A Workbench for Developing Natural Language Processing Tools

Carlos Amaral, Helena Figueira, Afonso Mendes, Pedro Mendes, Cláudia Pinto
e-mail:{cma, hgf, amm, prm, cp}@priberam.pt
Priberam Informática
Av. Defensores de Chaves, 32 – 3º Esq.
1000-119 Lisboa
Tel.: +351 21 781 72 60
Fax: +351 21 781 72 79

Abstract
This paper presents a workbench built by Priberam Informática for the development of the company's natural language processing technology. This workbench includes a set of linguistic resources and software tools that have been applied in a considerable number of practical purposes, covering proofing tools, text processing and information retrieval.

1 Introduction

The primary purpose of the workbench built by Priberam Informática was to assist a multidisciplinary team of computer engineers and linguists in the development of the company's natural language processing (NLP) technology. It was designed as a flexible modular system. Presently, several of its components have been integrated individually in other systems, products and services.
This article intends to provide an insight on how the workbench's resources and tools were developed. In the next section, we describe in detail the various resources used and their role. In section 3, we provide an overview of the software tools that interact with those resources and, finally, in section 4, we present and discuss some of the main applications, present and future.

2 Linguistic Resources

The linguistic resources of this workbench comprise a lexicon, an ontology and a grammar. In this section, we will look at these resources in more detail. We will use European Portuguese examples whenever relevant.

2.1 Lexicon

The lexicon, a major component of any NLP system, had its starting point in a monolingual dictionary[1], which was automatically converted into a classical relational database (Access). The traditional lexicographic model was upgraded to a more flexible computational lexicon, capable of providing machine understandable word knowledge, where structurally related fields declaratively list information, predetermined and encoded by linguists.
The standard entry of a given lexical unit (LU) α can be represented as (1):

```
(1)  <LU  ID="No." SPELL="α">
         <Cats>
             <CAT POS="Pos" FEAT=" PosFeat">
                 <Morph ID="No." FLEX="FlexCode" DER="DerCode"/>
                 <SynSem ID="No." SENSE="SenseNo" DEF="SenseGloss" SEM="SemFeat" SYN="SynFeat">
                     < Onts>
                         <ONT ID="OntCode" TERM="TermCode" TRANSL1="Eng" TRANSL2="Fr"/>
                     </Onts>
                     <Rels>
                         <REL SentRelID="No." Word="spelling" RelType="LexRel"/>
                     </Rels>
                 </SynSem>
             </CAT>
         </Cats>
     </LU>
```

Under SPELL we encode the spelling of the canonical forms of simple and compound word forms (either agglutinated or hyphenated) of nouns and proper names, verbs, adjectives, adverbs, functional words, etc.

[1] As a result of a partnership with Porto Editora publishing house.

Under CAT, we group linguistic information, Morph and SynSem. We encode morphological information (Morph), which varies according to each category, following pre-established inflectional and derivational paradigms. Verb inflections, nominal gender and plural inflections, nominalizations and diminutive derivations, for instance, are generated by the application of code rules in FLEX and DER. Derivations by prefixation are not encoded, since they are generated heuristically.
We also encode syntactic and semantic information (SynSem), which includes a sense index (SENSE), a gloss (DEF), semantic features (SEM), subcategorization and selection restrictions (SYN), ontology domains (ONT) (see section 2.2), terminological domains (TERM), English and French equivalents (TRANSL1 and TRANSL2) (see section 2.2) and lexical-semantic relations (REL) with other words (e.g., synonymy, antonymy, etc.). The following are abbreviated entries of the Portuguese words *porco* and *revistar*:

(2)
```
<LU  ID="50681" SPELL="porco">
     <Cats>
         <CAT POS="N" FEAT="MASC,SING">
             <Morph ID="69" FLEX="5" DER="701"/>
             <SynSem ID="70390" SENSE="s1" DEF="animal" SEM="ANIMAL" SYN="0">
                 < Onts>
                       <ONT ID="26.3.2.9" TERM="78" TRANSL1= "pig" TRANSL2= "cochon"/>
                 </Onts>
                 <Rels>
                       <REL SentRelID="60790 Word=””suíno" RelType="synon"/>
                 </Rels>
             </SynSem>
         </CAT>
         <CAT POS="A" FEAT="MASC,SING">
             <Morph ID="69" FLEX="5" DER="701"/>
             <SynSem ID="51170" SENSE="s1" DEF="sujo, imundo" SEM="CONCR" SYN="0">
                 < Onts>
                       <ONT ID=" 28.1.2.6" TERM="79" TRANSL1= "dirty" TRANSL2= "cochon"/>
                 </Onts>
                 <Rels>
                       <REL SentRelID="51159 "Word="porcalhão" RelType="synon"/>
                 </Rels>
             </SynSem>
         </CAT>
     </Cats>
</LU>
```

(3)
```
<LU  ID="55883" SPELL="revistar">
    <Cats>
        <CAT POS="V" FEAT="TR">
            <Morph ID="6769" FLEX="1" DER="0"/>
            <SynSem ID="56441" SENSE="s1" DEF="passar em revista"
                                              SEM="{SUJERG|SUJHUM}, {ODHUM| ODGHUM}"
                                              SYN="SUJS,OD,NPRED, HAN, SERN ">
                < Onts>
                    <ONT ID="2.1.4.10" TERM="78" TRANSL1="review" TRANSL2="passer en revue"/>
                </Onts>
                <Rels>
                    <REL SentRelID="36867" Word=”inspeccionar” RelType="synon"/>
                </Rels>
            </SynSem>
        </CAT>
    </Cats>
</LU>
```

The semantic information encoded so far in the lexicon covers a brief definition and semantic features that distinguishes each sense. The syntactic information covers the number and the semantic nature of arguments, syntactic categories of their realizations (NP, PP, etc.), syntactic functions (subject, object, etc.) and lexical constraints on argument realization (preposition heading the PP that follows a given noun, verb, etc.).

Covering a wide range of vocabulary, from general to terminological domains, the role of the lexicon is significant in FLiP, *Ferramentas para a Língua Portuguesa*, Priberam's proofing tools package for Portuguese[2]. The spell checker, for instance, uses the spelling and inflection information to accept or correct word forms. In order to keep track of the actual use of language, the lexicon database is regularly updated, mainly by large corpora search, but also by screening custom dictionaries uploaded by FLiP users. The morphological analyser uses the spelling and the morphological information for POS tagging. Finally, the thesaurus of FLiP, a source of lexical enrichment, uses the lexical-semantic relations to present synonyms (in the case of the European Portuguese FLiP) and synonyms and antonyms (in the case of the Brazilian Portuguese FLiP). Synonyms are also used in query expansion for information retrieval (see next section).

2.2 Ontology

Another major component of this workbench is the ontology[3], which is also implemented in an Access database. It was initially designed by Synapse Développement, the French partner of TRUST[4], *Text Retrieval Using Semantic Technologies*, an EU co-financed project[5]. The aim of this project was the development of a semantic and multilingual search engine capable of processing natural language (NL) questions in English, French, Italian, Polish and Portuguese.
Our starting point was the translation of the 195 000 English entries of the ontology. We used bilingual (English / Portuguese and French / Portuguese) dictionaries for automatic translation, which covered successfully 25 000 entries. Better results could have been achieved if the dictionaries had a higher number of fixed expressions, which are quite numerous in the ontology. The rest of the translation was done manually and is being perfected and revised, as senses are being encoded in the lexicon.
Briefly, the ontology can be regarded as a lattice whose nodes are the available ontological domains or levels (28 top levels which expand into several sublevels leading to 3387 terminal nodes), and whose entries correspond to the senses of the entries of the lexicon. The organising principle of this ontology consists on building conceptual fields or domains, combining in the same category things that might appear together, such as the actor, the action, the instrument, the environment, the place, the manner of acting, and so on. The principle also takes into account semantic relations as hyperonymy/hyponymy (e.g., since the noun "climbing" is a kind of sport, it is included in the domain *sports*), as well as proximity relations ("climbing" is also included in the domain *vertical movement*).
Each entry in the ontology is a structure {Portuguese word, part-of-speech, sense index, ontological domain, English word, French word} that enables Portuguese-English, English-Portuguese, Portuguese-French and French-Portuguese translations through the ontology domain. The following are examples of the entries for the Portuguese word *porco*:

(4)

{porco, N, 1, 26.3.2.9, pig, cochon}	[animal]
{porco, N, 2, 28.1.2.7, pig, porceau}	[dirty person]
{porco, N, 3, 27.2.2.5, swine, cochon}	[vulgar person]
{porco, N, 4, 22.2.1.14, pork, porc}	[meat]
{porco, A, 1, 28.1.2.6, dirty, cochon}	[dirty]
{porco, A, 2, 7.3.2.9, dirty, cochon}	[obscene]

The connection between entries in the lexicon and the levels of the ontology was partially done using automatic processes. Whenever an entry co-occurred with its synonyms in an ontology sublevel, a relation was established between that sense and the ontological domain.
Terminological domains can assist in semantic disambiguation if used in combination with ontological domains and their relation with each sense of an entry in the lexicon. A word can appear in different technical and non-technical ontological sublevels and domains. The terminology acts as a restriction of this eventual thematic ambiguity.
The ontology was particularly useful for TRUST, where the combination of all of its versions provided a bidirectional word/expression translation mechanism, having English as an intermediate. This enabled to perform tasks in cross-language information retrieval, namely to obtain answers in Portuguese to questions formulated in French, or vice-versa.

[2] FLiP includes a grammar checker, a spell checker, a thesaurus and a hyphenator that enable proofing different levels – word, sentence, paragraph and text – of European and Brazilian Portuguese. An online version is available at www.flip.pt.
[3] As in other cross-linguistic projects [12], ontologies seem to play, more and more often, an important role in NLP systems. This particular ontology was shared by all the members of the TRUST project. In a near future, additional languages will be made available by the original project participants or other companies.
[4] http://www.trustsemantics.com
[5] Cooperative Research (CRAFT) project number IST-1999-56416.

2.3 Grammar

At the beginning of our work with the grammar checker, we faced the problem of joining the vision of programmers and that of linguists in a tool that could be used by both. With this in mind, our first goal was to define a descriptive language that would permit the linguists to describe the grammar in such a way that the software could handle it.

Features, Nodes and Productions

The language we defined is somewhat similar to the one used by Yacc [8], where the linguists can build the grammar rules. As in any context-free grammar (CFG) formalism [1, 7], one can define terminal nodes, nonterminal nodes and a start node. A terminal or nonterminal node is declared with a set of features. A feature is as set of discrete values that can be applied to the node. For example, this is a simple definition for a noun:

(5)
```
Node     N                {Number, Gender}
```

And the features Number and Gender can be defined as:

(6)
```
Feature Gender [MASC, FEM]
Feature Number [SING, PLU]
```

This means that the node N can be instantiated like N(SING, MASC) for a singular masculine noun (ex.: *porco* [pig]) or, for instance, N(SING|PLU, MASC) for a noun which is both singular and plural and masculine (ex.: *lápis* [pencil]).
Nonterminal symbols are also defined by a set of rewriting rules, which we call productions. A production is a sequence of nodes (terminal or not) that permits to build the nonterminal node. For instance, we could write a simple rule to describe a noun phrase:

(7)
```
NP : N ADJ        (P1)
   | DET N        (P2)
```

This NP could be rewritten as (P1) or, alternatively, as marked by "|" symbol, as (P2). Then we could use this NP on a rule such as (8):

(8)
```
S : NP VP
```

There is also the possibility to define optional and repetitive nodes.

Feature Inheritance and Unification

Feature inheritance and unification is the method for passing feature values between the nodes of a production and its left node. A simple programming language was defined to perform this task. This language allows comparisons between feature values, conditional statements and attribution of values to features on the left node. Besides the set of feature values defined by the grammar writer, two special values were defined: "confl" and "undef". The first value means that no value is defined for the feature or that, somehow, the nodes contained in the production form an ungrammatical production. The second value means that all the feature values are present. This mechanism provides us with a way to prune overgenerated parse trees and to implement agreement for a grammar checker. Thus, we could write the following rule for the second production of the NP above:

(9)
```
NP  : N ADJ
    {…
    }
    | DET N
    {
        If ($1.Gender != $2.Gender)
        {
            $$.Gender = confl;
        }
        Else
        {
            $$.Gender = $1.Gender;
        }
    }
```

Alternatively, we could simply write:

(10)
```
| DET N
{
        $$.Gender = CommonValue ($1.Gender, $2.Gender);
}
```

In (9) and (10), $$ stands for the node NP on the left, $1 and $2 stand for the DET and N, respectively. Common values between the gender of the two nodes are unified by the function "CommonValue".

Conflict Resolution, Overgenerated Parse Trees and Pruning

Let us assume that in (10) we wanted to analyse the phrase "a carro" [the(FEM) car(MASC)] and that the start node is the NP. In this situation, the phrase would produce the terminal nodes after disambiguation of "a" and we would have a parse tree like (11):

(11) NP(SING, confl) : DET("a", SING,FEM) N("carro", SING,MASC)

For a grammar checker, this would mean that there is a conflict in gender between the nodes DET and N that should be solved by either changing the gender of the DET or that of the N. Since in this case we could not produce a feminine for "carro", then the determiner "a" should be changed to "o", the masculine form. The situation presented above is quite straightforward because there was no overgeneration involved, since (10) is a simple rule. When the grammar becomes more complex and recursion is involved, we can get several representations for the same set of terminal nodes. To overcome this, the following rule was defined: "if for two disjoint possible parse trees there is a production α with a conflict in one of its features, and another with no conflict, then α is removed".

Ambiguous Node Attachment

One of the most frequent reasons for overgenerated trees is the possibility of different attachments between nodes. For instance in the sentence "A mesa de vidro redonda" [The round glass table], with a grammar like (12)[6]:

(12)
```
NP : DET? N ADJ? PP
PP : P NP
```

The parse tree would be:

(13)
```
NP("A mesa de vidro redonda") : DET("a", SING, FEM) N("mesa", SING, FEM) PP ("de vidro redonda")
    PP ("de vidro redonda") : P("de") NP ("vidro redonda", SING, confl)
        NP("vidro redonda", SING, confl) : N("vidro", SING, MASC) ADJ("redonda", SING, FEM)
```

With this analysis, we would get a conflict in the NP("A mesa de vidro redonda") and the grammar checker would wrongly ask the user to change "redonda" to "redondo". That would be wrong because in this sentence the adjective is not in its canonical position and it is not qualifying "vidro", it is qualifying "mesa". In order to overcome this situation, we defined a new type of feature called node features. A node feature contains a node (terminal or not) which can be moved through the parse tree until a suitable attachment is found. In the example above, the first NP would appear as follows:

(14) NP("A mesa de vidro redonda") : DET("a", SING, FEM) N("mesa", SING, FEM) PP ("de vidro", ADJ("redonda", SING, FEM))

With this approach, "redonda" is attached to "a mesa", forming the noun phrase "a mesa redonda", and the conflict is avoided. The example above is quite a simple one and does not take into account semantic features nor others that we might want to implement but many ambiguities could be solved in the same manner.

Structural Restrictions

Structural restrictions apply over nonterminal nodes after the complete analysis of the input. These restrictions are used to eliminate branches of the generated tree or, if the restricted branches are the only ones that cover the

[6] The "?" marks an optional node.

part of the input, to flag a syntactical error in the input. When building a grammar checker, we have to predict the possibility of the input sentence being ill formed and so some productions are allowed to produce ungrammatical trees. In such cases, we can build a structural restriction that will be used to inform the user of the ungrammaticality.
For instance, if inside a VP we allow the possibility of two adjacent gerunds and the only representation that covers the two is a VP, then in the grammar checker a warning is displayed to the user. If, on the other hand, there are alternate representations where this restriction does not apply, the VP and the entire dependent tree is eliminated. In (15) we can see an example of this kind of restriction in (15):

```
(15)    Restriction adjac VP: V (,,,GER) V (,,,GER)
                {
                        msg=VGERDUPLO
                        sug="[$1 e $2]||[$1, $2]"
                }
                ;
```

There are two types of restrictions: adjacency and existence. In the first case (see (15)), the restriction is applied if, at the bottom level of the tree, the terminal nodes occur immediately one after the other. In the second case, the restriction is applied if the terminal nodes on the restriction exist in any order in the input.

Contextual and Proximity Rules

Contextual and proximity rules are used to apply transformations on the input string of terminal nodes. These can be used to perform morphological disambiguation, detection of fixed expressions (dates, adverbial expressions, noun and verbal expressions) and detection of named entities, as will be explained in section 3.1. The following is an example of a disambiguation rule:

```
(16)    Word( OS ) | Word( AS )= PRONDEM : $$ Cat(P)
                        = ART : Start $$
                        …
```

A rule like (16) states that the ambiguous word forms "Os" or "As" (which can be an article, a demonstrative pronoun or a clitic) always behave as a pronoun before a preposition and as an article in the beginning of a sentence. As we can see on the example, the left side of the rule (before the equal sign) is one token or a sequence of tokens of the input string; the right side of the rule (after the equal sign), a terminal node, is the category that is applied if the contextual conditions bellow are satisfied.
The following is an example of a rule to detect multiword expressions, in this case, a fixed verbal expression:

(17) Root(levar) Word(a) Word(sério) : V($1.Number, $1.Pessoa, $1.Tempo, $1.Modo, ….)

Rule (17) creates the expression "levar a sério" [to take seriously] and behaves in the same way as (16), although it states no contextual conditions. It states that all sequences where the first word has for lemma the verb "levar", followed by the words "a" and "sério", form a fixed expression with category V, which inherits its features from the form of the verb "levar". We are now working on some enhancements to allow writing rules like this for sense disambiguation. The rules are similar to those used for morphological disambiguation algorithms, but we will be able to state new conditions, based on ontology domains, and assign sense numbers instead of POS.

The Grammar Compiler and Interpreter

The syntactic structure of the input is analysed by an Earley-like [4] left-corner bottom-up parser, which determines if and how the input can be derived from the rules of the grammar. The algorithms applied to conflict handling and tree pruning were designed internally but its description is out of the scope of this paper. The approach to implement the contextual rules is somehow related to the one described in [3] but the implementation and the coverage are broader.
The grammar language described in this section is not dependent on our lexicon and can be used with any morphological analyser, as long as the input respects a set of structures defined to describe the tokens, categories and features of the input string. The whole system is coded in C++ and was originally written in UNIX. Later it was ported to Microsoft Windows and recently some parts of the system have been ported to MAC OS X.
The language described in this section was used for developing European and Brazilian Portuguese grammars, which are being used for FLiP's grammar checkers and also as part of the pipeline of the TRUST Portuguese language module.

3 Software Tools

The workbench also contains a group of software tools that have been developed for helping in NLP tasks. They include tools for information extraction from corpora (in order to automatically extract proper names, neologisms, collocations, named entities, word frequency, fixed expressions, etc.) and tools for editing and testing grammars. Besides the tools described in this section, which are mainly for helping the linguistic development, the workbench also contains a set of compilers to produce optimized release versions suitable for a given tool or product.

3.1 Corpora tools

One of the main tasks when dealing with computational linguistics is to extract information from corpora. As part of our workbench, we use tools available on the market and public domain but we have also designed a set of tools to assist some specific needs. Among others, we developed:

- Word frequency extractor
- Named entity extractor
- Collocations extractor
- Automatic POS tagger and training tool
- Manual POS tagger
- Rule builder for morphological disambiguation
- Concordance extractor with configurable statistics

Other tools are being implemented and tested with the intention of assisting the task of sense disambiguation. We will now give a little insight on the implementation of some of the tools.

Named Entity Extractor

The named entity extractor was built using the contextual rules described in section 2.3. The extractor is capable of detecting and tagging a large amount of named entities (NE). The NE tagger tries to find a sequence of two or more proper nouns, recognizing it as a single token and classifies the NE thus created according to some criteria, namely the POS established in our lexicon for each element of the entity (e.g., *Luís Vaz de Camões* will be classified as an anthroponym). It also uses groups of conceptually gathered words that will help in the classification of NE: for instance, a sequence of proper nouns preceded by a common noun such as *rio* [river] will be classified as a toponym (e.g., *rio de São Domingos*). Sometimes, for purposes of semantic disambiguation, it also considers the context, checking what words precede or follow the NE. The NE extractor is used at runtime in tools like the grammar checker, TRUST's Portuguese language module or the stemmer.

Collocations Extractor

The collocations extractor was based on the statistical algorithm described in [10]. Our collocation extractor is capable of extracting collocations between words, lemmas and POS. The tool reads a set of corpora files and produces output for every word that presents systematic behaviours given a set of parameters as defined in [10]. We have found this tool very useful, for instance, for extracting verbal subcategorization and general or terminological fixed expressions. The rules thus acquired are inserted into the contextual rules of the grammar and then edited and tested by linguists, proving to be an efficient way of enriching both the lexicon and the grammar.

Rule Builder for Morphological Disambiguation

The rule builder for POS disambiguation was developed based on the collocations extractor. The purpose is that given a specific ambiguity, for instance N/V, the rule builder would produce a set of contextual rule conditions that would be applied in the order of the strength found by the collocations extractor. The big advantage of this kind of approach is that after learning we can modify the rules and understand exactly why some disambiguation was done.

3.2 Morphological and Word Sense Disambiguation

Morphological disambiguation is done in two stages: first, the contextual rules referred in the previous section are applied; then, remaining ambiguities are suppressed with a statistical POS tagger based on a second-order hidden Markov model (HMM). This turns out to be a fast and efficient approach using the Viterbi algorithm ([9, 11]). The prior contextual and lexical probabilities were estimated by processing large, partially tagged corpora. Lexical probabilities are encoded for each lemma*,* rather than for each word. To achieve this, we calculated, for

each lemma, its frequency and the relative frequency of its inflections. Then, those lemmas with similar distributions for their inflections are grouped into a smaller number of classes. Clustering techniques based on competitive learning [6] are used to choose the number of classes, group the lemmas, and characterize each class. The advantage of working with these clusters is that, on one hand, we can extend the behaviour to words that are not so frequent in our corpora, and, on the other hand, we can compress the information we need at runtime.
Word sense disambiguation is still at an early stage. Currently, it is implemented as a set of rules that use the semantic features of the lexicon and the ontology domains. We are also investigating the use of the rule builder by using the collocations of the monosemous synonyms of a polysemous word as the basis for comparison with the collocations of the polysemous word we want to disambiguate. The same system is being tried using parallel corpora instead of synonyms.

3.3 Grammar Editor, Tester and Compiler

Priberam's SintaGest program is used to build and test a grammar that describes the syntax of a particular natural language. This tool was successfully used by Priberam's linguistic and programming teams to develop European and Brazilian Portuguese grammars. It would be cumbersome to cover all the functionalities of SintaGest in this article, so we will focus on its main features[7].
The interactive mode is where the user can write, edit and test the grammar. It is typically used on a sentence by sentence basis in which the user can test the grammar rules that are being written. Several tools are provided to make the user's work easier. These include a graphical view of the morphological analysis, parsing and disambiguation, the possibility of running the grammar in a step by step mode with breakpoints, the automatic synchronization between the grammar text files and sentence parsing, as well as the possibility of including comments in the grammar.
The second group of SintaGest's functions are the ones that run in batch mode typically on a corpus. They are used to automatically test the grammar against a corpus and generate reports which are analysed or compared to the reports from different versions of the grammar. As the grammar gets more and more complex, this is a fundamental tool because changing one rule can have unforeseen side effects in other rules. This can also happen because of changes in the software. This is a way of comprehensively test grammar and software versions with real texts, to evaluate the extent of a given change. This batch tool is also used to profile the software without the noise introduced by other parts of the proofing tools systems (word processor and user interface) in order to find out how to improve it in terms of speed, which is very important in parsing intensive applications like the ones in information retrieval.
One of the most complex modules of SintaGest is the grammar compiler that generates a file with the lexicon and grammar rules information in a computationally optimized format for the specific purpose of the target tool. For instance, if the syntactic analyser being built will be used for information retrieval, no suggestion information or user messages will be included as opposed to the file for a grammar checker.
SintaGest provides a stable environment for the development of grammars in a CFG based formalism, allowing the user to input, inspect and modify the grammar, perform consistent checks by processing example sentences, and to perform grammar improvements.
Just as any programming language development system (for C++ for instance), SintaGest enables its users to develop fully functional grammars and applications like a grammar checker without coding them from scratch. It is not just for testing some ideas or language structures; it builds real world NLP applications in a much more interesting approach than by coding all the linguistic knowledge in the software.

4 Applications

The linguistic resources and the software tools described in the previous sections interact at several levels and are being used in different areas of NLP applications, namely text processing and information retrieval.
Some of FLiP's modules have been integrated in several text processing applications, such as Microsoft Office, newspaper editorial systems, subtitling editors, automatic correction of web/database queries, etc.
As for information retrieval, morphological components have been integrated in the indexing and query mechanism of Priberam's LegiX law databases and in Microsoft Index Server, as a word breaker and stemmer. The word breaker splits a text into words and expressions and is the first step for indexing. It normalizes different representations of numbers, dates and time: different expressions of numbers, dates and time are turned into a single format, which makes the search procedure easier. It is also responsible for processing accent (or lack of accent) and for expanding abbreviations and acronyms. As for the stemmer, it returns all the words that have the same radical. For instance, it allows a search by "expropriação" (expropriation) to retrieve documents that include "expropriações" (expropriations) ou "expropriado" (expropriated). For languages with a rich morphological system, such as Portuguese, this tool has quite an impact in search performance. Priberam

[7] Some of them will be shown in a demo session to be presented at this conference.

implemented Microsoft's IWordbreaker and IStemmer APIs in 32 and 64 bits, which allowed a simpler integration of these components in Microsoft SharePoint Portal Server and in Microsoft SQL Server.
As mentioned previously, Priberam has also engaged its NLP technology in TRUST's search engine, namely in monolingual and cross-language tasks (see [2] for general outline and experimental results). This technology is presently being integrated in the web version of our law databases[8], in order to allow prompter and more accurate searches. TRUST technology is also being used for implementing a SPAM filter and an automatic email response system.
Software components are available for integration by programmers in third-party applications. The components available so far – FSP: spell checking; FMR: morphological analyser; FGR: grammar and style checker – allow the integration of robust technology developed over ten years and tested by tens of thousands of users. They can be integrated on-line or off-line enabling a set of new functionalities with a low investment. These components can also be used, at no cost, for academic and research purposes [5].

5 Conclusions

The NLP tools described in this paper were conceived as a modular set and may be reused separately. The modularity of this workbench makes it easy to adapt solutions to meet different customers' needs, as well as speed up and simplify integration with third-party products. The work in NLP at Priberam started a decade ago and is presently the basis of several products and applications developed and commercialized both internally and externally, by companies like Microsoft.
Future work includes defining a finer semantic classification, accounting for sense generation and systematic polysemy, as well as increasing the lexical relations encoded. It also includes increasing the multilingual lexical links, namely through the ontology. In order to achieve a better ambiguity resolution, we will augment the subcategorization and selection restrictions with collocates, especially when extracted from large corpora. Reutilization of existing lexical resources and automatic production of more information to enrich them is also an item for future work.

6 References

1. Aho, A., R. Sethi, J. Ullman (1986), *Compilers: Principles, Techniques and Tools,* Reading: Addison-Wesley.
2. Amaral, C., D. Laurent, A. Martins, A. Mendes, C. Pinto (2004), Design and Implementation of a Semantic Search Engine for Portuguese. In Proceedings of LREC 2004, Lisbon, Portugal. Also available at http://www.priberam.pt/docs/LREC2004.pdf
3. Brill, E. (1995), Unsupervised Learning of Disambiguation Rules for Part of Speech Tagging. In *Proceedings of the Third Workshop on Very Large Corpora*, pp. 1-13.
4. Earley, J. (1970), An Efficient Context-Free Parsing Algorithm. In *Communications of the ACM*, 13(2): pp. 94-102.
5. Freitas, D. et al. (2002), A Project of Speech Input and Output in an E-Commerce Application. In Ranchhod, Elisabete, N. Mamede, (eds.), *Advances in Natural Language Processing*, Proceedings of the 3rd International Conference, PorTal 2002, Faro, Portugal, June 2002, Berlin: Springer-Verlag, (LNAI n.º 2389), pp.141-150.
6. Haykin, S. (1994), Self-Organizing Systems II: Competitive Learning. In *Neural Networks: A Comprehensive Foundation*. New York: Prentice Hall, pp. 397-443.
7. Hopcroft, J., J. Ullman (1979), *Introduction to Automata Theory, Languages and Computation*, Readin: Addison-Wesley.
8. Johnson, S. C. (1975), *YACC - Yet another compiler compiler*, Computing Science Technical Report 32, AT & T, Bell Laboratories Murray Hill,N J.
9. Manning, C., H. Schütze (2000), *Foundations of Statistical Natural Language Processing*. Massachusetts: The MIT Press.
10. Smadja, Frank (1993), Retrieving Collocations from Text: Xtract. *Computational Linguistics* 19(1): pp. 143-177. Also available at http://smadja.us/J93-1007.pdf
11. Thede, S. M., M. P. Harper (1999), A Second-Order Hidden Markov Model for Part-of-Speech Tagging. In *Proceedings of the 37th Annual Meeting of the ACL*. Maryland: College Park, pp. 175-182. Also available at http://acl.ldc.upenn.edu/P/P99/P99-1023.pdf
12. Vossen, P. (1998), (ed.), *EuroWordnet: a Multilingual Database with Lexical Semantic Networks*. Dordrecht: Kluwer Academic Publishers.

[8] www.legix.pt