Deliverable D 4.1

Language Processing Chains

Grant Agreement number: 250467

Project acronym: ATLAS

Project title: Applied Technology for Language-Aided CMS

Project type: ■Pilot B

Deliverable D 4.1 S Language Processing Chains 30.12.2011

Project coordinator name, title and organisation:

Anelia Belogay, CEO, Diman Karagiozov, CTO,

Tetracom Interactive Solutions

Tel: +35924950444

Fax: +35924950443

E-mail:anelia@tetracom.com, diman@tetracom.com

Project website address: www.atlasproject.eu

Authors:

Polivios Raxis, Ioannidis Dimosthenis (Atlantis), Svetla Koeva, Angel Genov (IBL-DCL), Maciej Ogrodniczuk, Adam Przepiórkowski (ICS PAS), Anelia Belogay, Emil Stoyanov, Diman Karagiozov (Tetracom), Cristina Vertan (UHH), Dan Cristea, Eugen Ignat (UAIC)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	X
C	Confidential, only for members of the consortium and the Commission Services	

Revision	Date	Author, organisation	Description
0.1	14/10/2010	M. Ogrodniczuk (ICS PAS)	First complete outline of the document.
0.2	21/10/2010	M. Ogrodniczuk, A. Przepiórkowski (ICS PAS)	Added draft documentation of the linguistic tools being integrated.
0.3	23/11/2010	M. Ogrodniczuk, A. Przepiórkowski (ICS PAS)	Additional notes on LPC usage.
0.4	27/09/2011	M. Ogrodniczuk, A. Przepiórkowski (ICS PAS)	Added sections concerning internal testing process and templates.
0.5	6/12/2011	M. Ogrodniczuk, A. Przepiórkowski (ICS PAS)	Document restructured after project meetings in Thessaloniki and Sofia.
0.6	27/12/2011	M. Ogrodniczuk, A. Przepiórkowski (ICS PAS), D. Cristea, E. Ignat (UAIC), D. Karagiozov, E. Stoyanov, A. Belogay (Tetracom), S. Koeva, A. Genov (IBL-DCL), D. Ioannidis, P. Raxis (Atlantis), C. Vertan (UHH)	Added documentation and integration notes of language-dependent parts of the LPCs.
0.8	29/12/2011	M. Ogrodniczuk, A. Przepiórkowski (ICS PAS), D. Karagiozov (Tetracom)	Added references to the latest test results after performance optimization.
1.0	4/01/2012	M. Ogrodniczuk, A. Przepiórkowski (ICS PAS)	Final version of the document.

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

D4.1 is the deliverable of the WP4 “Language Processing Chains” (LPCs) of the ATLAS project.

This document includes:

- description of language processing chains in ATLAS for Bulgarian, English, German, Greek, Polish and Romanian,
- short guide for integration and usage of heterogeneous language-dependent tools,
- description of the integration test process and references to detailed test results.

1 DOCUMENTATION OF INTEGRATED LINGUISTIC TOOLS

1.1 Common pre-processing tools

The main task performed by the pre-processing phase of the linguistic framework in ATLAS is to extract the textual content from various document sources and send it to the corresponding language processing chain (LPC). This process consists of a mime detector, text extractor and language detector.

The architecture of the pre-processing platform is based on the OSGi implementation from Eclipse – Equinox and allows flexible, plugin- and feature-oriented deployment. In other words, one component, or subcomponent, can be easily replaced with an alternative implementation which provides the same functionality.

MIME type detector

The mime type detector is responsible for detecting the correct mime type of an unknown document sources. The implementation of this sub-component is based on two external tools, available only for Linux platforms

- “/usr/bin/file” – when invoked with „-i” or „--mime” and a filename the tool returns the detected mime type of the file and its character encoding;
- „gnomevfs-info” – this tool provides a lot more information about the file, including its mime type. The precision of the „gnomevfs-info” tool is higher than the the „/usr/bin/file” and is preferred mime type detection program if gnome shell is available. Below is an example, how the mime type is extracted from the fill information of a file: gnomevfs-info -s /tmp/video | awk '{FS=":"} /MIME type/ {gsub(/^[\t]+|[\t]+$/, "",$2); print $2}'

Text extractor

Once the mime type of the document source is detected, pre-processing engine is able to extract the text from the source, using mime-specific text extractor. The table below enlists the currently available text extractors in ATLAS.

MIME type	3rd party Java library or external tool	Remarks
application/ x-fictionbook+xml application/x-chm application/ x-mobipocket-ebook application/prs.plucker	Calibre http://calibre-ebook.com/	The e-books are converted to ePub format and then the ePub text extractor is used for providing the document text.
image/vnd.djvu	/usr/bin/djvutxt	
application/epub+zip	Epublib – a Java epub library http://www.siegmann.nl/ epublib	
text/html application/xhtml+xml	Jericho HTML Parser http://jericho.htmlparser.net	
application/x-ms-reader application/x-obak	/usr/bin/lit2epub	The LIT file is converted to ePub format and then the ePub text extractor is used for providing the document text.
application/ x-mimearchive message/rfc822		Java standard implementation of javax.activation and javax.mail is used.
application/vnd. ms-office application/msword application/vnd. ms-powerpoint application/vnd. ms-excel	Apache Jackrabbit http://jackrabbit.apache.org/	Text extraction from modern (2008-2010) MS office files is provided by a separate plugin.

MIME type	3rd party Java library or external tool	Remarks
application/vnd.openxmlformats-officedocument.wordprocessingml.document application/vnd.openxmlformats-officedocument.spreadsheetml.sheet application/vnd.openxmlformats-officedocument.presentationml.presentation	Apache POI http://poi.apache.org/	
application/vnd.oasis.opendocument.text	Apache Jackrabbit http://jackrabbit.apache.org/	
application/pdf	Apache PDFBox http://pdfbox.apache.org/ /usr/bin/pdftotext	The default PDF text extractor, based on PDFBox, provides better quality compared to the external tool. However, sometimes text cannot be extracted at all, thus the pdftotext is used as fallback extractor.
text/rtf application/rtf	Apache Jackrabbit http://jackrabbit.apache.org/	
Fallback text extractor	/usr/bin/unoconv	The external executable is using OpenOffice headless installation to convert documents from various formats to plain test. This extractor requires significantly more resources (CPU and Heap), thus it is used as a last-resort to retrieve the text from a document.

Language recognizer

The language detection sub-component is based on NGramJ Java library (http://ngramj.sourceforge.net/). The n-gram language profiles have been extended with a model for the Croatian language, built on a corpus of Croatian Wikipedia articles.

1.2 Language processing tools for Bulgarian

The Bulgarian language processing chain consists of: BG sentence splitter, BG tokenizer, BG Part of Speech (POS) tagger, BG lemmatizer, BG Noun phrase (NP) extractor, BG Named entity (NE) recognizer, BG Word sense (WS) disambiguator and BG Stop word (SW) recognizer. All tools are self-contained and designed to work in a chain, i.e. the output of the previous component is the input for the next component, starting from the sentence splitter and following the strict order for the tokenizer, POS tagger, and lemmatizer. The rest tools use the lemmatizer output and there is no dependencies in their execution order. Each tool associates tokens from the input text with different sets of annotation tags. The tools exchange data among the chain using so called vertical format.

The common form of the vertical format is:

```
tok1\tTag1\tTag2\tTag3 .....\tTagk\n
tok2\tTag1\tTag2\tTag3 .....\tTagk\n
.....
tokn\tTag1\tTag2\tTag3 .....\tTagk\n
```

Figure 1. Example of the format for the exchange of data in the Bulgarian LPC

In the vertical format the tokens are separated by a newline, whereas the annotation tags – by a tab character (\t). One and the same tool can assign tags with a complex structure (marked with delimiters), different types of annotation separated in different columns as well as an annotation to a group of tokens. Each tool accumulates tags in fixed positions at one or several columns.

BG sentence splitter

The sentence splitter marks the sentence boundaries in raw Bulgarian text. The sentence splitter applies regular rules and lexicons. Both - regular rules and lexicons - are manually crafted by an expert. For example the general rule for sentence splitting is: ([\.\?\!]\n?\s*)(\-?\s?[A-Я1-9A-Z]). Lists of lexicons (for recognizing abbreviations after which there must be or there might be a capital letter, a number, etc. in the middle of the sentence) are applied before the regular rules. The lexicons are compiled by a separate tool - the Lexicon compiler, as minimal acyclic final state automata which allows an effective processing.

Details:

- Input format: raw text UTF-8 encoded
- Output format:
  - The sentence border is marked by inserting an annotation, for instance <S>.
  - Sentence borders are represented as a position and length which allows the incoming text to be kept unchanged as well as an easy integration in different systems for annotation.
- UIMA integration class: dcl.bas.uima.SentenceSplitter class in Bulgarian Sentence splitter primitive engine as part of the Bulgarian Aggregate
- Required language resources: lexicons compiled as finite state automata (optional) and regular rules (mandatory)
- Programming language: C++
- Related tools: Lexicon compiler
- Other dependencies: no
- Access condition: as a source code and precompiled binary for 32 and 64 bit Linux platform
- Copyright: DCL
- Licensed under: GPL

BG tokenizer

The Bulgarian tokenizer demarcates strings of letters, numbers, punctuation marks, special symbols, combinations of them and empty symbols. Regular patterns are used to recognize some simple cases of named entities that mean dates, fractions, emails, internet addresses, abbreviations, etc. The tokenizer classifies each recognized token (for example: small Cyrillic letters, capital Latin letters, etc.). The tokenizer utilizes finite state transducers for token recognition and type matching. The token demarcating and token classifying rules are defined and compiled as finite state transducers with a separate tool – the ParseEst.

Details:

- Input format: raw text UTF-8 encoded
- Output format: vertical format

  For example if the text: 'Писмо до Ижан Йоцов от Враца. То е последното Ботево писмо.' (A letter written to Ivan Yotsov from Vratsa. This is the last letter written by Botev.) is passed through the tokenizer, the output in a vertical format will be:

```
Писмо       TOK_FUCA 0,5
до          TOK_LCA  6,2
Иван        TOK_FUCA 9,4
Йоцов       TOK_FUCA 14,5
от          TOK_LCA  20,2
Враца       TOK_FUCA 23,5
.           TOK_FS   28,1
То          TOK_FUCA 30,2
е           TOK_LCA  33,1
последното  TOK_LCA  35,10
Ботево      TOK_FUCA 46,6
писмо       TOK_LCA  53,5
.           TOK_FS   58,1
```

Figure 2. Example of the output format of the BG tokenizer

  Here the first column contains the graphical representation of tokens, the second column consists of the associated token tags and the third column represents the position and length of tokens.

- UIMA integration class: dcl.bas.uima.Tokenizer class in Bulgarian Tokenizer primitive engine as part of the Bulgarian Aggregate
- Required language resources: token demarcating and token classifying rules compiled as finite state transducers
- Other dependencies: no

- Programming language: C++
- Related tools: ParseEst
- Access condition: as a source code and precompiled binary for 32- and 64-bit Linux platform
- Copyright: DCL
- Licensed under: GPL

BG POS tagger

The Bulgarian POS tagger marks up each word with the most probable Part of Speech and unambiguous morphosyntactic information among the set of tags associated with a given word. The tagger is based on SVM (Support Vector Machines) learning. The tagger predicts the POS tag of a word based on a set of features describing the word and its context. These features are words, word bigrams and trigrams within a window of words around the currently tagged word; POS tags, POS tags bigrams and trigrams in the current window, and information about suffixes, prefixes, capitalization, hyphenation etc. for the unknown words. The tagger is trained and tested on manually POS disambiguated corpus. The strategy chosen for training Bulgarian tagger is two passes in both directions; a window of five tokens, the currently tagged word being on the second position; two and three-grams of words or tags or ambiguity classes, lexical parameters as prefixes, suffixes, sentence borders, and capital letters. The trained model is applied to disambiguate texts. The precision of the tagger up to the moment is 96,58%.

The tagger exploits the SVMTool, an open source utility for training of tagger models and their application for POS disambiguation. To improve the robustness of the SVMtool an alternative disambiguation module has been developed in C++. The new implementation provides an integration with the lower levels of annotation, full Unicode support and improves the model loading speed.

The BG POS tagger is executed in two modes:

- 'Command line' mode in which the tagger is run with a command line argument containing the name of the input file. The generated output is returned to the standard output.
- 'Server' mode in which the tagger listens on a TCP socket for client connections. The input and output data are provided by the TCP socket. Concurrent client connections are allowed.

Details:

- Perquisites: sentence split and tokenized text
- Input format: vertical format
- Output format: vertical format with accumulated POS annotation

  For example, if the tokenised output of the above sentences is processed by the tagger the output in a vertical format will be as follows:

Писмо	Ns
до	R
Иван	NHs
Иоцов	NHs
от	R
Враца	Ns
.	U
То	Ps
е	Vs
последното	As
Ботево	Ns
писмо	Ns
.	U

Figure 3. Example of the output format of the BG POS tagger

Here the last column contains the POS tags determined by the POS tagger.

- UIMA integration class: dcl.bas.uima.POSTagger class in Bulgarian POS tagger primitive engine as part of the Bulgarian Aggregate
- Required language resources: trained SVM language model for POS tagging
- Programming language: C++
- Related tools: no
- Other dependencies: no
- Access condition: as a source code and a precompiled binary for 32- and 64-bit Linux platform
- Copyright: DCL
- License: LGPL

BG lemmatizer

The Bulgarian lemmatizer determines for a given word form its lemma and detailed morphosyntactic annotation. The lemmatization is based on an unambiguous association between the tagger output and information encoded in a large grammatical dictionary of Bulgarian language. At the tagging a reduced tagset is used (75 word classes compering to 1029 unique grammatical tags in the dictionary) compiled in a way that the minimum necessary information for unambiguous association with the respective lemma to be ensured. A small number of rules and preferences are also implemented to limit the ambiguity in

lemmatization. The grammatical dictionary is represented as a finite state automaton which itself provides a very efficient lookup. The dictionary is part of the executable file.

Details:

- Perquisites: POS tagged text
- Input format: vertical format
- Output format: vertical format with accumulated annotation for word lemma and detailed morphosyntactic annotation

If the output of the tagger (above example) is processed by the lemmatizer the output in a vertical format will be as follows:

Писмо	писмо	NCNson
до	до	R
Иван	иван	NHMsom
Иоцов	Иоцов	NHs
от	от	R
Враца	враца	NHsom
.	.	U
То	аз	PHi3sn
е	съм	VLINr3s
последното	последен	Asnd
Ботево	ботев	Asno
писмо	писмо	NCNson
.	.	U

Figure 4. Example of the output format of the BG lemmatizer

The fourth column contains the respective lemmas while the fifth one represents the extended POS tags assigned by the lemmatizer.

- UIMA integration class: dcl.bas.uima.Lemmatizer class in Bulgarian Lemmatizer primitive engine as part of the Bulgarian Aggregate
- Required language resources: no
- Programming language: C++
- Related tools: no
- Other dependencies: no
- Access condition: as a precompiled binary for 32-bit Linux platform
- Copyright: DCL
- Licensed under: DCL license

BG NP extractor

Bulgarian NP extractor recognizes and annotates noun phrases and their heads in the output text. The extractor is rule based parser and exploits a manually crafted grammar designed according to the following criteria: to recognize unambiguous phrases, to exclude pronouns as modifiers as well as relative clauses. The rules are defined in ParseEst XML based formalism as context-free or context-dependent rules, unlimited to the number of constituents, based on the part of speech tags and values of grammatical categories of word forms and providing annotation for phrase boundaries and heads. As a result an extensive number of noun phrases and their heads are unambiguously annotated - the number varies in different types of texts. The generic tool used for Bulgarian (as well as for English) NP extraction is ParseEst, a system for compiling and processing linguistic rules. It consists of two modules: lr_builder and lr_engine. ParseEst lr_builder compiles linguistics rules into a finite state transducers where the input is an XML file with rules definitions and the output – compiled finite state transducer and meta symbol definitions. For a given rule a finite state transducer is constructed via the ParseEst. For each rule group there is a corresponding transducer, constructed by composition of all single rule transducers. The resulted transducers are composed according to their priority to result in a single transducer at the end. The transducers are applied with the ParsesEst lr_engine over a lemmatized text and they add syntactic information represented by means of annotations, such as brackets and labels for noun phrase head.

Details:

- Perquisites: lemmatized text
- Input format: vertical format
- Output format: vertical format with marked begin and end indexes

  The same example is give bellow with a marked noun phrase “the last letter from Botev”:

То	аз	PHi3sn	
е	съм	VLINr3s	
последното	последен	Asnd	<<<
Ботево	ботев	Asno	
писмо	писмо	NCNson	>>> NP
.	.	U	

Figure 5. Example of the output format of the BG NP extractor

- Required language resources: ParseEst compiled NP grammar
- UIMA integration class: dcl.bas.uima.npextractor. BulgarianNounPhraseExtractor class in Bulgarian NP extractor primitive engine as part of the Bulgarian Aggregate
- Programming language: C++
- Related tools: ParseEst

- Other dependencies: no
- Access condition: as a precompiled binary for 32- and 64-bit Linux platform
- Copyright: DCL
- Licensed under: DCL license

BG NE recognizer

The tool recognizes and marks different types of named entities (NE) in the input text. Named entity recognition is performed by the generic tool ParseEst for compiling and processing linguistic rules. The rules are defined in ParseEst XML based formalism as context-free rules, based on different kind of input information (words, lemmas, part of speech tags and values of grammatical categories of lemmas or word forms, lexicons), and providing annotation for named entity boundaries and types. ParseEst lr_builder compiles linguistics rules into a finite state transducers where the input is an XML file with rules definitions and the output – compiled finite state transducer and meta symbol definitions. The transducers are applied with the ParseEst lr_engine over a lemmatized text. Both – lr_builder and lr_engine operate with the compiled lexicons produced by a separate tool – the Lexicon compiler. As a result NE defined to cover dates, money, percentage and time expressions, names of organizations, locations and persons are recognized.

Details:

- Perquisites: lemmatized text
- Input format: vertical format
- Output format: vertical format with accumulated annotation for boundaries and types of recognized named entities
- The same example is given bellow with marked named entities for a person and location.

Писмо	писмо	NCNson	
до	до	R	
Иван	иван	NHMsom	<<<
Иоцов	Иоцов	NHs	>>> Person
от	от	R	
Враца	враца	NHsom	<<< >>> Location

Figure 6. Example of the output format of the BG NE recognizer

- Required language resources: ParseEst compiled NE grammar
- UIMA integration class: dcl.bas.uima.NERecognizer class in Bulgarian NE recognizer primitive engine as part of the Bulgarian Aggregate
- Programming language: C++

- Related tools: ParseEst, Lexicon compiler
- Access condition: as a precompiled binary for 32- and 64-bit Linux platform
- Copyright: DCL
- Licensed under: DCL license

BG WS disambiguator

Bulgarian WS disambiguator is a web service for identifying the most appropriate sense (among the senses defined in Bulgarian wordnet) for a given word in a given context. The service is based on a multi-component algorithm for word sense disambiguation developed by the DCL. At present five independent "weak" classifiers (two knowledge based and three statistical) and an ensemble one (combining all of them) are used for the disambiguation. Each classifier provides a confidence distribution over the senses for a particular single word or multiword expression (lists of pairs: <sense, confidence>, where the sum of the confidences is 1, are generated). The ensemble classifier uses a weighted sum of the five weak ones. The current version outperforms the calculated random sense baseline (~40%) by 24 points with an overall precision of ~65%. The ensemble disambiguator shows a good overall improvement in terms of precision outperforming the best of the weak classifiers by approximately 5 points (~65% vs. ~60%). Although some of the algorithms process only part of the words in a given text, the coverage of the system is near 100%.

Details:

- Perquisites: lemmatized text
- Authentication method: digest authentication URL: http://dcl.bas.bg/wsd
- Input format: vertical format
- Output format: vertical format
- UIMA integration class: dcl.bas.uima.BgWSD class in Bulgarian WSD primitive engine as part of the Bulgarian Aggregate
- Required language resources: no
- Other dependencies: no
- Programming language: Python
- Related tools: no
- Access condition: web service with restricted access
- Copyright: DCL
- Licensed under: DCL license

BG SW recognizer

Stop words are closed class words with high frequency such as prepositions, conjunctions, pronouns, etc. The BG SW recognizer marks such words in a given text such as they can be filtered by the high level processing components. The tool exploits a stop list lexicon which is

compiled as a finite state automaton for an efficient lookup. Similarly with the lemmatizer the automaton is embedded into the executable file, which makes the tool self-contained.

Details:

- Perquisites: tokenized text
- Input format: vertical format
- Output format: vertical format with annotated stop words
- UIMA integration class: dcl.bas.uima.StopWords class in Bulgarian SW recognizer primitive engine as part of the Bulgarian Aggregate
- Required language resources: no
- Other dependencies: no
- Programming language: C++
- Related tools: no
- Access condition: as a precompiled binary for 32- and 64-bit Linux platform
- Copyright: DCL
- Licensed under: DCL license

ParseEst

The BG tokenizer, BG NP extractor and BG NE recogniser use ParseEst – a generic tool for crafting, compiling and applying linguistic rules. The tool can be used for other languages and for different tasks involving development of language grammars. The rules are formulated in the ParseEst XML based formalism. ParseEst consists also of two main modules: lr_builder and lr_engine.

ParseEst lr_builder

A tool for compilation of linguistics rules as a finite state transducer.

Details:

- Input: XML file containing the rules definitions
- Output: compiled finite state transducer, meta symbols definitions
- Programming language: C++
- Access condition: as a precompiled binary for 32- and 64-bit Linux platform
- License: DCL license
- Copyright: DCL

ParseEst lr_engine

A tool for linguistic rules application over an annotated text.

Details:

- Input: lemmatized text

- Output: vertical format
- Required language resources: linguistic rules compiled by the lr_builder
- Programming language: C++
- Access condition: as a precompiled binary for 32- and 64-bit Linux platform
- License: DCL license
- Copyright: DCL

Lexicon compiler

The BG sentence splitter and BG NE recogniser use the Lexicon compiler – a generic tool for compilation of large lexicons that can be exploited as a relatively language independent tool for different purposes. A lexicon is a list of words, collected according to certain criteria – i.e. family names, proper names, etc. The lexicon definition file specifies available lexicons and defines union operations among them. All lexicons are compiled as a finite state automaton.

Details:

- Input: lexicons and lexicon definition file
- Output: compiled finite state automata
- Programming language: C++
- Access condition: as a precompiled binaries for 32- and 64-bit Linux platform
- License: DCL license
- Copyright: DCL

1.3 Language processing tools for English

The English LPC consists of the following components, executed in a sequence:

- Paragraph splitter – based on regular expressions „((^.*\S+.*$)+)". More information can be found in the com.tetracom.uima.text.ParagraphSplitter class code.
- URL and Email annotator – based on regular expressions. The URL and emails contains „." (dot) which confuses the subsequent components. Thus, URLs and Emails found in the text are annotated as named entities and skipped by the other annotators in the chain.
- Sentence splitter – the English sentence splitter uses OpenNLP[1] SentenceDetectorME from the opennlp.tools.sentdetect package for splitting up raw text into sentences. A maximum entropy model is used to evaluate the characters ".", "!", and "?" in a string and to determine if they signify the end of a sentence. More information can be found at: http://opennlp.sourceforge.net/api/opennlp/tools/sentdetect/ SentenceDetectorME.html.

[1] Sentence splitter, Tokenizer, POS tagger and primary entity recognizer for English are based on OpenNLP project (http://opennlp.sourceforge.net/). OpenNLP hosts a variety of Java-based NLP tools which perform sentence detection, tokenization, POS tagging, chunking and parsing, named-entity detection, and coreference. All OpenNLP tools are working with Penn Treebank tagset (http://bulba.sdsu.edu/jeanette/thesis/PennTags.html).

- Tokenizer – the tokenizer uses OpenNLP TokenizerME from the opennlp.tools.tokenize package. Current implementation of the tokenizer instance is not thread safe, thus a separate tokenizer must be instantiated for each thread. However, the TokenizerModel instance can be reused for each of the tokenizer instances in order to save memory. More information can be found at: http://opennlp.sourceforge.net/api/opennlp/tools/tokenize/TokenizerME.html.
- POS Tagger – uses OpenNLP POSTaggerME from the opennlp.tools.postag package. All punctuation characters are marked with „PU". More information could be found at: http://opennlp.sourceforge.net/api/opennlp/tools/ postag/POSTaggerME.html
- Lemmatizer – uses the Morphological Analysis tool from the RASP (Robust Accurate Statistical Parsing) system (RASP System second distribution RASPv2). The development of the RASP system was funded by the UK EPSRC within the project "Robust Accurate Statistical Parsing (RASP)" (grants GR/N36462 and GR/N36493). Since the end of that project it has continued to be extended and enhanced on an on-going basis. The tagset this tool uses is close to CLAWS C7 although it is in fact a cut-down version of the CLAWS C2 tagset. The POS tagset, used by the OpenNLP POS tagger, has to be converted to CLAWS C2 tagset in order to use the RASP lemmatizer in the LPC. A new version of RASP system became available at the time of writing this document. The new version 3 will be adopted in the English LPC by the end of the project.
- Noun phrase extractor – the grammar and structure of the English noun phrase are described in a set of 14 rules, following the format of ParseEst sub-component.
- Named entities recognizer – NEs are extracted using the OpenNLP NameFinderME from the opennlp.tools.namefind package. The tool recognizes seven different types of named entities – date, time, location, money, organization, percentage and person. Tetracom added two additional named entities to be recognized, using regular expressions – e-mails and URLs.

1.4 Language processing tools for German

The language processing chain for German rely on two type of components:

- modules belonging to the Weighted Constraint Dependency Parser developed at the University of Hamburg (http://nats-www.informatik.uni-hamburg.de/view/CDG/WebHome)
- open source modules already integrated in the GATE Architectural Framework (http://gate.ac.uk/)

General Description of the Weighted Constraint Dependency Parser (WCDG)

The WCDG-System is based on the Weighted Constraint Dependency Grammar formalism which describes natural language exclusively as dependency structure, i.e. ordered, labelled pairs of words in the input text. It performs natural language analysis under the paradigm of constraint optimization, where the analysis that best conforms to all rules of the grammar is returned. The rules are explicit descriptions of well-formed tree structures, allowing a modular and fine grained description of grammatical knowledge. In general these constraints

are defeasible, since many rules about language are not absolute, but can be pre-empted by more important rules. The strength of constraining information is controlled by the grammar writer: fundamental rules must always hold, principles of different import have to be weighted against each other and general preferences that only take effect when no other disambiguating knowledge is available can be formulated in a uniform way. Among the main features of the WCDG we can enumerate the following:

- Supports multiple levels of dependencies (e.g., Syntactic, Semantic roles or References of relative pronouns)
- Arbitrary information sources can be integrated into the grammar as predictors (e.g. POS-Taggers, PP-Attachers, Chunkers, visual context information or even other parsers)
- Anytime capable, i.e., the parser provides an analysis even if interrupted early
- Supports interactive grammar development
- Can assist in corpora annotation
- Diagnostic features: provides the grammatical constraints violated by a dependency analysis
- Incremental parsing mode available
- Syntactic prediction for incomplete sentences

The system is based on a hand-crafted set of approximately 1000 constraints, describing in fact the German grammar, as described in [Foth04][2].

As described in [Fothetal.04][3] Weighted constraint dependency grammar (WCDG) [Schröder, 2002][4] is an extension of the CDG formalism first described by [Maruyama, 1990][5]. It describes the structure of natural language as a set of labelled subordinations: each word is either subordinated to exactly one other word or considered the root of the syntax tree (also called a NIL subordination). See Figure 7 for an example.

[2] Kilian A. Foth. Writing weighted constraints for large dependency grammars. In Recent Advances in Dependency Grammar, Workshop COLING 2004, 2004.

[3] Kilian Foth, Michael Daum, and Wolfgang Menzel. A broad coverage parser for German based on defeasible constraints. In H. Christiansen, P. R. Skadhauge, and J. Villadsen, editors, Proc. Constraint Solving and Language Processing, Workshop Proceedings, Datalogiske Skrifter No. 99, pages 88-101, Roskilde Universitetscenter, Denmark, 2004.

[4] Ingo Schröder. *Natural Language Parsing with Graded Constraints*. PhD thesis, Dept. of Computer Science, University of Hamburg, Germany, 2002.

[5] Hiroshi Maruyama. 1990. Constraint dependency grammar. Technical Report RT0044, IBM Research, Tokyo Research Laboratory.

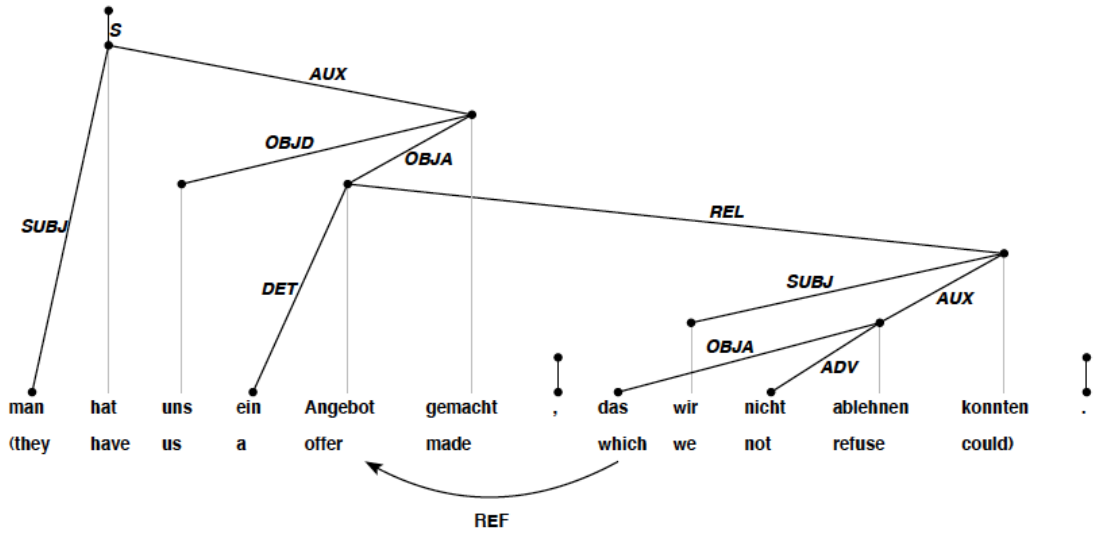


Figure 7. Dependence analysis of a German sentence

Each subordination is annotated with one of a fixed set of labels such as 'subject', 'direct object' or 'indirect object', but no larger groups of words carry labels. This means that there is no direct equivalent to the constituents postulated by phrase structure grammar. Since there are no constituents, there are also no generative rules along the lines of S→ NP VP; these are often problematic for languages with free or semi-free word order since they mingle dominance principles with linear precedence rules. Instead, the grammar rules take the form of declarative constraints about permissible subordinations. These constraints can reference the position, reading and lexical features of the concerned word forms, as well as features of neighbouring dependency edges. Every subordination that is not forbidden by any constraint is considered valid. The goal of the parser is to select a set of subordinations that satisfies all constraints of the grammar. As an example of a constraint, consider the rule that normal nouns of German require a determiner of some kind, either an article or a nominal modifier, unless they are mass nouns. This rule can is in WCDG formulated as in Figure 8:

```
{X:SYN} : 'missing determiner' : 0.2 :
  X@cat = NN
  ->
  exists(X@mass_noun) |
  has(X@id, DET) |
  has(X@id, GMOD);
```

Figure 8. Example of a constraint rule in WCDG

It states that for each subordination on the syntax level (SYN), a word with the category 'normal noun' (NN) must either bear the feature 'mass noun' or be modified by a determiner

(label DET) or a genitive modifier (label GMOD). Each constraint bears a score between 0.0 and 1.0 that indicates its importance relative to other constraints. The acceptability of an analysis is defined as the product of the scores of all instances of constraints that it violates. This means that constraints with a score of 0.0 must be satisfied if at all possible, since violating them would yield the worst possible score.3 Note that the score of the determiner constraint is 0.2, which means that missing determiners are considered wrong but not totally unacceptable.

The WCDG grammar uses constraint at the syntactical and semantical level. For the purposes of the German LPC w use only the syntactic level. A particularity of the system is also that the parsing process can be stopped at any time, or can be run up to a certain limit of time. In this case only a partial parsing will be performed, the result is not completely accurate but is obtained fast. In fact a first prediction is produced quite fast (seconds) while the optimisation process takes quite long. this optimisation process is not relevant for the LPC as we are not producing a full parse tree.

The WCDG system works with independent sentences. It integrates a tokenizer, and can be used together with a POS tool. Within the WCDG system an open source version of the TnT system is used. This was further integrated in the German LPC. As lexical resources the system uses a lexicon of about 23 000 full forms of closed classes as well as patterns for recognising morphological features of nouns, verbs etc.

The German LPC system general architecture

We used the main components of the WCDG as follows:

- the tokenizer was modified in order to recognize also sentence boundaries
- the POS tagger gives a first prediction of a possible part of Speech with a certain probability.
- WCDG is launched under a time constraint and following the results POS or token boundaries are corrected (especially verb particles vs. prepositions)
- Lemmas and Proper Names are extracted based on the lexical resource included in the WCDG

The architecture is described in Figure 9.

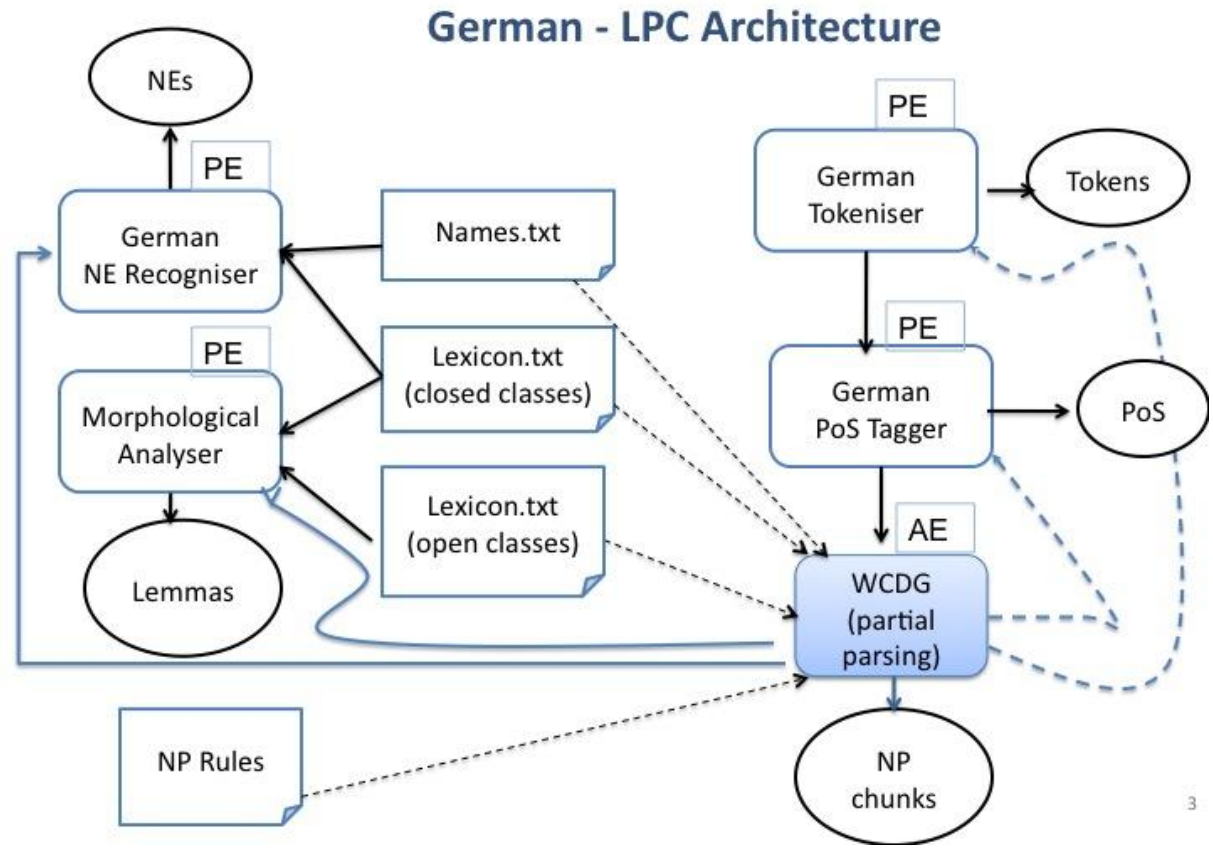


Figure 9. German LPC Architecture

In the following sentence we will describe shortly each component.

Sentence splitter and tokenizer

The sentence splitter and tokenizer is a Perl programme based on regular expression.
Input: a file in text form.
Output: one token per line in XML Form where sentences and tokens are marked:

```
<sent_bound>
 <tok>An</tok>
 <tok> </tok>
 <tok>die</tok>


 <tok> </tok>
 <tok>UNESCO</tok>
```

```
<tok> </tok>
<tok>und</tok>
<tok> </tok>
<tok>die</tok>
<tok> </tok>
<tok>Unterzeichner-Staaten</tok>
<tok> </tok>
<tok>der</tok>
<tok> </tok>
<tok>Welterbekonvention</tok>
<tok>:</tok>
</sent_bound>
```

Figure 10. Example of the sentence splitter and tokeniser output

The programme is based on a dictionary of Abbreviations specific for German. More empty lines between sentences are ignored.

POS Tagger

The POS Tagger is an open source reimplementation of the TnT Tagger (http://www.coli.uni-sb.de/~thorsten/tnt/). TnT Tagger is based on second order Markov models, thus looking two words into the past. The states of the model represent tags, outputs represent the words, and transition probabilities depend on the states which consist of pairs of tags in this case. The TnT tagger is a trigram tagger where the probability of a tag depends on the previous two tags. The German Model is trained on the NEGRA Corpus (http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/negra-corpus.html). It uses the Stuttgart-Tübingen Tagset (STTS). In order to se the list of the used tags please refer to (http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/stts.asc)

The input of the POS tagger is one token per line.

The output is one token followed by one or more pairs of (POS probability) separated by white spaces.

Input:

Der Affe isst das Brot.

Output:

Der ART 1

Affe NN 1

```
frisst VVFIN 0.756 ADV 0.116 VVPP 0.041 VAFIN 0.038 ADJD 0.036
   VMFI7.380e-03 PTKNEG 5.329e-03
gern ADV 0.967 ADJD 0.033
Brot NN 1
. $. 1
```

Figure 11. Example of the POS tagger input and output

Lemmatiser

As mentioned the WCDG system is based on a full form lexicon consisting of more parts (files) as follows:

- Nomen.txt: Nouns declared explicitly
  - approx. 27500 Entries
  - each Noun is classified in one of 38 declination classes (see below)
  - additionally following features are specified:
    - sg: singulare tantum, (no plural forms possible)
    - pl: plurale tantum (no singular forms possible)
    - sto: nouns declaring materials
  - The entries are ordered alphabetically and grouped like Nouns with counting meaning, nouns which define objects etc.
- Adjectives.txt (5800 entries), similarly constructed,
- Verbes.txt (8670 entries).
- Lexicon.cdg containing all closed word classed as well as patterns for digits (in Roman and Arabic dates).

```
noun Ehrgeiz m1 obj:ic sg
noun Eid m1 obj:ic
noun Eifersucht f17 obj:c sg sto
noun Billiarde f16 sort:number
noun Billion f17 sort:number

noun Augenblick m1 sort:timeunit
noun Beginn m1 sort:timepoint sg
noun Kilogramm n20/n32 sort:measure
noun Kilohertz n20/n32 sort:measure
```

```
noun Absatzplus n32 sg
noun Absatzprognose f16
noun Absaugen n23 sg deverbal:yes
```

Figure 12. Example of the noun lexicon file

Based on this files dedicated PERL programmes take as input a word an produce its morphological analysis.

Input:

```
warf
```

Output:

```
warf_VVFIN_third := warf : [
 base:werfen, cat:VVFIN, avz:allowed, person:third,
 number:sg, tense:past, mood:indicative, perfect:haben,
 valence:a, extravalence:dscni
];

warf_VVFIN_first := warf : [
 base:werfen, cat:VVFIN, avz:allowed, person:first,
 number:sg, tense:past, mood:indicative, perfect:haben,
 valence:a, extravalence:dscni
];

warf_FM := warf : [
 pattern:FM, cat:FM, case:bot, number:bot, gender:bot,
 person:third
];

warf_ADJD := warf : [
 cat:ADJD, pattern:ADJD, degree:positive
];

warf_NE := warf : [
 cat:NE, pattern:NE, person:third, number:bot, gender:bot,
 case:bot
];
```

Figure 13. Sample output of the morphological analyser

One can specify also as input the word and its POS. In this case the search space is restricted and only the entries corresponding to the given POS is selected. This morphological analysis is used in order to select the features for the Lemmatizer of the German LPC.

NP-Chunker

As mentioned before the WCDG is a dependency parser so no constituent structures are produced. However we can use the results produced by the WCDG in order to produce NP Constituents., by defining rules of combining POS belonging to the same sub-dependency tree. Our NP-Chunker is based on a set of about 10 rules.

NE Recogniser

The German LPC use two NE recognisers. The first one is based on the File Names.txt included in the WCDG system, containing 30100 first and family names, explicitly declared. The second one is the open source MunPex (Multilingual NER) http://www.semanticsoftware.info/munpex, variant for German. This is a component developed for GATE-Framework and adapted to the UIMA Framework.

German LPC based on GATE components

GATE pipeline has also been integrated and ported to UIMA with the following specific German components:

1. Tree Tagger (http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/) performing PoS Tagging using the STTS tagset (http://www.ims.uni-stuttgart.de/projekte/corplex/TagSets/stts-table.html).
2. NE Recogniser integrating JAPE regular expressions.
3. MuNPEx NP Chunker (Multi-Lingual Noun Phrase Extractor, http://www.semanticsoftware.info/munpex) – fast, robust, customizable and well-tested JAPE implementation of noun phrase (NP) chunker for GATE, currently supporting English, German, and French. MuNPEx requires a part-of-speech (POS) tagger to work and can additionally use detected named entities (NEs) to improve chunking performance.
4. Drum Lemmatizer (http://www.rene-witte.net/german-lemmatization) – a self-learning lemmatizer capable of automatically creating a full-form lexicon by processing German documents; uses a German lexicon (http://www.semanticsoftware.info/system/files/delexicon.txt) with about 84320 entries.

   The lemmatization system has two main components, an algorithm and a lexicon. The algorithm lemmatizes German nouns depending on morphological classes. The lexicon is generated from the nouns that have been processed by this algorithm, with some additional capabilities for self-correction.

   The Drum Lemmatization system is based on following GATE components and resources:

   - The Case Tagger, adding case information to nouns,

- The POS-based Number Tagger, add number information,
- The MuNPEx noun phrase chunker for German.

1.5 Language processing tools for Greek

Greek Tokenizer and Sentence Splitter

The tokenizer is one of the core pre-processing modules of the Greek natural language processing chain. It may run either in training or in run time mode. In both cases the tool first applies tokenization to the input text or file, treating any non-alphanumeric character as a separate token. Furthermore, the tool supports the recognition of Greek and Latin characters, which are split in the pre-processing phase. Finally, the tokenizer uses an svm-based sentence splitter to locate punctuation symbols in order to mark the end of sentences. The built-in classifier can detect whether punctuation denotes an abbreviation or the end of a sentence.

Overall, the tokenizer recognizes tokens, abbreviations, non-Greek (Latin) tokens and sentence boundaries and composes the first part of the Greek LPC chain. The tool is part of the open source tools provided by the AUEB Natural Language Processing Group (http://nlp.cs.aueb.gr/software.html) and specifically of the Greek Part-of-Speech tagger (version 1).

Details:

- Required language resources: only the classifier file in a proprietary format that allows the classification of punctuations for sentence boundaries detection.
- Users: for linguistic research and as a component of a Natural Language Processing chain.
- Programming language: Java, platform independent.
- Access conditions: open source under GNU GPL.
- Copyright: AUEB (original code), ATLANTIS (extensions and improvements for the ATLAS project).
- UIMA integration class: com.atlantis.uima.SentenceSplitter class in Greek Sentence Splitter primitive engine as part of the Greek Aggregate.
- Input format: raw Text in UTF-8.
- Output format:
  - Proprietary XML format (see below for the short description of the proprietary format used)
  - UIMA data types for token & sentence annotation, namely Java package eu.atlas.anno.core.text.Token and eu.atlas.anno.core.text. Sentence correspondingly.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
   elementFormDefault="qualified"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="ARTICLE">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="SENTENCE">
     <xs:complexType>
      <xs:sequence>
       <xs:element maxOccurs="unbounded" name="token"
           type="xs:string" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure 14. Schema of the output of the Greek tokenizer and sentence splitter (standalone version)

Greek POS Tagger

The Greek part-of-speech (POS) tagger uses a k-nearest neighbour classifier in order to automatically determine the part of speech (e.g. noun, adjective, verb, etc.) of each word occurrence in Greek texts. It can be used however to tag each word with additional grammatical information, such as the gender, number, and case of each noun, the voice, number and tense of each verb, etc. The tool is open source and as a standalone application it includes a GUI and active/passive learning functionalities. Although a pre-trained set of resources exist, the POS tagger could be retrained and can be configured to use alternative settings depending on the incoming text content. The original tool (version 1) has been extended and optimized in terms of performance, reliability, bug removal, etc. and in terms of adding multi-core support in sections where this was feasible. Within ATLAS a UIMA wrapper has been developed and integrated to the Greek Language Processing Chain as a primitive engine.

Details:

- Required language resources: For both the standalone and the UIMA-wrapper, a set of resource files are needed:

  - A directory containing the sentence classifier, which is the same one used in the Greek Tokenizer & Sentence Splitter tool described above.
  - A directory containing the training files for the classification of the token to the corresponding part-of-speech.
  - For the standalone only version: the resources for the GUI are needed, which are distributed with the source code of the tool (http://nlp.cs.aueb.gr/software_and_datasets/AUEB_Greek_POS_tagger.tar.gz).
- Programming language: Java, platform independent for the run time and Windows for the (re)training of the POS tagger classifier.
- Access conditions: open source under GNU GPL.
- Copyright: AUEB (original code), ATLANTIS (extensions and improvements for the ATLAS project).
- UIMA integration class: Java wrapper class in Greek Sentence Splitter primitive engine as part of the Greek Aggregate.
- Input: raw Text in UTF-8.
- Output format:
  - Proprietary XML format (see below for the short description of the actual format used in the standalone version 1)
  - UIMA data types for POS tagging annotation, namely Java packages eu.atlas.anno.core.text.Token and eu.atlas.anno.core.text. Markable.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
   elementFormDefault="qualified"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="folder">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="document">
     <xs:complexType>
      <xs:sequence>
       <xs:element maxOccurs="unbounded" name="sentence">
        <xs:complexType>
         <xs:sequence>
          <xs:element maxOccurs="unbounded" name="token">
           <xs:complexType>
            <xs:simpleContent>
```

```
        <xs:extension base="xs:string">
         <xs:attribute name="POS"
          type="xs:string" use="required" />
         <xs:attribute name="function"
          type="xs:string" use="optional" />
         <xs:attribute name="case"
          type="xs:string" use="optional" />
         <xs:attribute name="gender"
          type="xs:string" use="optional" />
         <xs:attribute name="number"
          type="xs:string" use="optional" />


         <xs:attribute name="voice"
          type="xs:string" use="optional" />
         <xs:attribute name="tense"
          type="xs:string" use="optional" />
         <xs:attribute name="mode"
          type="xs:string" use="optional" />
        </xs:extension>
       </xs:simpleContent>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
 <xs:attribute name="file" type="xs:string"
     use="required" />
</xs:complexType>
</xs:element>
</xs:schema>
```

Figure 15. Schema of the output of the Greek POS tagger (standalone version)

Part-of-speech tagset	Additional Attributes
adjective	numbers cases genders
adverb	
article	numbers cases genders
conjunction	
determiner	
interjection	
noun	numbers cases genders
numeral	
particle	
preposition	
pronoun	
verb	
punctuation	

Other Attributes	
Numbers	sin plu
Cases	nom gen acc voc dat inf
Genders	m f n
Persons	first second third
Moods	indicative subjunctive optative imperative infinitive
Voices	active passive
Tenses	present imperfect past future perfect pastperfect futureperfect
Extras	abbreviation acronym foreign_word symbol other

Table 1: List of attributes and POS tags in the Greek Tagset (Remark: the same tagset has been utilized in the Greek Noun Phrase Extractor described below)

Greek Lemmatizer

The Greek lemmatizer has been developed as a separate primitive engine of the Greek Aggregate LPC chain and is executed after performing tokenization, sentence extraction and POS tagging in the input text. This module is a morphological lemmatizer for the Greek language, thus for a given word/token, the tool forms the exact corresponding lemma taking into account the grammatical information assigned to the token.

Details:

- Required language resources: inflectional dictionary of the Greek language in SQLite and MySQL format. SQLite is enabled by default with UTF-8 collation.
- Application areas: Natural Language Processing Chains (e.g. summarization tools, language pre-processing, etc.)
- Programming language: Java.
- OS compatibility: Windows, Unix.
- Access conditions: binary freely available, source code only for the ATLAS partners.
- Copyright: ATLANTIS.
- UIMA integration class: com.atlantis.uima.Lemmatizer class in Greek Lemmatizer primitive engine as part of the Greek Aggregate.
- Input: a token annotated with POS tag (e.g. noun) in UTF-8 encoding.
- Output format:
  - text of the lemma.
  - As lemma text in UIMA data type for token (lemma property), defined in Java package eu.atlas.anno.core.text.Token.
- Status: stable

Greek Noun Phrase Extractor

The noun phrase extractor is based on the Spejd tool (Java version v0.84), an open source shallow parsing and disambiguation engine (http://zil.ipipan.waw.pl/Spejd) that was adapted to support the Greek grammar and the corresponding set of rules for identifying Noun-Phrases in a text. The tool has been extended for the Greek language within ATLAS project and can be further enhanced in order to include additional shallow parsing annotations given an input text (e.g. verb phrases, etc.)

Details:

- Required language resources: a set of resources for the Greek language, summarized below:
  - A file with the tagset specification of the Greek Language based on the format needed by the Spejd tool.

  - A file defining the rules for extracting noun phrases from the input file (XML, UTF-8). The input XML file is based on the XCES standard (http://www.xces.org/) and is one of the input formats supported by the Spejd tool.
- Programming language: Java.
- OS compatibility: Windows, Unix.
- Access conditions: GNU GPL license (version 2)
- Copyright: IPI PAN (Spejd tool), ATLANTIS (extension for the Greek language).
- UIMA integration class: com.atlantis.uima.GreekNPExtractor class in Greek Noun Phrase primitive engine as part of the Greek Aggregate.
- Input: XCES XML file in UTF-8 encoding
- Output format:
  - the same XML format as input, with annotation for the noun phrases found in the input file (standalone version)
  - In the UIMA framework a Java wrapper class is used to annotate the input to the appropriate UIMA data type, namely eu.atlas.anno.core.text.NP.
- Status: stable

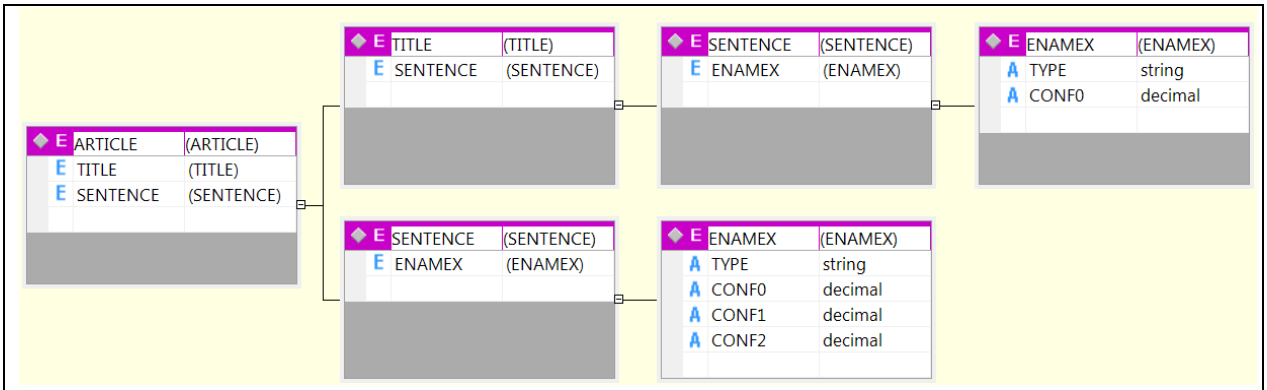
Greek Named Entity Extractor

The Greek Named Entity Extractor is partially based on the open source library "Named-entity recognizer for Greek texts" (version 2) developed by the AUEB group (http://nlp.cs.aueb.gr/software.html). The original tool supports extraction of temporal expressions, person names, organization names using semi-automatically produced regular expression patterns for the temporal expressions and an ensemble of Support Vector Machines (SVMs) for person and organization names. The extended version developed within ATLAS identifies additional location names based on the same algorithms used for the person and organization names and regular expression patterns for money, percentage extraction from an input text. The named entity extractor utilizes internally a sentence splitter (if needed), which is the same described above (Greek Tokenizer & Sentence Splitter). The software of the named-entity recognizer is released under the GNU GPL, and it requires LIBSVM, which is available from: http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

Details:

- Required language resources: A set of files which are summarized in the following:
  - The trained model files for the person, organization and location names.
  - A file containing the temporal expressions for the Greek language.
- Programming language: Java, platform independent.
- External libraries and tools: Uses libsvm (open source) for the training and the run-time.
- OS compatibility: Windows and Linux.
- Access conditions: GNU GPL2.

- Copyright: AUEB (original NE), ATLANTIS (extended version within ATLAS project).
- UIMA integration class: com.atlantis.uima.GreekNERecognizer class in Greek Named Entity primitive engine as part of the Greek Aggregate.
- Input: raw text in UTF-8 encoding.
- Output format:
  - A file in UTF-8 containing identified Named Entities using MUC-7 format (standalone version)
  - UIMA data type for NE annotation, namely eu.atlas.anno.core.text. NamedEntity




```
<?xml version="1.0" encoding="windows-1253"?>
 <xs:schema attributeFormDefault="unqualified"
    elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ARTICLE">
   <xs:complexType>
    <xs:sequence>
     <xs:element maxOccurs="unbounded" name="TITLE">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="SENTENCE">
         <xs:complexType mixed="true">
          <xs:sequence minOccurs="0">
           <xs:element name="ENAMEX">
            <xs:complexType>
             <xs:simpleContent>
              <xs:extension base="xs:string">
               <xs:attribute name="TYPE"
                    type="xs:string"
                    use="required" />
```

```
          <xs:attribute name="CONF0"
              type="xs:decimal"
              use="required" />
         </xs:extension>
        </xs:simpleContent>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element maxOccurs="unbounded" name="SENTENCE">
 <xs:complexType mixed="true">
  <xs:sequence minOccurs="0">
   <xs:element maxOccurs="unbounded" name="ENAMEX">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:string">
       <xs:attribute name="TYPE" type="xs:string"
           use="required" />

       <xs:attribute name="CONF0" type="xs:decimal"
           use="required" />
       <xs:attribute name="CONF1" type="xs:decimal"
           use="optional" />
       <xs:attribute name="CONF2" type="xs:decimal"
           use="optional" />
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Figure 16. Schema of the output of the Greek NE recognizer (standalone version)

Greek Anaphora Resolver

The anaphora resolver is based on the Robust Anaphora Resolution Engine (RARE)6 tool provided by UAIC (partner of the ATLAS project). The RARE tool is a framework allowing the development and testing of anaphoric references that occur in a text. Its engine can output co-referential chains for the first mentioning of characters/object in a text and is language independent given that a set of resources are adapted for the target language. In that context, the Greek anaphora resolver engine utilizes the core engine of RARE with resources created for the Greek language. The resources can be categorized as follows:

- Pre-processor resources, which contain files to further pre-process the extracted annotated text by the primitive engines of the Greek Aggregate. These are:
  - Person Names (male and female),
  - Family Names,
  - Copulative verbs,
  - and language-dependent determiners.

The format for each tool is provided within the RARE library documentation.

- Core engine resources
  - An XML file that defines all the pronoun forms for the singular and plural at first, second and third person. The format of the file is defined within the documentation of the RARE tool.

Details:

- Required language resources: A set of files are needed for the RARE engine as described above. Further details are provided in the section where the RARE engine is described in detail.
- Programming language: Java, platform independent.
- OS compatibility: Windows and Linux.
- Access conditions: open source.
- Copyright: UAIC (RARE Engine), ATLANTIS (extension to support the Greek Language in the context of the ATLAS project).
- UIMA integration class: com.atlantis.uima.GreekAREngine class in Greek Anaphora Resolver primitive engine as part of the Greek Aggregate.
- Input: XML text in UTF-8 encoding after passing through the Greek Aggregate engine analysis.
- Output format:
  - An XML file in UTF-8 containing the co-referent chains found in the input file (standalone version)

[6] [Cristea, Postolache, 2002a]

  - UIMA data type for Markable annotation, namely eu.atlas.anno.core.text. Markable
- Status: a first prototype is available. The Greek Anaphora resolver will be finalized in the context of WP5, where the summarization engine of the ATLAS project will be delivered.

Short description of the tagsets used in the Greek Anaphora Resolver input files:

- Numeral:
  - "SG"
  - "PL"
  - "SG,PL"
- POS:
  - nouns: "N"
  - pronouns: "PRON"
  - relative pronouns: "PRONREL"
  - adjective: "A"
  - adverb: "ADV"
  - conjunction: "CC"
  - prepositions: "PREP"
  - punctuation: "PUNCT"
  - verbs: "V"
  - verb, past participle: "EN"
  - determiners, articles: "DET"
  - verbs in the -ing form: "ING"

Greek Stopword Recognizer

The Greek Stopword Recognizer includes a list of words that can be ignored in special situations such as in finding a category of an input article, in summarization process, etc..

Details:

- Programming language: Java.
- OS compatibility: Windows, Unix.
- Access conditions: binary freely available, source code only for the ATLAS partners
- Copyright: ATLANTIS
- UIMA integration class: Java wrapper class, namely com.atlantis.core. GreekStopWords
- Input: -
- Output format: An array of strings containing stop words for the Greek Language.

Greek Stemmer

The Greek Stemmer takes as input a word and removes its inflexional suffix according to a rule based algorithm. The algorithm follows the known Porter algorithm for the English language and it is developed according to the grammatical rules of the Modern Greek language (see http://www.salix.gr/downloads/GreekStemmer.class.zip and http://gelaligo.org/stemmer/stemmer.php.txt).

Details:

- Required language resources: -.
- Programming language: Java
- Access conditions: binary freely available, source code only for the ATLAS partners.
- Copyright: ATLANTIS.
- UIMA integration class: not applicable
- Input: a token with optional POS tag annotation.
- Output format: the stemmed word as string token.

1.6 Language processing tools for Polish

Sentence splitter, tokenizer, lemmatizer, morphological analyser, tagger

The linguistic processing on segmentation, lemmatization and tagging levels is executed by Pantera tagger tightly integrated with Morfeusz SGJP morphological analyser.

Pantera (http://code.google.com/p/pantera-tagger/) is a recently developed morphosyntactic rule-based Brill tagger of Polish. It uses an optimized version of Brill's algorithm adapted for specifics of inflectional languages. The tagging is performed in two steps, with a smaller set of morphosyntactic categories disambiguated in the first run (part of speech, case, person) and the remaining ones in the second run. Due to the free word order nature of Polish, the original set of rule templates as proposed by Brill has been extended to cover larger contexts.

Morfeusz SGJP (http://sgjp.pl/morfeusz/index.html.en) is a morphological analyser for Polish, also used for sentence- and token-level segmentation and lemmatisation of texts before the morphological part is applied. It uses positional tags starting with POS information followed by values of morphosyntactic categories corresponding to the given part of speech. Current version of the tool is based on linguistic data coming from The Grammatical Dictionary of Polish by Zygmunt Saloni.

Pantera is available on GNU GPL v3 license and Morfeusz SGJP on FreeBSD license.

NP extractor and MWU lemmatizer

NP extraction is performed by Spejd – an Open Source Shallow Parsing and Disambiguation Engine (http://zil.ipipan.waw.pl/Spejd/) based on a fully uniform formalism both for constituency partial parsing and for morphosyntactic disambiguation — the same grammar rule may contain structure-building operations, as well as morphosyntactic correction and disambiguation operations. The formalism and the engine are more flexible than either the usual shallow parsing formalisms, which assume disambiguated input, or the usual unification-based formalisms, which couple disambiguation (via unification) with structure building. Current applications of Spejd include rule-based disambiguation, detection of multiword expressions, valence acquisition, and sentiment analysis. The functionality can be further extended by adding external lexical resources.

Spejd processing is powered by a grammar of Polish developed by Katarzyna Głowińska for the task of syntactic group recognition in the National Corpus of Polish (http://www.nkjp.pl/). MWU lemmatizer is a set of extensions to this grammar developed by Łukasz Degórski especially for ATLAS.

Spejd is available on GNU GPL license.

NE recognizer

Named entity recognition is performed by Nerf (http://clip.ipipan.waw.pl/Nerf), a statistical CRF-based named entity recognizer trained over 1-million manually annotated subcorpus of the National Corpus of Polish and successfully used in the process of automated annotation of its total 1 billion segments.

Nerf annotation model was recreated to maintain consistency with general ATLAS annotation framework, defined to cover dates, money, percentage and time expressions, names of organizations, locations and persons. Normalized versions of entities are provided to facilitate extraction and comparisons (e.g. for dates and time: values conforming to xsd:date and xsd:time types, for money: value with ISO currency code).

1.7 Language processing tools for Romanian

Sentence splitter, tokenizer, lemmatizer, POS tagger

The Romanian POS Tagger automatically performs 4 tasks, sentence splitting, tokenizing, POS tagging and lemmatizing.

Details:

- Required language resources: language model for the POS tagger, a language dictionary, a set or rules, used in the redactor rules system
- Programming language: Java
- Access condition: open source
- Copyright: UAIC
- Input format: raw text in UTF-8
- Output format: proprietary XML format, memory-based in Java

NP extractor

The Romanian NP Chunker, uses GGS (Graphical Grammar Studio http://sourceforge.net/projects/ggs/), a visual tool for describing grammars. A Romanian grammar has been developed which allows fully recursive NP chunks.

Details:

- Required language resources: language grammar
- Programming language: Java
- Access condition: open source
- Copyright: UAIC
- Input format: proprietary XML format annotated at tokens, POS tags, lemma levels
- Output format: proprietary XML format, memory-based in Java

NE recognizer

The Romanian NE recognizer uses the GATE plugin ANNIE which also has the Romanian language implemented.

We are also using the JRC-Names.

Details:

- Required language resources: GATE dependency files (GATE must be installed on the working machine)
- Programming language: Java, Jape
- Access condition: open source
- Copyright: GATE
- Input format: raw text in UTF-8
- Output format: memory-based in Java

Anaphora resolver

The Romanian Anaphora resolver is called RARE (Robust Anaphora Resolution Engine).

Details:

- Required language resources: lists of names, a list of stopwords, a pronounConstraint.xml file, a list of copulative verbs, a list of determiners.
- Programming language: Java
- Access condition: GNU
- Copyright: UAIC
- Input format: proprietary XML format annotated at tokens, POS tags, lemma, NP chunks, and other levels
- Output format: proprietary XML format, memory-based in Java

1.8 Common post-processing tools

The last component in each language processing chain is the “Performance report” provider. The annotated text is enriched with a performance reports containing the overall processing time (in milliseconds) for the whole document, as well as processing time for each primitive engine (annotator in the chain). Each performance report is then stored in a database so that the LPC can be evaluated in terms of productivity.

2 INTEGRATION OF INDIVIDUAL TOOLS

2.1 Bulgarian tools

Bulgarian language processing chain is implemented in a such a way that to be easily integrated in different NLP systems. All individual tools are designed and implemented to be as platform independent as possible. The tools can be supported virtually by all major 32 and 64 bits platforms such as MacOS, Linux, Windows. The list of the required third party libraries for each tool is listed below.

Tool name	Required libraries
BG sentence splitter	libboost_regex, libpthread
BG tokenizer	libboost_regex, libpthread, libz
BG POS tagger	libboost_system, libpthread
BG lemmatizer	none
BG NP extractor	See ParseEst lr_engine.
BG NE recognizer	See ParseEst lr_engine.
BG stopword recogniser	libz
Lexicon compiler	none
ParseEst lr_builder	libboost_regex, libz, libpthread, libxml2
ParseEst lr_engine	libboost_regex, libz, libpthread

Command line arguments for each tool are as follows:

Tool name	Command-line arguments
BG sentence splitter	ssplitter <rules1> <lexicon> <alphabet> <rules2> <plain|mark> <input> [log_file_name] – rules1 – name of the text file containing the first set of rules – lexicon – name of the file containing the lexicon finite state automata – alphabet – name of the file containing the alphabet definition of the lexicon's automata – plain – sentence borders will be marked with <S> – mark – sentence borders will be marked as with sentence position and length – input – name of the input file – log_file – (optional) file name of the generated log file. The result is on the standard output.

Tool name	Command-line arguments
BG tokenizer	tokenizer <tokrules> <toktypes> <plain|mark|split> <input> [log_file_name] – tokrules – name of the file containing the finite state transducer (rules for token splitting) – toktypes – name of the file containing the finite state transducer (for token type definitions) – plain – returns tokens and token type annotation – mark – returns tokens, token type annotation and token positions – split – returns tokens only – input – name of the input file – log_file – (optional) file name of the generated log file The result is on the standard output.
BG POS tagger	svmtagger <model> <input> – model – name of the trained tagger model – input – name of the input file The result is on the standard output. When the tagger is run in server mode an additional parameter '-port' should be specified, i.e: svmtagger <model> -port <port> – model – name of the trained tagger model – port – TCP port for a communication
BG lemmatizer	lemmatizer <alphabet> <reverse_tag_set> <input> [log_file_name] – alphabet – name of the file containing the grammatical dictionary automata alphabet – reverse_tag_set – name of the file containing the tagset transformation – input – name of the input file – log_file – (optional) file name of the generated log file The result is on the standard output.
BG NP extractor	See ParseEst lr_engine.
BG NE recognizer	See ParseEst lr_engine.
BG stopword recogniser	bgstopwords <alphabet> <input> – alphabet – name of the file containing the stop words dictionary automata alphabet – input – name of the input file

Tool name	Command-line arguments
Lexicon compiler	lexcompiler <lex_compound_file> – lex_compound_file – name of the file containing the lexicon definitions
ParseEst lr_builder	lrbuilder <rule_definition> <taglist> <lex_map> <transducer> <meta_def> – rule_definition – name of the XML file with rules definitions – taglist – name of the file containing the list with all possible POS tags – lex_map – name of the file with generated lexicon classes – transducer – name of the file where the transducer is stored – meta_def – name of the file where the meta symbols definitions are stored
ParseEst lr_engine	lrengine <alphabet> <lexicon> <class_def> <meta_def> <transducer> <plain|mark> <input> – alphabet – name of the file containing the alphabet definition of the lexicon's automata – lexicon – name of the file containing the lexicon's automata – class_def – name of the file with generated lexicon classes – meta_def – name of the file with the generated by the lr_builder meta symbols definitions – transducer – name of the file where the transducer is stored – plain – identified tokens are marked by '<<<' and '>>>' symbols – mark – identified tokens are marked by their position and length – input – name of the input file The result is on the standard output.

2.2 Constraint Dependency Grammar

The CDG is primarily developed for Linux and runs on any recent distribution (debian/sarge, SUSE 9.1). To compile and run CDG, the following utilities and libraries apart from GNU development tools are required: gcc, flex, bison, m4, awk, sed, make.

2.3 Greek Sentence Splitter and the Greek Aggregate Config file

The sentence splitter, as well as the POS Tagger and the Named Entity Recognizer, all require libsvm (open source) for the training and the run-time execution. Both windows and Linux operating systems are supported in the running time. For the training procedure, the corresponding svm library shall be compiled and used (either as "dll" library in Windows or "so" library in Linux). The Green Sentence Splitter is part of the Greek Aggregate engine and

the resources that utilizes can be configured using the corresponding config file. This Config file is being used by all primitive engines of the Greek LPC. Its structure is presented below:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
 <!-- Resources root directory for the Greek Language -->
 <resourcesDir>.</resourcesDir>

 <!-- Sentence Splitter & Tokenizer Config -->
 <sentenceSplitterSelectedAlgorithm>
  <value>ALGORITHM_USING_POS_TAGGER</value>
 </sentenceSplitterSelectedAlgorithm>


 <!-- POS Tagger Config -->
 <POSTAGGER_Enable>
  <value>true</value>
 </POSTAGGER_Enable>
 <POSTAGGER_ConfigPath>Greek.SentenceSplitter/resources/POS_Tagger/
 </POSTAGGER_ConfigPath>
 <sentenceSplitterPOSTAGGER_ClassifierName>Greek.SentenceSplitter/
  resources/POS_Tagger/classifier/SENTENCE_CLASSIFIER
 </sentenceSplitterPOSTAGGER_ClassifierName>
 <POSTAGGER_UseThreads>
  <value>false</value>
 </POSTAGGER_UseThreads>
 <POSTAGGER_Capacity>
  <value>50</value>
 </POSTAGGER_Capacity>

 <!-- Lemmatizer Config -->
 <Lemmatizer_Enable>
  <value>true</value>
 </Lemmatizer_Enable>

 <!-- Threading related -->
 <Lemmatizer_UseThreads>
  <value>false</value>
 </Lemmatizer_UseThreads>
 <Lemmatizer_Capacity>
  <value>100</value>
 </Lemmatizer_Capacity>
```

```
<!-- Caching mechanism related -->
<Lemmatizer_Use_Cache>
 <value>true</value>
</Lemmatizer_Use_Cache>

<Lemmatizer_Cache_Configuration>Greek.Lemmatizer/resources/
 Lemmatizer/EhCache_el.xml</Lemmatizer_Cache_Configuration>

<Lemmatizer_Cache_Alias>
 <value>Caching_Lemmatizer</value>
</Lemmatizer_Cache_Alias>

<!-- Storing capabilities -->
<Lemmatizer_DB_Type>
 <value>org.sqlite.JDBC</value>
</Lemmatizer_DB_Type>



<!-- org.sqlite.JDBC, com.mysql.jdbc.Driver -->
<MorphoLexico>Greek.Lemmatizer/resources/Lemmatizer/
 lexiko_el.db</MorphoLexico>

<!-- NPEXTRACTOR Config -->
<NPExtractor_Enable>
 <value>false</value>
</NPExtractor_Enable>
<NPExtractor_IniFile>Greek.NPExtractor/resources/Spejd/
 config_spejd_el.ini</NPExtractor_IniFile>

<!-- NERECOGNIZER Config -->
<NERecognizer_Enable>
 <value>false</value>
</NERecognizer_Enable>
<NERC_ConfigPath>Greek.NERecognizer/resources/Atlantis_NERC/
 files_UTF-8/</NERC_ConfigPath>
<!-- Anaphora Resolver Config -->
<GreekAREngine_Enable>
 <value>false</value>
</GreekAREngine_Enable>
</config>
```

Figure 17. Config file structure

The above configuration file contains paths for the resources and the possibility to enable/disable primitive engines and tools. When the aggregate engine is invoked to accomplish a process, checks to verify if all required primitive engines are enabled; if not, an error will be reported to the corresponding log facility (e.g. console, file, etc.)

2.4 Greek POS Tagger and Named Entity Recognizer

Both tools require the libsvm for the training procedure and the run-time execution. Regarding the POS tagger, it should be noticed that it can be trained either in passive or active mode. It is preferred to use the passive mode, as it can be used to further improve the reliability and accuracy of the POS tagging without having the original training dataset.

Swing-layout-1.0.jar library is required by the AUEB version of the tools (as standalone applications) for their successful compilation.

2.5 Resources for the Integration of the Greek Anaphora Resolver

The Greek Anaphora resolver primitive engine utilizes the RARE core and pre-processor modules in order to annotate the input. The following libraries are needed for the successful compilation of the engine:

- jdom.jar, which is used by pre-processor and the core RARE engine for parsing the input XML file (UTF-8).
- scorer_java.jar, which provides the necessary functionalities to validate the automatic annotation with a given gold file.

The location of the resources for the Greek Anaphora resolver is configured through the Greek Aggregate Config file. These file-based resources are the following:

- pronounConstraints.xml, which is the file that defines all the pronoun forms in the Greek language,
- stopwords.txt file, which contains a list of common stop words for the Greek language including punctuation and related signs,
- copulativeVerbs.txt file, which is part of the pre-processor and define the Greek copulative verbs,
- MaleNames.txt, FemaleNames.txt and FamilyNames.txt files, which are part of the pre-processor RARE library and are used to further enhance the annotation of the tokens and for the purposes of the co-reference chain provided by the RARE engine.

2.6 Greek NP Extractor Integration and Resources

The Green Noun Phrase extraction module has been integrated in a UIMA wrapper as a separate primitive engine and supports the annotation of a word group (simple or synthetic) as a noun phrase. It is a Spejd-based tool requiring the commons-cli-1.0.jar library which provides an API for parsing command line options passed to the Spejd engine.

The resources (location) of the Greek NP extractor are configured through the Greek Aggregate Config file. These file-based resources are the following:

- config_spejd_el.ini file, which contains the necessary configurations needed by the Spejd tool in order to find the Greek NP rules, tagsets and the input format (XML) of the file used. It also contains acronyms for the Greek language as well as a directory for the log files,
- morfologik_el.cfg file, which contains the POS tagset used by the Greek LPC tools and their attributes (e.g. sense, mood, etc),
- rules_el.sr file, which contains the NP rules used for the detection and annotation of noun phrases in the input file.

2.7 JRC Names annotator

JRC Names[7] is a highly multilingual named entity resource for person and organisation names (called 'entities'). It consists of large lists of names and their many spelling variants (up to hundreds for a single person), including across scripts (Latin, Greek, Arabic, Cyrillic, Japanese, Chinese, etc.). The JRC names resources are updated on a daily basis.

The JRC Names annotator is a UIMA wrapper for the JRC names resources. Being a multilingual named entity resource the JRC Names annotator is suitable to be added in each of the LPCs in the project.

3 LPC INTEGRATION TESTS

This section documents all actions related to planning and executing tests of the language tools integrated into ATLAS platform.

3.1 Test scope and methodology

The tests executed in WP4 intended to verify the integration coherence of the tools only, i.e.

- prove that the language processing chains (LPCs) composed of individual language tools and integrated into ATLAS framework provided results consistent with their offline execution outside the framework – i.e. that the framework does not interfere with document formats, encodings, sizes etc. and that the result of their execution provides types of annotations expected from the individual tools,
- measure the execution time of individual components of the chain depending on the size of processed document (in tokens).

Other potentially valuable issues related to testing such as suitability of tools (e.g. evaluation of their properties, tagsets etc.), performance of LPCs other than measuring the basic execution time, scalability of the architecture or user impressions are outside the scope of WP4 and will not be evaluated.

3.2 Implementation plan

Following implementation plan was adopted to assure high quality of provided language processing chains:

1. The test data was gathered by ICS PAS basing on the information from all partners even before the implementation started.

[7] http://langtech.jrc.it/JRC-Names.html

2. The Web application for testing execution of the LPC for input text and uploaded file (WebCASDebugger) was implemented by Tetracom and made available for all partners.
3. Every partner wrapped and unit-tested their tools on ATLAS test servers.
4. Tetracom and ICS PAS were constantly monitoring the deployment status of the LPCs and periodically (or on partner request) tested their integration.
5. Tetracom prepared infrastructure for testing the LPCs and measuring their execution times (with a relational database gathering test results).
6. The official integration test results together with initial performance results were presented at the project meeting in Thessaloniki.
7. Performance optimizations were applied by partners.
8. The final tests were run at the end of the workpackage implementation period and the results were recorded.
9. Reference to final test results were included in D4.1 deliverable – LPC documentation (this document).

3.3 Testing levels

The implementation plan implied execution of tests on multiple levels:

- Unit tests were executed for each tool and the whole chain by each partner to check their proper integration with the test platform. This process was outside the scope of the current document and was carried out independently by partners.
- Intermediate LPC integration tests were executed by Tetracom and ICS PAS using test documents provided by partners and test infrastructure provided by Tetracom.
- Final LPC integration tests were executed by Tetracom and ICS PAS and their results were officially recorded.

3.4 Internal test data

Internal tests were periodically executed by partners using WebCASDebugger environment. In order to facilitate them, the test data representing documents in all project languages with formats and encodings used for particular language and different sizes have been gathered and used in the process of testing.

The description of data used for internal tests is available online at

http://atlasproject.eu/atlas/project/wp4/results#test-data-internal.

3.5 Final test data

Apart from verification of the proper integration of the tools constituting LPCs, the goal of the testing process was to roughly assess their performance and the stability, taking into account their incomparability caused by different annotators, implemented on different platforms

(Java, C++, Perl). Nevertheless, such information is quite important for further optimization of the tools making the chains and also for future integrators of the LPCs in their applications.

In order to provide sound testing environment a parallel corpus of documents based on EurLEX[8] database have been collected. It consists of over 360K documents of various sizes for all project languages. The documents were divided into 9 classes according to number of tokens:

Class	Number of tokens
c0	0 – 1000
c1	1001 – 2000
c2	2001 – 4000
c3	4001 – 8000
c4	8001 – 16000
c5	16001 – 32000
c6	32001 – 64000
c7	64001 – 128000
c8	128001 and more

The description of data used for final tests is available online at

http://atlasproject.eu/atlas/project/wp4/results#test-data-final.

3.6 Final test infrastructure

The test infrastructure intended to provide a complete environment for running and measuring execution time of all individual tests.

The test data has been imported in ATLAS, as part of the EUDocLib service. Each document was represented as a multi-lingual content item identified by its CELEX number. These content items were added to a "selection" so that only the documents from the test corpus could be processed. i-Publisher user interface provided a functionality for sending a group/selection of documents to be process as a batch which made triggering the processing with the above-described settings very straightforward.

Processing of a single document included the following steps:

- MIME type detection, text extraction, language detection,
- processing through an LPC, performance recording,
- storing the results in a data store; storing the performance report.

[8] http://eur-lex.europa.eu/en/index.htm

The raw performance report for each document has been stored in a database table which made the data easy to query, aggregate and chart.

3.7 Final integration and test results

Before running the tests the integration status of all LPCs has been verified. The following language processing components were integrated for the project languages:

LPC component	Language					
	BG	DE	EL	EN	PL	RO
Sentence splitter	✔	✔	✔	✔	✔	✔
Tokenizer	✔	✔	✔	✔	✔	✔
Lemmatizer	✔	✔	✔	✔	✔	✔
POS tagger	✔	✔	✔	✔	✔	✔
NP extractor	✔	✔	✔	✔	✔	✔
NE recognizer	✔	✔	✔	✔	✔	✔

The final integration tests have been executed on 28 December 2011 by Tetracom. The detailed results are available online at

http://atlasproject.eu/atlas/project/wp4/results#test-results