A NLP engine from the lab to the iPhone

Sébastien Paumier
Université Paris-Est,
LIGM, France
paumier@univ-mlv.fr

François Liger,
Gilles Vollant,
Anastasia Yannacopoulou
Ergonotics, France
francois@ergonotics.com
gilles@ergonotics.com
anastasia@ergonotics.com

Sylvain Surcin
Kwaga, Paris, France
surcin@kwaga.com

Abstract

This paper describes a successful collaboration between the academic world in NLP and industry in order to develop real-life applications that benefit from both theoretical research and engineering skills. This collaboration led to the development of a real-time text analyzer for a PDA note taking application and a server-based mail parsing application. The Open Source philosophy suited this partnership perfectly and did not conflict with trade secret.

1 Introduction

In some NLP applications, a gap exists between what is known to be feasible according to the academic state of the art and the actual use of those technologies in industrial products. A major cause is that researchers often develop prototypes that are typically proofs of concept, but not reliable and reusable components ready to be integrated in industrial workflows. As a consequence, the only available options for industry developers are too often either to recode engines from scratch, which requires highly qualified manpower, or to accept the limitations of available tools.

However, there is an evolution in relations between industrial and academic worlds, for it had become clear that there are mutual advantages to this kind of partnership, as shown in (Etzkowitz et al., 1998), (Okubo and Sjoberg, 2000) and (Newell et al., 2006). In this paper, we describe such a fruitful collaboration, with significant benefits for both. We will first analyse the preconditions that made this possible. Special attention will be paid to the advantages of the Open Source model, and we will show that it has not been incompatible with trade secret, as one may fear. Then, we will describe the way this collaboration actually took place and how all partners influenced the NLP engine development roadmap.

Finally we will present experimental results that are so impressive that they open new perspectives that academics would not have dreamt of, demonstrating that optimization is not relevant to engineering only, but that it can lead to progress in science too.

2 Preconditions

First of all, let us introduce the background of this work. The idea was to produce two applications: a commercial note taking application capable of analyzing user input on the fly in order to perform tasks such as automatically inserting meetings into his/her agenda, and another application performing a semantic analysis of user's mail to pinpoint and synthesize relevant information. To achieve these goals, one of the steps was to parse the input and to do so, the developers had to choose an existing NLP toolbox. Based on the previous experience of one of them, they chose Unitex, a corpus-processing tool developed at University Paris-Est Marne-la-Vallée[1]. This engine is designed for applying electronic dictionaries and grammars to texts. Its major use is to perform advanced linguistic queries, more complicated than regular expressions over characters. As a consequence, it is very well suited to information extraction tasks such as those required by both projects. However, some conditions had to be met before this tool could be used in a production environment: availability, reliability and efficiency.

The first condition (availability) was obviously met thanks to the engine's LGPL license. The second one (reliability) stemmed from the wide use of Unitex in many teaching and scientific projects. Since its inception in 2001, this system has been extensively tested. The last point (efficiency) was by far the most critical one, because of specific hardware constraints. Unitex was designed for

[1] http://igm.univ-mlv.fr/~unitex

dealing with large data on computers with large memory and no time constraints, not for parsing very small texts on handheld machines such as the iPhone, or parallel processing of very large amounts of small corpora under time constraints. Thus, it was necessary to investigate the code in order to determine if there was a sufficient margin for optimization that could reduce Unitex's runtime and memory consumption below the critical threshold for the projects. Once again, due to its Open Source license, this code review was possible and the conclusion of the feasibility test was positive.

3 Collaboration, Open Source and trade secret

As explained by Raymond (2001), Open Source has been proven to be able to produce reliable software, despite its uncentralized nature at the opposite of classical software engineering techniques like those discussed in (Brooks, 1995). This new way of producing software has been studied, in particular to analyse why this uncommon model seems to work (Johnson, 2006), (Lee et al., 2003).

As argued in (Pedersen, 2008) and (Paumier, 2009), Open Source should be the natural way of producing academic software. One of the main reasons is that enriching the science toolbox and software engineering are very distinct activities. In order to benefit from academic research, it is often easier for an industrial developer to adapt an available (and often ill-fitted) existing piece of software than to try to redevelop the whole thing from an obfuscated paper published in a scientific journal. Open Source can guarantee access to academic code, and it does not necessary imply that the industrial product should be Open Source itself, thanks to licenses like the LGPL[2]. So, if researchers can have their work used in real-life applications and industry can exploit the state of the art, both benefit from Open Source.

However, a thorny problem remains: a license such as the LGPL allows anyone to use a piece of software as a black box without any restriction, but modifications made to the code itself must be released as Open Source. This may raise practical problems if the open source engine is not perfect, which appeared to be the case with Unitex. A major optimization opportunity was to save on I/O operations by using persistent dictionaries and grammars, kept in memory for as long as needed. Such a modification was not a simple black box use case, since it had of course to be performed on the core Unitex code. The problem was that the technology used to manage such persistent objects had to be protected as an important part of the company's savoir-faire, which seemed contradictory with the LGPL.

This apparent deadlock was in fact quite easily solved. Unitex was provided with a callback system that calls the private library if present, and Unitex normal routines if not. Thus, the trade secret in the library remain protected without violating the LGPL. Of course, it required the callback system to be inserted into Unitex, but as it is Open Source, anyone could do so. The interesting point here is that it could have been done legally without the Unitex authors' consent, leading to a derivation of the original system. However, it was far more interesting to modify the main trunk of the system since a forked branch would not easily benefit from future development. So was done by requiring and obtaining access to the Unitex SVN server, thus initiating a real collaboration and not simply a one-way exploitation of the system.

4 Concrete cooperation

Once full SVN access obtained by the companies, the first step was to commit all the bug fixes produced during the code review. This was an opportunity for Unitex's main author to test the trustfulness of the new developers, as there is nothing worse than to give full powers to someone that messes up your own work. Mutual trust was certainly one of the most important success factors.

The next step was a discussion about the roadmap, since current research objectives were quite divergent from the industrial ones. We had to define priorities because some core modifications required for I/O optimizations had to be carefully planned to avoid conflicts at the lowest levels of the engine. Some parts of the code had to be marked as out of bounds until some ongoing research work was done, and some others were flagged as deprecated in order to avoid unnecessary code review.

Then the optimization started. Memory leaks were fixed and wasteful allocations, such as unnecessary local arrays in recursive functions, were removed. Performance optimizations came from several areas. First, the Unitex I/O library was se-

[2] `http://www.gnu.org/licenses/lgpl.html`

riously boosted by a wise use of buffering strategies. Then some objects were made persistent, which saved many operations. This part required discussions between partners in order to determine which data could be safely made persistent. It appeared to be the case for electronic dictionaries, but grammars had to be manipulated with caution, since some are modified at runtime depending on the text they are applied to. Without close cooperation, it would have been difficult to implement all those optimizations, since a deep knowledge of the code was necessary to foresee potential traps.

Another major improvement was obtained by gathering all Unitex code into a shared library, in order to minimize system time previously spent on launching a process for each Unitex program call. As a side effect, a big effort had to be made to clean up the code and free dangling objects, fix name conflicts and remove global variables, thus producing thread-safe code, which is always a pleasant plus when evaluating the quality of a piece of software.

5 Improving practices

Reliability is a priceless quality, but it is not acquired forever since any modification can compromise the safety of the system at any time. This is the reason why industrial development workflows use test sets in order to ensure that modifications do not lead to regression. However, this seducing and securing process is not worth the effort if the project is either small or controled by a very few developers, as often in research software development. So, even computer scientists that teach the arts of programming and software engineering seldomly practice themselves those wise rules, and it is thus a great opportunity for industrial developers to influence positively the academic world.

In Unitex, it took the form of a log system capable of recording any operation in order to replay it as needed. Test sets were prepared this way. Now, any modification can be tested along the whole test set to see whether it seems safe or not. And if a modification or a use case reveals a new bug that was not detected by the test set, we add a new test case.

This change obviously improves the software quality, but as a major side effect, it helps to educate users. A problem for developers of all kinds are useless ”Doesn’t work!” bug reports closer to spam than to usable feedback. With such a logger system, users are offered the possibility to build themselves test data that they can submit with useful comments like ”it worked with version X but not with version Y”. So, the introduction of such a continuous integration tool had helped to improve not only the product, but also its users’ responsibility, which is priceless.

6 Evaluation

6.1 Note taking application

This application uses two grammars, the second one being applied only when the first one does not match anything:

- `firstPass`: 435 subgraphs, 6 336 states, 12 426 transitions, 291 516 bytes
- `secondPass`: 367 subgraphs, 5 689 states, 11 196 transitions, 256 854 bytes

All results have been obtained on a Mac mini core 2 duo 1.6 GHz under Snow Leopard, with GCC 4.2. Times are given in milliseconds. Revision numbers are those on Unitex SVN server (with public anonymous read-only access[3]).

To begin with, here are 3 results obtained on the sentence *go to the cinema tonight, alarm 7pm* with simple optimizations, before the step when objects were made persistent:

- Revision 837 (the first 2.1 beta version, with no optimization): 811 ms
- Revision 911 (gcc’s -O2 option): 618 ms
- Revision 913 (optimization of hash tables): 176 ms

Table 1 shows the evolution of performance on the test corpus (1363 sentences, 222 of them needing the 2 graphs, about 6.5 words per sentence). Given times correspond respectively to the fastest, slowest and average processing of a sentence. The revision 1237++ row shows results obtained with optimizations made on the external private libraries. As one can see, the best results shown in the last row are far better than those obtained with version 2.0 of Unitex (more than 800 ms on a simple sentence).

[3] `https://svnigm.univ-mlv.fr/svn/unitex`

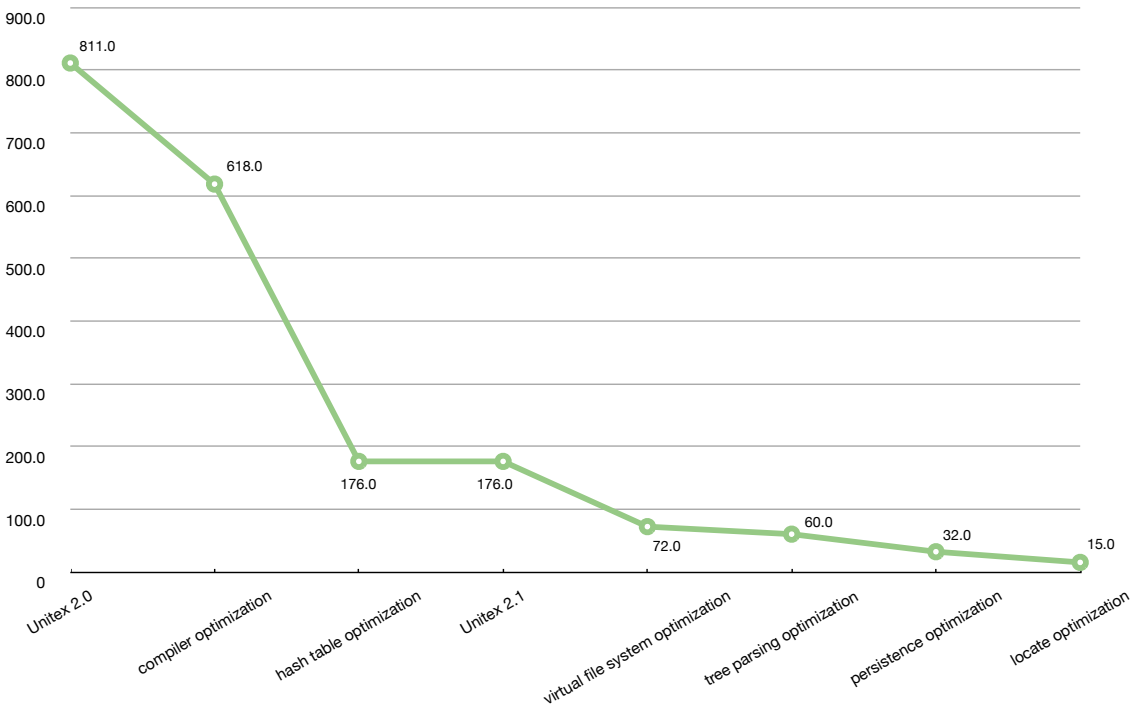

Figure 1: Average time spent processing a sentence in ms

Version	Min.	Max.	Aver.
Revision 1123, no virtual file	71	1101	176
Revision 1123, only virtual files	51	407	72
Revision 1123, persistant files	25	355	44
Revision 1237, virtual files	49	191	60
Revision 1237, persistant files	23	151	32
Revision 1237++, persistant files	8	127	15

Table 1: Performance evolution on the PDA note taking application

6.2 Mail parsing application

This application uses 20 grammars for each language it works on. In average, each grammar has 183 states and 20 transitions.

Tests have been made on a MacBook Unibody Intel Core 2 Duo 2.4 GHz with 4 Gb RAM. The test corpus is made of 143 mails of about 4 Kb per mail. As the application excludes some sections from the mails like quotations, it process only about 72 words per mail. Results are given in mails processed per minute and in millisecond per mail.

The first two rows of Table 2 show results obtained with regular versions of Unitex. The later ones shows results obtained with Unitex used as a dynamic library called from a JNI (Java Native Interface). The last row shows results obtained with optimizations made on the external private libraries.

7 Feedback to Open Source

Academic Open Source can provide industrials with the very handy practicality of available software solutions that can be tuned or upgraded to the level of industrially usable software components with less effort than developing them from scratch. And we have already seen that in return, the academics benefit from the industrialization process through the improvement of the quality, robustness

Version	Mails/mn	ms/mail
Unitex 2.0	5	11962
Unitex 2.1β, rev. 1237	23	2564
Library, rev. 1030	430	139
Library, rev. 1123	470	127
Library, rev. 1125	472	127
Library, rev. 1129	553	108
Library, rev. 1237	566	106
Library, rev. 1237++	711	84

Table 2: Performance evolution on the mail parsing application

and efficiency of their code.

Further cooperation is possible, and the industry may also produce Open Source software. This is the case of our cooperation as one of the industrial partners will provide very soon a Java encapsulation of the Unitex library, and a UIMA[4] annotator, together with a type system, built on top of it.

8 Conclusion

We have described a collaboration between the industrial and academic worlds built over an Open Source software project. The industrials have benefited from the academic knowledge about natural language processing with finite-state technologies. The academics have not only seen their software hugely improved (54 times faster for the PDA note taking application, 142 times faster for the mail parsing one), but they also have inherited good practices, a UIMA wrapper and tools that will help educate users. Thus, there are mutual benefits to such a collaboration.

However, there may be another, subtler, long-term academic benefit, since those impressive optimizations are a real performance scale change. As a consequence, new experiments and new processing workflows become possible. For example, guessing the language of very small texts can be hazardous: a new method for this could be to use resources for different languages in parallel and to determine the language in function of the results. Thus, a major speed gain can actually open new perspectives for researchers.

Open Source has been proven to be able to produce reliable software. Moreover, it provides an interesting frame for cooperation with the industry, through the use of scientific software. Open Source does not necessarily conflict with trade secret and promotes idea exchanges. It should be always considered as a valuable option when starting a software project.

[4] `http://incubator.apache.org/uima`

References

Frederick P. Brooks, Jr. 1995. *The mythical man-month (anniversary ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Henry Etzkowitz, Andrew Webster, and Peter Healey, editors. 1998. *Capitalizing knowledge: New Intersections of Industry and Academia*. State university of New York Press, Albany.

Justin P. Johnson. 2006. Collaboration, peer review and open source software. *Information Economics and Policy*, 18(4):477 – 497.

Samuel Lee, Nina Moisa, and Marco Weiß. 2003. Open source as a signalling device - an economic analysis. Working Paper Series: Finance and Accounting 102, Department of Finance, Goethe University Frankfurt am Main, March.

Alan F. Newell, Anna Dickinson, Mick J. Smith, and Peter Gregor. 2006. Designing a portal for older users: A case study of an industrial/academic collaboration. *ACM Trans. Comput.-Hum. Interact.*, 13(3):347–375.

Yoshiko Okubo and Cecilia Sjoberg. 2000. The changing pattern of industrial scientific research collaboration in sweden. *Research Policy*, 29(1):81 – 98.

Sébastien Paumier. 2009. Why academic software should be Open Source. *INFOtheca: Journal of informatics and librarianship*, X(1-2):51–54, June.

Ted Pedersen. 2008. Empiricism is not a matter of faith. *Comput. Linguist.*, 34(3):465–470.

Eric S. Raymond. 2001. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc., Sebastopol, CA, USA. Foreword By-Young, Bob.

Appendix A - About the companies

Ergonotics is a software company located in northern France. The company was started by François Liger, Anastasia Yannacopoulou and Gilles Vollant with a simple goal: build and market intelligent software that adapts to its user rather than the usual other way around. The note taking application will be available on both MacBook and iPhone.

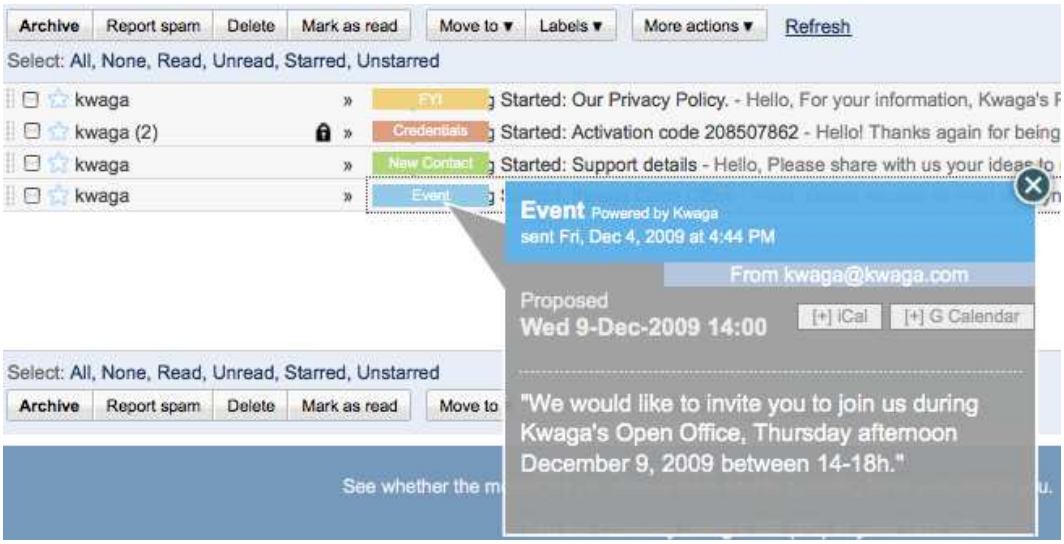

Figure 2: Screenshot of Kwaga's application

Kwaga (`http://www.kwaga.com`) is a company that provides a semantic-based solution for email users to help them focus on essential emails and information, and follow-through with key decisions in a few clicks. This solution comes with a plug-in for Firefox working with the user's Gmail account. The plug-in adds a lightweight display layer over the native Gmail interface (see Figure 2). It displays the information extracted and inferred by the Kwaga server in such a way that it is easier for the user to keep track of relevant information, meeting and actions.

Kwaga's technology relies on a server-side processing chain combining, through the UIMA framework, a linguistic analysis module based on Unitex and local grammars for French and English (note: Spanish and Chinese being the next steps), and various inference modules using Artificial Intelligence techniques to combine linguistic and structured information to extract the relevant information from the user's emails.

Kwaga benefits from collaborations with industrial partner Ergonotics and universities such as Université Paris-Est and provide use cases adapted to the context of emails processing (several hundred thousands mails processed each day), encapsulate Unitex into UIMA and develop parallel algorithms for massive parallel linguistic processing.