LIMA: A Multilingual Framework for Linguistic Analysis and Linguistic Resources Development and Evaluation

Romaric Besançon, Gaël de Chalendar, Olivier Ferret, Faiza Gara,
Olivier Mesnard, Nasredine Semmar

CEA, LIST, Vision and Content Engineering Laboratory,
Fontenay-aux-Roses, F-92265, France;
`firstname.lastname@cea.fr`

Abstract

The increasing amount of available textual information makes necessary the use of Natural Language Processing (NLP) tools. These tools have to be used on large collections of documents in different languages. But NLP is a complex task that relies on many processes and resources. As a consequence, NLP tools must be both configurable and efficient: specific software architectures must be designed for this purpose. We present in this paper the LIMA multilingual analysis platform, developed at CEA LIST. This configurable platform has been designed to develop NLP based industrial applications while keeping enough flexibility to integrate various processes and resources. This design makes LIMA a linguistic analyzer that can handle languages as different as French, English, German, Arabic or Chinese. Beyond its architecture principles and its capabilities as a linguistic analyzer, LIMA also offers a set of tools dedicated to the test and the evaluation of linguistic modules and to the production and the management of new linguistic resources.

1 Context and objectives

In this article, we present the LIMA (CEA List Multilingual Analyzer) platform which is, as GATE (Cunningham et al., 2002), together an architecture, a set of tools and resources and an environment for developing applications based on Natural Language Processing (NLP). This platform was developed by the LVIC laboratory of CEA LIST with the following requirements:

- multilingualism, with the objective of dealing with a broad spectrum of languages;
- a large diversity of applications. LIMA aims at being used as a basic component for various applications that can be text-based applications such as automatic summarization or question-answering but can also be applications dealing with multimedia documents;
- extensibility, that is to say the ability to support the addition of new functionalities. As illustrated by Section 2.2, the current version of LIMA mainly performs analyses up to syntactic analysis but we also aim at extending it to semantic and discourse analyses;
- the need for efficiency. A platform such as LIMA must be able to process very large corpora both because the processing of such corpora is more and more required by work in Computational Linguistics (see (Pantel et al., 2009) for instance) and because it also has to be used in an industrial context.

The first three requirements make necessary to design an architecture based on modularity and flexibility at a high degree. All languages are not characterized by the same set of linguistic phenomena and as a consequence, their processing doesn't rely on the combination of the same elementary analyses. Moreover, even if a linguistic analysis module can be used for two different languages, the linguistic resources it relies on are generally specific to each language. The same need for modularity and flexibility comes from the diversity of applications LIMA has to deal with: using the same system for lemmatizing a set of keywords from a base of images, a newspaper article or the transcription of a phone conversation is not the best means to have good results in each of these three contexts. Finally, the main difficulty LIMA had to face was to fulfill these requirements without sacrificing efficiency.

Several kinds of architectures were already proposed to address these different issues. Process-oriented architectures focus on the combination and the control of a set of modules together with the communication between them. They generally implement a loosely integration by the means of a "glue" that can be a multi-agent system as in TalLab (Wolinski et al., 1998) or a client-server architecture as in FreeLing (Carreras et al., 2004). Data-oriented architectures also represent a weak type of integration by concentrating on the normalization of data between modules, as in the MULTEXT project (Ide and Véronis, 1994) or the LT XML Library (Brew et al., 1999). The TIPSTER-like architectures (Grishman, 1997) go a step further by imposing both a shared representation of data, often by annotation graphs (Bird and Liberman, 1999), and a uniform interface for modules. The GATE platform (Cunningham et al., 2002) is the typical representative of this kind of architectures but TEXTRACT (Neff et al., 2004) and its most recent descendant, UIMA (Ferrucci and Lally, 2004), also belong to this category. Finally, the highest degree of integration is reached by formalism-oriented architectures in which both data and processes are represented through a declarative formalism associated to a kind of inference engine. This approach was initially dedicated to the development of grammars as in ALVEY (Grover et al., 1993) but was also applied more widely through platforms such as ALEP (Simkins, 1994), NooJ (Koeva et al., 2007) or Outilex (Blanc et al., 2006).

As it will be illustrated in the following sections, we chose a TIPSTER-like architecture for LIMA as it represents the best trade-off between modularity and efficiency, which are the main requirements LIMA must fulfill. However, we particularly emphasized the design of the framework for supporting processes to enable the use of LIMA with various kinds of interfaces.

2 Presentation of LIMA

2.1 Principles

In order to have a flexible architecture, allowing to handle multiple languages, and thus take into account various linguistic phenomena in the analysis, the choice was made to use the principle of a configurable processing chain (or pipeline) applying in cascade several treatments corresponding to different stages or different levels of linguistic analysis. The various treatments use specific linguistic resources and share and modify a common data structure. Moreover, this processing chain is conceived to integrate LIMA in applications based not only on a direct integration but also in a client-server environment, either CORBA or SOAP based.

Figure 1 gives an illustration of the flexibility of LIMA through its configuration. One master file points at a configuration file for each language. Each language configuration file defines a set of pipelines; in our example, one dedicated to the processing of open-domain documents and one dedicated to medical documents. A pipeline is described as a list of processing units and a processing unit can refer to a set of resources. In the example of Figure 1, each of two pipelines contains a rule-based named entity tagger but the two corresponding processing units, while relying on the same type of tagger, are different as they refer to two different resources: the rules for recognizing medical named entities such as diseases or drugs are different from the rules for recognizing persons or locations. Each resource has also its own configuration data, which are limited in our example to the file in which the compiled rules for the general or medical named entity tagger are stored. Pipelines can also share components: in Figure 1, the named entity logger, a component that dumps named entities in a XML format, is shared by the general and the medical pipelines as it is not specific to a domain.

The main design pattern used for implementing this flexibility is the Factory pattern. All parts of the system have factories that allow to select them at runtime based on configuration files stating the names of the modules to use. Not only processing units are based on factories, but also the resources associated to them. So, a simple change in a configuration file allows to replace a component by another, different kind of tokenizers for example, or to make a given pipeline unit to use a different resource, e.g. replacing a named entities rules file by another one specific to the corpus processed. In the same manner, a process unit can be inactivated or a completely different chain can be selected at runtime.

Linguistic resources are dynamically initialized: the initialization of each process unit triggers the loading of the resources needed by this treatment. So, only the resources that are useful for the given analysis session will be initialized. Moreover, the resources are handled by an independent manager, which allows to share resources between different processing units and thus avoid to load a same resource several times.

A collection of data structures is transmitted all along the processing chain. The main data structure is an analysis graph (built around the `boost::graph` C++ library) whose nodes are tokens associated to several annotations, such as tokenization status (e.g. capitalization), or morphosyntactic categories. There is two kind of edges: sequential links between adjacent tokens and syntactic links bearing syntactic dependency information between the tokens. One should also note that contrary to annotation graphs used in other systems, our graphs are lattices, allowing to keep some kind of segmentation ambiguities before a dedicated process unit can remove them. Along this main graph comes another graph, a generic annotation graph whose nodes and edges can be associated to the nodes and edges of the analysis graph and that can handle any data structure thanks to the `boost::any` C++ library.

A specific unit (called "dumper") is dedicated to the output of the analysis, given the content of the analysis data structures. According to the applications, different levels of details or different formats are to be considered: each output format needed can be implemented by a corresponding dumper and specified at runtime.

2.2 Examples of LIMA configurations

In this section, we illustrate the configurability of LIMA with the detailed description of two different pipelines, for French and Arabic standard textual documents.

2.2.1 LIMA configuration for French analysis

The French pipeline uses the following treatments covering segmentation, morphological analysis and morphosyntactic analysis:

- *tokenization*: performed using a character-based automaton with a context of up to three characters on both side. The tokenization is absolute, meaning that it does not consider tokenization ambiguity at this step. But later steps will be able to group some tokens or split other ones;

- *dictionary check*: each word is checked against a full-form dictionary (a dictionary including all the known forms of single words of the language obtained from a list of lemmas and inflection rules), and possible lemmas and part-of-speech (POS) are associated with it;

- *hyphenated words*: this unit performs a special treatment to associate lemmas and categories of hyphenated words not present in the dictionary (parts of the split word are looked up in the dictionary);

- *idiomatic expressions*: the full-form dictionary contains only single words, compound expressions are recognized with this specific unit (which allows to reduce the ambiguity before POS tagging by considering the expression as a whole for the rest of the analysis). This step uses a generic pattern recognition unit based on finite state automata.

- *unknown words*: the unknown words are given default POS using a guesser based on typographical clues;

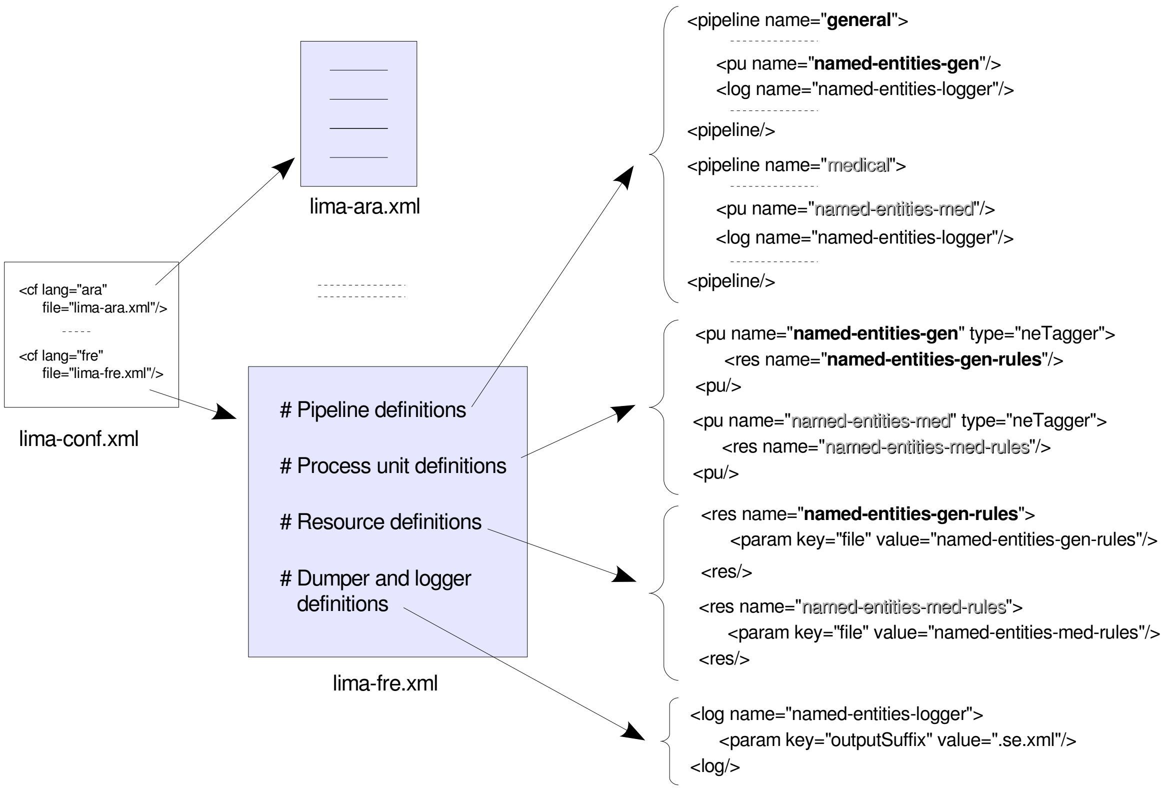

lima-ara.xml
<cf lang="ara" file="lima-ara.xml"/>
<cf lang="fre" file="lima-fre.xml"/>
lima-conf.xml
# Pipeline definitions
# Process unit definitions
# Resource definitions
# Dumper and logger definitions
lima-fre.xml
<pipeline name="general">
<pu name="named-entities-gen"/>
<log name="named-entities-logger"/>
<pipeline/>
<pipeline name="medical">
<pu name="named-entities-med"/>
<log name="named-entities-logger"/>
<pipeline/>
<pu name="named-entities-gen" type="neTagger">
<res name="named-entities-gen-rules"/>
<pu/>
<pu name="named-entities-med" type="neTagger">
<res name="named-entities-med-rules"/>
<pu/>
<res name="named-entities-gen-rules">
<param key="file" value="named-entities-gen-rules"/>
<res/>
<res name="named-entities-med-rules">
<param key="file" value="named-entities-med-rules"/>
<res/>
<log name="named-entities-logger">
<param key="outputSuffix" value=".se.xml"/>
<log/>

Figure 1: Configuration of LIMA

- *specific entity recognition*: the same pattern recognition unit is used with different rules to recognize numbers, dates and named entities (the treatment is performed before POS-tagging because it reduces the ambiguity);

- *POS-tagging*: default POS-tagging implemented in LIMA uses a Viterbi algorithm on POS trigrams, with a fallback to bigrams, unigrams and lemma a priori probabilities;

- *syntactic analysis*: it is performed using a dependency grammar implemented as a set of simple rules executed by the generic pattern recognition unit (Besançon and Chalendar (de), 2005). This step is itself splitted in sub-steps, each able to handle specific kinds of relations, such as local or distant ones.

2.2.2 Adding a specific process unit for Arabic analysis

The Arabic pipeline has a similar skeleton to the French pipeline, but without the special treatment for hyphenated words and with and additional unit dedicated to some specificities of Arabic, namely the fact that Arabic texts are often completely unvowelled or partially vowelled and a large number of proclitics and enclitics can be attached respectively to the beginning or the end of a word (Gregory Grefenstette, 2005). The following treatment is then added:

- *clitic stemmer*: this unit uses two additional dictionaries for clitic (containing 77 proclitics and 65 enclitics) in order to split Arabic agglutinated words into proclitics, simple forms and enclitics. It proceeds as follows:
  - Several vowel form normalizations are performed: the vowel symbols اٍ اِ اٌ اُ اً اَ are removed, the characters آ إ أ are replaced by the character ا and the final characters ي ئ ؤ or ة are replaced by the characters ى يء وء or ·ه;
  - All clitic possibilities are computed by using proclitics and enclitics dictionaries;
  - A radical, obtained by removing these clitics, is checked against the full form lexicon. If it does not exist in the full form lexicon, re-write rules are applied, and the altered form is checked against the full form dictionary. For example, consider the token "بكرته" (with its ball) and the included clitics ب (with) and ه (its), the computed radical كرت does not exist in the full form lexicon but after applying one of the re-write rules, the modified radical "كرة" (ball) is found in the dictionary and the input token is segmented into root and clitics as: ه + كرة + ب = بكرته (with + its + ball);
  - The compatibility of the grammatical tags of the three components (proclitic, radical, enclitic) is then checked. Only valid segmentations are kept and added into the graph of words.

2.2.3 Reusing and combining process units

The clitic stemmer had been developed for Arabic but treats a general linguistic phenomenon that is useful for other languages: for instance, the same processing is used in Spanish, in which the verb forms can have pronominal enclitics. For example, the imperative form "*dame*" (gives me) is a form composed of agglutinated radical "*da*" and the enclitic "*me*". The Spanish pipeline if hence configured to use the clitic stemmer with an enclitic dictionary.

Similarly, the German pipeline has two additional specific units dedicated to treat compound words. German has the ability to form compound words by combining all possible words together: nouns, adjectives, adverbs, participles, verbs, prefixes, prepositions. Specific elements can be used for coupling the words: -e ; -es ; -s ; -n ; -en. These additional units are:

- before POS-tagging, a processing unit seeks if a word is an agglutination of several lexical units, possibly separated by a delimiter, and assigns a grammatical tag to the compound word. Lexical units are checked against full form lexicon and delimiters are specified in another resource. All grammatical tags of the last unit are assigned to the compound word.
- after POS-tagging, another processing unit is used to separate compound word components, to be treated separately in the rest of the analysis.

For example, the word "*Vorlesungsbetrieb*" contain "'*Vorlesung*" + ("*s*") + "*betrieb*". "*Vorlesung*" and "*betrieb*" are found in the full form lexicon and 's' is a delimiter. "*betrieb*" can be VERB or NOUN. The grammatical tags of the last unit (VERB or NOUN) are assigned to the compound word. Separating word components is done after POS-tagging in order to decrease lexical ambiguity.

When dealing with new languages, the different process units developed to treat specific linguistic phenomenons can be combined. For instance, Hungarian is a highly agglutinative language: for example, "*hazamban*" is the equivalent of the English prepositional phrase "*in my house*". Hungarian has also the ability to compose words like German. For example, "*anyanyelv*"="*anya*" ("*mother*")+"*nyelv*" ("*language*") is a composed word and means "*native language*". In order to develop a pipeline for Hungarian text processing, we integrated the clitic stemmer developed for Arabic and the decompounder developed for German. The only additional work needed is the development of specific resources for these pipeline units for the Hungarian language.

2.3 Performances

2.3.1 Part-of-speech tagging

We present in Table 1 an evaluation of the performances of the Part-Of-Speech tagging of LIMA on different languages. These evaluation measures have been obtained using a ten-fold cross validation on the reference tagged corpus that is used as training for the default n-gram model used by LIMA for its POS tagging module. We use in the LIMA POS-tagging a rich tagset with many features, which makes the POS-tagging a more complex task and makes necessary larger corpus for training (which we do not have for all languages). We present the results for the fine-grain tagset, and the results on the main part-of-speech categories, limited to 12 tags.

lang	corpus size	POS-tagging precision	
	(words)	fine-grain	12 tags
eng	30086	91.4% (136 tags)	93.5%
fre	28828	87.7% (168 tags)	94.4%
ara	13257	80.2% (85 tags)	86.1%
spa	30088	80.6% (114 tags)	84.1%
ger	20155	63.5% (423 tags)	89.1%

Table 1: Evaluation of LIMA POS-tagging

The performance of LIMA POS tagger is a bit below state-of-the-art taggers for English language (around 96 or 97%), but has reasonable performances on the other languages. The modular architecture of LIMA allows to easily change the model of a particular module, and we are currently testing other paradigms of POS-taggers for English (SVM and MaxEnt models, which achieve the best performance in the litterature).

2.3.2 Syntactic analysis

LIMA has been evaluated during two evaluation campaigns of French syntactic analyzers, Easy in 2005 and Passage in 2009. In these two campaigns, analyzers were evaluated on non recursive minimal chunks and on relations between these chunks, while LIMA produces a pure dependency analysis. Thus, we had to design a converter able to produce Easy/Passage formats from our dependencies. Table 2 shows the F-measures on groups and relations for LIMA and the best system for each campaign. Note that the Passage results are still unofficial and very preliminary as they concern only a small subset of the evaluated material, namely 989 words out of the final 430,000 final reference.

Campaign	LIMA	Best system
Easy Groups	.82	.89
Passage Groups	.85	.94
Easy Relations	.50	.59
Passage Relations	.60	.68

Table 2: Evaluation of LIMA during Easy and Passage (very preliminary results) campaigns ; F-measure on groups and relations

We can see that although LIMA is still not at the level of the best system, it has progressed between the two campaigns, particularly on relations. The progress was obtained thanks to a work on resources eased by the development environment described in section 3.1.

2.3.3 Speed of processing

Since the LIMA framework has been developed with the goals of both configurability and efficiency to treat large amounts of data, the time required to process texts is an important criterion of LIMA evaluation. We present in Table 3 the evaluation of speed processing, in number of words per second (CPU time), for different languages, and for two levels of processing: one for the part-of-speech tagging only, and one for the complete syntactic analysis and concept extraction. These measures have been obtained on a standard workstation (Intel Core2 Duo 3Ghz, 4Go RAM).

Language	POS tagging	Syntactic analysis
eng	12066 w/s	4332 w/s
fre	8646 w/s	6580 w/s
spa	23282 w/s	11819 w/s
ger	4854 w/s	4394 w/s
ara	436 w/s	146 w/s

Table 3: Evaluation of speed of LIMA processing (in number of words per second)

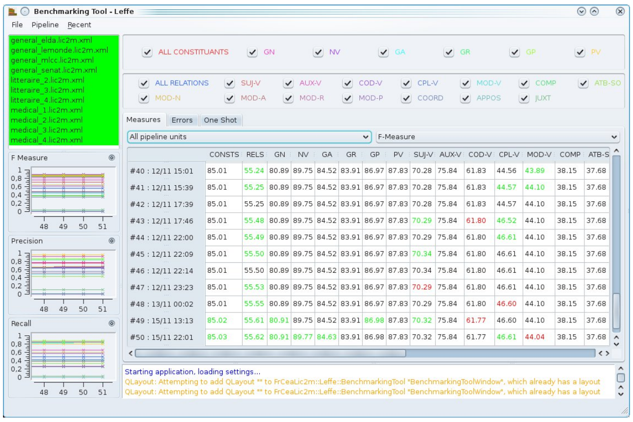
These results are quite good, compared to other NLP processors[1]. However, the figures show that speed performances are different across languages, which can be explained by the differences of treatments and the complexity of the resources used (morphological analysis, dictionary sizes, grammar sizes). For instance, in German and even more in Arabic, the morphological analysis is costly due to clitic treatments and complex words decompouding.

3 Developing with LIMA

This section describes the three steps involved in making LIMA a useful tool for Natural Language Processing tasks. First of all, one has to be able to develop resources used by the analyzer (section 3.1). Then, LIMA must communicate with the application using it. Section 3.2 shows the various ways in which this communication can be handled. Finally, section 3.3 lists a few applications in which LIMA has been integrated.

3.1 LIMA Integrated linguistic analysis development environment

As other real-size linguistic analyzers, LIMA is a large-scale software, whose development involved many man-years of work, in terms of both coding and resource development. Given, for example for French, a dictionary of 110k lemmas, a few hundred syntactic analysis rules, 20k ngrams matrices and other resources, it is difficult to predict what will be the impact of adding a new possible category to a given verb or a new syntactic rule. Any modification may imply, besides what was expected, unforeseeable side-effects and the complexity of the system makes it difficult to guess the overall impact of even small changes. Thus, in association with the analyzer, we need tools that allow to easily edit the resources and to iteratively evaluate the quality of the analysis using a reference corpus. We have developed two such tools, a benchmarking tool and a resources editor (de Chalendar and Nouvel, 2009). Both have been developed using the Qt 4.5 GUI library.
The benchmarking tool (Figure 2) allows to run an evaluation against a reference corpus, to see immediately the performance, globally and for each of a set of different evaluation dimensions. Its specificity is its genericity. It is not limited to the evaluation of LIMA but can be configured to use a different analyzer, a different evaluation program and a different set of evaluation dimensions. Besides the evaluation results themselves, the tool allows to compare an evaluation run with the previous one or with the reference by displaying the analysis graph for both using the Syanot tool[2], a graphical view of the syntactic analysis results (Figure 3). Note that Syanot is also an editor allowing to graphically annotate texts.

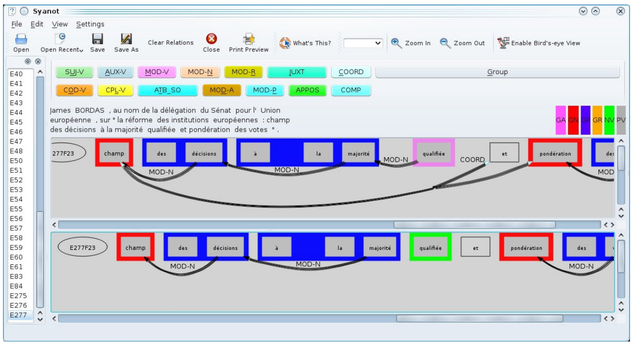


general_elda.lic2m.xml
general_lemonde.lic2m.xml
general_mlcc.lic2m.xml
general_senat.lic2m.xml
litteraire_2.lic2m.xml
litteraire_3.lic2m.xml
litteraire_4.lic2m.xml
medical_1.lic2m.xml
medical_2.lic2m.xml
medical_3.lic2m.xml
medical_4.lic2m.xml

	CONSTS	RELS	GN	NV	GA	GR	GP	PV	SUJ-V	AUX-V	COD-V	CPL-V	MOD-V	COMP	ATB-S
#40 : 12/11 15:01	85.01	55.24	80.89	89.75	84.52	83.91	86.97	87.83	70.28	75.84	61.83	44.56	43.89	38.15	37.68
#41 : 12/11 15:39	85.01	55.25	80.89	89.75	84.52	83.91	86.97	87.83	70.28	75.84	61.83	44.57	44.10	38.15	37.68
#42 : 12/11 17:39	85.01	55.25	80.89	89.75	84.52	83.91	86.97	87.83	70.28	75.84	61.83	44.57	44.10	38.15	37.68
#43 : 12/11 17:46	85.01	55.48	80.89	89.75	84.52	83.91	86.97	87.83	70.29	75.84	61.80	46.52	44.10	38.15	37.68
#44 : 12/11 22:00	85.01	55.49	80.89	89.75	84.52	83.91	86.97	87.83	70.29	75.84	61.80	46.61	44.10	38.15	37.68
#45 : 12/11 22:09	85.01	55.50	80.89	89.75	84.52	83.91	86.97	87.83	70.34	75.84	61.80	46.61	44.10	38.15	37.68
#46 : 12/11 22:14	85.01	55.50	80.89	89.75	84.52	83.91	86.97	87.83	70.34	75.84	61.80	46.61	44.10	38.15	37.68
#47 : 12/11 23:23	85.01	55.53	80.89	89.75	84.52	83.91	86.97	87.83	70.29	75.84	61.80	46.61	44.10	38.15	37.68
#48 : 13/11 00:02	85.01	55.55	80.89	89.75	84.52	83.91	86.97	87.83	70.29	75.84	61.80	46.60	44.10	38.15	37.68
#49 : 15/11 13:13	85.02	55.61	80.91	89.75	84.52	83.91	86.98	87.83	70.32	75.84	61.77	46.60	44.10	38.15	37.68
#50 : 15/11 22:01	85.03	55.62	80.91	89.77	84.63	83.91	86.98	87.83	70.32	75.84	61.77	46.61	44.04	38.15	37.68

```
Starting application, loading settings...
QLayout: Attempting to add QLayout "" to FrCeaLic2m::Leffe::BenchmarkingTool "BenchmarkingToolWindow", which already has a layout
QLayout: Attempting to add QLayout "" to FrCeaLic2m::Leffe::BenchmarkingTool "BenchmarkingToolWindow", which already has a layout
```

Figure 2: Benchmarking tool showing the measures panel on all evaluated files and with all groups and relations selected



James BORDAS , au nom de la délégation du Sénat pour l' Union européenne , sur " la réforme des institutions européennes : champ des décisions à la majorité qualifiée et pondération des votes " .

Figure 3: Syanot used to compare changes between two runs in the Benchmarking Tool

The benchmarking tool allows also to run the analyzer on a given sentence ("One Shot" tab) in order to observe its behaviour in handling a specific linguistic phenomenon. The results are shown in Syanot but also as raw XML showing the full dump of the data structures and in KgraphViewer (Figure 2) to show the internal representation of the analysis graph. KgraphViewer is available for all in the KDE extragear module[3].
The resource development tool is currently tied to LIMA but we plan to make it more generic. It offers a clean and easy to use interface to the dictionary, the part-of-speech tagging learning corpus and the syntactic analysis grammar rules. It has been designed to be easily usable by non computer specialists. Figure 5 shows the tab allowing to dig into the part-of-speech tagging learning corpus. It is centered around the (filterable) list of trigrams. When selecting a trigram, the user sees below the occurrences of this

[1] It would for instance be in second place behind the TreeTagger in Matthew Wilkens' evaluation of POS-taggers, on a similar computer: http://workproduct.wordpress.com/2008/11/08/evaluating-pos-taggers-speed/

[2] Syanot is a graphical syntactic annotations editor developed under an open source licence. It will soon be publicly available.

[3] http://extragear.kde.org/apps/kgraphviewer/

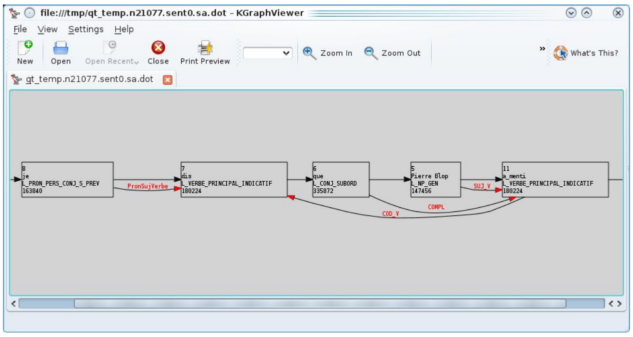
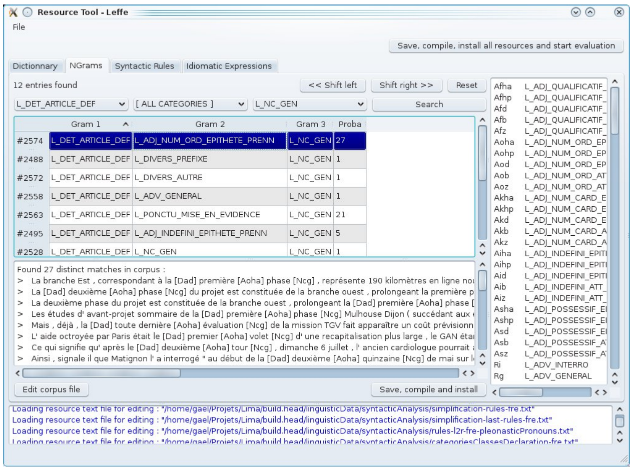


Figure 4: KGraphViewer

trigram in the corpus. The right part shows a correspondence table between two tag sets used respectively during the compilation of the matrices and during the analysis by LIMA.

Three other editors are available in other tabs: one allowing to edit the dictionary that is used to produce the final full-form dictionary; a second one allowing to edit the syntactic analysis rules and a third one to edit the rules recognizing idioms. Each interface permits to save, compile and deploy the respective resources.



12 entries found

	Gram 1	Gram 2	Gram 3	Proba
#2574	L_DET_ARTICLE_DEF	L_ADJ_NUM_ORD_EPITHETE_PRENN	L_NC_GEN	27
#2488	L_DET_ARTICLE_DEF	L_DIVERS_PREFIXE	L_NC_GEN	1
#2572	L_DET_ARTICLE_DEF	L_DIVERS_AUTRE	L_NC_GEN	1
#2558	L_DET_ARTICLE_DEF	L_ADV_GENERAL	L_NC_GEN	1
#2563	L_DET_ARTICLE_DEF	L_PONCTU_MISE_EN_EVIDENCE	L_NC_GEN	21
#2495	L_DET_ARTICLE_DEF	L_ADJ_INDEFINI_EPITHETE_PRENN	L_NC_GEN	5
#2528	L_DET_ARTICLE_DEF	L_NC_GEN	L_NC_GEN	1

```
Found 27 distinct matches in corpus :
>   La branche Est , correspondant à la [Dad] première [Aoha] phase [Ncg] , représente 190 kilomètres en ligne nou
>   La [Dad] deuxième [Aoha] phase [Ncg] du projet est constituée de la branche ouest , prolongeant la première p
>   La deuxième phase du projet est constituée de la branche ouest , prolongeant la [Dad] première [Aoha] phase [
>   Les études d' avant-projet sommaire de la [Dad] première [Aoha] phase [Ncg] Mulhouse Dijon ( succédant aux e
>   Mais , déjà , la [Dad] toute dernière [Aoha] évaluation [Ncg] de la mission TGV fait apparaître un coût prévisionn
>   L' aide octroyée par Paris était le [Dad] premier [Aoha] volet [Ncg] d' une recapitalisation plus large , le GAN étai
>   Ce qui signifie qu' après le [Dad] deuxième [Aoha] tour [Ncg] , dimanche 6 juillet , l' ancien cardiologue pourrait a
>   Ainsi , signale il que Matignon l' a interrogé " au début de la [Dad] deuxième [Aoha] quinzaine [Ncg] de mai sur l
```

Afha	L_ADJ_QUALIFICATIF_
Afhp	L_ADJ_QUALIFICATIF_
Afd	L_ADJ_QUALIFICATIF_
Afb	L_ADJ_QUALIFICATIF_
Afz	L_ADJ_QUALIFICATIF_
Aoha	L_ADJ_NUM_ORD_EP
Aohp	L_ADJ_NUM_ORD_EP
Aod	L_ADJ_NUM_ORD_EP
Aob	L_ADJ_NUM_ORD_AT
Aoz	L_ADJ_NUM_ORD_AT
Akha	L_ADJ_NUM_CARD_E
Akhp	L_ADJ_NUM_CARD_E
Akd	L_ADJ_NUM_CARD_E
Akb	L_ADJ_NUM_CARD_A
Akz	L_ADJ_NUM_CARD_A
Aiha	L_ADJ_INDEFINI_EPITI
Aihp	L_ADJ_INDEFINI_EPITI
Aid	L_ADJ_INDEFINI_EPITI
Aib	L_ADJ_INDEFINI_ATT_
Aiz	L_ADJ_INDEFINI_ATT_
Asha	L_ADJ_POSSESSIF_EI
Ashp	L_ADJ_POSSESSIF_EI
Asd	L_ADJ_POSSESSIF_EI
Asb	L_ADJ_POSSESSIF_AT
Asz	L_ADJ_POSSESSIF_AT
Ri	L_ADV_INTERRO
Rg	L_ADV_GENERAL

```
Loading resource text file for editing : "/home/gael/Projets/Lima/build.head/linguisticData/syntacticAnalysis/simplification-rules-fre.txt"
Loading resource text file for editing : "/home/gael/Projets/Lima/build.head/linguisticData/syntacticAnalysis/simplification-last-rules-fre.txt"
Loading resource text file for editing : "/home/gael/Projets/Lima/build.head/linguisticData/syntacticAnalysis/rules-l2r-fre-pleonasticPronouns.txt"
Loading resource text file for editing : "/home/gael/Projets/Lima/build.head/linguisticData/syntacticAnalysis/categoriesClassesDeclaration-fre.txt"
```

Figure 5: Resource tool

3.2 Interfaces of LIMA

As stated above, LIMA has been conceived to be very modular. It has also been conceived to be modularly integrated in various architectures and to be easily portable to other platforms, even if its primary development platform is GNU/Linux. Its factory based API allows for a direct integration but it can also be used in a UNIX-style script by writing to files using various dumpers dedicated to specific tasks, such as the Easy/Passage dumper writing in the syntactic analysis Passage format (de la Clergerie et al., 2008). Three kinds of client/server integration modules also exist:

- simple socket: a Perl TCP server waits for commands using an ad-hoc protocol and calls the analyzer;
- CORBA: LIMA can act as a CORBA server using a dedicated API;
- SOAP Web Services: we implement several of the services specifications defined by the WebContent project[4], including `NamedEntitiesExtraction`, `SemanticAnnotation`, etc.

This versatility makes LIMA very easy to integrate into larger applications.

To illustrate the flexibility of the general architecture, we present how we used LIMA to analyze the output from speech-to-text transcription tools. This can be made without difficulties by simply configuring the reader of LIMA and making it accept as input documents with format specific to the tool used to perform transcription. Document are considered in this configuration as any other written text.

In order to better take into account the nature of the source document and to fully exploit the results of the transcription step, we propose another configuration. The idea is to keep not only the most likely words, as computed by the final stage of transcription, but to keep also other hypothesis with their respective weights as inputs for the linguistic processing performed by LIMA. To be able to keep these hypotheses it is necessary to import a lattice. In this lattice, a node is created to represent each word hypothese. The word is associated with a value wich reflects its occurrence probability according to the acoustic model. To perform this import, we do not need to change anything in the architecture or recompile existing code. We have written a process unit which reads the outputs from speech-to-text tool and builds a graph. To integrate the new process unit into the pipeline we have substituted it to the tokenizer in the pipeline configuration.

3.3 Applications based on LIMA

Several applications developed by the CEA LIST use the LIMA framework as a core module for linguistic analysis: (1) a multimedia search engine, in which LIMA is used for the analysis of both textual documents and queries (with different pipelines); this search engine has been used in various CLEF campaigns (Besançon et al., 2004); (2) the ŒDIPE question answering system, in which LIMA is used for the question analysis and in the search engine used to pre-select the documents; this QA system has participated in CLEF and EQUER campaigns (Besançon et al., 2007); (3) the CHORAL summarization system, in which LIMA is used for the linguistic analysis of the text to identify the most important concepts used for sentence selection and to provide a syntactic analysis that can be used for syntactic simplification of kept sentences (Chalendar (de) et al., 2005; Garcia-Flores and Chalendar de, 2008). The efficiency of LIMA was also illustrated by its use in the Semantic Map project (Grefenstette, 2007) where its syntactic analyzer processed 2 million French Web pages to build a large network of syntactic co-occurrences that now supports work about the automatic building of semantic resources such as word senses (Mouton et al., 2009) or semantic frames.

[4] http://www.webcontent-project.org

4 Conclusion and future work

In this article, we have presented LIMA, a platform dedicated to the implementation of Natural Language Processing based applications. LIMA was designed to offer a large flexibility of configuration without sacrificing processing speed and quality of results. On one side, the emphasis in LIMA on flexibility comes from two needs: taking into account the problem of multilinguality and covering a wide range of applications. On the other side, the need for processing very large corpora with the deeper and deeper levels of analysis required by advanced search tools such as question-answering systems justifies the need for efficiency in LIMA. Furthermore, we have seen that LIMA is not only an architecture but also gathers a set of tools for several languages, going from tokenization to syntactic analysis[5] and taking into account various linguistic phenomenons such as the absence of delimiters, agglutination, composition or the lack of vowels. LIMA was also used in a large set of applications as various as information retrieval, ontology population, automatic summarization or question-answering.

Although LIMA is already usable in a wide set of contexts, we plan to develop several extensions for enlarging its capabilities. From the architecture viewpoint, this enlargement focuses on execution control. First, the control structures for defining a pipeline are currently very limited in LIMA since a pipeline is restricted to a list of processing units. Introducing conditional or iteration structures as in UIMA would be a first way for a pipeline to adapt the processing of documents to their content. Furthermore, we would like to make LIMA able to modify its configuration dynamically: current LIMA pipelines are only statically configurable and a dynamic configuration would enable the modification of a pipeline according to the results of a module, which is a powerful means for adapting the processing of a document according to its content. Finally, the widespread use of clusters and multi-core processors makes necessary for LIMA to define ways of exploiting parallelism.

From a wider perspective, we apply LIMA not only to textual documents but more and more to multimedia documents, which has led us to generalize and extend the architecture of LIMA for supporting the multimedia search engine we currently develop (Delezoide et al., 2010 to appear). We plan to release this extended version of LIMA as a open-source software.

5 References

Romaric Besançon and Gaël Chalendar (de). 2005. L'analyseur syntaxique de LIMA dans la campagne d'évaluation EASY. In *actes de la 12e conférence annuelle sur le Traitement Automatique des Langues Naturelles, TALN 2005*, Dourdan, France, June.

Romaric Besançon, Gaël de Chalendar, Olivier Ferret, Christian Fluhr, Olivier Mesnard, and Hubert Naets. 2004. In *4th Workshop of the Cross-Language Evaluation Forum, CLEF 2003*, chapter Concept-based Searching and Merging for Multilingual Information Retrieval: First Experiments at CLEF 2003, pages 174–184. Springer Verlag.

Romaric Besançon, Mehdi Embarek, and Olivier Ferret. 2007. In *Evaluation of Multilingual and Multi-modal Information Retrieval - 7th Workshop of the Cross-Language Evaluation Forum, CLEF 2006*, volume 4730 of *Lecture Notes in Computer Science*, chapter Finding Answers in the Œdipe System by Extracting and Applying Linguistic Patterns. Springer Berlin / Heidelberg, September.

Steven Bird and Mark Liberman. 1999. Annotation graphs as a framework for multidimensional linguistic data analysis. In *ACL Workshop Towards Standards and Tools for Discourse Tagging*, pages 1–10.

Olivier Blanc, Matthieu Constant, and Eric Laporte. 2006. Outilex, plate-forme logicielle de traitement de textes écrits. In *13^{e} conférence annuelle sur le Traitement Automatique des Langues Naturelles (TALN 2006).*

Chris Brew, David McKelvie, Richard Tobin, Henry Thompson, and Andrei Mikheev. 1999. The XML Library LT XML version 1.1 User documentation and reference guide. Technical report, Language Technology Group.

Xavier Carreras, Isaac Chao, Lluí Padró, and Muntsa Padró. 2004. Freeling: An open-source suite of language analyzers. In *4th International Conference on Language Resources and Evaluation (LREC'04).*

Gaël Chalendar (de), Romaric Besançon, Olivier Ferret, Gregory Grefenstette, and Olivier Mesnard. 2005. Crosslingual summarization with thematic extraction, syntactic sentence simplication, and bilingual generation. In *Crossing Barriers in Text Summarization Research Workshop, RANLP-2005.*

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. 2002. GATE: A framework and graphical development environment for robust NLP tools and applications. In *40th Annual Meeting of the Association for Computational Linguistics (ACL'02).*

Gaël de Chalendar and Damien Nouvel. 2009. Modular resource development and diagnostic evaluation framework for fast NLP system improvement. In *Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing (SETQA-NLP 2009)*, pages 65–73, Boulder, Colorado, June. Association for Computational Linguistics.

Eric V. de la Clergerie, Olivier Hamon, Djamel Mostefa, Christelle Ayache, Patrick Paroubek, and Anne Vilnat. 2008. Passage: from french parser evaluation to large sized treebank. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08).*

Bertrand Delezoide, Hervé Le Borgne, Romaric Besançon, Gaël de Chalendar, Olivier Ferret, Faiza Gara, Patrick Hède, Meriama Laib, Olivier Mesnard, Pierre-Alain Moëllic, and Nasredine Semmar. 2010, to appear. MM: modular architecture for multimedia information retrieval. In *8th International Workshop on Content-Based Multimedia Indexing (CBMI 2010), demo session*, Grenoble, France.

David Ferrucci and Adam Lally. 2004. UIMA: an architec-

[5] The highest level of analysis is not the same for all the languages processed by LIMA.

tural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348.

Jorge Garcia-Flores and Gaël Chalendar de. 2008. Syntactico-semantic analysis: a hybrid sentence extraction strategy for automatic summarization. In *MICAI 2008: Advances in Artificial Intelligence*, Atizapán, México, November 2008. Springer, Lecture Notes in Artificial Intelligence.

Gregory Grefenstette. 2007. Conquering language: Using nlp on a massive scale to build high dimensional language models from the web. In *8th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing 2007*, pages 35–49, Mexico City, Mexico.

Faiza Elkateb-Gara Gregory Grefenstette, Nasredine Semmar. 2005. Modifying a natural language processing system for european languages to treat arabic in information processing and information retrieval applications. In *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, pages 31–37, Michigan, USA, June.

Ralph Grishman. 1997. Tipster architecture design document version 2.3. Technical report, DARPA.

Claire Grover, John Carroll, and Ted Briscoe. 1993. The Alvey Natural Language Tools grammar (4^{th} release). Technical Report 284, Computer Laboratory, Cambridge University.

Nancy Ide and Jean Véronis. 1994. Multext : Multilingual text tools and corpora. In *15^{th} International Conference on Computational Linguistics (COLING'94)*, pages 588–592.

Svetla Koeva, Denis Maurel, and Max Silberztein. 2007. *Formaliser les langues avec l'ordinateur : De Intex à Nooj*. Les Cahiers de la MSH Ledoux. Presses Universitaires de Franche-Comté.

Claire Mouton, Guillaume Pitel, Gaël Chalendar (de), and Anne Vilnat. 2009. Unsupervised word sense induction from multiple semantic spaces with locality sensitive hashing. In *7th Conference on Recent Advances in Natural Language Processing (RANLP 2009) - poster session*, Borovets, Bulgaria.

Mary S. Neff, Roy J. Byrd, and Branimir K. Radev. 2004. The Talent system: TEXTRACT architecture and data model. *Natural Language Engineering*, 10(3-4):307–326.

Patrick Pantel, Eric Crestan, Arkady Borkovsky, Ana-Maria Popescu, and Vishnu Vyas. 2009. Web-scale distributional similarity and entity set expansion. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 938–947, Singapore, August. Association for Computational Linguistics.

Neil K. Simkins. 1994. An open architecture for language engineering. In *First CEC Language Engineering Convention*.

Francis Wolinski, Frantz Vichot, and Olivier Gremont. 1998. Producing nlp-based on-line contentware. In *Natural Language Processing and Industrial Applications (NLP+IA'98)*, volume 2, pages 253–259.