FreeLing 3.0: Towards Wider Multilinguality

Lluís Padró
Software Department – TALP Research Center
Universitat Politècnica de Catalunya
padro@lsi.upc.edu

Evgeny Stanilovsky
FreeLing project developer
stanilovsky@gmail.com

Abstract

FreeLing is an open-source multilingual language processing library providing a wide range of analyzers for several languages. It offers text processing and language annotation facilities to NLP application developers, lowering the cost of building those applications. FreeLing is customizable, extensible, and has a strong orientation to real-world applications in terms of speed and robustness. Developers can use the default linguistic resources (dictionaries, lexicons, grammars, etc.), extend/adapt them to specific domains, or –since the library is open source– develop new ones for specific languages or special application needs. This paper describes the general architecture of the library, presents the major changes and improvements included in FreeLing version 3.0, and summarizes some relevant industrial projects in which it has been used.

Keywords: Language analyzers, Open source, Industrial applications.

1. Introduction

FreeLing[1] is an open-source multilingual language processing library providing a wide range of analysis functionalities for several languages.

The FreeLing project was undertaken at the TALP research center[2] to provide advances towards general availability of basic NLP tools and resources. That availability should enable faster advances in research projects and lower costs in industrial development of NLP applications.

The project is conceived as a library that can be called from a user application needing analysis services. The software is distributed as open-source under a GNU General Public License[3], and dual-licensed to companies that embed it in their commercial products.

The open-source approach has been very fruitful during the eight years of life of the project (first version was released on 2003). Version 2.2 has been downloaded over 88,000 times since its release in September 2010, and the source for the alpha release for version 3.0 has been downloaded 530 times in the first five weeks it has been published. Direct SVN downloads of development versions are not registered, so this figure is probably higher.

This wide user community has extended the initial three languages (English, Spanish and Catalan) to nine (adding Galician, Italian, Welsh, Portuguese, Asturian, and Russian) plus the diachronic variant of Spanish from 12th to 16th centuries (Sánchez-Marco et al., 2011). Also, the open-source nature of the project and its modular architecture have made it possible to include code from other similar projects, such as the Word Sense Disambiguator module based on UKB (Agirre and Soroa, 2009).

Current version of the package supports –to different extents– the following languages: Asturian, Catalan, English, Galician, Italian, Portuguese, Russian, and Spanish. Functionalities covered for each languages can be found in Table 1.

[1] http://nlp.lsi.upc.edu/freeling
[2] http://www.talp.upc.edu
[3] http://www.gnu.org/copyleft/gpl.html

Section 2. describes the main modules and services in FreeLing. Next, the most relevant novelties in version 3.0 are presented in section 3., and section 4. summarizes some of the industrial projects the library has been used in. Finally, we sketch some conclusion and future work lines.

2. Data Structures and Linguistic Analysis Services

FreeLing is conceived as a library on top of which powerful NLP applications can be developed, and oriented to ease the integration of language analysis services into higher level applications.

Its architecture consists of a simple two-layer client-server approach: A basic linguistic service layer which provides analysis services (morphological analysis, tagging, parsing, ...), and an application layer which, acting as a client, requests the desired services from the analyzers, and uses the result according to the application goal.

The internal architecture of the system is based on two kinds of objects: linguistic data objects and processing objects.

2.1. Linguistic Data Classes

The basic classes in the library are used to contain linguistic data (such as words, PoS tags, sentences, parse trees, documents...) resulting from performed analysis. Any client application must be aware of those classes in order to be able to provide to each processing module the right data, and to correctly interpret the module results.

The linguistic classes supported by the current version are:

- `analysis`: A tuple <lemma, PoS tag, probability, sense list>.
- `word`: A word form with a list of possible `analysis` objects.
- `sentence`: A list of `word` objects known to be a complete sentence, it may include also a parse tree and/or a dependency tree.

	as	ca	cy	en	es	gl	it	pt	ru
Tokenization	X	X	X	X	X	X	X	X	X
Sentence splitting	X	X	X	X	X	X	X	X	X
Number detection		X		X	X	X	X	X	X
Date detection		X		X	X	X		X	X
Morphological dictionary	X	X	X	X	X	X	X	X	X
Affix rules	X	X	X	X	X	X	X	X	
Multiword detection	X	X	X	X	X	X	X	X	
Basic named entity detection	X	X	X	X	X	X	X	X	X
B-I-O named entity detection				X	X	X		X	
Named Entity Classification				X	X	X		X	
Quantity detection		X		X	X	X		X	X
PoS tagging	X	X	X	X	X	X	X	X	X
Phonetic encoding				X	X				
WN sense annotation		X		X	X				
UKB sense disambiguation		X		X	X				
Shallow parsing	X	X		X	X	X		X	
Full/dependency parsing	X	X		X	X	X			
Co-reference resolution					X				

Table 1: Analysis services available for each language.

- `paragraph`: A list of `sentence` known to be an independent paragraph.
- `document`: A list of `paragraph` that form a complete document. It may contain also co-reference information about the entity mentions in the document.

Figure 1 presents a UML diagram with the linguistic data classes.

2.2. Processing Classes

Apart from classes containing linguistic data, the library provides classes able to transform them, usually enriching the structures with additional information. Figure 3 shows an UML diagram with the processing classes described below:

Most processing classes are derived from an abstract class `processor`, and offer the same API (depicted in Figure 2). The API allows to analyze a sentence or a list of sentences, and obtain the results either as enrichment of the original data, or as a copy:

Only three modules do not follow this pattern: Those that are used early in the analysis chain and deal with plain text and lists of tokens before they are organized into sentences:

- `lang_ident`: Language identifier. Receives plain text and returns a sorted list of pairs <language,probability>.
- `tokenizer`: Receives plain text and returns a list of `word` objects.
- `splitter`: Receives a list of `word` objects and returns a list of `sentence` objects.

All other modules are classes derived from `processor`, and offer the above described API:

- `morfo`: Receives a list of `sentence` and morphologically annotates each `word` of each sentence in the list. This class is actually a meta-module that applies a cascade of specialized processors (number detection, date/-time detection, multi-word detection, dictionary search,

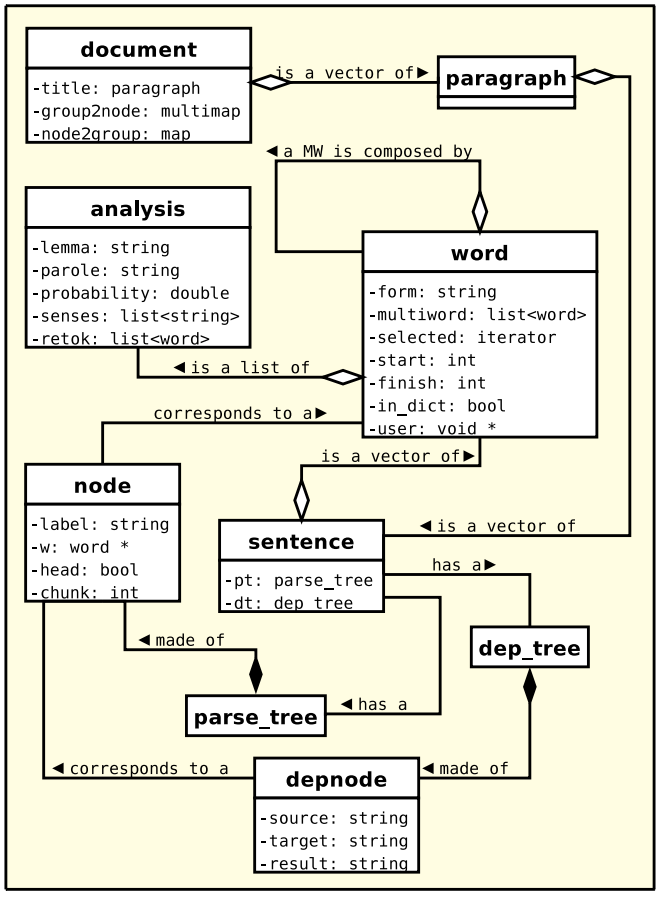

Figure 1: Linguistic Data Classes in FreeLing-3.0.

etc.) each of which is in turn a `processor` derived class that can be called independently if needed. Available processors are:

– `user_map`: User-defined regular expression matcher, that allows the direct assignation of pairs lemma/PoS to words matching certain patterns

```
class processor {
  public:
    /// constructor
    processor();
    /// destructor
    virtual ~processor() {};

    /// analyze and enrich sentence
    virtual void analyze(sentence &)=0;
    /// analyze and enrich sentences in list
    void analyze(std::list<sentence> &);
    /// analyze sentence, return enriched copy
    sentence analyze(const sentence &);
    /// analyze sentences in list, return enriched copy
    std::list<sentence> analyze(const std::list<sentence> &);
};
```

Figure 2: API for `processor` abstract class.

  - `locutions`: Multiword recognizer.
  - `dictionary`: Dictionary look-up and suffix handling.
  - `numbers`: Numerical expressions recognizer.
  - `dates`: Date/time expressions recognizer.
  - `quantities`: Recognized for ratio and percentage expressions, physical magnitudes, and monetary amounts.
  - `punts`: Punctuation symbol annotator.
  - `probabilities`: Lexical probabilities annotator and unknown word handler.
  - `ner`: Proper noun recognizer. Two modules are provided for this task. A fast and simple pattern–matching module based on capitalization (which yields an accuracy near 90%), and a NE recognizer based on the CoNLL-2002 shared task winning system (Carreras et al., 2002), rather slower, but with an accuracy over 94%.

- `tagger`: Receives a list of `sentence` and disambiguates the PoS of each `word` in the given sentences. If the selected analysis carries retokenization information, the word may be split in two or more new words (e.g. *del* → *de+el* in Spanish, or *isn't* → *is+not* in English). FreeLing offers two PoS taggers with state-of-the-art accuracy (97%-98%): One HMM–based tagger implemented following (Brants, 2000) and another based on relaxation labelling (Padró, 1998) which enables the combination of statistical and hand-written rules.
- NE `classifier`: Receives a list of `sentence` and classifies all `word` tagged as proper nouns in the given sentences. This module is based on the CoNLL-2002 shared task winning system (Carreras et al., 2002).
- `Sense annotator`: Receives a list of `sentence` and adds synset information to the selected `analysis` for each `word`.
- `Word sense disambiguator`: Receives a list of `sentence` and ranks the possible senses for for each `word` selected `analysis` in the given context. This module is a direct inclusion of the UKB project code (Agirre and Soroa, 2009).
- `chunk parser`: Receives a list of `sentence` and enriches each of them with a `parse_tree`. This module consists of a chart parser, and is a reimplementation of (Atserias and Rodríguez, 1998).
- `dependency parser`: Receives a list of parsed `sentence` and enriches each of them with a `dependency_tree`. This module uses a set of hand–written rules to build a dependency tree: First, completion rules are applied to transform the output of the chunk parser into a full parse tree. Then, the tree is converted to dependencies, and the functions are annotated. This module is an extension of that described in (Atserias et al., 2005).
- `co-reference solver`: Receives a document formed by parsed `sentence` and enriches it with co-reference information. This module is based on the system proposed by (Soon et al., 2001).

3. What's new in FreeLing 3.0

Version 3.0 presents some major changes that aim to make the tool more flexible, usable, and to ease its installation. These changes can be grouped in three main types: Changes related to multilingual support and extension, changes to the Machine-Learning based components of the library, and changes related to the engineering aspects of the project.

3.1. Extending Multilingual Support

The first relevant contribution to extending the coverage of FreeLing –with regard to the number and variety of languages it can process– is the development of linguistic data for morphological analyzer and PoS tagger for Spanish of 12th to 16th centuries (Sánchez-Marco et al., 2011). This work uses the default data for Spanish with appropriate adaptations and extensions to process the orthographic variations present in ancient Spanish. Also, a corpus to train the tagger has been developed, and the resulting tool was successfully used in a linguistic study about the evolution of the use of the verb *haber* (*to have*) (Sánchez-Marco and Evert, 2011; Sánchez-Marco, 2012)

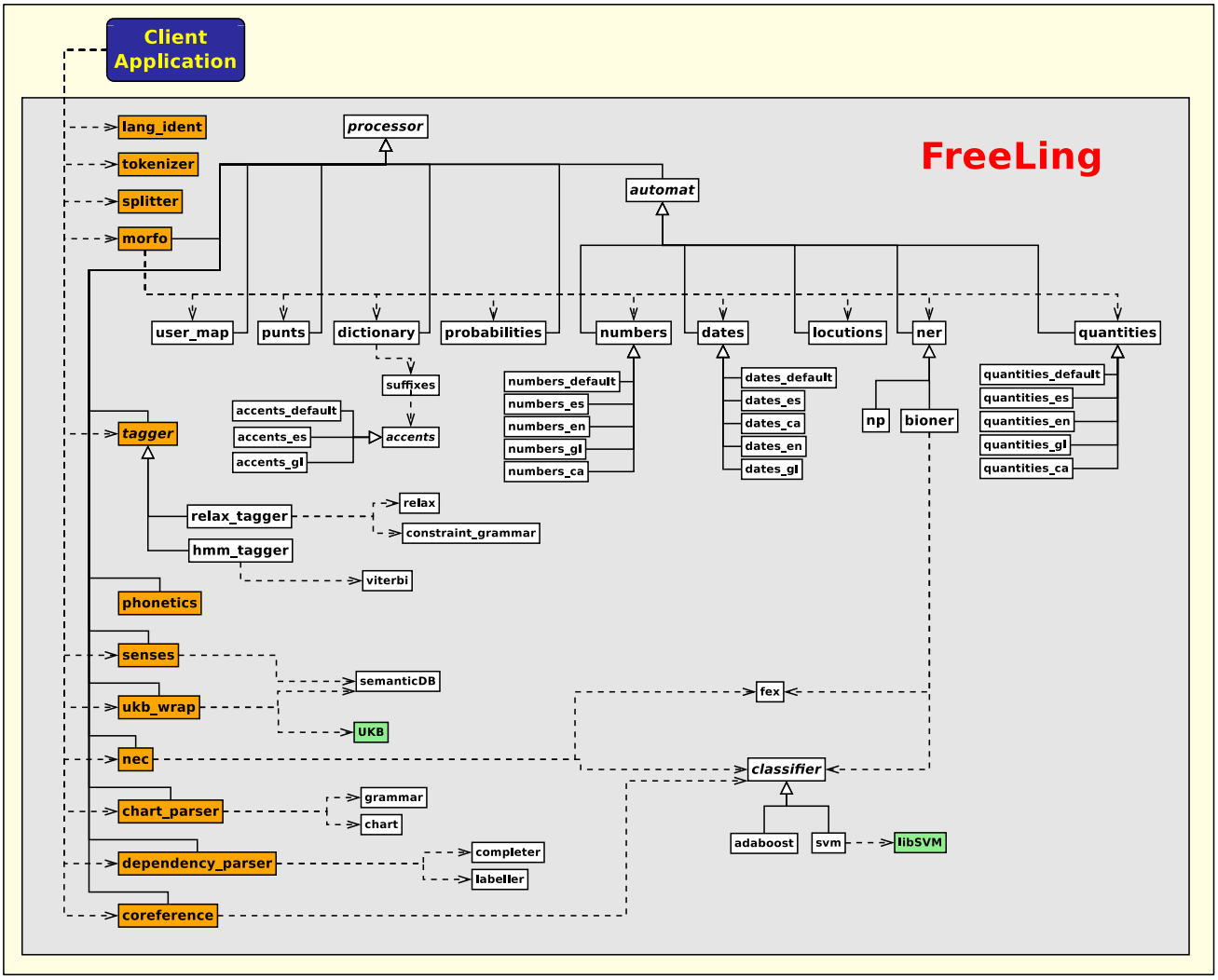

Figure 3: Processing classes in FreeLing-3.0.

Another modifications in FreeLing version 3.0 is the full support of Unicode (UTF8) character encodings. This is a major change, and one of the main reasons for the change of version number.

Previous FreeLing versions supported multiple languages, but the developer of the linguistic data for each language should take care of which encoding should be used, and of maintaining the consistency of the encoding for different resource files (lexicon, grammars, multiword lists, etc.) and configuration rules (e.g. regular expressions in tokenizer rules) according to the selected encoding. Also, since each language might use a different encoding, it was not easy to integrate a language identifier able to handle texts in different alphabets.

With the full support of UTF8 encodings, the same application may deal with texts in different languages and alphabets. This extension enabled several functionality improvements:

- A new module for language identification based on (Padró and Padró, 2004) has been integrated in the library.
- Ability to change the application locale to match the language of the processed text, even if it is different from the default system locale configuration.
- The regular expressions used either in configuration files or hardwired in the code are now more expressive and easy since POSIX extensions can be used. For instance, the tokenizer regular expression to match a word made of alphabetical characters in Spanish used to be `[A-Za-záéíïóúüñÁÉÍÏÓÚÜÑ]+`, and now can be written as `[[:alpha:]]+`. Note that this same expression can be used to match an alphabetical word in any language or alphabet, just changing the current locale of the application.

The use of UTF8 encoding cleared the path for developers interested in adding support for languages with non-latin alphabets. This is the case of Russian developers, who achieved a very complete morphological analyzer and a competitive PoS tagger that wouldn't have been possible in previous versions. Additionally, not only linguistic data has been provided for Russian, but also actual code for language-dependent modules such as number or date recognition.

3.2. Improving Machine-Learning Based Modules

Another major change in FreeLing 3.0 architecture is the organization and content of the Machine Learning modules: The feature extraction engine, and the learning/classification algorithms themselves.

In previous versions, those functionalities were provided by two libraries external to FreeLing: *Omlet&Fries*[4]. In version 3.0 the code for *Omlet* library is included in the FreeLing package, which provides a clearer code organization, and a reduction in the number of dependencies easing installation process.

The feature extraction module that used to be provided by *Fries* library has been completely rewritten, providing a clearer formalism for extraction rules and a more flexible API for those developers wishing to include their own feature functions. Also, the repertoire of available ML algorithms has been extended with Support Vector Machines (SVM), thanks to the open-source project `libSVM` (Chang and Lin, 2011). The `libSVM` code has been integrated with FreeLing under a common wrapper with the other existing classifiers. This integration has the additional advantage of avoiding the addition of a new item to the list of dependencies needed to build and install FreeLing.

Finally, the Named Entity Recognition and Classification modules have been retrained using the new architecture. Machine-learning based NE detection and classification are now available for Spanish, English, Galician, and Portuguese. Both AdaBoost and SVM models are provided for the former two, and just AdaBoost models for the later.

3.3. Technical Modifications

The third type of changes on FreeLing library are technical issues related to the external dependency organization, the porting to platforms other than Linux, and the use of FreeLing in server mode.

3.3.1. External Dependencies

One important aspect of these engineering modifications is the management of the external library dependencies required by FreeLing. This is a main reason –together with the change to Unicode encodings described above– for the major version number upgrade.

An important effort has been made in the direction of reducing and simplifying the dependency list, in order to ease building and installation of the library, as well as its dual licensing.

The dependencies of the previous versions were:

- BerkeleyDB - For fast access to on-disk dictionary files.
- PCRE - Regular Expression management.
- libcfg+ - Option management for main program `analyzer`.
- Omlet&Fries - Machine Learning modules (discussed above).

BerkeleyDB is no longer used: Dictionaries are loaded into RAM in either prefix trees or STL map structures. Time performance is roughly the same, and the increase in memory consumption does not pose any problem to a modern machine. Moreover, the greater simplicity in installation (less dependencies, no dictionary indexing needed at installation time), and in dictionary management (no reindexing needed after modifying a dictionary) pays off this small cost.

PCRE and libcfg+ are no longer used either: The functionalities related to regular expressions and configuration options have been transferred to *boost C++* libraries[5]. This has two advantages: Dependencies are unified under a single provider, and installation is easier since `libboost` is part of all Linux distributions.

3.3.2. MS-Windows Compilation

Using FreeLing under MS-Windows used to be a difficult enterprise. Emulators or cross-compilers such as MinGW[6] or CygWin[7] had to be used, and the obtained results were not always easily integrable in a MS-Windows application. All C++ code in version 3.0 has been adapted to be compiled by MS-Visual C++. Project files are provided for that environment, greatly simplifying the building and use of FreeLing in MS-Windows, since obtained binaries are native libraries for that system.

3.3.3. Improved Server Mode

Finally, a minor technical improvement is the multiclient ability that the `analyzer` demonstration program presents in the new version.

In previous versions, the server was conceived as a mean to avoid repeatedly initializing the analysis modules if many small files had to be processed. For that, all client requests were served sequentially, with the consequent inefficiency if many clients were to be used simultaneously.

In the new version, the code follows a standard Linux server architecture: A *dispatcher* process waits for client requests listening in a socket. When a client connection is established, the dispatcher forks a new *worker* process that will take care of the client, while the dispatcher returns to listen for new incoming requests.

This makes it possible to use FreeLing in parallel processing (e.g. to process huge amounts of text in a multiprocessor machine, or to use it as a server in a multiuser web application), which was not possible in the previous versions. The server mode in previous versions was devised only as a mean to avoid repeated start-ups if many small files had to be processed, and all client request were attended sequentially by the same server.

Nevertheless, the new server doesn't currently limit the maximum number of connected clients, nor has a queue for waiting requests. Thus, applications with a potentially massive amount of clients should adapt the server code to safely handle a pending request queue.

3.4. Other improvements

Other interesting improvement that can be found in FreeLing 3.0 are the following:

[4] `http://nlp.lsi.upc.edu/omlet+fries`

[5] `http://www.boost.org`

[6] `http://www.mingw.org`

[7] `http://www.cygwin.com`

- *UserMap*: A new module that enables the user to define a set of regular expressions and assign to each of them a pair <lemma,PoS> to be assigned to words/tokens matching the pattern. The goal of this module is to ease the application developer the specific treatment of cases not covered by the other library modules. For instance, an application processing *Twitter* posts could require annotating as proper nouns the tokens matching the pattern `@name`. Instead of re-training the NE recognizer, one can simply add an ad-hoc rule:
  ```
  @[a-z][a-z0-9]* $$ NP00000
  ```
  that will detect those tokens and assign them their own form as lemma, plus `NP00000` as PoS.

- *Forbidden trigrams*: The HMM-based tagger performs smoothing on transition probabilities, in order to accept unobserved combinations. This smoothing allocates some probability mass for any unobserved trigram, even for linguistically impossible ones (e.g. as a determiner followed by a finite verb form, or an Spanish auxiliary *have* followed by something other than a participle). such cases may be explicitly listed in the tagger model file, excluding them from the smoothing process and forcing their probability to be zero, therefore reducing the error rate of the tagger.

- *General* NER *module*: Named Entity Recognition modules have been wrapped under a factory class `ner`. Depending on the options specified in the configuration file, either a basic `np` recognizer or a ML–based `bioner` module is instantiated, transparently for the calling application-

- *Flexible Semantic Database*: Class `semanticDB` handles access to sense repository for any module requiring access to such data (e.g. `senses` and `ukb_wrap`). This class is now capable of accepting sense mapping rules that enable to bridge between criteria and PoS tags used in morphological dictionary and those used in the sense repository. For instance, one can state the following configuration:

  ```
  <WNposMap>
  N n L
  J a L
  R r L
  V v L
  VBG a F
  </WNposMap>
  ```

  This will cause that for words with PoS starting with `N` the lemma (`L`) is searched in the sense repository (WordNet in this case) with PoS `n`. For words with PoS starting with `J` the lemma is searched with PoS `a`, etc. The last rule states that for words with PoS `VBG`, their form (`F`) is also looked up with PoS `a`.

  With this last rule we can achieve that a word like *falling*, which does not appear in the morphological dictionary as adjective, but it does in WordNet, is assigned the existing adjective senses in addition to those corresponding to verb *to fall*.

- *Phonetics*: Another new service in FreeLing is the phonetic codification of a word. This module uses a transcription rule file that translate the text to its encoding in SAMPA *de-facto* standard notation[8]. It is also able to use a whole-word transcription dictionary for exceptions o for languages –such as English or Russian– with a highly irregular ortography.

4. Some projects using FreeLing

Alpha releases for FreeLing 3.0 are available in the project web page. The latest development version is also downloadable from project SVN. Version 3.0 has already been used in several industrial projects, of which we briefly summarize the most relevant:

- Ruby Reader: iPhone app that helps Japanese speakers to understand English texts. Developed by CA Mobile (`http://www.camobile.com`).
- Vi-Clone: Impressive virtual assistants for corporate web pages. Some FreeLing components are being integrated in the dialog system. Vi-Clone is funding the development of the spell correction module which will enable FreeLing to process user dialog utterances in non-standard writing. `http://www.vi-clone.com`.
- TextToSign: Translator from Spanish text to sign language, which uses FreeLing for text processing. `http://www.textosign.es`.
- Dixio: Intelligent dictionary able to help the reader of a text providing contextualized definitions. Developed by Semantix (`http://www.semantix.com`).
- Aport News: News portal using FreeLing as a preprocessor to enrich Russian text. The annotations are used in news classification and clustering tasks (`http://news.aport.ru`).

5. Conclusions and Further Work

We have presented the main improvement and changes undertaken in FreeLing version 3.0, and some industrial projects it has been used in.

Thanks to this changes, and to the active community around this project, we expect to continue improving the number of supported languages, the provided functionalities, and the usability of these analyzers in industrial NLP applications.

One of the most interest-awakening work lines is the inclusion of a spelling correction module, which –as a part of a robust analysis chain– will constitute a keystone for the development of applications targeting non-standard texts such as Internet chats, forums, micro-blogs, etc.

Other engineering–related improvement could be the development of thread–based processors, so each module can be run in a different thread, thus taking advantage of the parallel processing capabilities of modern computers.

Finally, we are considering using FOMA[9] finite–state engine to replace date, number, and quantities recognition modules, which are the only language dependent C++ code in FreeLing. This step would mean the complete independence of FreeLing code and linguistic data, and ease the

[8] `http://www.phon.ucl.ac.uk/home/sampa`
[9] `http://code.google.com/p/foma/`

development of these kind of modules both for existing and new languages.

Acknowledgments

This work has been partially funded by the Spanish Government through projects KNOW-2 (TIN2009-14715-C04-03/04) and OpenMT-2 (TIN2009-14675-C03-01), and by the European Union through projects FAUST (FP7-ICT-2009-4) and X-LIKE (FP7-ICT-2011-288342). We also thank ViClone (footnotesize`www.vi-clone.com`) for funding part of FreeLing development.

6. References

Eneko Agirre and Aitor Soroa. 2009. Personalizing pagerank for word sense disambiguation. In *Proceedings of the 12th conference of the European chapter of the Association for Computational Linguistics (EACL-2009)*, Athens, Greece.

Jordi Atserias and Horacio Rodríguez. 1998. Tacat: Tagged corpus analizer tool. Technical report lsi-98-2-t, Departament de LSI. Universitat Politècnica de Catalunya.

Jordi Atserias, Elisabet Comelles, and Aingeru Mayor. 2005. Txala un analizador libre de dependencias para el castellano. *Procesamiento del Lenguaje Natural*, (35):455–456, September.

Thorsten Brants. 2000. Tnt - a statistical part- of-speech tagger. In *Proceedings of the 6th Conference on Applied Natural Language Processing, ANLP*. ACL.

Xavier Carreras, Lluís Màrquez, and Lluís Padró. 2002. Named entity extraction using adaboost. In *Proceedings of CoNLL Shared Task*, pages 167–170, Taipei, Taiwan.

C.C. Chang and C.J. Lin. 2011. Libsvm : a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, April.

Muntsa Padró and Lluís Padró. 2004. Comparing methods for language identification. *Procesamiento del Lenguaje Natural*, (33):155–162, September.

Lluís Padró. 1998. *A Hybrid Environment for Syntax–Semantic Tagging*. Ph.D. thesis, Dep. Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya, February. http://www.lsi.upc.es/˜padro.

W.M. Soon, H. T. Ng, and D.C.Y. Lim. 2001. A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521–544.

Cristina Sánchez-Marco and Stefan Evert. 2011. Measuring semantic change: The case of spanish participial constructions. In *Proceedings of 4th Conference on Quantitative Investigations in Theoretical Linguistics (QITL-4)*, Berlin, Germany, March.

Cristina Sánchez-Marco, Gemma Boleda, and Lluís Padró. 2011. Extending the tool, or how to annotate historical language varieties. In *Proceedings of the 5th ACL-HLT Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, pages 1–9, Portland, OR, USA, June. Association for Computational Linguistics.

Cristina Sánchez-Marco. 2012. *Tracing the development of Spanish participial constructions: An empirical study of language change*. Ph.D. thesis, Universitat Pompeu Fabra, Barcelona, Spain. (forthcoming).