open source. *Lecture Notes on Computer Science*, 4394, pp. 374--384.
Alegria, I., Casillas, A., Diaz de Ilarraza, A., Igartua, J., Labaka, G., Lersundi, M., Mayor, A., Sarasola, K. (2008). Spanish-to-Basque MultiEngine Machine

Translation for a Restricted Domain. In *Proc. of the 8th Conference of the Association for Machine Translation in the Americas (AMTA-2008)*. Hawai, USA: AMTA, pp. 57--69.

Alegria, I., Ansa, O., Arregi, X., Otegi, A., Soraluze, A. (2009). Ihardetsi: A Question Answering system for Basque built on reused linguistic processors. In *Proc. SALTMIL 2009 workshop: Information Retrieval and Information Extraction for Less Resourced Languages*. Donostia: SALTMIL, pp. 37--43.

Ansa, O., Arregi, X., Otegi, A., Soraluze, A. (2008). Ihardetsi question answering system at QA@CLEF 2008. In *Working Notes of the Cross-Lingual Evaluation Forum*. Aarhus: CLEF, pp. 369--376.

Areta, N., Gurrutxaga, A., Leturia, I., Alegria, I., Artola, X., Diaz de Ilarraza, A., Ezeiza, N., Sologaistoa, A. (2007). ZT Corpus: Annotation and tools for Basque corpora. In *Proceedings of Corpus Linguistics 2007*. Birmingham: University of Birmingham.

Castelruiz, A., Sánchez, J., Zalbide, X., Navas, E., Gaminde, I. (2004). Description and design of a web accessible multimedia archive. In *Proc. of 12th IEEE Mediterranean Electrotechnical Conference (MELECON)*. Dubrovnik: IEEE, pp. 681--684.

Chaudiron, S., Mariani, J. (2006). Techno-langue: The French National Initiative for Human Language Technologies (HLT). In *Proc. of fifth international conference on Language Resources and Evaluation (LREC)*. Genoa: ELRA, pp. 767--772.

D'hallewey, E., Odijk, J., Teunissen, L., Cucchiarini, C. (2006). The Dutch-Flemish HLT Programme STEVIN: Essential Speech and Language Technology Resources. In *Proc. of fifth international conference on Language Resources and Evaluation (LREC)*. Genoa: ELRA, pp. 761--766.

Díaz de Ilarraza, A., Igartua, J., Sarasola, K., Sologaistoa, A., Casillas, A., Martinez, R. (2007). Spanish-Basque Parallel Corpus Structure: Linguistic Annotations and Translation Units. In *Proceedings of TSD 2007 Conference*. Plzen: TSD, pp. 230--237.

Gurrutxaga, A., Saralegi, X., Ugartetxea, S., Lizaso, P., Alegria, I., Urizar, R. (2004). A XML-based term extraction tool for Basque. In *Proc. of fourth international conference on Language Resources and Evaluation (LREC)*. Lisbon: ELRA, pp. 1733--1736.

Hernáez, I., Navas, E., Murugarren, J.L., Etxebarria, B. (2001). Description of the AhoTTS conversion system for the Basque language. In *Proceedings of 4th ISCA Tutorial and Research Workshop on Speech Synthesis*. Edinburgh: ISCA, paper 202.

Leturia, I., Gurrutxaga, A., Alegria, I., Ezeiza, A. (2007a). CorpEus, a 'web as corpus' tool designed for the agglutinative nature of Basque. In *Proceedings of Web as Corpus 3 workshop*. Louvain-la-Neuve: ACL-SIGWAC, pp. 69--81.

Leturia, I., Gurrutxaga, A., Areta, N., Alegria, I., Ezeiza, A. (2007b). EusBila, a search service designed for the agglutinative nature of Basque. In *Proceedings of iNEWS'07 workshop*. Amsterdam: ACM-SIGIR, pp. 47--54.

Leturia, I., San Vicente, I., Saralegi. X. (2009). Search engine based approaches for collecting domain-specific Basque-English comparable corpora from the Internet. In *Proceedings of 5th International Web as Corpus Workshop (WAC5)*. Donostia: ACL-SIGWAC, pp. 53--61.

Maegaard, B., Fenstad, J., Ahrenberg, L., Kvale, K., Mühlenbock, K., Heid, B. (2006). KUNSTI - Knowledge Generation for Norwegian Language. In *Proc. of fifth international conference on Language Resources and Evaluation (LREC)*. Genoa: ELRA, pp. 757--760.

Mayor, A., Alegria, I., Diaz de Ilarraza, A., Labaka, G., Lersundi, M., Sarasola, K. (2009). Evaluación de un sistema de traducción automática basado en reglas o por qué BLEU sólo sirve para lo que sirve. *Procesamiento del Lenguaje Natural*, 43, pp. 197--208.

Navas, E., Hernáez, I., Castelruiz, A., Luengo, I. (2004). Obtaining and evaluating an emotional database for prosody modelling in standard Basque. *Lecture Notes on Computer Science*, 3206, pp. 393--400.

Sainz, I., Erro, D., Navas, E., Hernáez, I., Saratxaga, I., Luengo, I., Odriozola, I. (2009). The AHOLAB Blizzard Challenge 2009 Entry. In *Proc. Blizzard Challenge 2009 workshop*. Edinburgh: Blizzard.

Sanchez, J., Luengo, I., Navas, E., Hernáez, I. (2006). Adaptation of the AhoTTS text to speech system to PDA platforms. In *Proceedings of the SPECOM 2006*. San Petersburg: SPECOM, pp 292--296.

Saralegi, X., Alegria, I. (2007). Similitud entre documentos multilingües de carácter científico-técnico en un entorno web. *Procesamiento del Lenguaje Natural*, 39, pp. 71--78.

Saralegi, X., San Vicente, I., Gurrutxaga, A. (2008). Automatic extraction of bilingual terms from comparable corpora in a popular science domain. In *Proceedings of Building and Using Comparable Corpora workshop*. Marrakech: BUCC, pp. 27--32.

Saralegi, X., López de Lacalle, M. (2009). Comparing different approaches to treat Translation Ambiguity in CLIR: Structured Queries vs. Target Co-occurrence Based Selection. In *Proceedings of the 6th International Workshop on Text-Based Information Retrieval*. Linz: TIR.

Saratxaga, I., Navas, E., Hernáez, I., Luengo, I. (2006). Designing and recording an emotional speech database for corpus based synthesis in Basque. In *Proc. of fifth international conference on Language Resources and Evaluation (LREC)*. Genoa: ELRA, pp. 2126--2129.

Stroppa, N., Groves, D., Way, A., Sarasola, K. (2006). Example-Based Machine Translation of the Basque Language. In *Proc. of the 7th Conference of the Association for Machine Translation in the Americas (AMTA-2007)*. Boston, USA: AMTA, pp. 232--241.

Taylor, P., Black, A., Caley, R. (1998). The architecture of the Festival Speech Synthesis System. In *Proc. 3rd ESCA Workshop on Speech Synthesis*. Jenolan Caves, ESCA, pp. 147--151.

Utilizing Web Service Technology to Create Danish Arabic Language Resources

Mossab Al-Hunaity
Center for Language Technology
University of Copenhagen
musab@hum.ku.dk

Abstract

Language resources are a major challenge for modern SMT applications. We propose a new model that utilizes the web service technology to create a bilingual text resource out of a monolingual corpus. We consider the Danish-Arabic pair as a case study for this model, yet it's possible to apply our model to any language pair. Our system compiles the monolingual corpus into many small segments and distributes them to network users. Received user translations are validated by both the system and human judges. Finally accepted translation is stored in bilingual translation memory.

1. Introduction

Building any statistical machine language (SMT) system between any language pair requires bilingual language resources. When translating between widely spoken languages pair like the case between English-Spanish or English-Arabic or Arabic-French it is common that SMT system developer can find bilingual resources easily like newspapers , blogs , parallel corpora , etc. The problem arises when we try to build SMT system between languages with limited common language resources like the case of Arabic and Danish. A possible solution for this problem is to use pivot technology where we develop two SMT systems with common language resources. For a language pair like the Arabic and Danish we build an Arabic-English SMT where Arabic is the source language and English is the target language. We build another English-Danish SMT system where English is the source language and Danish is the target language. We pipeline the result of the AR-EN system as an input to the EN-DA system. Finally we receive a Danish translation for our source Arabic text. English plays a pivot role in this approach. Although the pivot strategy provides a solution for the above problem it still has many drawbacks. Pivot approach translation output quality is less than direct approaches; usually translating sentences between many SMT (pivoting) will cause a partial loss of the meaning and sometimes may lead to a completely different meaning of the source sentence. A clear example of this is Google Translate[1] which relies on pivot approach when translating between languages with limited common resources like the case between Arabic and Danish pair. SMT systems based on Pivot approach sometimes produce results that are far away from the correct meaning. Building a quality SMT system requires a common bilingual resource. We propose a method that utilizes the web service technology to build bilingual text language resources for languages pair with limited parallel resources like the case with Arabic and Danish. Our framework is composed of five major web services that communicate with each other. Our major services are: Segmentation, Distribution and Replication, Translation collection, Validation, and Update translation memory. Figure 1 describes our model components. The five web services communicate with each other to process a monolingual corpus and finally produce a bilingual translation memory for any selected languages pair. We apply our experiments on the Danish–Arabic languages pair. Yet it can be applied to any languages pair**.** The services communicate with each other in a sequential order. The segmentation service will partition the monolingual corpus into documents, which will also be processed into sentences. Each sentence will be represented as a segment and it will be stored in a special XML format, as demonstrated in figure 2. The replication and distribution web service will create many copies of each segment and will disseminate segments through the web. Translation collection web service collect users translation feedback and send them in a proper XML format to the validation service, which in return will validate if the output is a possible candidate translation of the source sentence. A human evaluator is recommended to approve the received translation. Finally the update web service will update the translation memory with a new bilingual Danish-Arabic Entry. In the next section we describe related work. Section 3 presents our system model and its various details. Finally we discuss our conclusions and future work in section 4.

1: http://translate.google.com/

2. Related Work

SOA (Service Oriented Architecture) and web services technology are playing a major role in today modern software development Hayashi (2010). Efforts now are spent to utilize SOA and web service technology with natural language processing techniques. The notion of SaaS (Software as a Service) is now being adopted in NLP projects as LRaaS (Language Resource as a Service) Hayashi (2010). In his work Sornlertlamvanich (2010) argues that internet is now considered the largest linguistic resource which demand the development of many dedicated API set. Grefenstette (2010) suggested directions on how we can utilize web 2.0 for creating linguistic resources, especially the linguistic knowledge available within web users and their ability to provide language support. Dini and Petras (2010) introduce many technical challenges for building language resources for European languages like integration, language resources maintenance and licensing, which demand a dynamic software infrastructure like the one web services have. Using the internet as a primary source for corpus building for under-resourced languages was the aim of pioneer projects like the Crúbadán project Scannel (2007), they built their resources mainly by crawling the web. Fairon et.al (2007) proposed to use the web as a linguistic resource, mostly via search engine queries or by using ad-hoc data crawling software. Sharoff (2006) used ad-hoc queries to create general corpora for many languages. Leturia et. al (2007) used search engine queries for collecting bilingual specialized comparable corpora from the Internet. Baroni et. al (2004) used the web to enhance corpora quality. Although many free source corpora are available, still there is a need for common and open corpora Steinberger (2006). Biemann et.al (2009) described the web as a major resource for data collection and building for their Leipzing Corpora, their effort should make it easy to build free access language resources using web as a source of data and web technologies as emerging tools. We believe our technique should make it easy to build language resources easily and effectively. We are different from previous mentioned studies in that we target quality of language resources and hence we prefer human involvement in language resources collection.

3. Model Introduction

The system deploys five services to generate a bilingual text translation memory (Danish-Arabic) from of a source monolingual corpus (Danish), see figure 1. Web services interact with each other using XML formatted SOAP messages. In Sections 3.1 through 3.5 we explain our model system components details.

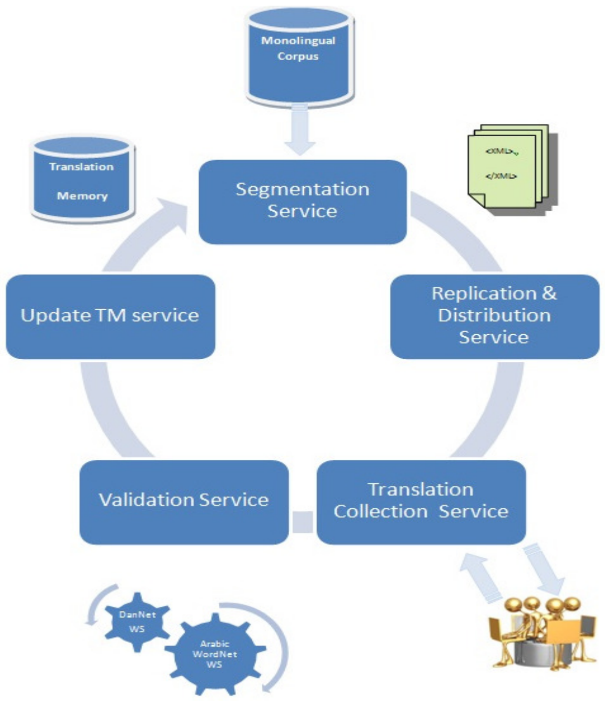


Figure 1: Model web services components

3.1 Segmentation Service

The service will process the monolingual corpus and compile it into a group of small XML files. Figure 2 represent a sample XML file extracted from the Danish monolingual corpus. The segmentation service will process this XML document into many segments that is ready to be distributed through the net work. Figure 3 represent a sample segment structure. Each segment contains only one small sentence. Segments include the sentence location details which makes it easy to map sentence to its original document location after translation.

3.2 Replication and Distribution Service

This service receives segments files produced by the segmentation service. Usually each file should contain one sentence. It will make many copies of each file and will start to disseminate these segments through the network.

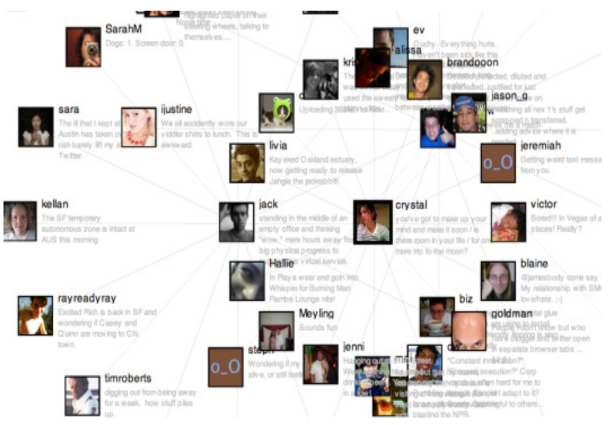
```
<?xml version="1.0" encoding="UTF-8"?>
<SRCSET setid="Climate_Change_Summit" srclang="DA">
      <DOC docid="1" genre = "text" >
         <seg id="1.1">
              de fire vigtige punkter, der bør rummes i en aftale i
               København
         </seg>
         <seg id="1.2">
             Hvor meget er industrilandene villige til at reducere deres
              udledning af drivhusgasser?
          </seg>
         <seg id="1.3">
              Hvor meget er toneangivende udviklingslande
               som Kina og Indien villige til at gøre for at begrænse
               stigningen i deres udledning?
       </seg>
      <seg id="1.4">
                Hvis København kan levere varen på de fire
               punkter, vil jeg være glad," siger Yvo de Boer.
      </seg>
   </DOC>
</SRCSET>
```

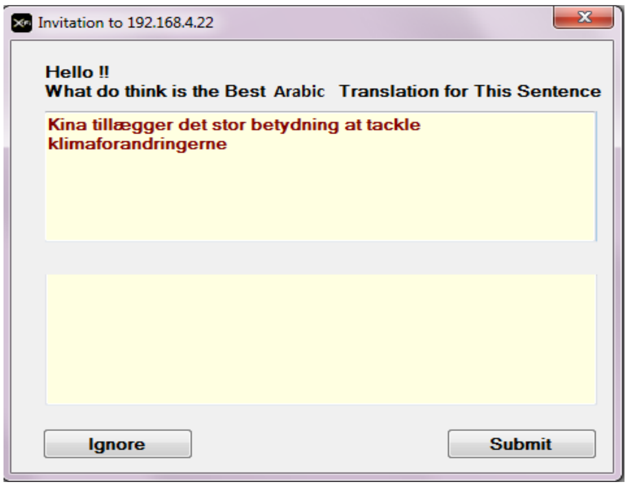
Figure 2: Sample document extracted from the Danish monolingual corpus.

```
<?xml version="1.0" encoding="UTF-8"?>
<SRCSET setid="Climate_Change_Summit" srclang="DA">
      <DOC docid="1" genre = "text" >
         <seg id="1.1">
              de fire vigtige punkter, der bør rummes i en aftale i
               København
         </seg>
   </DOC>
</SRCSET>


<?xml version="1.0" encoding="UTF-8"?>
<SRCSET setid="Climate_Change_Summit" srclang="DA">
      <DOC docid="1" genre = "text" >
         <seg id="1.2">
              Hvor meget er industrilandene villige til at reducere deres
             udledning af drivhusgasser?
         </seg>
   </DOC>
</SRCSET>
```

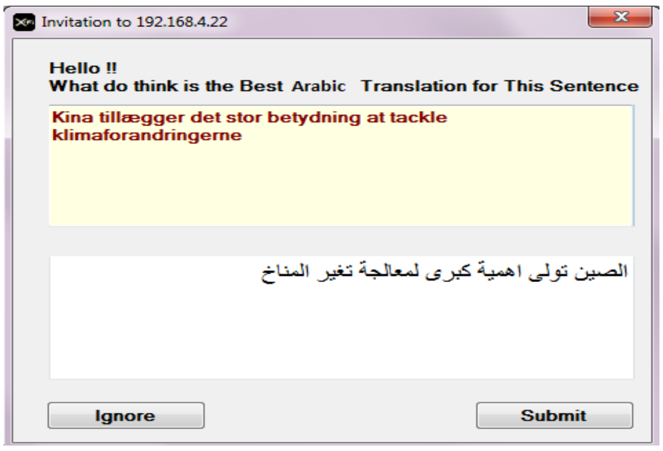
Figure 3: Segments produces from a corpus document, ready to be sent to network users.





Hello !!
What do think is the Best Arabic Translation for This Sentence

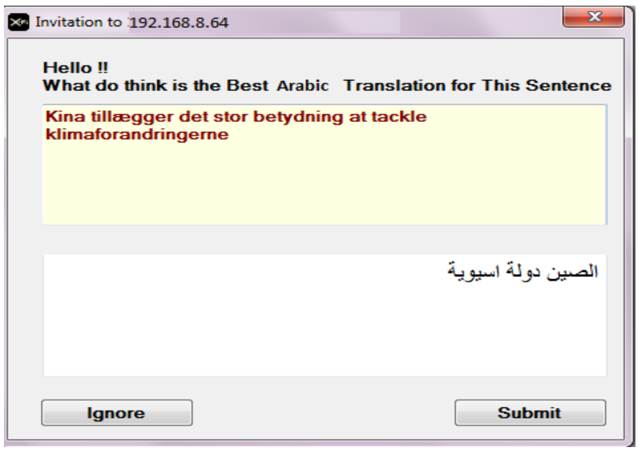
Kina tillægger det stor betydning at tackle klimaforandringerne



Hello !!
What do think is the Best Arabic Translation for This Sentence

Kina tillægger det stor betydning at tackle klimaforandringerne

الصين تولى اهمية كبرى لمعالجة تغير المناخ



Hello !!
What do think is the Best Arabic Translation for This Sentence

Kina tillægger det stor betydning at tackle klimaforandringerne

الصين دولة اسيوية

Figure 4 shows two possible feedbacks received from network users

Network users will receive segment as a message asking them to translate a sentence. Network users can be any target language (Arabic) literate user. Figure 4 illustrates this process. User can respond to the request by translating the sentence or by ignoring the request. The service will send many replica of the same segment to many users. The more segment replicas on the network are sent the higher the system chances to receive a response from users to the translation request. Receiving many translations to the same sentence will give us the chance to choose the best translation for our translation memory.

3.3 Translation Collection

The Translation Collection service will collect translation from network users who agreed to respond to the translation request. For each target sentence sent to network users the service will group all received translation into one XML file. Figure 5 describe users translation file for one sentence. Every translation is identified with a sequence identifier (ID) , original sentence ID (Seg) and user IP address (User) . This structure will make it easy for the system to organize user translation feedback.

```
<?xml version="1.0" encoding="UTF-8"?>
<SRCSET setid="Climate_Change_Summit" srclang="DA">
    <DOC docid="1" genre = "text" >

    <trans id="1" seg ="1.1"    user=”192.168.4.22”>
         الصين تعلق  أهمية كبيرة  على التصدي  لتغير المناخ
    </trans>

    <trans id="2" seg ="1.1"    user=”192.168.8.36”>
               الصين تهتم  بموضوع  المناخ
    </trans>

    <trans id="3" seg ="1.1"    user=”192.168.8.64”>
               الصين دولة اسيوية
    </trans>


  </DOC>
</SRCSET>
```

Figure 5: All users received translation for a text segment.

3.4 Translation validation service

After collecting translation text from network users, a validation service will ensure that the received translation is valid and can be considered as a candidate translation for the source sentence. For example Figure 5 shows three user translations for the source sentence. Translations received from network users can be valid or invalid, in our example we can find two valid translations(trans id 1 and 2). We inspect four main features we find most interesting to decide how valid the translation is. We consider translation length, similarity matching between source and destination sentences, translation accuracy and translation fitness. The validation service is a useful to filter candidate user translation from other none useful or incorrect translation we may receive from network users. This should save the human evaluators times and help increase their efficiency, so now human evaluators have only to verify what the system suggest as a correct translation.

3.4.1 Length Validation

Let S and D be two different sentences. Both S and D are composed of a group of words where S= {s1, s2,... , sm} and D= {d1, d2, … , dn}.We define L (X) to be a length function that returns the number of words each sentence have. We compare the length of the original sentence (S) to the length of the translated sentence (D) as formulated below.

$$r = \frac{\mathrm{L(S)}}{\mathrm{L(D)}} \qquad ..1$$

The service accept a sentence D as a possible translation for sentence S if r is greater than 0.75. Otherwise D is too short to be a candidate translation for sentence S. the observation underlying this believe is that a long sentence tend to be translated into long sentences, and a short sentence tend to be translated into long sentence.

3.4.2 Similarity Matching

Now for each sentences pair (S, D) we compare their similarity. We construct an alignment matrix for each pair (S, D) as it appears in Table 1. Alignment matrix has two dimensions. One represent source language sentence (S) and the other represent user translation (D) for that sentence. Matrix elements can have two values {0,1}. {0} value mean there is no match between S(i) and D (j) while {1} means the opposite. A match function will search for a possible match in the dictionary between Si and Dj as we describe in Table 1. So for example for the word couple {tillægger, تعلق } the match function will search the Danish Arabic dictionary directly for a meaning. If it manage to find that the Danish word “tillægger” has an Arabic meaning of “تعلق”word then the search will return the value of {1} as a sign of success. If the match function didn't find a direct meaning in the Danish Arabic dictionary then it will search the Danish DanNet web service for other concepts the Danish word { tillægger } has. Again the match function will try to find a match in the Danish Arabic dictionary for the Danish word { tillægger } and the other concepts it receives from the Danish DanNet. Figure 6 below simulate these steps. We apply the match function for all alignment matrix elements. For example If the sentence S was: "Kina tillægger det stor betydning at håndtere klimaændringerne "which means in English "China attaches great importance to tackling climate change". A possible user translation may be represented in sentence D which is "الصين تعلق أهمية كبيرة على التصدي لتغير المناخ ". We construct the alignment matrix between the two sentences as described in Table 1. For each entry we call the match function described above. For the Danish-Arabic words pair like {tillægger, تعلق }, we can find a value of {1} which means the "tillægger" word is a possible translation for "تعلق" word. While for the pair { betydning , الصين } we find the value of {0} which means that " betydning" word is not a possible translation for the "الصين" word. To estimate the overall similarity between the sentences pair, source Danish sentence and the target Arabic translation we introduce the Sim function which can be empirically calculated as below.

$$\mathrm{Sim}\ (\mathrm{S,D}) = \frac{\sum_{J=1}^{n}\ \sum_{i=1}^{m} \mathrm{match}\ (\mathrm{Si,Dj})}{\max(L(S), L(\ D))} \quad ..2$$

The S represents the source language which is Danish. The D represents the destination language which is Arabic. The match (Si,Dj) function will check if a dictionary meaning match is possible between any selected matrix elements as described above. The Sim function count number of pairs with {1} value , and compare that with the longest sentence length. To demonstrate we apply this formula to the example we have in table 1. We have two sentences (dimensions), each composed of eight elements. This will form an alignment matrix of 64 pair match. Eight of them of got the value {1}.we apply formula 2. Sim (S,D) = $\frac{8}{8}$.

S **D**	**Kina**	**tillægger**	**det**	**stor**	**betydning**	**at**	**handler**	**klimaændringerne**	$\sum$ match (Si,Dj)
الصين	1	0	0	0	0	0	0	0	1
تعلق	0	1	0	0	0	0	0	0	1
أهمية	0	0	0	0	1	0	0	0	1
كبيرة	0	0	0	1	0	0	0	0	1
على	0	0	0	0	0	1	0	0	1
التصدي	0	0	0	0	0	0	1	0	1
لتغير	0	0	0	0	0	0	0	1	1
المناخ	0	0	0	0	0	0	0	1	1

Table 1: Alignment matrix between source (Danish) and destination (Arabic) sentences.

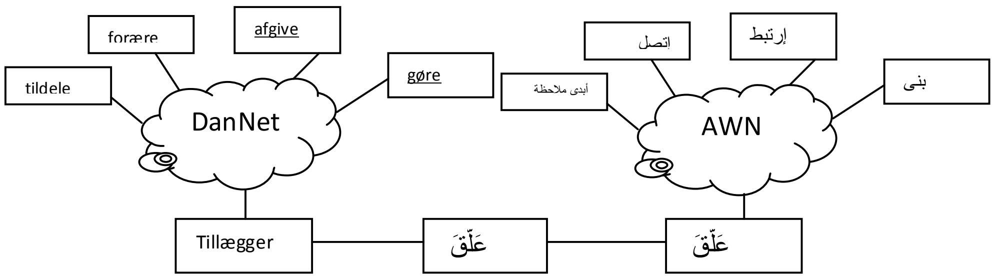


Figure 6: Using Arab wordNet (AWN) and DanNet to

This score mean every word at the source sentence has a match at the destination sentence. The result is a good hint that sentence D is a good candidate translation for the source sentence S.

3.4.3 Translation Accuracy

Another interesting issue we consider is how accurate user translation is. By accurate we mean what is the probability that user selected a translation that conveys the best word meaning. We use Arabic WordNet (AWN) web services to verify that. The AWN web service will provide us list of all possible concepts or meaning for the selected word as it appears in figure 6. If a word has for example four meanings, that means the probability that this word is the suitable translation selection would be $\frac{1}{4} = 0.25$. We apply formula 3 to calculate the general probability that the destination sentence D is an accurate translation of a sentence S. For example in Table 2 we calculate the average probability for the sentence according to formula 3: Acc (D)= $\frac{4.08}{8}$ =0.51.

الصين	تعلق	أهمية	كبيرة	على	التصدي	لتغير	المناخ	$\sum$
1/1	**1/11**	**1/2**	**1/7**	**1/1**	**1/4**	**1/10**	**1/1**	4.08

Table 2: Translation sentence words accuracy values.

Consequently this means that the average probability that this sentence is a suitable translation is 0.51. The AWN represents the Arabic WordNet function. It will return the number of possible concepts to the selected word. The m represents the number of words in a selected sentence.

$$Acc(D) = \frac{\sum_{j=1}^{m}\left(\frac{1}{AWN(D(j))}\right)}{m} \quad ..3$$

Table 2 show how the Accuracy is calculated for translated sentence D.

3.4.4 Translation Fitness

A fitness measure is required in translation validation to indicate the quality level of the translated sentences. For any sentences pair (S,D) we consider both similarity and accuracy values to estimate translation fitness. Fitness value between sentence pair (S, D) can be formulated as follows:

$$Fitness\,(S,D) = Sim(S,D) * Acc\,(D) \; ..4$$

Sim (S,D) and Acc (D) are calculated as explained in section 3.4.2 and section 3.4.3 respectively. According to our observations to the sample data sets a fitness threshold greater of (0.35) is quite acceptable to be considered for human evaluation. For our example presented earlier the Fitness (S, D) = 1 * 0.51 = 0.53 which indicate that this is a good candidate translation and can be considered for human evaluation.

3.5 Human Evaluation

System validation indicates whether the destination is a good candidate translation for the source sentence or not. It doesn't mean that the sentence is accepted. Human validation is still needed to finally approve translation and make sure that translation is acceptable and free of syntactical and grammatical errors. Our validation technique will only aid the human evaluators and help filter invalid received translation from network user especially when we have too many feedbacks for the same segment or sentence. Human evaluators can easily select or modify a candidate sentences to be the source sentence translation.

3.6 Update the Translation memory

Now a new entry with both the source language sentence and the translated sentence language (S,D) can be added to the translation memory.

```
<?xml version="1.0" encoding="UTF-8"?>
<SRCSET setid="Climate_Change_Summit" srclang="DA",
dstlang="AR">
     <DOC docid="1" genre = "text" >

     <trans id="1" seg ="1.1" srcdoc ="1"    >
          Kina tillægger det stor betydning at tackle
klimaforandringerne
     </trans>

    <trans id="1" seg ="1.1"    user="192.168.4.22">
          الصين تعلق أهمية كبيرة على التصدي لتغير المناخ
     </trans>

  </DOC>
</SRCSET>
```

Figure 7: Sample translation memory input.

It is possible that not all of the monolingual corpus segments would be translated. This will make fragmented segments to appear in translation coverage of the monolingual corpus. Translated segments will be used to feed our translation memory which can be used to enhance SMT system learning. Figure 7 describes a sample entry to the translation memory after being processed by the system

4. Conclusion and future Work

We presented a model that can utilize the web service technology to create bilingual text resources out of other monolingual data source. We selected the Danish- Arabic pair to be model input, yet any languages pair can be selected. We presented a sample scenario on how that model can be deployed through the network target language literate users. We demonstrated a simple validation service that helps human judges to verify translated sentence quality. Our model interacts with common popular language web services like the DanNet and Arab wordNet while validating user translation. In the future we plan to expand our model to interact with other software agents that can provide translation, like Google Translate. We plan to use agents translation as a translation hint for our human network users which will make it easy for them to translate segments they received from the system. We also intend to make the process of translation validation fully automatic and minimize the human role in that process. We intend to test more similarity matching techniques between the source sentence and user translated sentence. We also plan to deploy our model with different language pair like English and Danish for the same monolingual corpus so that we can get a multilingual resource for our developed translation memory.

5. References

C. Biemann, G. Heyer, U. Quasthoff, and M. Richter. The Leipzig corpora collection - mono-lingual corpora of standard size. In Proceedings of Corpus Linguistic 2007, Birmingham, UK, 2007

C. Fairon, H. Naets, A. Kilgarriff. Building and exploring web corpora, Proceedings of the 3rd Web as Corpus Workshop, incorporating Cleaneval. Presses Universitaires de Louvain, Louvain 2007.

G. Grefenstette. Proposition for a web 2.0 version of linguistic resource creation . In proceeding of FLaReNet Forum 2010: Language Resources of the future, 2010.

I .Leturia , A.Gurrutxaga, I.Alegria, A. Ezeiza. A ‘web as corpus’ tool designed for the agglutinative nature of Basque. In Building and exploring web corpora, Proceedings of the 3rd Web as Corpus workshop, Belgium, 2007.

K. Scannel ,The crúbadán project: corpus building for under-resourced languages. Proceedings of the 3rd Web as Corpus Workshop, 2007

L. Dini and V.Petras The challenge of multilinguality in Europeana: web services as language resources. In proceeding of FLaReNet Forum 2010: Language Resources of the future, 2010.

M .Baroni, S.Bernardini, Bootstrapping corpora and terms from the web. In Proceedings of LREC 2004. Lisbon, Portugal 2004.

R. Steinberger, B. Pouliquen, A. Widiger, C. Ignat, T. Erjavec, D. Tufiş, D. Varga. The JRC-Acquis: A multilingual aligned parallel corpus with 20+ languages. In Proceedings of the 5th International Conference on Language Resources and Evaluation . Genoa, Italy, 2006.

S. Sharoff. Creating general-purpose corpora using automated search engine queries. Baroni &S. Bernardini (Eds.), WaCky! Working papers on the Web as Corpus. Bologna, Italy, 2006

V. Sornlertlamvanich .Will Language as a Service (LaaS) increase the interoperability in language resources and applications? . In proceeding of FLaReNet Forum 2010: Language Resources of the future, 2010

Y. Hayashi. Toward a standardized set of language service Web APIs. In proceeding of FLaReNet Forum 2010: Language Resources of the future, 2010.

Technology-Neutral Machine Translation with an Abstracted Technology Stack

Joachim Van den Bogaert
Woodrow Wilsonplein 7, 9000 Gent, Belgium
E-mail: joachim@crosslang.com

Abstract

Web Services allow easy integration of Machine Translation (MT) into existing workflows. The use of XML facilitates MT implementation across platforms. The abundance of different interfaces, however, makes it sometimes difficult to switch technologies easily. Each existing MT service needs to be called with its own specific parameters. Moreover, the inner workings of different MT systems may be so different that parameters are not interchangeable. As a result, workflows need to be re-engineered for each new MT integration. The difficulties involved in switching MT technologies, become clear when pre-processing and post-processing technologies are introduced. The efforts it takes to re-target existing code and resources to a new MT system are considerable and can be reduced. In this paper we will present the design and implementation of a Technology-Neutral MT Gateway Service with an abstracted technology stack that makes MT integration of different systems easier and maximises the re-use of existing pre-processing and post-processing routines and resources.

1. Introduction

Most popular MT systems are offered in the form of Web Services. Google Translate (Google Inc., 2009), Systran (Systran Inc., 2009), Language Weaver (Language Weaver Inc., 2009), Apertium (Corbí-Bellot, et al., 2005) and Moses (Koehn, et al., 2007), for example, have interfaces that can be called over the web. The standard procedure of setting up communications which such systems consists of automatically creating a client proxy class that talks to the interface by generating it from the Web Service's WSDL page (Web Services Description Language (Christensen, Curbera, Meredith, & Weerawarana, 2001)). This only requires a WSDL-capable conversion tool that is included in most Web Services frameworks, for example WSDL2Java in the Java Axis2 framework (The Apache Software Foundation, 2009) or svcutil for the .NET framework (Microsoft Inc., 2010). The result is code that can be used almost immediately. Although the use of Web Services greatly reduces the effort required to integrate MT into an existing system, it sometimes is difficult to switch MT technologies quickly. This may be necessary because of quality issues, the availability of language pairs, or when comparing the performance of different systems. To be able to do this, some kind of abstraction of the MT interface is required, as described in Figure 1.

It is not so hard to come up with a solution on the client side that meets this requirement - a unified/abstract[1] MT interface is easily set up. However, when extra functionality is introduced, things tend to get more complex. Suppose that we wanted to add Named Entity Recognition (NER) to our workflow in order to get better MT output (Babych & Hartley, 2003). We then would have to redesign the procedure for each existing MT system we are using, because, unfortunately, not only MT interfaces differ between existing MT systems, but also the mechanisms they employ to process meta-data – if such mechanisms are available at all. This situation is described in Figure 2.

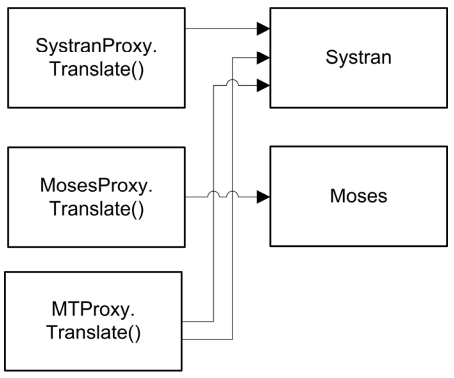


Figure 1

A more integrated solution is required here. To accommodate for such scenarios, we have designed and implemented a Gateway Service that resolves these kind of issues at the service side. Similar solutions have been proposed, amongst others by Sánchez-Cartagena & Pérez-Ortiz (2010), but their work focuses on harnessing existing MT engines and improving scaleability. The ScaleMT architecture they describe also deals with issues on a lower implementation level, such as sentence segmentation and routing. Other frameworks which connect different MT systems are, for example, (Heafield & Lavie, 2010) and (Barrault, 2010), but these architectures are aimed at combining the output of different MT systems to create new and better output.
Our system focuses on the integration of a translation pipeline over different MT engines. In the system we produced, an extra layer of functionality is added to each Web Service separately, while the additional functionality

[1] We will use the term 'abstract' instead of 'unified', because 'abstract' is used in the object-oriented software engineering literature to refer to processes which provide a generic interface to specialized objects.

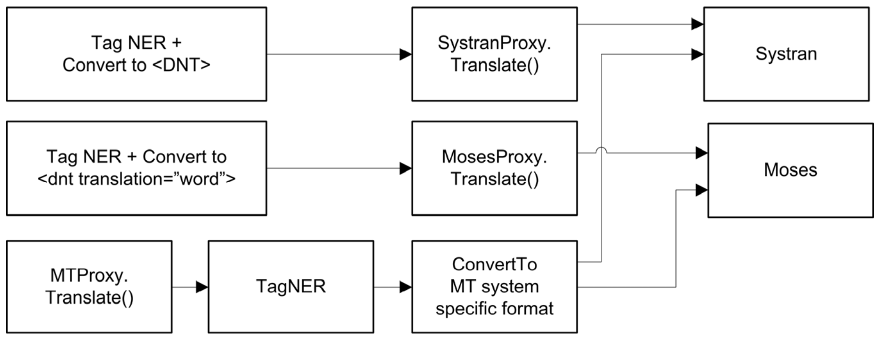

Tag NER + Convert to <DNT>
SystranProxy. Translate()
Systran
Tag NER + Convert to <dnt translation="word">
MosesProxy. Translate()
Moses
MTProxy. Translate()
TagNER
ConvertTo MT system specific format

Figure 2

can still be called over the abstract interface. The advantage is that a pre-processing or post-processing procedure has to be implemented *only once* for all available MT systems. More importantly, all created resources can also be re-used, without the need to re-target them to a different MT system.

2. Example: NER

We can illustrate how this works with Named Entity Recognition, as proposed in the previous section. Jurafsky & Martin (2009, p. 762) describe Named Entity Recognition as "the task in which proper names mentioned in a text are detected and categorised. Commonly, the notion of a named entity is extended to include things that aren't entities per se, but nevertheless have practical importance and do have characteristic signatures that signal their presence; examples include dates, times, named events, and other kinds of temporal expressions as well as measurements, counts, prices, and other kinds of numerical expressions". Traditionally, special tags are used to mark the named entities in a text, such as the SGML ENAMEX tag set (see for example (Grishman & Sundheim, 1996)). For an MT system to use and recognise these tags, a tagging routine and a tag conversion routine are required, as described in Figure 3. The tagging routine intercepts the text that needs to be translated and adds mark-up to it. The advantage over the non-abstract interface is that the NER can be implemented after the call to the abstract Translate() method. In an abstract MT interface it is also easy to add a TagNamedEntities(string sourceText) method. But the problem we now have is that each MT system handles meta-data differently (if it handles meta-data at all). For example, Systran requires tags that are formed according to its DNT (Do-Not-Translate) standard (Listing 1), while Moses uses a different format (Listing 2).

```
<!-- DNT -->
This text should not be translated.
<!-- /DNT -->
```

Listing 1

```
das ist ein kleines <n translation="dwelling"
prob="0.8">haus</n>
```

Listing 2

In order to accomplish our NER tagging and its processing by the MT systems, a conversion routine for each MT system needs to be added. Note that if no abstract MT interface would have been available, also the tagging routine would have to be implemented for each MT system separately. To make it even more interesting from a localisation-industry point-of-view, we add a typical file format, such as TTX (Brockmann, 2009) to the flow. TTX is a proprietary xml-format that basically wraps xml formats for translation. It is not an official standard, like XLIFF (Schnabel, et al., 2008) for example, but because of the widespread use of SDL's TagEditor (SDL Inc., 2009) – the most popular translation tool among translators – with which the format is associated, TTX has become the de facto standard for translation. We

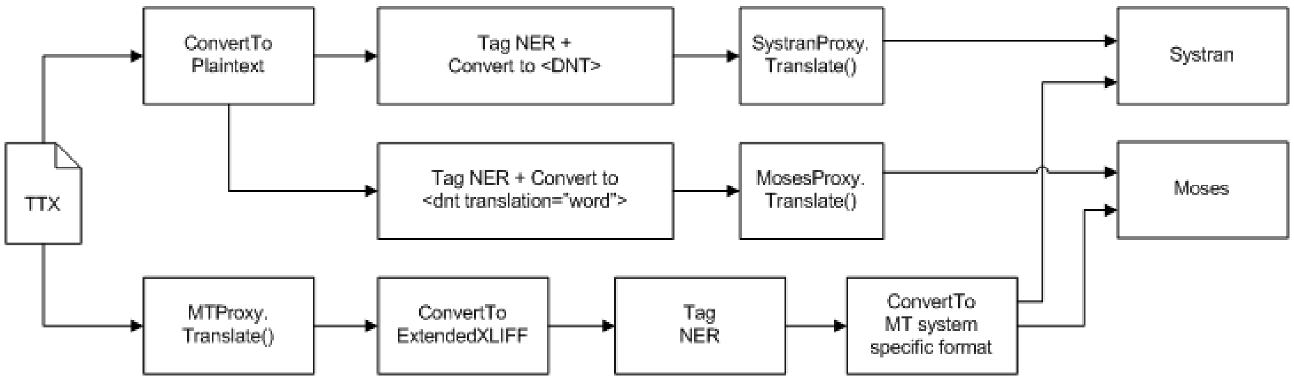

TTX
ConvertTo Plaintext
Tag NER + Convert to <DNT>
SystranProxy. Translate()
Systran
Tag NER + Convert to <dnt translation="word">
MosesProxy. Translate()
Moses
MTProxy. Translate()
ConvertTo ExtendedXLIFF
Tag NER
ConvertTo MT system specific format

Figure 3

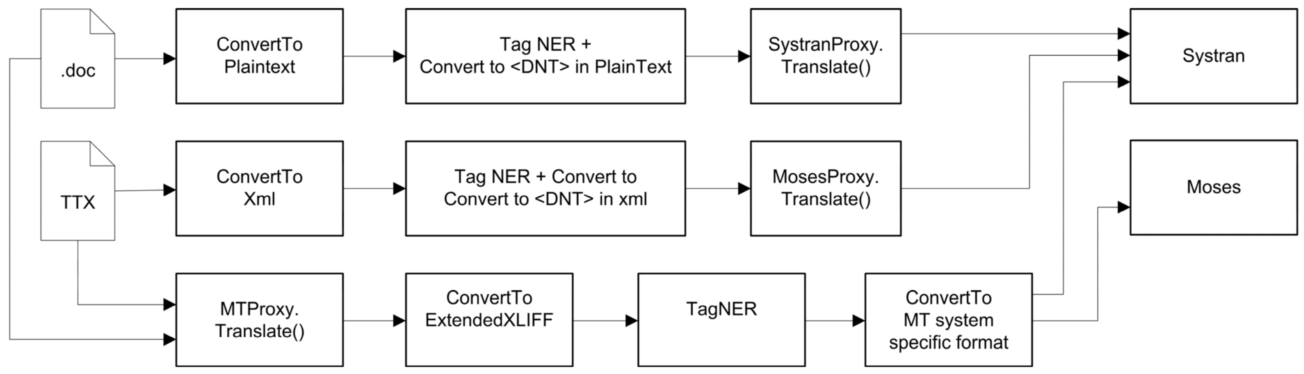

.doc
TTX
ConvertTo Plaintext
Tag NER + Convert to <DNT> in PlainText
SystranProxy. Translate()
Systran
ConvertTo Xml
Tag NER + Convert to Convert to <DNT> in xml
MosesProxy. Translate()
Moses
MTProxy. Translate()
ConvertTo ExtendedXLIFF
TagNER
ConvertTo MT system specific format

Figure 4

now also need a conversion routine, as most current MT systems only accept plaintext, xml or html. At this moment, the usefulness of an abstract MT interface becomes very clear. Figure 3 illustrates how the abstract MT interface differs from an implementation-specific MT interface. In the example TTX, a NER module exposed by a Web Service, and the Systran Web Service are used.

3. MT abstraction

To abstract the MT interface, the most common operations for MT were defined in a generic interface, while systems-specific settings were isolated in a configuration framework. The design objective was to keep the translation interface as tidy as possible while keeping the MT systems highly configurable. This was achieved by decoupling translation and configuration. Note that this design easily allows for versioning as new configuration parameters can be added once the architecture is in place. All changes to engines are absorbed by the configuration class and do not propagate further. In practice, this means that upgrades to an MT system will not affect the code base that uses this MT system. In the following example we will use the dictionary parameter – which can be configured for the Systran translation system (pseudo-code in Listing 3), but which is meaningless to Moses – to demonstrate interface abstraction. Listing 4 shows the resulting pseudo-code, Listing 5 shows the contents of the configuration.

```
ProxySystran.Translate("Das ist ein kleines
Haus.", Dictionaries.GeneralDictionary)
```

Listing 3

```
IMTSystem m_MTSystem =
MTSystemFactory.getMoses();
IConfig m_Config = Config.load(config.xml);
m_MTSystem.setConfig(m_Config);
m_MTSystem.config();
```

Listing 4

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<dictionary
file="GeneralDictionary.txt" />
</configuration>
```

Listing 5

The next step in the abstraction process was to develop an internal format, based on XLIFF, to avoid an exponential explosion of conversions when accommodating the Gateway System for the most frequently used translation formats. The resulting conversion produces a format that can be used to have meta-data converted to MT system-specific format implementations. Note that a very poor plaintext format could be used for this purpose, but this would result in the loss of important mark-up information. This procedure still requires a convertor for each MT system at the output side of the conversion module, but at the input side the abstract MT interface reduces the amount of conversions that normally would be required, with factor *n* as shown in Figure 4.

4. MT standardisation

For some obscure reason, MT service providers do not stick to language naming conventions. In the systems we used, a mixture of ISO 639 (Library of Congress, 1988) names and ad-hoc non-standard names were defined. This was a problem that also could be addressed by abstracting the MT interface. The system we developed only takes ISO 639-2 language names and ISO 3166-1 (ISO, 2007) country names as input. Internally, these codes are converted to system-specific language names. For our integration purpose, it was best to leave out language names and language pairs altogether, as the idea is that translation is carried out by using a translation configuration, consisting of an MT system, a set of processors (the NER module in our example) and a language pair, instead of an MT system with all sorts of system-specific parameters. Each configuration is assigned a key, so that a call to a Translate method, would only require the text and the configuration key. This makes the language names as required parameters obsolete. For historical reasons though, we added methods that include language names to our abstract interface, this also allows us to optimise for speed, as no database queries need to be performed when initialising the translation configuration. Some other speed optimisations will be discussed in the following section.

5. Shortcuts and REST API

For the sake of processing speed, shortcuts were added to allow for immediate translation of plaintext, html and xml. This resulted in better throughput for some applications in our current set-up, which require up to 20,000 document translations per day, as no conversion filters are required. In terms of coding effort, this added some overhead, because conversion filters had to be added for each file

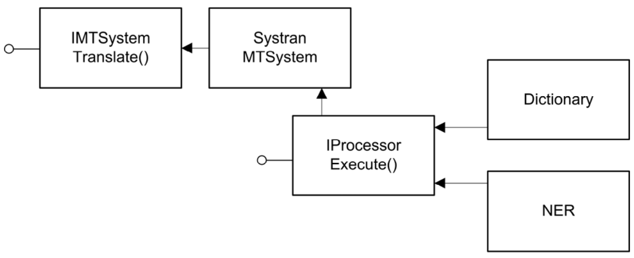


Figure 5

type, which would not be required if we would start from the modified XLIFF format, but business requirements could not justify a pure system. On the other hand, the programming overhead was minimal since the xml and html input types already inherently encode meta-information.
At the same time a REST (Fielding, 2000) API was implemented to avoid the protocol overhead associated with SOAP (Gudgin, et al., 2007). Again, this proved to produce better response and throughput times. Implementing the API itself was not so difficult, as we could build it on top of our Web Services framework (GlassFish 3 (Sun Microsystems Inc., 2009)).
Eventually we ended up with a double system as illustrated in Figure 6: for more complex workflows, the pure system is used, because of the file compatibility layer. For workflows that are focussed on speed, we can use the simplified APIs combined with the REST API. Note that it is still possible to use complex workflows with the *shortcut* API.

6. Processor abstraction

For the NER example to work, some sort of GetNamedEntities(string text) method needs to be provided in the abstract MT interface, but this would conflict with the design goal to keep the MT interface as tidy as possible. At the same time it would make it difficult to update the system when new processors are added. If, for example, a dictionary is added to the MT system, all references to the MT object would need to be updated with the new MT object that contains the new, say for example, ReplaceDictionaryEntriesWithDictionaryTranslations (string text) method.
This is a very common problem in software engineering. To avoid it, we implemented the Strategy pattern (Gamma, et al., 1995), which is a generic solution to these kinds of issues. As a result, we were able to isolate the abstract MT code from the additional pre-processing and post-processing code as described in Figure 5.

We deliberately chose the ReplaceDictionaryEntriesWithDictionaryTranslations(string text) method to demonstrate how generic and MT-technology-independent our solution is. Systran provides dictionaries within its translation system, and allows for customised dictionary encoding (which is a very cumbersome task), whereas Moses, for example, does not. By using the generic dictionary processor on top of the generic MT interface, we were able to bypass the proprietary Systran dictionary mechanism, and at the same time it allowed us to re-use the dictionary for any other MT system, without any additional effort.

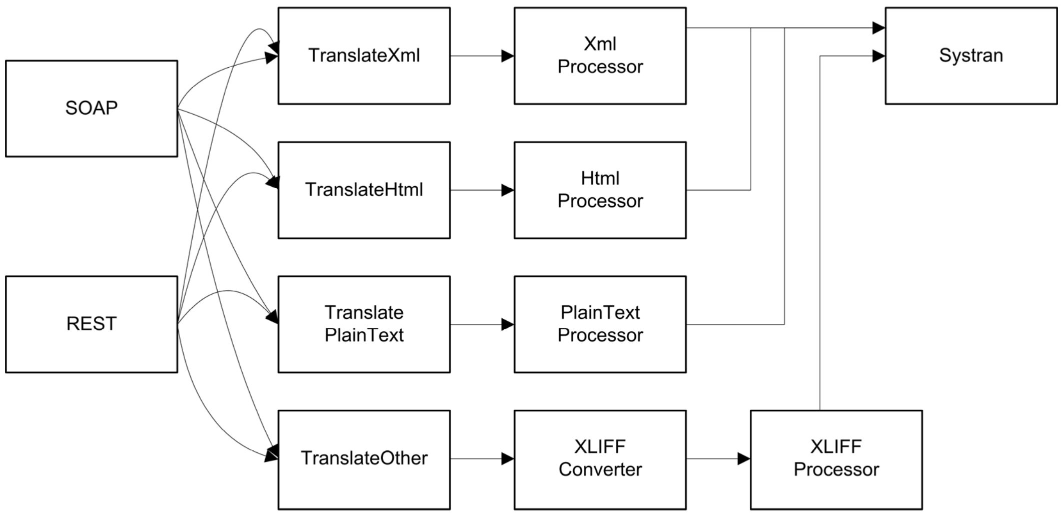


Figure 6

7. Support for research and production experiments: configuration at runtime

In order to provide a real-world solution for experimenting with MT extension, we made the whole pipeline configurable at runtime. We integrated the processing as a plug-in system that allows users to add any module they see fit, without the need to restart or recompile the server. Suppose, for example, that a NER module is required for Spanish. With the plug-in system, it is possible to look for all available Spanish NER solutions, write a processor for each of them, upload them to the server, attach each of them to the pipeline and do an experiment. After the experiment has been carried out and the results have been analysed, the best-performing configuration can be selected. Note that this makes the whole processing system vendor-technology neutral: it allows a client to search for the best components and assemble them in a fully customised pipeline. This opens up quite some possibilities, as often only vendor-specific, non-portable and non-reusable customisation is offered as a black-box solution.

8. Conclusion and future work

We presented an MT-technology-neutral framework that allows vendor-technology-neutral customisation. The design can be conceived as a technology stack as illustrated in Figure 7.

The first layer takes care of MT abstraction and exposes all MT methods, such as Translate(). The second layer takes care of file conversions. The third layer adds processing capabilities. The fourth layer, finally, converts all data to the MT-system-specific formats.
Each layer is isolated from the other layers, which allows for flexible extension. As a result, language resources created for the customisation of one specific MT system can now be ported to any other MT system within the framework. Additionally, the system provides a plug-in system, which facilitates experimenting with different configurations of pipelines. At the same time it provides the benefit that MT users can become independent from technology vendors.
In a next release, we are planning to integrate the GATE (Cunningham, et al., 2002) framework into the Gateway system. This will allow users with a linguistic background but no programming experience to add custom created functionality and resources to the translation pipeline.

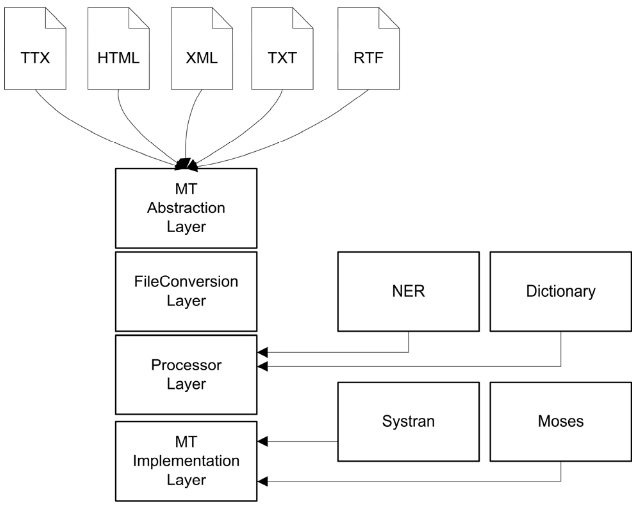


Figure 7

9. References

Babych, B., & Hartley, A. (2003). Improving Machine Translation Quality with Automatic named Entity Recognition. *Proceedings of the 7 th International EAMT workshop on MT*, (pp. 1-8). Budapest.

Barrault, L. (2010). MANY. Open Source Machine Translation System Combination. *The Prague Bulletin of Mathematical Linguistics* , 147-155.

Boitet, C., Bey, Y., & Kageura, K. (2005). Main research issues in building web services for mutualized, non-commercial translation. *Proceeding of the 6th Symposium on Natural Language Processing, Human and Computer Processing of Language and Speech, SNLP-05* .

Brockmann, D. (2009). *TTX Compatibility Guide for SDL Trados Studio 2009.* SDL Trados.

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001, March 15). *Web Services Description Language (WSDL) 1.1.* Retrieved March 2, 2010, from W3c.org: http://www.w3.org/TR/wsdl

Library of Congress. (1988). *ISO 639 Language Code List.* Retrieved March 2, 2010, from ISO 639 Language Code List: http://www.loc.gov/standards/iso639-2/php/code_list.php

Corbí-Bellot, A. M., Forcada, M. L., Ortiz-Rojas, S., Pérez-Ortiz, J. A., Ramírez-Sánchez, G., Sánchez-Martínez, F., et al. (2005). An Open-Source Shallow-Transfer Machine Translation Engine for the Romance Languages of Spain. *Proceedings of the European Associtation for Machine Translation, 10th Annual Conference*, (pp. 79-86). Budapest.

Cunningham, H., Maynard, D., Bontcheva, K., & Tablan, V. (2002). GATE: A framework and graphical development environment for robust NLP tools and applications. *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, (pp. 168-175). Philadelphia, USA.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Irvine: University of California.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

Google Inc.. (2009). *Google AJAX Language API.* Retrieved March 2, 2010, from Google AJAX Language API: http://code.google.com/intl/nl-BE/apis/ajaxlanguage/

Grishman, R., & Sundheim, B. (1996). Design of the MUC-6 Evaluation. *Proceedings of the TIPSTER Text Program: Phase II* (pp. 413-422). Vienna, Virginia, USA: Association for Computational Linguistics.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., et al. (2007, April 27). *SOAP Specifications.* Retrieved March 2, 2010, from SOAP Version 1.2 Part 1: Messaging Framework (Second Edition): http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

Heafield, K., & Lavie, A. (2010). Combining Machine Translation Output with Open Source. The Carnegie Mellon Multi-Engine Machine Translation Scheme. *The Prague Bulletin of Mathematical Linguistics* , 27-36.

Hoang, H., & Koehn, P. (2008). Design of the Moses Decoder for Statistical Machine Translation. *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, (pp. 58-65). Ohio.

ISO. (2007, September 21). *ISO - Maintenance agency for ISO 3166 country codes.* Retrieved March 2, 2010, from ISO - Maintenance agency for ISO 3166 country codes: http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.htm

Jurafsky, D., & Martin, J. H. (2009). *Speech and Language Processing* (Second Edition ed.). New Jersey: Pearson Education.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., et al. (2007). Moses: Open Source Toolkit for Statistical Machine Translation. *Annual Meeting of the Association for Computational Linguistics (ACL).* Prague.

Language Weaver Inc.. (2009). *LW Translation On Demand.* Retrieved March 2, 2010, from LW Translation On Demand: http://www.languageweaver.com/translator-software

Löwy, J. (2007). *Programming WCF Services.* O'Reilly Media.

Microsoft Inc.. (2010). *Microsoft .NET Framework.* Retrieved March 2, 2010, from Microsoft .NET Framework: http://www.microsoft.com/uk/net/Default.aspx

Nasierding, G., Xiang, Y., & Dai, H. (2004). Towards Extended Machine Translation Model for Next Generation World Wide Web. *International conference on grid and cooperative computing*, (pp. 1017-1020). Wuhan.

Peiris, C., & Mulder, D. (2007). *Pro WCF Practical Microsoft SOA Implementation.* New York, USA: Apress.

Sánchez-Cartagena, V. M., & Pérez-Ortiz, J. A. (2010). ScaleMT: a Free/Open-Source Framework for Building Scalable Machine Translation Web Services. *The Prague Bulletin of Mathematical Linguistics* , 97-106.

Schnabel, B., Jewtushenko, T., Savourel, Y., Reid, J., & Raya, R. M. (2008, February 1). *XLIFF 1.2 Specification.* Retrieved March 2, 2010, from XLIFF 1.2 Specification: http://docs.oasis-open.org/xliff/xliff-core/xliff-core.html

SDL Inc. (2009). *SDL Trados Studio 2009 Professional.* Retrieved March 2, 2010, from SDL Trados Studio 2009 Professional: http://www.sdl.com/en/sites/sdl-trados-solutions/desktop-products/sdl-trados/

Sun Microsystems Inc.. (2009, January 23). *Project GlassFish.* Retrieved March 2, 2010, from Project GlassFish: https://glassfish.dev.java.net/

Systran Inc.. (2009). *Translation Server - Sytran Enterprise Server 7.* Retrieved March 2, 2010, from Translation Server - Sytran Enterprise Server 7: http://www.systran.co.uk/translation-products/server/systran-enterprise-server

The Apache Software Foundation. (2009, October 23). *Apache Axis2.* Retrieved March 2, 2010, from Apache Axis2: http://ws.apache.org/axis2/