Modular resource development and diagnostic evaluation framework for fast NLP system improvement

Gaël de Chalendar, Damien Nouvel
CEA, LIST, Multilingual Multimedia Knowledge Engineering Laboratory,
F-92265 Fontenay-aux-Roses, France.
{Gael.de-Chalendar,Damien.Nouvel}@cea.fr

Abstract

Natural Language Processing systems are large-scale softwares, whose development involves many man-years of work, in terms of both coding and resource development. Given a dictionary of 110k lemmas, a few hundred syntactic analysis rules, 20k ngrams matrices and other resources, what will be the impact on a syntactic analyzer of adding a new possible category to a given verb? What will be the consequences of a new syntactic rules addition? 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. We present here a framework designed to effectively and iteratively improve the accuracy of our linguistic analyzer LIMA by iterative refinements of its linguistic resources. These improvements are continuously assessed by evaluating the analyzer performance against a reference corpus. Our first results show that this framework is really helpful towards this goal.

1 Introduction

1.1 The evaluation framework

In Natural Language Processing (NLP), robustness and reliability of linguistic analyzers becomes an everyday more addressed issue, given the increasing size of resources and the amount of code implied by the implementation of such systems. Beyond choosing a sound technology, one must now have efficients and user-friendly tools around the system itself, for evaluating its accuracy. As shown by (Chatzichrisafis et al., 2008), where developers receive daily reports of system's performance for improving their system, systematic evaluation with regression testing has shown to be gainful to accelerate grammar engineering.

Evaluation campaigns, where several participants evaluate their system's performance on a specific task against other systems, are a good mean to search for directions in which a system may be able to improve its performance. Often, these evaluation campaigns also give possibility for participants to run their analyzer on test data and retrieve evaluation results. In this context, parsers authors may rely on evaluation campaigns to provide performance results, but they should also be able to continuously evaluate and improve their analyzers between evaluation campaigns. We aim at providing such a generic evaluation tool, using evaluation data to assess systems accuracy, this software will be referenced as the "Benchmarking Tool".

Approaches concerning Natural Language Processing involve everyday more and more resource data for analyzing texts. These resources have grown enough (in terms of volume and diversity), that it now becomes a challenge to manipulate them, even for experienced users. Moreover, it is needed to have non-developers being able to work on these resources: it is necessary to develop accessible tools through intuitive graphical user interfaces. Such a resource editing GUI tool represent the second part of our contribution, called the "Resource Tool".

The overall picture is to build a diagnostic framework enabling a language specialist, such as a linguist, to status, almost in real-time, how modifica-

Proceedings of the NAACL HLT Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing, pages 65–73,
Boulder, Colorado, June 2009.

tions impact our analyzer on as much test data as possible. For analyzers, each resource may have an effect on the final accuracy of the analysis. It is often needed to iterate over tests before understanding what resource, what part of the code needs to be improved. This is especially the case with grammar engineering, where it is difficult to predict the consequences of modifying a single rule. Ideally, our framework would allow the manipulator to slightly alter a resource, trigger an evaluation and, almost instantaneously, view results and interpret them. With this framework, we expect a large acceleration in the process of improving our analyzer.

In the remaining of this introduction, we will describe our analyzer and Passage, a collaborative project including an evaluation campaign and the production of a reference treebank for French through a voting procedure. Section 2 will describe our evaluation framework; its architecture, its two main modules and our first results using it. Section 3 describes some related works. We conclude in section 4 by describing the next steps of our work.

1.2 The LIMA linguistic analyzer

Our linguistic analyzer LIMA (LIc2m Multilingual Analyzer, (Besancon and de Chalendar, 2005)), is implemented as a pipeline of independent modules applied successively on a text. It implements a dependency grammar (Kahane, 2000) in the sense that produced analysis are exclusively represented as binary dependency relations between tokens.

The analyzer includes, among other modules, a tokenizer segmenting the text based on punctuation marks, a part of speech tagger, short and long distance dependencies extractors based on finite-state automata defined by contextualized rules. The latter rules express successions of categories, augmented with constraints (on words inflexion, existence of other dependencies, etc.). The analyzer also includes modules to find idiomatic expressions and named entities that, once recognized, are merged into a single token, thus allowing grammar rules to apply on those. Furthermore, modules may be specialized in processing language-specific phenomena, e.g. Chinese tokenization, German compounds, etc. Currently, the analyzer is able to process more or less deeply ten languages, including English, Spanish, Chinese, Arab, French and German.

1.3 The Passage Project

Our work is part of the Passage project (Clergerie et al., 2008b). The objectives of this project are twofold. Firstly, it organizes two evaluation campaigns of syntactic analyzers (around 15 participating systems) for the French language. Secondly, it aims at producing a large scale reference treebank for French by merging the output of all the participating parsers, using a Rover (Recognizer Output Voting Error Reduction) (Fiscus, 1997) approach.

Within this project, syntactic annotations are produced in a common format, rich enough to represent all necessary linguistic features and simple enough to allow participating parsers (using very different parsing approaches) to represent their analysis in this format. It is an evolution of the EASy campaign format, mixing simple non recursive chunks and dependency relations between chunks or tokens. It respects two proposed ISO specifications: MAF (ISO 24611) and SynAF (ISO 24615). The chunks and dependencies types are issued from the ISO data category registry, DCR[1], currently using the French language section names. The syntactic analysis of a corpus in the Passage format provides information about:

- Segmentation of the corpus into sentences
- Segmentation of sentences into forms
- Non-recursive typed (listed in Table 1) chunks embedding forms
- Labeled typed (listed in Table 2) dependencies that are anchored by either forms or chunks

Type	Explanation
GN	Nominal Chunk
NV	Verbal Kernel
GA	Adjectival Chunk
GR	Adverbial Chunk
GP	Prepositional Chunk
PV	Prepositional non-tensed Verbal Kernel

Table 1: Chunks types

Within the EASy project, parsers have been evaluated against a reference, which itself was a small subset of the available corpora. The reference was

[1] http://www.isocat.org

Type	Explanation
SUJ-V	Subject-verb
AUX-V	Aux-verb
COD-V	Direct objects
CPL-V	Other verb arguments/complements
MOD-V	Verb modifiers (e.g. adverbs)
COMP	Subordinate sentences
ATB-SO	Verb attribute
MOD-N	Noun modifier
MOD-A	Adjective modifier
MOD-R	Adverb modifier
MOD-P	Preposition modifier
COORD	Coordination
APPOS	Apposition
JUXT	Juxtaposition

Table 2: Dependencies types

created by human annotation of random sentences within the corpora. Thus, once this evaluation campaign had been finished, the annotated corpora reference was released for participants to test and improve their parser. Currently, we use this reference for benchmarking our analyzer.

1.4 Metrics for parsing evaluation

We are constantly recalled that evaluation metrics and methodologies evolve and are subject to intense research and innovation (Carroll et al., 2002). Discussing these metrics is not in the scope of this paper, we only need to be able to work out as many metrics as possible on the entire corpus or on any part of it. The evaluation is supposed, for each document d and for each type (of chunk or of dependency) t within all types set T, to return following counts:

- Number of items found and correct - $fc(d,t)$
- Number of items found - $f(d,t)$
- Number of items correct - $c(d,t)$

With this approach, we are able to compute common Information Retrieval (IR) metrics (Rijsbergen, 1979): precision, recall, f-measure. We also introduce a new metric that gives us indications about what types are the most lowering overall performance, called "Type error ratio":

$$\frac{f(d,t) + c(d,t) - 2.fc(d,t)}{\sum^{t \in T} f(d,t) + c(d,t) - 2.fc(d,t)} \quad (1)$$

This metric counts the number of errors and misses for a given type reported to the total number of errors and misses. It allows us to quantify how much an improvement on a given type will improve the overall score. In our case, scores are computed for chunks on the one hand, and for dependencies on the other hand. For instance, we have notices that GN errors represent 34.6% of the chunks errors, whereas PV only represent 2.2%: we are thus much more interested in improving detection of GN than PV regarding current evaluation campaign.

2 The evaluation framework

2.1 Architecture

We need our framework to be portable and to be implemented using an agile approach: each new version should be fully functional while adding some more features. It also must be user-friendly, allowing to easily add eye-candy features. Consequently, we have chosen to implement these tools in C++, using the Qt 4.5 library[2]. This library satisfies our requirements and will allow to rely on stable and open source (LGPL) tools, making it feasible for us to possibly deliver our framework as a free software.

This approach allows us to quickly deliver working software while continuously testing and developing it. Iterations of this process are still occurring but the current version, with its core functions, already succeeded in running benchmarks and in beginning the improvement of our linguistic resources while regularly delivering upgraded versions of our framework. First results of this work will be presented below in this paper.

The open architecture we have chosen implies to use externals tools, for analysis and evaluation on the one hand, for compiling and installing resources on the other hand. These tools may then be considered as black boxes, being externals commands called with convenient parameters. In particular, the Benchmarking Tool relies on two commands: the analyzer command, receiving input file as a parameter and producing the analyzed file, the evaluation command, receiving the analyzed file and the reference file as parameters and outputting counts of found, correct, found and correct items for each dimension. This allows, for example, to replace our

[2] http://www.qtsoftware.com/

analyzer with another one, by just wrapping the latter in a thin conversion layer to convert its inputs and its outputs.

2.2 Benchmarking Tool

The Benchmarking Tool, which architecture is depicted in Figure 1, is responsible of executing analysis and evaluation on pairs of data and reference files, using commands stored in benchmarking configuration. For each pair of files, the registered analysis command is executed followed by the evaluation one. In our case, those commands apply to the task of annotating files for syntactic chunks and dependencies.

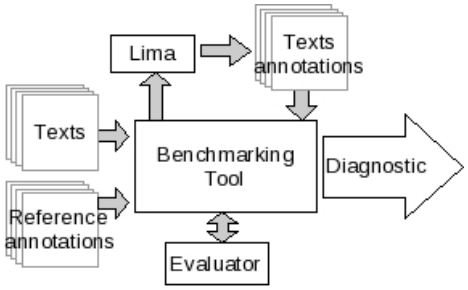

Figure 1: Benchmarking Tool data flow

We may consider the type of chunks and dependencies as dimensions of an evaluation. To a certain extent, these may be associated to linguistics phenomena which are tested, as proposed within the TSNLP project (Balkan et al., 1994) or, more recently, for Q/A systems by (Paiva et al., 2008). But in these projects, focus is also made on the evaluation tool, where we do not implement the evaluation tool but rely on an external program to provide accuracy of analysis.

The pairs of data and reference files are inserted inside a structure implemented as a pipeline, which may be modified (adding, removing, reordering units) with common GUI interfaces. After creation of the pipeline, the user may trigger a benchmarking (progress is shown by coloring pipeline units), which may be suspended, resumed or restarted at any moment. For note, the current version of the framework uses the local machine's processors to analyze pipeline units in parallel, but we intend to distribute the analyzes on the available nodes of a cluster soon. As soon as results are received, tables and graphics are updated on screen within a view showing previous and current results for each evaluated dimension. To refine diagnosis, the user may choose what dimensions are displayed, what metrics should be computed, and what pipeline units are used. Finally, any evaluation may be deleted if the corresponding modification did not increase performance and should be reverted.

Upon demand, the tool saves current benchmarking configuration and results as an XML file. Conversely, it loads a pipeline and results from file, so as to resume or switch between evaluations. The parsed output of the evaluator tool is recorded for each pipeline unit and for each dimension, so that metrics based on those quantities are computed for each pipeline unity or for the overall corpus. Besides, the date and a user comment for each evaluation are also saved for these records. Writing comments has proved to be very helpful to keep track of what changes have been made on code, linguistic resources, configuration, parameters, etc.

As an example within the Passage project, running evaluation with the Benchmarking Tool allowed us to notice that we had difficulties in recognizing verb-auxiliary dependencies. Considering previous results, we detected that this issue appeared after having introduced a set of idioms concerning pronominal verbs. Unit testing showed that the analysis of past perfect progressive for pronominal verbs was buggy. Patching the code gave us a 10 points f-measure gain for AUX-V dimension and 0.3 for all dependencies dimensions (AUX-V having a 2.6% global error rate within dependencies). Thus, benchmarking results have been saved with appropriate comment and other improvements or deficiencies could be examined.

With these features, the tool offers the possibility to have an overall view on evaluation results and on their evolution across time, given multiple data, dimensions of analysis and computed metrics. Therefore, it helps us, without any complex manipulation, to get a visual report on what implication on evaluation results has a modification to the analysis process. Furthermore, those tests allow to search for errors in resources as well as in code, so as to find how to enrich our linguistic resources or to identify deficiencies in our code.

Figure 2 shows a benchmarking using a set of 24 evaluation files (left part) to improve the analyzer's

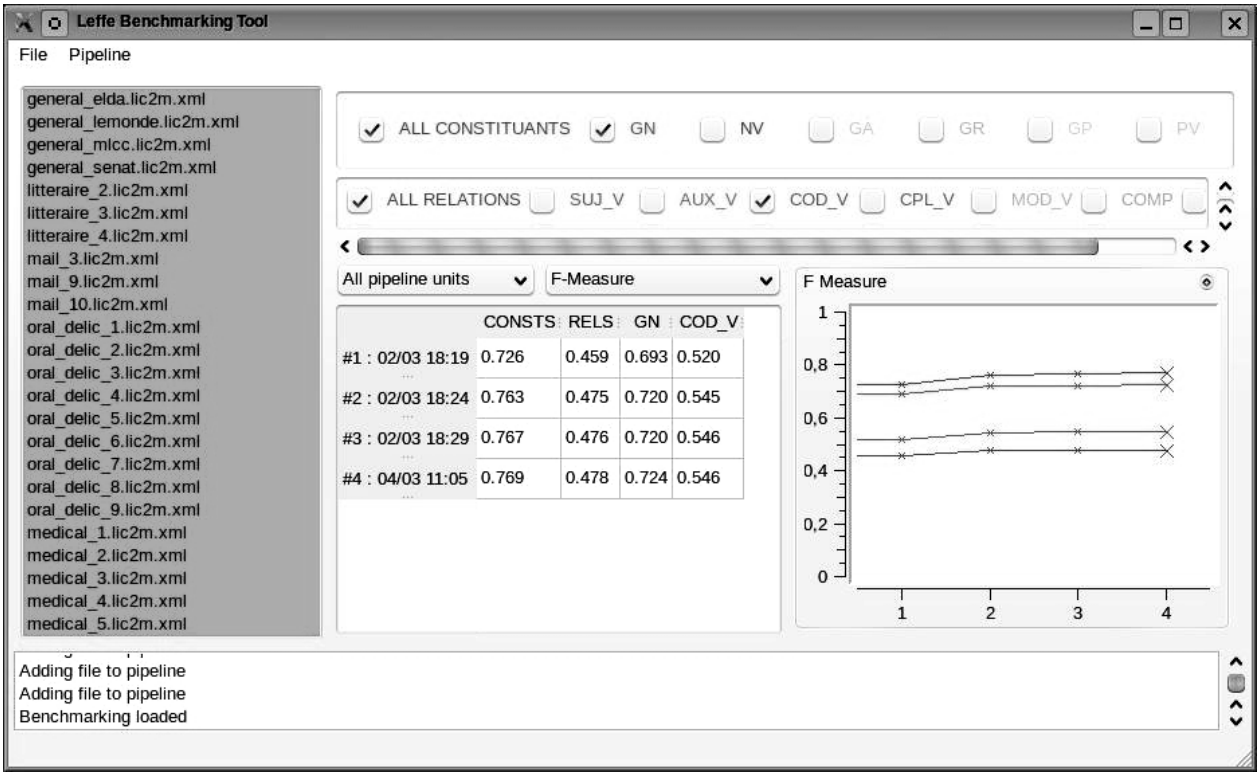

Figure 2: Chunks (CONSTS), dependencies (RELS), nominal chunks (GN) and direct objects dependencies (COD_V) f-measure results evolution through 4 evaluations on a 24 files corpus

results. The central table shows the measures corresponding to 4 successive evaluations, displaying results for the dimensions selected on the top most part (check-boxes). The right-hand side shows graphically the same data, successive evaluations being displayed as its abscissa and measures as its ordinate.

2.3 Resource Tool

The Resource Tool, which modular design is depicted in Figure 3, aims at making resources editing accessible for people who have neither a deep knowledge of the system internals nor computer programming skills. Enriching our resources implies having people, either specialized in linguistics or in testing to interact with the resources, even if not accustomed to our specific storage format for each resource.

In its current version, the Resource Tool allow to edit the following resources:

- Dictionary: items and their categories
- Syntactic rules: syntactic dependency detection

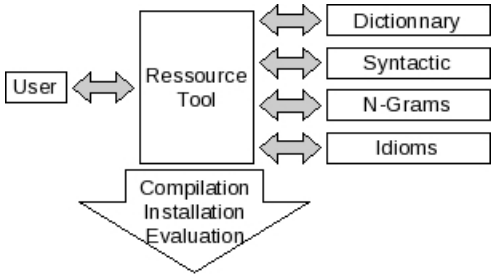

Figure 3: Resource Tool modular design

- Part-of-speech tagger learning corpus: tagged examples of *ngrams* disambiguation matrices
- Idioms: language dependent fixed expressions

Those resources are presented in a tabbed view, each having a dedicated interface and access functions. Within each resource, a search feature is implemented, which has shown to be really useful, especially for dictionary. The tool also provides simple means to save, compile and install resources, once they have been modified. This has to be very transparent for the user and we just provide a “Save” button and another “Compile and install” button. The current version of Resource Tool is quite ba-

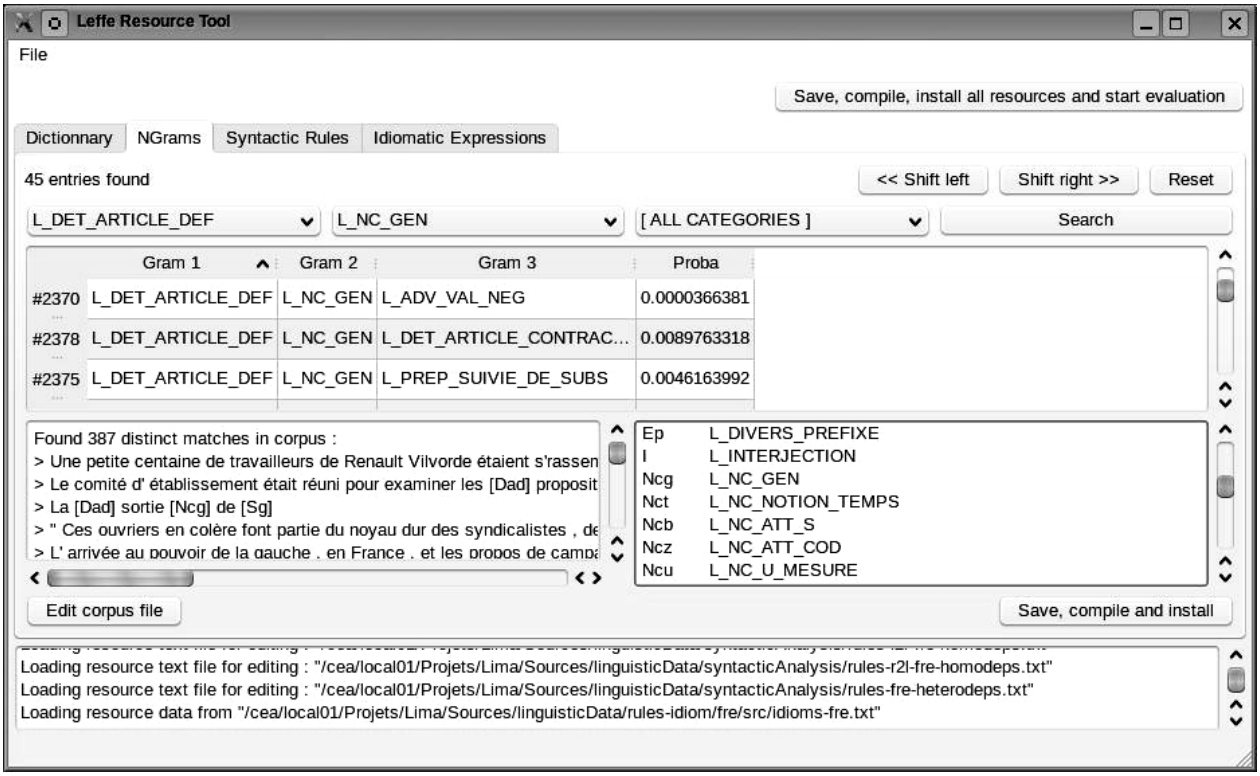

Figure 4: Viewing and editing disambiguation matrices: probabilities and examples for articles followed by nouns

sic in terms of edition capacities. Dictionary has a dedicated interface for editing words and their categories, but ngrams, syntactic rules and idioms resources may yet only be changed through a basic text editor.

Figure 4 shows the resource tool interface for the annotated corpus that allows to build part-of-speech disambiguation matrices. The top most tabs allow to switch between resources among editable ones. The data table shows the computed 3-grams (from our own tag set). The left part text field shows a list of sentences, where occurrences of the ngrams selected in the above table appear. The right part text field shows correspondences between two tag sets. Eventually, the "Edit corpus file" button opens an editor for the user to add sentences or to modify sentences in the tagged corpus.

The Resource Tool and the Benchmarking Tool communicate together through two signals: on the one hand when resources are installed, the Resource Tool may trigger an evaluation in the Benchmarking Tool, on the other hand when the evaluation has finished, the Resource Tool is notified and warns the user. Being aware of their respective status, we also warn the user for dangerous operations, like when trying to install resources while a benchmarking is still running, or when quitting the application before last benchmark is finished.

While these two applications are connected to be aware of benchmarking and resource installation status, no more interaction has been implemented for the moment to link evaluation and resource edition together. We have considered implementing a feature making possible to automatically do unit testing resource modifications, but, from our point of view, this has to be implemented with following restrictions: the Benchmarking Tool should remain generic (modifying configuration and resources should not be part of the tool) ; amount of required disk space should remain minimal (only differences between evaluations should be stored).

2.4 Preliminary results

We recently finished the first implementation iteration. The evaluator itself is provided by a partner laboratory. Its measurement methodology is deeply presented in (Paroubekr, 2006). From our point of view, we are only concerned in the fact that these

Chunks			Dependencies			Modifications
F	P	R	F	P	R	
72.6	72.0	73.2	45.9	54.2	39.8	Initial evaluation
76.3	76.2	76.3	47.5	56.1	41.1	Code reengineering / debugging
76.7	76.7	**76.7**	47.6	56.2	41.3	New set of syntactic rules
76.9	76.9	76.9	47.8	**56.7**	41.4	Specified preposition detection rules

Table 3: Benchmarking results, f-measure (F), precision (P), recall (R)

measures are relevant for improving the quality of analysis produced by our parser.

We applied our resource improvement methodology on a small annotated corpus of approximately 80.000 words, delivered after the EASy campaign, among 27 thematic files. For information, the whole process (analysis and evaluation for each file) is 5 minutes long on a bi-processor: this allows the software to be used intensively on a personal computer. Results in Table 3 show that the use of our framework already allowed us to introduce modifications of the linguistic resources with the Resource Tool; these changes lead to a slight improvement of the overall score of the system.

First, we obtained confirmation that some code reengineering and some debugging was required. These tasks, associated with iterative evaluation, have allowed us to detect parts of the code which did not give entire satisfaction, especially in the step transforming output from our analyzer to the expected Passage format. We also found a bug within the evaluation scripts, which, once corrected, forced to restart evaluation measures from the beginning: this shows the importance of having a stable environment apart analyzer (evaluation process, valid data and reference file). These results show that iterating over time and saving history may help to reveal potential weaknesses of the code and to detect what goes wrong.

Secondly, these tools where well-suited for evaluating the impact of a new set of syntactic rules, for which we did not have opportunities to do precise evaluation before. For this set of 20 rules, we systematically tried each rule separately, then kept the combination of the rules increasing scores. This improvement may appear as minimal, but these rules where written in the context of an ongoing work on our grammar. It gave an intuitive idea that this approach is not a dead-end and may be further explored. Besides, methodologies have been suggested to test the impact of each rule in the entire set of rules by systematically testing combinations of rules. But, currently, this is beyond our goal.

Finally, we also introduced some "syntactic sugar", by grouping some expressions within rules, and successfully obtained insurance that these modification did not lower scores. This is an important result for us in the sense that we ensure that the same set of rules expressed differently (with rules more concise thus more readable) do not introduce regressions.

3 Related works

We have previously described the test suite approach, along with the TSNLP project. This approach was concerned with identifying and systematically testing linguistic phenomena. As a conclusion of TSNLP, (Oepen et al., 1998) points out the necessity "to assess the impact of individual contributions, regularly evaluate the quality of the overall grammar, and compare it to previous versions". This project thus showed how positive it is to identify deficiencies and improve grammars by iterating tests over time. This is the goal we intend to reach with our framework.

More recently, in biomedical domain, (Baumgartner et al., 2008) describes implementation of a framework and, although it is applied to a text mining task, the approach remains quite close in its foundations (evaluation oriented, iterative testing, modular framework, open source, corpora based, etc.) to ours and encourages these kind of initiative by showing the importance of continuous evaluation while coding parser and engineering grammar. This work present the interest to rely on the UIMA framework, thus allowing a good modularity. In the future, we should study the interest to give the ability to our framework to integrate UIMA-ready modules.

Close to our Benchmarking Tool, some projects aim at building frameworks for text analysis, annotation and evaluation, which projects encourage people to use a common architecture, as openNLP or GATE. Those may also be used for benchmarking and evaluation tasks (Cunningham et al., 2002) as part of their process. But, while these framework often provide evaluation and regression testing tools, they are rarely well-suited for only implementing specific diagnostic tasks. We would appreciate that such frameworks focusing on evaluating, benchmarking and diagnosing, as generic as possible across IR tasks, become more widely available. If our Benchmarking Tool appears to be appropriate for other systems evaluations, we will consider making it available for the IR community.

4 Conclusions and future work

From our first use of the framework, we are convinced of the importance of diagnostic for accelerating the improvement of our analyzer, by making linguistic resources accessible and by iterating tests and comparing results obtained over time. We also concluded that this generic framework would be useful in other tasks, such as Information Retrieval. Especially, image retrieval is a very active and growing field of research, and we currently consider applying the Benchmarking Tool for accelerating the improvement of the image retrieval system developed in our laboratory (Joint et al., 2004).

This work also emphasizes the great distinction between performance evaluation and diagnostic evaluation. In our case, the association of the Benchmarking Tool and the Resource Tool used in conjunction with unit and regression testings helps to identify what part of the analysis process is concerned and, for grammar engineering, what rule or set of rules have to be questioned in order to improve the overall system performance.

Future directions of our work include the parallelization of the analysis on a cluster, so as to retrieve evaluation results as quickly as possible. This should allow us to use evaluation results from a larger annotated corpus. We also intend to focus on visualization of results for better identification and interpretation of errors, in order to access directly erroneous analysis and involved resources. A second development iteration will include the development of more user friendly resources editors.

We also plan to work on automatic syntactic rules inference, based on previous work in our laboratory (Embarek and Ferret, 2008). For this goal, continuous benchmarking will be even more important as the system will rely on experts tuning parameters for learning rules, the syntactic rules themselves being not necessarily edited nor viewable for the expert.

Acknowledgments

This work was partly funded by the French National Research Agency (ANR), MDCA program 2006.

References

Lorna Balkan, Klaus Netterz, Doug Arnold, Siety Meijer, 1994. *Test Suites for Natural Language Processing.* Proceedings of the Language Engineering Convention (LEC'94), 17–22.

William A Baumgartner, Kevin Bretonnel Cohen, Lawrence Hunter, 2008. *An open-source framework for large-scale, flexible evaluation of biomedical text mining systems.* Journal of Biomedical Discovery and Collaboration 2008, Vol. 3, pp 1.

Romaric Besançon, Gaël de Chalendar, 2005. *L'analyseur syntaxique de LIMA dans la campagne d'valuation EASY.* Actes des Ateliers de la 12e Confrence annuelle sur le Traitement Automatique des Langues Naturelles (TALN 2005), Vol. 2, pp 21.

John Carroll, Anette Frank, Dekang Lin, Detlef Prescher, Hans Uszkoreit, 2002. *Proceedings of the workshop beyond parseval - toward improved evaluation measures for parsing systems.* Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC'02).

Nikos Chatzichrisafis, Dick Crouch, Tracy Holloway King, Rowan Nairn, Manny Rayner, Marianne Santaholma, 2007. *Regression Testing For Grammar-Based Systems.* Proceedings of the GEAF07 Workshop, pp 128–143.

Eric V. de la Clergerie, Olivier Hamon, Djamel Mostefa, Christelle Ayache, Patrick Paroubek, Anne Vilnat, 2008. *PASSAGE: from French Parser Evaluation to Large Sized Treebank.* Proceedings of the Sixth International Language Resources and Evaluation (LREC'08).

Eric V. de la Clergerie, Christelle Ayache, Gaël de Chalendar, Gil Francopoulo, Claire Gardent, Patrick Paroubek, 2008. *Large scale production of syntactic*

annotations for French. In Proceedings of the international workshop on Automated Syntactic Annotations for Interoperable Language Resources, Hong-Kong.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, 2002. *GATE: A framework and graphical development environment for robust NLP tools and applications.* Proceedings of the 40th Anniversary Meeting of the ACL, 2002.

Mehdi Embarek, Olivier Ferret, 2008. *Learning patterns for building resources about semantic relations in the medical domain.* 6th Conference on Language Resources and Evaluation (LREC'08), Marrakech, Morocco.

Jonathan G. Fiscus, 1997. *A Post-Processing System to Yield Reduced Word Error Rates: Recognizer Output Voting Error Reduction (ROVER).* Proceedings IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU97), pp 347–352.

Magali Joint, Pierre-Alain Moellic, Patrick Hede, Pascal Adam, 2004. *PIRIA: a general tool for indexing, search, and retrieval of multimedia content.* Proceedings of SPIE, Vol. 5298, 116 (2004), San Jose, CA, USA.

Sylvain Kahane, 2000. *Les grammaires de dpendance.* Traitement Automatique des Langues, Vol. 41.

Stephan Oepen, Daniel P. Flickinger, 1998. *Towards systematic grammar profiling. Test suite technology ten years after.* Special Issue on Evaluation 12, 411–436.

Valeria de Paiva, Tracy Holloway King, 2008. *Designing Testsuites for Grammar-based Systems in Applications.* Proceedings of the GEAF08 Workshop, pp 49–56.

Patrick Paroubek, Isabelle Robba, Anne Vilnat, Christelle Ayache, 2006. *Data, Annotations and Measures in EASY, the Evaluation Campaign for Parsers of French.* 5th Conference on Language Resources and Evaluation (LREC'06), Genoa, Italy.

C. J. van Rijsbergen, 1979. *Information Retrieval, 2nd edition.*