Heterogeneous Natural Language Processing Tools via Language Processing Chains

Diman Karagiozov
Tetracom IS Ltd.
diman@tetracom.com

Abstract

One of the most recent developments in NLP is the emergence of linguistic annotation meta-systems which make use of existing processing tools and implement pipelined architecture. In this paper we describe a system that offers a new perspective in exploiting NLP meta-systems by providing a common processing framework. This framework supports most of common NLP tasks by chaining tools that are able to communicate on the basis of common formats. As a demonstration of the effectiveness of the system to manage heterogeneous NLP tools, we developed an English processing chain, pipelining OpenNLP-based and C++ NLP implementations. Furthermore, we conducted experiments to test the stability and measure the performance of the English processing chain. A baseline processing chain for the Bulgarian language illustrates the capabilities of the system to support and manage processing chains for more languages.

1 Introduction

Increasingly complex digital content needs to be retrieved, stored and aggregated for future access. In addition, it should be organized, annotated and structured. However, it is difficult to manage the information flow because of its volume, rapidly evolving structure and its multilinguality.

The usage and integration of natural language processing and understanding tools (NLP and NLU) is vital for processing digital content. The different input and output formats, supported operating systems and programming languages determine the existence of the wide range of NLP tools. Furthermore, the choice of available tools makes their integration in content management systems, analytical tools and in-house systems very difficult.

One of the latest developments in NLP is the emergence of linguistic annotation meta-systems which make use of existing processing tools and implement pipelined processing architecture (Cristea and Pistol, 2008). This paper describes a system that exploits NLP meta-systems and provides a common processing framework capable to host a variety of tools for different natural languages that are able to communicate on the basis of common formats. Furthermore, our system provides a well-defined integration API, so that 3rd party software components can use the NLP services provided by the system.

The paper is organized as follows: Section 2 overviews related work, Section 3 describes system architecture, Section 4 presents the language processing chains method, Section 5 discusses implementation, evaluation and results, Section 6 describes the scope of LPC for Bulgarian and Section 7 sketches further work and conclusion.

The work reported in sections 3 (NLP System Architecture), 4 (Language Processing Chain) and 5 (UIMA Implementation of LPC) was designed, developed, evaluated and analyzed by the author of this paper.

2 Related Work

Several standardization approaches have been made towards the interoperability of the NLP tools (XCES[1], TEI[2], GOLD[3]). None of the proposed standards have been universally accepted, leading to the development of resources and tools according to the format of each research project.

[1] http://www.xml-ces.org/
[2] http://www.tei-c.org/index.xml
[3] http://www.linguistics-ontology.org/gold.html

More notably, two systems that facilitate the access and usage of existing processing tools have emerged. GATE (Cunningham et al., 2002) is an environment for building and deploying NLP software and resources that allows integration of a large amount of built-ins in new processing pipelines.

UIMA (Unstructured Information Management Application) (Ferrucci and Lally, 2004) offers the same general functionalities as GATE but once a processing module is integrated in UIMA it can be used in any further chains without any modifications (GATE requires wrappers to be written to allow two or more modules to be connected in a chain). Currently, UIMA is the only industry OASIS standard [4] (Ferrucci et al., 2006) for content analytics.

3 NLP System Architecture

The processing of unstructured text in system is split into three independent subtasks, executed sequentially.

Pre-processing – at this stage the text is extracted from the input source (documents in OpenOffice, PDF, MS Office, HTML, ePub, FB2 and other formats). Details of the implementation of the pre-processing engine are not in the scope of this article.

Processing – at this stage the text is annotated by several NLP tools, chained in a sequence. We call the implementation of the processing engine for a specific language a ‘Language Processing Chain’ (LPC).

Post-processing – at this stage the annotations are stored in a data store, such as file system, relational or NoSQL database. Details of the implementation of the post-processing engine are not in the scope of this article.

The overall performance of an NLP task depends on the performance of the atomic NLP tools, used in the processing engine and the size of the input text. As the classical request-response chain cannot be used for such tasks because the response time cannot be predicted, we use an asynchronous, message-based, communication channel between the components in the system.

A pre-processing engine detects the mime type of the input source, extracts the text from it, detects the language of the text and sends it to a language-specific queue.

[4] http://www.oasis-open.org/committees/uima/

One (of the several) language processing chains checks-out a message, processes it and sends the annotated text to an output queue where a post-processing engine stores the text annotations in a data store.

“Figure 1” depicts the top-level architecture of the NLP components in the system.

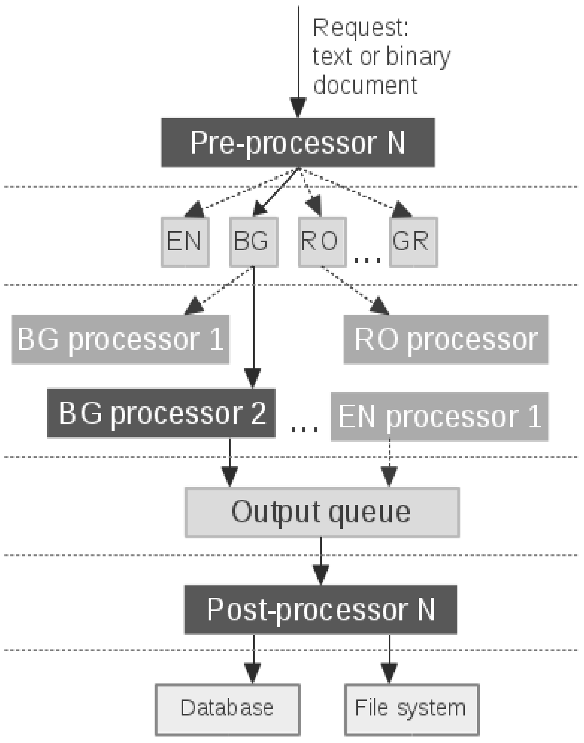

Figure 1. Top-level architecture.

4 Language Processing Chain

In order to achieve a basic set of low-level text annotations the following atomic NLP tools have to be executed in sequence (Cristea and Pistol, 2008): *Paragraph splitter* (splits the raw text in paragraphs) → *Sentence splitter* (splits each paragraph in sentences) → *Tokenizer* (splits each sentence into tokens) → *POS tagger* (marks up each token with its particular part of speech tag) → *Lemmatizer* (determines the basic form of each token) → *Word sense disambiguation* (disambiguates the meaning of each token and assigns a sense to it) → *NounPhrase Extractor* (marks up the noun phrases in each sentense) → *NamedEntity Extractor* (marks up named entities in the text).

“Figure 2” overviews the components and the sequence of execution of the atomic NLP tools, which are part of a LPC.

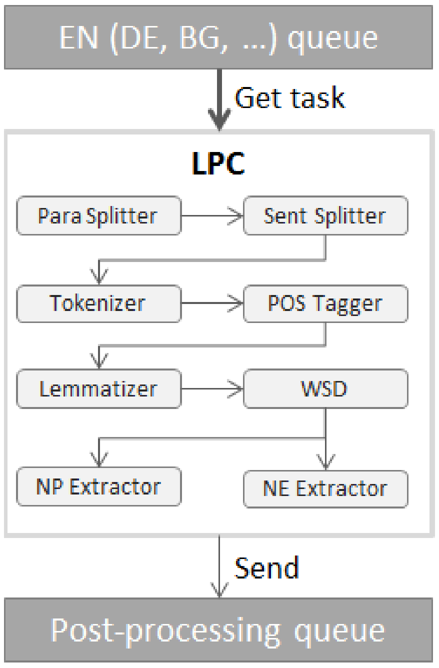

Figure 2. Components of a language processing chain.

The key requirements to our system are the possibility to use heterogeneous NLP tools for different languages, transparent horizontal scalability, and transparent hot-swap of linguistic components. Last but not least is the requirement of a minimal installation footprint.

After evaluating both GATE and UIMA meta-systems, in respect to the above requirements, we based the implementation of the processing engine on the UIMA framework (JAVA version). We wrapped the UIMA base framework with an OSGi shell (OSGi Alliance, 2009), making it available to the rest of the components in the system. The horizontal scalability of the NLP functionalities and the transparent hot-swap of the linguistic components are empowered by a network-distributed architecture based on ActiveMQ[5].

5 UIMA Implementation of LPC

A typical UIMA application consists of: *a Type System Descriptor*, describing the annotations that will be provided by the components of the application; one or more *Primitive Analysis Engines*, each one providing a wrapper for a NLP tools and adding annotations to the text; *an Aggregate Engine*, defining the execution sequence of the primitive engines (Gordon et al., 2011).

[5] http://activemq.apache.org/

5.1 Type System Descriptor

In order to put the atomic NLP tools in a chain, they need to be interoperable on various levels. The first interoperable level, the compatibility of formats of linguistic information, is supported by a defined scope of required annotations, described as a UIMA Type System Descriptor.

The uniform representation model, required by the UIMA type system, provides normalized heterogeneous annotations of the component NLP tools. Within our system, it covers properties that are critical for the further processing of annotated data, e.g. lemma, values for attributes such as gender, number and case for tokens necessary to run coreference module to be subsequently used for text summarization, automatic categorization and machine translation.

In order to facilitate the introduction of further levels and types of annotation, a general *markable* type has been introduced, carrying subtype and reference to another *markable* object. In this way we can test and later include new annotation concepts into the core annotation model.

“Table 1” enlists the annotations which are available in the Type System Descriptor of the system. The parameters of each annotation type, listed in “Parameters” column, extend the standard UIMA annotation set of parameters (begin offset, end offset and covered text).

Annotation type	**Parameters**
Paragraph	–
Sentence	–
Token	POS; MSD (lemma, gender; number, case); Word sense
Noun Phrase	Head, Lemma
Named Entity	Type (one of: date, location, money, organization, percentage, person, time); Normalized value
Markable	Type; Reference

Table 1: Summary of the text annotations and their parameters

5.2 UIMA LPC Components

We have built a reference LPC for English in order to illustrate the integration of English NLP tools into a processing chain.

Tool type	Based on
Paragraph Splitter	Regular expressions
Sentence Splitter	OpenNLP[6]
Tokenizer	OpenNLP
Lemmarizer	RASP[7]
POS tagger	OpenNLP
Word sense disambiguation	C++ LESK (Banerjee, 2002)[8]
NP extractor	Rules engine
NE recognizer	OpenNLP

Table 2: Tools, wrapped into UIMA primitive engines, contained in the English LPC.

We have successfully pipelined JAVA-based NLP tools and external C++ tools into a single LPC. A challenge, solved during the integration process, was the different sets of POS tags used by the OpenNLP and RASP tools. We created a rule-based converter between the Penn Treebank and CLAWS tagsets in order to achieve the interoperability of the tools.

5.3 Evaluation

Furthermore, we extended the standard UIMA functionalities to measure the performance of the whole LPC and each individual primitive engine. We based the current evaluations on the processing of a corpus of 27’085 EU law documents from EUR-Lex[9]. “Table 3” gives an overview of the contents of the processed corpus.

	Number of tokens (N)	Docs	Avg tkns[10]
C1	$N \in [1,1000)$	8’900	520
C2	$N \in [1000,2500)$	4’863	1’600
C3	$N \in [2500,7500)$	7’589	4’600
C4	$N \in [7500,12500)$	2’485	9’600
C5	$N \in [1250,25000)$	2’082	17’300
C6	$N \in [25000,50000)$	834	34’800
C7	$N \geq 50000$	332	82’600

Table 3: Distribution by number of tokens of documents in the processed corpus.

“Figure 3” depicts the average processing time (in milliseconds) of documents belonging to each of the above categories (C1-C7). The performance of the English LPC is linearly related to the number of tokens in the processed documents.

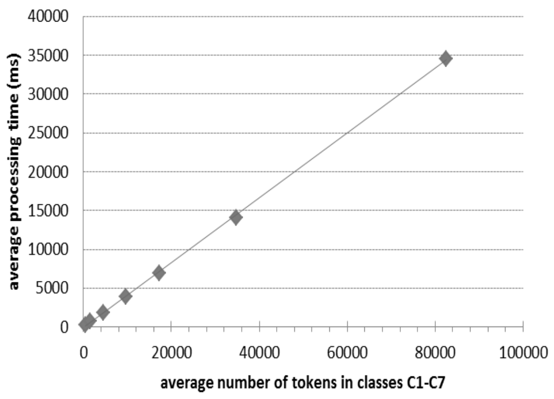

Figure 3. Average processing time of a document compared to the average number of tokens in documents in categories C1 to C7.

“Figure 4” shows the average processing time (in milliseconds) for each UIMA primitive engine (PE) for documents in categories C3 and C4. The performance of each PE is also linearly related to the number of tokens in the processed documents. The UIMA overhead time, caused by the CAS flow controller, is less than 1% of the total execution time and thus it is not represented at the “Figure 4”.

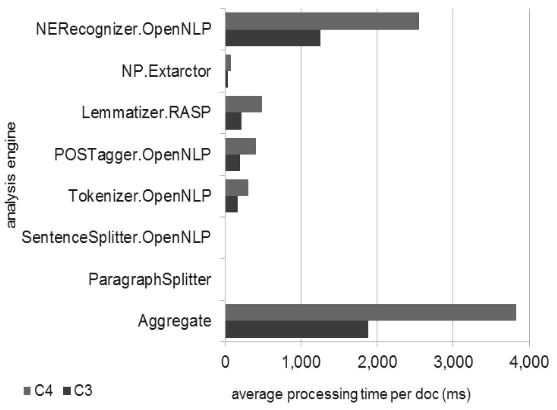

Figure 4. Average processing time of the primitive engines in the English LPC.

The results show that the Named Entitiy (NE) Recognizer (NERecognizer.OpenNLP) is a bottleneck in the English LPC mainly because the recognitions of the 7 different NE types (date, location, money, organization, percentage, person, and time) are executed sequentially.

[6] http://incubator.apache.org/opennlp/
[7] http://www.informatics.sussex.ac.uk/research/groups/nlp/rasp/
[8] We managed to achieve 30 time better performance of the C++ version compared to the initial Perl LESK tool.
[9] http://eur-lex.europa.eu/
[10] Average number of tokens in a document in a class

Possible solution to this problem is to run the recognition process in parallel for all 7 NE types. Another approach that will be evaluated in the process of further development of the English LPC is to replace the OpenNLP statistical NE recognizer with a solution, using language specific rules and lexicons.

6 Bulgarian LPC

We developed a Bulgarian language processing chain in order to demonstrate the ability of the system architecture to support more languages. The UIMA primitive engine wrappers, within the Bulgarian LPC, are the same as in the English one. The baseline NLP tools are developed by the Department of Computational Linguistics[11] at the Bulgarian Academy of Sciences. The Bulgarian NLP tools, integrated in our system, are based on the theory of finite-state language processing (Komani, 1999). The tools are implemented in C++ and are external for the JAVA-based UIMA environment.

The evaluation of the Bulgarian LPC was based on the processing of 200 Bulgarian fiction books, resulting in an average number of 100’000 tokens per document. The data, however, cannot be compared with the English LPC in terms of performance (average processing time per document) because of the different platforms, available tools and implementation approaches. The evaluation only demonstrates the capabilities of our system to support and manage LPCs for different languages.

7 Conclusion and Further Work

The described architecture of a language processing chain and its implementation in our system goes towards the direction of standardized multilingual online processing of language resources. The framework can be extended by integration of new types of tools and new languages and thus providing wider online coverage of linguistic services in a standardized manner.

A future extension of our system is the implementation of processing chains for other languages. The final version of German, Greek, Polish and Romanian LPCs will be available by the end of 2011.

The core LPC annotation set will be extended to support annotation of coreference chains by anaphora resolution tools and the results will be effectively used to improve text summarization and recognition process of named entities.

Last but not least, the LPC framework will be made available to a wider range of platforms and programming languages such as PHP and .Net via API implementation. Furthermore, we will provide a LPC engine web service in order to enable the integration with 3rd party systems in other languages, such as Python, Ruby, and Perl.

The source code of the pre-processing, processing and post-processing engines, as well as the core annotation schema, will be released as open-source under the GPL3 license as soon as it becomes mature enough for the open-source community.

Acknowledgements

The work reported in this paper laid the foundation of the “Applied Technology for Language-Aided CMS” project co-funded by the European Commission under the Information and Communications Technologies (ICT) Policy Support Programme (Grant Agreement No 250467). We would like to thank to consortium members for their guidance and valuable feedback for refinement, extensions and standardization of the NLP architecture of system.

References

A. Kornai, 1999. *Extended Finite State Models of Language.* Cambridge University Press. ISBN 0-521-63198-X.

Banerjee, Satanjeev. 2002. *Adapting the Lesk algorithm for word sense disambiguation to WordNet.* University of Minnesota, Duluth. Master's thesis.

H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan. 2002: *GATE: A framework and graphical development environment for robust NLP tools and applications.* In Proceedings of the 40th Anniversary Meeting of the ACL (ACL’02). Philadelphia, US.

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

D. Ferrucci, A. Lally, D. Gruhl, E. Epstein, M. Schor, J.W. Murdock, A. Frenkiel, E. Browm, T. Hampp. 2006, *Towards an Interoperability Standard*

[11] http://dcl.bas.bg/en/home_en.html

for text and Multi-Modal Analytics, IBM Research Report, RC24122 (W0611-188).

Cristea D., Pistol, I. 2008. *Managing Language Resources and Tools using a Hierarchy of Annotation Schemas.* In Proceedings of Workshop 'Sustainability of Language Resources and Tools for Natural Language Processing', organized in conjunction with LREC 2008

OSGi Alliance., *OSGi Service Platform, 2009 Core Specification, Release 4, Version 4.2.*

I. Gordon, A. Wynne, Y. Liu. 2011, *Engineering High Performance Service-Oriented Pipeline Applications with MeDICi,* ICSOC 2010 Workshops, LNCS 6568, pp. 88-99.