Aspects of Dictionary Making Developing an In-House Text Parsing Tool

LIVIA SANGEORZAN[1], MARINELA BURADA[2], KINGA KISS IAKAB[1]
[1]Department of Computer Science
[2]Department of Foreign Languages and Literatures
Transilvania University of Brasov
B-dul Eroilor, Nr. 25, 500030, Brasov
ROMANIA
sangeorzan@unitbv.ro, marinela.burada@gmail.com, kissjakab@gmail.com http://www.unitbv.ro

Abstract: - This paper reports on particular aspects of ongoing research funded by the National University Research Council and conducted by an interdisciplinary team of academics from *Transilvania* University of Braşov, Romania, of which the authors of the present contribution are members. Based on the results yielded by a large-scale survey of seventy online bilingual/multilingual dictionaries involving the English and Romanian domains, we begin with an assessment of the *status quo* in the area of glossaries and dictionaries available on the internet; we then focus on one particular aspect of dictionary design, *i.e.* the development and operation of a flexible, customizable scanner-parser that we designed with a view to optimizing the work associated with data collection and dictionary compiling.

Key-Words: - accessibility, customizability, interstructure, Java, macrostructure, online dictionary, parsing tool

1 Introduction

The specialist literature provides ample evidence of the impact that online dictionaries have in the knowledge-based era. Their emergence and prospective development have been hailed ([1]) as a qualitative leap from a finished, static product to a dynamic, flexible service apt to promptly meet the greatest variety of cognitive demands and, at the same time, fall in step with the ongoing process of language change.

The discussion below focuses on a particular aspect in the research work behind the design and implementation of a specialized bilingual online dictionary, which is the main objective of our research project. By and large, this research consists of three main stages: first, the survey of a number of seventy online dictionaries - a qualitative approach to their *macrostructure*, *micro-structure*, and *interstructure*; second, the development of an in-house software support system designed to assist in the various activities associated with online dictionary compilation and implementation and third, piloting and field-testing the online dictionary that we created. In this context, the topic of this paper specifically refers to research work conducted in the second stage of the project: the description of a customized parsing tool used in the selection of the relevant lexicographic input to be included in our specialized bilingual dictionary.

As already mentioned, the investigation conducted in the first stage of our project has been based on a set of qualitative criteria subsumed under three main coordinates, *i.e.* the *macrostructure*, *microstructure*, and *interstructure* of online dictionaries, which are wide enough in scope to target both the lexicographic input and the computer programming effort required by the design and implementation of online dictionaries

The rationale behind this investigation has been, firstly, to identify, diagnose, and typify the problems commonly encountered when using such internet resources and secondly, to find solutions and design tools aimed at amending them. The ultimate goal of our research project is to optimise both the process and the product of online dictionary design by suggesting a set of reference criteria and standards that online dictionaries involving the Romanian domain should meet in order to increase their reliability and accessibility.

2 Problem Formulation

As stated above, the design and implementation of a professional online dictionary implies a preliminary step of selecting the set of words which will become entries in our dictionary. We propose to optimize this process of headword selection by implementing an automated text parsing tool. What does the concept of text parsing mean? Informally, parsing refers to the process of analyzing a sentence or language statement. Parsing breaks down texts written in natural language into grammatical units like sentences, words, etc. This parsing tool has the effect of a lexical analyzer combined with a syntactic analyzer [2]. The goal is to recognize sentences and words so that they can serve as input for processing and analysis by the team of lexicographers and be subsequently introduced in the database of our dictionary.

A retrospective overview of compilers shows that there are various lexical analyzer generators and syntactic analyzer generators. First there were Lex [6] and Yacc [5] and later appeared different versions like: Flex, Rex, Jlex, Jflex respectively Bison, Byacc, Lalr, Byacc/J [3] for different programming languages with different extensions. All types of scanner generators and parser generators were developed for the lexical and, respectively, the syntactic analysis of any kind of input. We chose not to use any of these tools, because we are trying to scan and parse particular genres of natural language; furthermore, our scope is to break down natural texts written in English into sentences and words for later analysis. The general scheme of this process targeted at building an online bilingual dictionary is illustrated in Figure 1.

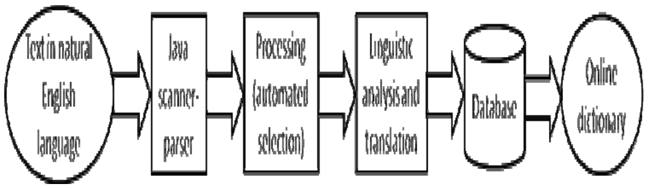

Text in natural English language
Java scanner-parser
Processing (automated selection)
Linguistic analysis and translation
Database
Online dictionary

Fig. 1 The complete process from input to output

Our parsing process relies on the concept of regular expressions which are closely related to regular grammars. In this context, relevant new work was done by S. Dumitrescu in [4] concerning replacement grammars. Some volume of research towards developing conditions for language modeling regular expressions is associated to Popa in [16]. Applications of regular expressions in the domain trainee-adaptive tests and evaluation can be found in [17].

2.1 Related work

From a survey of the literature devoted to dictionaries in general and to IT-supported dictionaries in particular it becomes obvious that a good amount of attention ([1], [9], [10]) has been paid to the technical capabilities of databases and lexicons, *i.e.* the optimal algorithms and mechanisms for optimal applications used or to be used in the design of online resources. Most of the authors seem to concern themselves with the assessment of the finished product, while fewer target particular aspects of dictionary design, *e.g.* search and navigation options, lexical access and associative networks building, text categorization techniques ([18]) *etc.*, highlighting the merits, the advantages and/or the drawbacks thereof. By and large, the responsibility for the lack or limited success in accessing and decoding the information required is shared jointly, as suggested in the literature ([11], [12], [13]), by the dictionary user on one hand, and by the dictionary itself, on the other. In sum, it appears that the higher the potential for information supply, the greater the demands, technology-wise, on the user, whose computer (and/or dictionary) literacy seems to be assumed *de facto*. While we agree that this may well be the case in practice, our claim is that this need not be so, if the complexity of the informational content is counterbalanced by effective data storage, as well as judicious word search and retrieval facilities.

2.2 Some problems with online dictionaries: the status quo

In what follows we shall briefly outline some of the more conspicuous **macrostructural** and **interstructural** flaws identifiable in the corpus of dictionaries we investigated. For obvious reasons, we will deliberately leave out the microstructural problems, which are linguistic in nature and hence not germane to the present discussion. It should also be mentioned from the outset that the research corpus has been considered and assessed from the perspective of the sophisticated user rather than from the computer specialist's standpoint. Moreover, as might be expected, the weaknesses highlighted below are not endemic to the research corpus in its entirety; despite that, they are frequent enough to provide a sense of typicality.

Following Burke ([1]), we are using the term *macrostructure* here to refer to the interface of online dictionaries and to possible interaction routines, *e.g.* the search options available to the dictionary user. From among the macrostructure-related flaws shortlisted as a result of our survey, we will include only two interrelated features with a direct bearing on user-friendliness: customizability, and accessibility.

Customizability is a direct result of the configurability options that the programmers include when developing the software for online dictionaries. Customizability problems are particularly manifest when users have little or no control over the search they perform, or when they obtain less or more information than originally requested. Therefore, while most dictionaries allow the reader to modify/adapt/refine the search, there are also static dictionaries which closely emulate their print counterparts in giving the user virtually no search options at all. Other information provided, such as the number of words contained by each lexicon, the most frequent/recent lookups have only statistical validity, and do not enable the access to the linguistic data in the dictionary. At the other end of the continuum, we may versions of the fuzzy matching option – extreme, in the sense that they indiscriminately include among the results all the occurrences of the headword, in all the definitions available in the dictionary.

Furthermore, one particular case that deserves attention here and which can be accounted for in terms of inaccurate use of the stemming method is the lack of discrimination between stems proper (which are typically used as headwords) and coincident segments found in other words: *e.g.* for the Romanian headword *tratat* (English *treaty*), a rather ludicrous result was 'barker: ciine la**tratat**or'. Stemming as one of the most popular feature-based extraction method has been discussed by Janakova [18], who outlines the constraints in its application when attempting to categorize texts produced in the Czech language. The points she makes are also pertinent to Romanian, whose highly inflectional language system sometimes makes stems rather difficult to recognize.

Apart from this, further illustrations of imperfect customizability are some multilingual dictionaries which by default present the search results simultaneously in all the languages they operate with, regardless of the target-language selected by the user.

A major drawback of online dictionaries is the surplus of information. This redundancy is determined by the results of the search and the way in which they are structured, either because they are repetitive, or because they are irrelevant in different ways. For example, the user asks for the English counterpart of a Romanian word and by default obtains the Italian, German, Spanish and French translations; or s/he searches for a certain word and also obtains results which only bear a formal resemblance to the original word, or results that are embedded in a large amount of information that has no connection to the original search.

Acessibility problems also relate to a number of design and implementation errors which have an adverse effect on the presentation/display of results, *i.e.* misspellings/ typographical errors, incomplete bracketing, inappropriate spacing, inadequate use of sign and symbols, lack of diacritics, all of which might be alienating to the end user. Moreover, some dictionaries seem to be designed for a restricted number of people that share some special, esoteric knowledge of data accessing. A design flaw common to almost all the dictionaries we investigated is the lack of diacritics, usually signaled on the home page or in the instructions. In the rest of the cases, the impossibility to distinguish between words whose forms are only slightly different because of the use (or non-use) of diacritics determines a higher number of search results making the whole process of selection longer and more difficult.

Apart from such macrostructural aspects, interstructure is also worth referring to. The term interstructure is used to denote how an entry links to resources outside the lexicon; in other words, to indicate how it integrates these external resources in order to provide more detailed information about the headword. This element is particularly relevant for the purposes of our dictionary. The reality of interstructure is a direct consequence of the online medium and represents a significant point of distinction between print and online lexicons. Moreover, Burke ([1]) emphasizes the usefulness of interstructure in bilingual on-line lexicons as carriers of extensive information about culture-specific terms and, we might add, highly specialized terminology. Obviously, this would be a desideratum for all bilingual/multilingual online dictionaries; none of the dictionaries under investigation, however, is even close to the notion of interstructure. Even with the very few dictionaries which actually make use of hyperlinks, their use is restricted to facilitating the user's access to the information available internally, that is, inside the dictionary itself. In other words, instead of being a means of obtaining new information, interstructure is, in this case, an alternative way of accessing the same information.The few points summarized above should be justificative enough of the need for improvement in the design and implementation of online dictionaries aimed at interlingual transfers between Romanian and English. As part of the remedial action undertaken to that effect, the following section provides a description of a text parsing programme designed for natural data processing. As already mentioned above, this tool is part of the software system devised for the development of an in-house bilingual, bidirectional dictionary of specialised terminology

3 Problem Solution

During the first stage of our research project the input texts are selected and categorized non-automatically into a number of predefined categories (legal/quasi-legal, trade, politics); these texts will be permanently stored in the dictionary database and will remain accessible via the dictionary's interstructure. Next, the input texts are parsed in order to select the headwords to become dictionary entries. In this word-based selection process the features are the frequency of use, but also the frequency of sense, in the case of polysemous items. In other words, the classification task is based on the commonly used 'bag-of-words' approach, according to which each distinct word in a text is a selection feature and the number of their occurrences in the text corresponds to their value. Apart from that, the organisational principle underlying our dictionary is the alphabetical sequence of headwords – a principle that replicates the linearity of printed dictionaries. Despite the fact that, according to some authors ([14], [15]), the thematic and pragmatic grouping of headwords might be closer to the way the human mind works, alphabetisation is still considered more practical, considering that dictionary users are already accustomed to it.

3.1 Text parsing

The formal goal of text parsing can be formulated as follows: Given a text T = $(x_1, \ldots, x_n)$ in a natural language L, derive the correct analysis for every sentence $x_i \in T$ and every word $y_j \in x_i$ for every sentence x_i.
From the theory of formal languages and automatons, we know that words of some languages can be recognized using grammars, automatons and regular expressions

Definition 1.[19]
A grammar *G* is formally defined as the ordered quad-tuple (N, Σ, P, S) with the following components:

- A finite set *N* of *nonterminal symbols.*
- A finite set Σ of *terminal symbols* that is disjoint from *N.*
- A finite set *P* of *production rules*, each rule of the form $(\Sigma \cup N)^* N (\Sigma \cup N)^* \rightarrow (\Sigma \cup N)^*$
- where * is the Kleene star operator and $\cup$ denotes set union. That is, each production rule maps from one string of symbols to another, where the first string contains at least one nonterminal symbol. In the case that the second string is the empty string – that is, that it contains no symbols at all – in order to avoid confusion, the empty string is often denoted with a special notation, often (λ, *e* or ε).
- A distinguished symbol $S \in N$ that is the *start symbol.*

Such a formal grammar is often called a rewriting system or a phrase structure grammar in the literature.

The Chomsky hierarchy
When Noam Chomsky first formalized generative grammars in 1956 [19], he classified them into types now known as the Chomsky hierarchy. The difference between these types is that they have increasingly strict production rules and can express fewer formal languages. Two important types are *context-free grammars* (Type 2) and *regular grammars* (Type 3). The languages that can be described with such a grammar are called *context-free languages* and *regular languages*, respectively. Although much less powerful than unrestricted grammars (Type 0), which can in fact express any language that can be accepted by a Turing machine, these two restricted types of grammars are most often used because parsers for them can be efficiently implemented [20]. For example, all regular languages can be recognized by a finite state machine, and for useful subsets of context-free grammars there are well-known algorithms to generate efficient LL parsers and LR parsers to recognize the corresponding languages those grammars generate.

Definition 2. [19]
The Chomsky hierarchy consists of the following levels:

- *Type-0 grammars (unrestricted grammars)* include all formal grammars. They generate exactly all languages that can be recognized by a Turing machine. These languages are also known as the recursively enumerable languages. Note that this is different from the recursive languages which can be *decided* by an always-halting Turing machine.
- *Type-1 grammars (context-sensitive grammars)* generate the context-sensitive languages. These grammars have rules of the form $\alpha A\beta \rightarrow \alpha\gamma\beta$ with *A* a nonterminal and α, β and γ strings of terminals and nonterminals. The strings α and β may be empty, but γ must be nonempty. The rule $S \rightarrow \varepsilon$ is allowed if *S* does not appear on the right side of any rule. The languages described by these grammars are exactly all languages that can be recognized by a linear bounded automaton (a nondeterministic Turing machine whose tape is bounded by a constant times the length of the input.)
- *Type-2 grammars (context-free grammars)*

generate the context-free languages. These are defined by rules of the form $A \to \gamma$ with A a nonterminal and γ a string of terminals and nonterminals. These languages are exactly all languages that can be recognized by a non-deterministic pushdown automaton. Context free languages are the theoretical basis for the syntax of most programming languages.

- *Type-3 grammars (regular grammars)* generate the regular languages. Such a grammar restricts its rules to a single nonterminal on the left-hand side and a right-hand side consisting of a single terminal, possibly followed (or preceded, but not both in the same grammar) by a single nonterminal. The rule $S \to \varepsilon$ is also allowed here if S does not appear on the right side of any rule. These languages are exactly all languages that can be decided by a finite state automaton. Additionally, this family of formal languages can be obtained by regular expressions. Regular languages are commonly used to define search patterns and the lexical structure of programming languages.

Definition 4. [21]
The set of generated words by a generative grammar G is usually called generated language, denoted by L(G). L(G)=$\{w \in \Sigma^* | S \Rightarrow w\}$ (where the meaning of the notation $S \Rightarrow w$ is the following one: the word w can be generated from S using the rules of the grammar G).

Notation 1.
Let us denote $\mathcal{L}_0$ the set of Type 0 generated languages or recursive languages, $\mathcal{L}_1$ the set of Type 1 generated languages or context-sensitive languages, $\mathcal{L}_2$ the set of Type 2 generated languages or context-free languages and $\mathcal{L}_3$ the set of Type 3 generated languages or regular languages.

Every regular language is context-free, every context-free language is context-sensitive and every context-sensitive language is recursive and every recursive language is recursively enumerable. These are all proper inclusions, meaning that there exist recursively enumerable languages which are not context-sensitive, context-sensitive languages which are not context-free and context-free languages which are not regular. These relations can be represented as follows in the next figure.

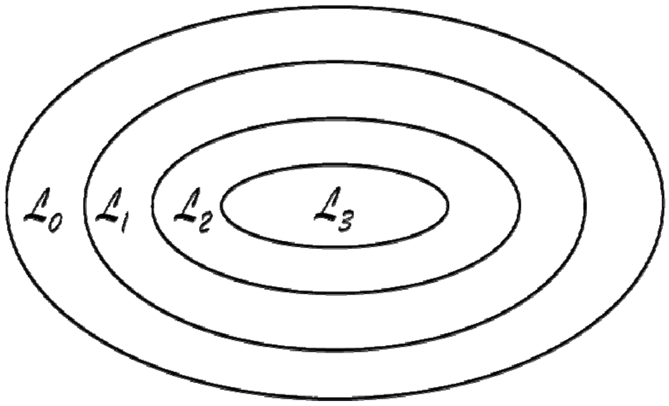


Fig. 2. Graphical representation of the Chomsky hierarchy

Definition 5. [22]
A finite-state machine is a quintuple (Σ, S, s_0, δ, F), where:

- Σ is the input alphabet (a finite, non-empty set of symbols).
- S is a finite, non-empty set of states.
- s_0 is an initial state, an element of S. In a nondeterministic finite state machine, s_0 is a set of initial states.
- δ is the state-transition function: $\delta: S \times \Sigma \to S$.
- F is the set of final states, a (possibly empty) subset of S.

Finite-state machine are generally used to determine if a word is can be generated by one regular grammar which is equivalent to the finite-state machine. So, the finite-state machines are the deductive counterparts of generative regular grammars.

Definition 6.[3]

Regular expressions can be defined by the following recursive rules:

1. Every symbol of an alphabet Σ is a regular expression;
2. The null symbol ε is a regular expression;
3. If r1 and r2 are regular expressions, so are (r1) (association), r1r2 (concatenation), r1 | r2 (alternation), r1* (repetition);
4. Nothing else is a regular expression.

Generative regular grammars, finite-state machines and regular expressions represent equivalent mechanism to generate/recognize Type 3, regular languages. Our approach uses regular expressions for the purpose of language structures and specially words recognition.

So, a regular expression is a rule that defines exactly the set of words that are valid tokens in a formal language. The rules are built up on three operators: concatenation, alternation and repetition.

One difficulty of parsing has traditionally been to achieve robustness [7], where robustness can be defined as the capacity of a system to analyze any input sentence. The shortcomings of grammar-driven systems in this respect can be traced back to the fact that some input sentences xi in a text T are not in the language L(G) defined by the formal grammar G.

We can distinguish two different cases where xi ∈ L (G). In the first case, xi is a perfectly well-formed sentence of the language L and should therefore also be in L(G) but is not. In the second case, xi is considered not to be part of L, and should therefore not be in L(G) either, but nevertheless has a reasonable syntactic analysis. However, even though there are many clear-cut examples of both kinds, there are also many cases where it is difficult to decide whether a sentence that is not in L(G) is in L, at least without resorting to a prescriptive grammar for the natural language L. According to [8], there are essentially two methods that have been proposed to overcome the robustness problem for grammar-driven systems. The first is to relax the grammatical constraints of G in such a way that a sentence outside L(G) can be assigned a complete analysis. The second is to maintain the constraints of G but to recover as much structure as possible from well-formed fragments of the sentence. We intend to solve this problem when we reach the final dictionary implementation stage.

3.2 A Java scanner parser

Our approach to text parsing is top-down, as follows: as a first step, the text is read in a text file which is parsed by means of Java regular expressions. During this step, the selection feature is the sentence, or rather, the word string extending between, say, two full stops. In other words, reading starts by selecting and separating the well-formed sentences in English, that is, the strings of words which start with a capital letter and end in a punctuation mark such as a full stop, exclamation/ question mark. The regular expression used to that effect is [A-Z].+?[\\.\\?\\!]. The sentences thus identified and selected and their number are saved in a file named ‘*inputFileName_sentences.txt*’. The next step is to identify individual words in the text; numbers, calendar dates, and suchlike are irrelevant to our purposes, so the programme will ignore these as well as other stop words like prepositions, conjunctions, and pronouns.

The words in the text are separated from each other by word boundaries marking the beginning and the end of each word. The word cannot include special symbols such as comma, apostrophe, semicolon, full stop, exclamation/question mark, round/square/curly brackets, numbers, and it must be at least one character long. The Java regular expression is \\b[^\\,\\'\\;\\.\\!\\?\\(\\)\\[\\]\\{\\}0-9   ]+?\\b.

Having identified all the words in the text, the programme generates lists of non-identical words because, obviously, there would be many recurring lexical items. The number of the non-identical words and their list is written in the file named “*inputFileName_differentwords.txt*”. The next step involves the elimination of non-words consisting of one, two and three characters, as already pointed out above. After this step, similar information about the remaining words is saved in the file named “*inputFileName_wordsafterelimination.txt*”. Next we build a TreeMap-type structure able to memorize the words and their frequency of occurrence in the input text.

Having built this essential information we have reached the stage at which the relevant results are generated. Overall, a number of fifty-two files are generated corresponding to the twenty-six letters of the alphabet: twenty-six files for the upper case letters and twenty-six files for the lower case ones. These files will hold the following information: for example, in the file “*inputFileName _c_small.txt*” the programme will write the list of words starting with lower case ‘c’. This folder will include information relative to the frequency of occurrence of each individual word as well as to the larger context (*i.e.* the sentence) in which each word occurs.

Relative to context display, we have designed a customized search allowing us to skip the mechanic display of each context containing sublemmata (*e.g.* ‘commonly’ in relation to the lemma ‘common’). To this effect, we have implemented a method which allows the selection of sentences which include the lemma, while opting out the contexts including the sublemma(ta) thereof.

The data in these files are further processed by the lexicographers in our team, who build up the lexicographic repertoire of the future online dictionary. This implies that parsing is a tool aimed at automating the selection of headwords and of their corresponding contexts.

Our Java Scanner-Parser parses a natural English text by determining the sentences and words of the text. The general scanning-parsing process involves the steps illustrated in Figure 3

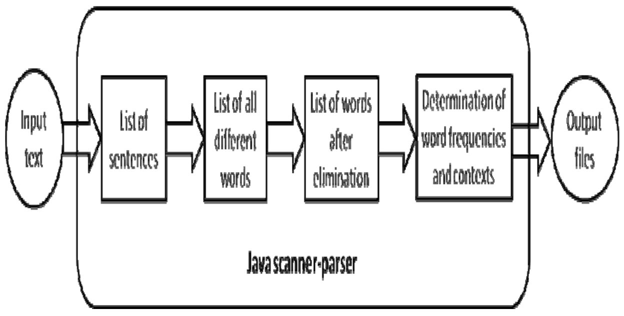

Input text
List of sentences
List of all different words
List of words after elimination
Determination of word frequencies and contexts
Output files
Java scanner-parser

Fig. 3. The scanner-parser process

In the parsing process the above listed actions from the scanner-parser process are executed in the previously specified order. The parsing tool is based on the Java implementation of the following parsing algorithm:

1. Reading input file into one string
2. Delimitation of sentences from the text
3. Delimitation of words form the sentences
4. Elimination of repeating words
5. Elimination of “short” words
6. Determination of word frequencies
7. Determination of word contexts
8. Writing output files for sentences and words with information

The Java tool has one text file as input file and several output file also with *txt* extension. The input-output diagram of the parsing tool is illustrated in the next figure:

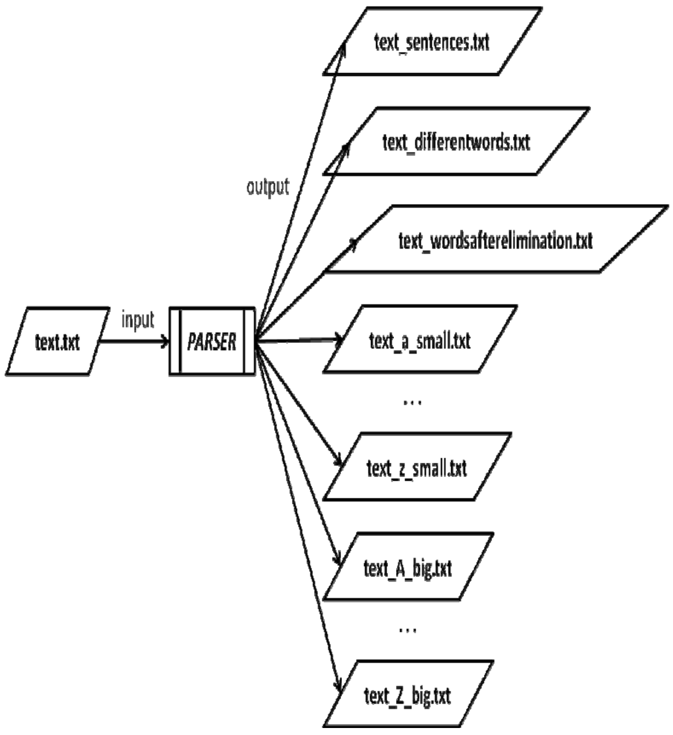


Fig. 4. Input-output diagram of the parsing tool

The usage of regular expressions in Java is facilitated by the classes *Pattern* and *Matcher* which are components of the package *java.util.regex*. The general usage pattern of these classes can be observed in the following sources code sequence, which parses the text and determines the list of all its sentences.

```
public Vector sentences()

{
  Pattern pattern=Pattern.compile(“[A-Z].+?
    [\\.\\?\\!]”);
  Matcher matcher=pattern.matcher(text);
  Vector sentences=new Vector();
  while(matcher.find())

  {
    sentences.add(matcher.group());
  }
  return sentences;
  }
```

Similarly, to recognize all the words in the text, we use the regular expression mentioned above as in the following Java-code sequence:

```
public Vector words()
{
  Pattern pattern=Pattern.compile(

"\\b[^\\,\\'\\;\\.\\!\\?\\(\\)\\[\\]\\
{\\}0-9]+?\\b");

Matcher matcher =
pattern.matcher(text);
  Vector words=new Vector();
  while (matcher.find())

 {
    words.add(matcher.group());
  }
  return words;
}
```

After determining the set of words in the text we eliminate repeating words, meaning with determine the list of different words in the text. This process can be realized with the following Java method:

```
public Vector diffwords()
{
  Vector w=words();
  Vector diffwords=new Vector();

  for(int i=0;i<w.size();i++)
    if (!diffwords.contains(w.get(i)))
      diffwords.add(w.get(i));

  return diffwords;
}
```

Finally, we eliminate the irrelevant "binding" words like: a, an, the, and, or, so, etc. In this process all words shorter than 4 characters are eliminated.

```
public Vector afterEliminationWords()
{
  Vector ew=new Vector();
  Vector dw=diffwords();

  for(int i=0;i<dw.size();i++)
    if (((String)dw.get(i)).length()>=4)
      ew.add(dw.get(i));
  return ew;
}
```

5 User's Guide for the Java Parser

The authors of the present paper are members of the NURC (The National University Research Council) funded project called LEXICA. The research results presented herein are part of the aforementioned project and therefore all information, results and advances in the project are freely available on the Internet under presented herein are part of the aforementioned project and therefore all information, results and advances in the project are freely available on the Internet under http://cerex.unitbv.ro/lexica/..
The structure and content of the project's website is outlined in Figure 5.

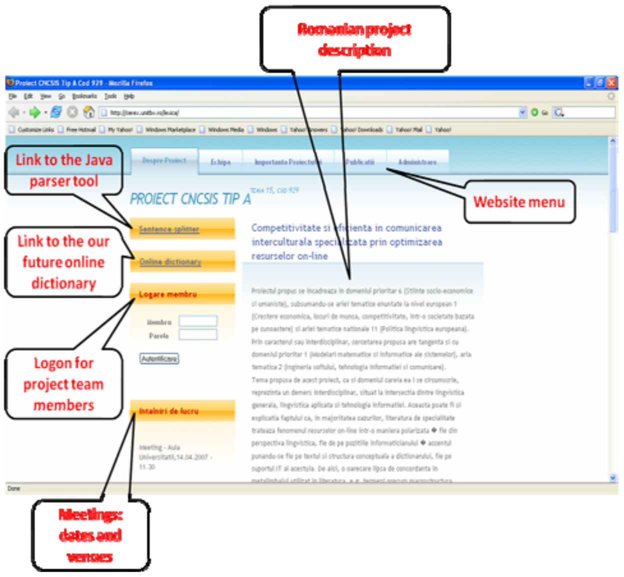
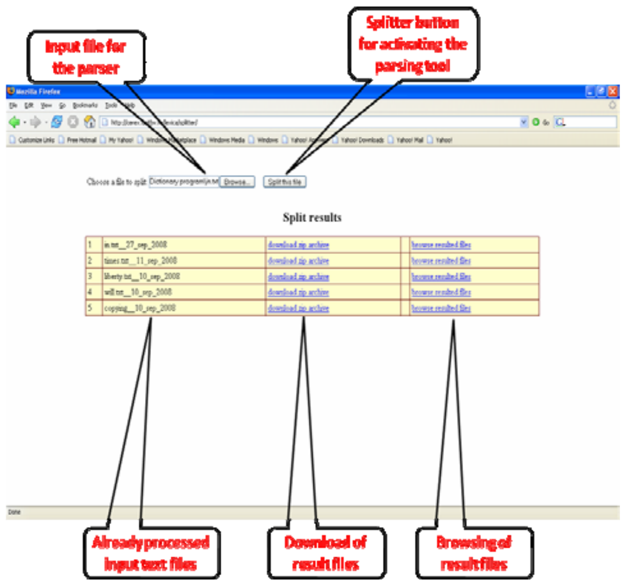

Romanian project description
Link to the Java parser tool
Website menu
Link to the our future online dictionary
Logon for project team members
Meetings: dates and venues
PROIECT CNCSIS TIP A
Sentence splitter
Online dictionary
Logare membru
Intalniri de lucru
Competitivitate si eficienta in comunicarea interculturala specializata prin optimizarea resurselor on-line

Fig. 5. Website of the LEXICA project

In order to use the Splitter parsing tool, the user has to upload his text in the form of a file with *txt* extension. After the upload, the Java parser is takes the *txt* file as an input and generates a set of output files. The web-interface allows the users to see previously processed input files and to browse their output files in the table from the Figure 6 below.

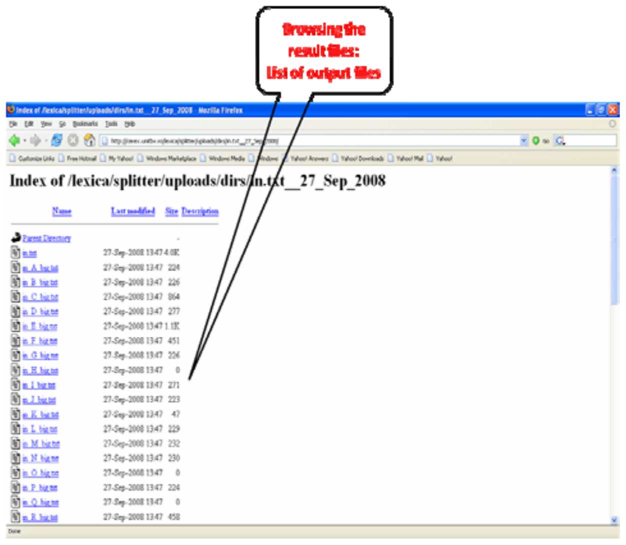

Input file for the parser
Splitter button for activating the parsing tool
Choose a file to split: Dictionary program\in.txt
Browse...
Split this file
Already processed input text files
Download of result files
Browsing of result files

Split results

1	in.txt__27_sep_2008	download zip archive	browse resulted files
2	times.txt__11_sep_2008	download zip archive	browse resulted files
3	liberty.txt__10_sep_2008	download zip archive	browse resulted files
4	will.txt__10_sep_2008	download zip archive	browse resulted files
5	copying__10_sep_2008	download zip archive	browse resulted files

Fig. 6. The web-interface of the “Splitter” parsing tool

When browsing output files, the user is allowed to view the list of all output files, the date of their creation and also their content. The editing of these files is not allowed.

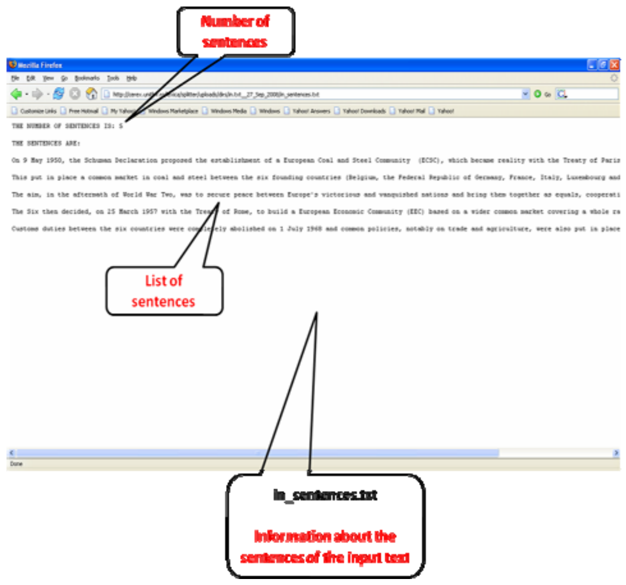


Index of /lexica/splitter/uploads/dirs/in.txt__27_Sep_2008

```
Name              Last modified       Size Description

Parent Directory                         -
in.txt            27-Sep-2008 13:47  4.0K
in_A_big.txt      27-Sep-2008 13:47   224
in_B_big.txt      27-Sep-2008 13:47   226
in_C_big.txt      27-Sep-2008 13:47   864
in_D_big.txt      27-Sep-2008 13:47   277
in_E_big.txt      27-Sep-2008 13:47  1.1K
in_F_big.txt      27-Sep-2008 13:47   451
in_G_big.txt      27-Sep-2008 13:47   226
in_H_big.txt      27-Sep-2008 13:47     0
in_I_big.txt      27-Sep-2008 13:47   271
in_J_big.txt      27-Sep-2008 13:47   223
in_K_big.txt      27-Sep-2008 13:47    47
in_L_big.txt      27-Sep-2008 13:47   229
in_M_big.txt      27-Sep-2008 13:47   232
in_N_big.txt      27-Sep-2008 13:47   230
in_O_big.txt      27-Sep-2008 13:47     0
in_P_big.txt      27-Sep-2008 13:47   224
in_Q_big.txt      27-Sep-2008 13:47     0
in_R_big.txt      27-Sep-2008 13:47   458
```

Fig. 7. The output files of the Splitter

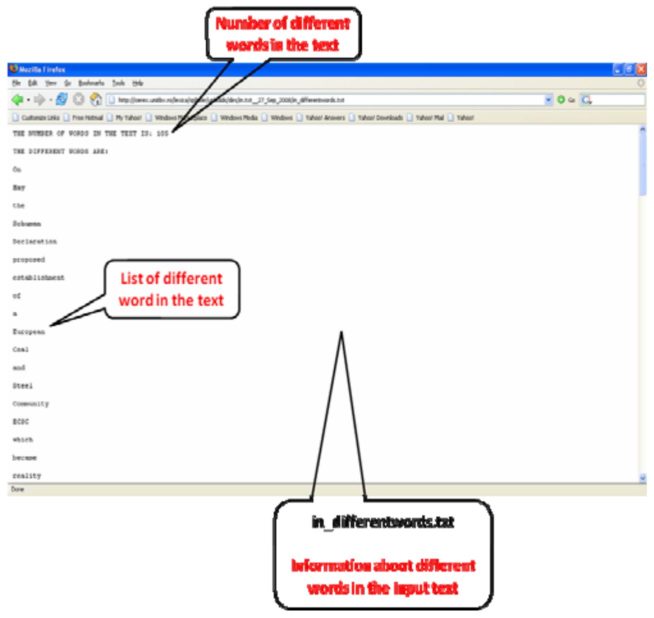
The first generated output file is the one with the sentences of the text. The parser separates all sentences of the input text and writes statistical information concerning the number of identified sentences and the sentences themselves in the output file.



```
THE NUMBER OF SENTENCES IS: 5

THE SENTENCES ARE:

On 9 May 1950, the Schuman Declaration proposed the establishment of a European Coal and Steel Community  (ECSC), which became reality with the Treaty of Paris
This put in place a common market in coal and steel between the six founding countries (Belgium, the Federal Republic of Germany, France, Italy, Luxembourg and
The aim, in the aftermath of World War Two, was to secure peace between Europe's victorious and vanquished nations and bring them together as equals, cooperati
The Six then decided, on 25 March 1957 with the Treaty of Rome, to build a European Economic Community (EEC) based on a wider common market covering a whole ra
Customs duties between the six countries were completely abolished on 1 July 1968 and common policies, notably on trade and agriculture, were also put in place
```

Fig. 8. Structure of output file in_sentences.txt

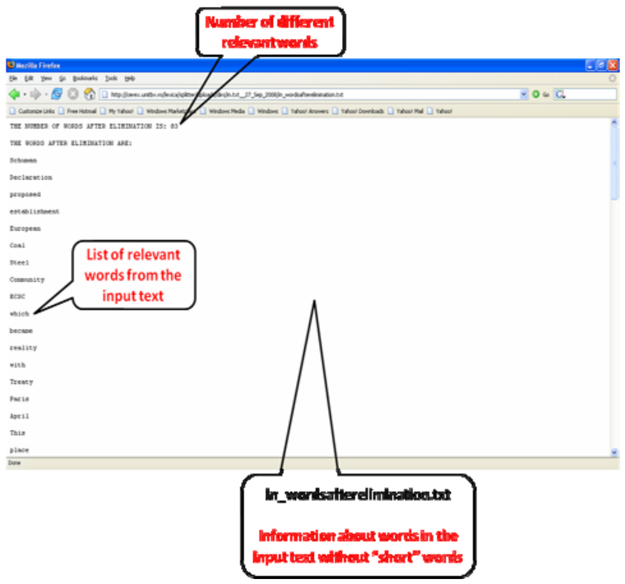
The next lexical units after sentences are words, so in the next step all words of the input text are identified. The output file regarding words contains the total number of different words in the text, respectively the list of them.


Number of different words in the text
List of different word in the text
in_differentwords.txt
Information about different words in the input text

```
THE NUMBER OF WORDS IN THE TEXT IS: 105

THE DIFFERENT WORDS ARE:

On
May
the
Schuman
Declaration
proposed
establishment
of
a
European
Coal
and
Steel
Community
ECSC
which
became
reality
```

Fig. 9. Structure of the output file in_differentwords.txt

Text written in natural English language contains many “binding” words like: a, the, so, and, or, etc. which are not relevant for our future online dictionary. This is why we first eliminate these words by eliminating all words which are shorter than 4 characters. The results after elimination can be viewed in the output file which name ends in “wordsafterelimination” as in the next figure.

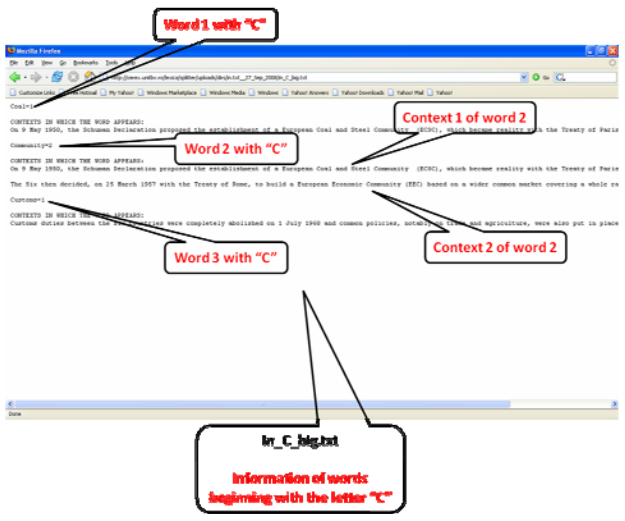


```
THE NUMBER OF WORDS AFTER ELIMINATION IS: 83

THE WORDS AFTER ELIMINATION ARE:

Schuman

Declaration

proposed

establishment

European

Coal

Steel

Community

ECSC

which

became

reality

with

Treaty

Paris

April

This

place
```

Number of different relevantwords
List of relevant words from the input text
in_wordsafterelimination.txt
Information about words in the input text without "short" words
Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Customize Links Free Hotmail My Yahoo! Windows Marketplace Windows Media Windows Yahoo! Answers Yahoo! Downloads Yahoo! Mail Yahoo!
Done

Fig. 10. Structure of the output file in_wordsafterelimination.txt

The final action of the parsing tool is the creation of output files with words and their contexts in the text classified by letters. For each letter in the English alphabet a file is written with all words which begin with that letter. First, the frequency of the word is listed and after that all possible contexts of these words are printed. This type of output files can be easily processed by the members of the linguistics team in the process of the dictionary development.



```
Coal=1

CONTEXTS IN WHICH THE WORD APPEARS:
On 9 May 1950, the Schuman Declaration proposed the establishment of a European Coal and Steel Community (ECSC), which became reality with the Treaty of Paris

Community=2

CONTEXTS IN WHICH THE WORD APPEARS:
On 9 May 1950, the Schuman Declaration proposed the establishment of a European Coal and Steel Community (ECSC), which became reality with the Treaty of Paris

The Six then decided, on 25 March 1957 with the Treaty of Rome, to build a European Economic Community (EEC) based on a wider common market covering a whole ra

Customs=1

CONTEXTS IN WHICH THE WORD APPEARS:
Customs duties between the Six countries were completely abolished on 1 July 1968 and common policies, notably on trade and agriculture, were also put in place
```

Word1 with "C"
Context 1 of word 2
Word 2 with "C"
Word 3 with "C"
Context 2 of word 2
in_C_big.txt
Information of words beginning with the letter "C"
Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Customize Links Free Hotmail My Yahoo! Windows Marketplace Windows Media Windows Yahoo! Answers Yahoo! Downloads Yahoo! Mail Yahoo!
Done

Fig. 11. Structure of the output file in_C_big.txt

4 Conclusion

In this paper we have attempted to make a case for the need of improvement in the area of online dictionary making targeting the English-Romanian domain. More specifically, we have presented a stage in our work to design and implement an online dictionary of specialized terminology. One aspect of this process has been the implementation of a scanning-parsing tool which, overall, has at least two advantages over standard variants of parsers: customizability and flexibility of use, which mainly refer to its ability to help build the corpus of specialized texts that our dictionary draws on, as well as to its ability to integrate with other programming languages used in the design of our dictionary’s IT-support system. In developing this tool we have used Java because Java has regular expression handling features, which entails that we do not have to allocate additional time to learn some type of scanner and parser generator languages. Furthermore, we believe that our Java source code can be easily integrated in the finished product of our work, *i.e.* a web-based dictionary.

References:

[1] Burke, S. M.: *The Design of Online Lexicons*, Northwestern University, Evanston, IL., 1998

[2] Aho, A. V., Ulman, J. D.: *The theory of parsing, translation, and compiling*, ACM Classic Books Series, 1972

[3] Aho, A. V., Sethi, R., Ullman, J. D.: *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986

[4] Dumitrescu, S.: *About Normal Forms for Hyperedge Replacement Grammars*, Proceedings of the 12th WSEAS International Conference on Computers, 2008

[5] Johnson, S. C.: *Yacc: Yet Another Compiler Compiler*, Computing Science Technical Report No. 32, Bell Laboratories, 1975

[6] Lesk, M. E., Schmidt, E.: *Lex — A Lexical Analyzer Generator*, Computing Science Technical Report No. 39, Bell Laboratories, 1975

[7] Nivre, J.: *Two strategies for text parsing*, A Man of Measure, Festschrift in Honour of Fred Karlsson, 2006, pp. 440-448

[8] Samuelsson, C., Wirén, M.: *Parsing techniques*, In Dale & al. (eds.), 2000, pp. 59–91

[9] Swanepoel, Piet, *Dictionary Quality and Dictionary Design: A Methodology for Improving the Functional Quality of Dictionaries*, on-line http

://www.sabinet.co.za/abstracts/lexikos/lexikos_v11_a12.xml

[10] Boas, H. C.: *Semantic Frames as Interlingual Representations for Multilingual Lexical Databases*, International Journal of Lexicography 2005 18(4): 445-478

[11] Tickoo, M. L.: *Learners' Dictionaries. State of the Art* Singapore: SEAMEO Regional Language Center, 1989

[12] Winkler, B.: *Students working with an English learners' dictionary on CD*-ROM, ITMELT 2001 Conference

[13] Scholfield, P.: *Dictionaries* on –line at http: //www.llas.ac.uk/resources/ goodpractice.aspx?resourceid=229

[14] McArthur, T: *Thematic Lexicography*, in 'The History of Lexicography', R.R.K. Harmann (ed), Amsterdam: J. Benjamins, 1986

[15] Nesi, H.: *Dictionaries on Computer: How Different Markets Have Created Different Products*, on-line at http://www.tu-chemnitz.de/phil/english/chairs/linguist/real/independent/llc/Conference1998/Papers/Nesi.htm

[16] Popa, E. M.: *Regular expressions of conditions for processing language modeling*, Proceedings 8th WSEAS International Conference on Mathematical Methods and Computational Techniques in Electrical Engineering, 2006

[17] Boboila, C., Boboila, M. S.: *Online Evaluation with Trainee-Adaptive Tests*, Proceedings of the 4th WSEAS/IASME International Conference on Engineering Education, 2007

[18] Janakova, H.: *Text categorization with Feature Dictionary – Problem of Czech Language*, WSEAS Transactions on Information Science and Applications Issue 1, Volume 1, 2004

[19] Chomsky, N.: *Three Models for the Description of Language*, IRE Transactions on Information Theory **2** (2), 113–123, 1956

[20] Grune, D., Jacobs, C. H.: *Parsing Techniques – A Practical Guide*, Ellis Horwood, England, 1990.

[21] Anderson, J. A.: *Automata Theory with Modern Applications*, Cambridge University Press, 2006

[22] Arbib, M. A.: *Theories of Abstract Automata*, 1st ed., Englewood Cliffs, N.J.: Prentice-Hall, Inc, 1969.