Computer Science Journal of Moldova, vol.**3**, no.1(7), 1995

The Roumanian spelling checker ROMSP: the project overview

A.Colesnicov

Abstract

Aspects of the Roumanian spelling checker ROMSP are presented: effective vocabulary representation, similar words detection algorithms, automatic word inflection, the user interface, supporting tools, further development. Problems of user interface engineering support by object oriented methods are of special interest.

1 Introduction

The development of the Roumanian spelling checker had posed many problems. Two main problems were those of the compact representation of the vocabulary including all word-forms, and of the fast search in the vocabulary. We discuss them in the sec. 2.

When the user asks to suggest the correct spelling for a misspelled word we have the problem of generating the list of the words existing in the dictionary and being in some sense "near" to the given word. The list is to be of reasonable size. The problem is discussed in the sec. 3.

It was found impossible to enter all word-forms manually, and we were to develop the automatized word inflection system. The system is also used by the end user to add words to his/her private vocabulary data base. We discuss the problem in the sec. 4.

Except the obvious problems of making the user interface friendly and comfortable we had met a specific for the object-oriented environments problem of the event stream synchronization. See sec. 5.

Engineering aspects of our product are discussed in sec. 6.

ROMSP is a developing system, and the perspectives of its development are presented in the final sec. 7.

2 Effective vocabulary representation

We give here only a general overview of the subject referring for details to [3, 4].

The lexicographical sourses [7]–[16] had supplied us with the base dictionary of more than 65,000 words and the resulting vocabulary of more than 660,000 word-forms. The current ROMSP version checks only separate words, and we are to represent only the word list.

2.1 Vocabulary decomposition

In the section we use a mathematical notion of the word as the arbitrary sequence of letters. Word sets are denoted by capital letters (V, E, R) and words by small letters (v, e, r).

Definition 1 *Let the vocabulary V be a non-empty word set. The binary decomposition of the vocabulary V consists of two sets of words, namely, a root set R and an ending set E, and a map φ, $R \xrightarrow{\varphi} 2^E$ from R to the set of all subsets of E, satisfying the following two conditions:*

- $\forall r \in R\ \forall e \in \varphi(r)$, *the concatenation* $re \in V$.
- $\forall v \in V\ \exists r \in R\ \exists e \in \varphi(r)$ *such that* $v = re$ *(r and e makes the decomposition of the word v).*

We didn't demand the uniqueness of the decomposition of the given word with the given map.

Another map φ may exist for given V, R and E. Two boundary cases are:

- $R = V$ and all endings are empty words (we use this for the user's private vocabulary);

- there is a single root – empty word, but all the elements of V serve as endings.

Of course, something intermediate is of interest. In our case, about 220,000 roots, 144 endings, and about 2,000 ending sets from 2^{144} were sufficient.

See [3, 4] on problems of the minimization of the decomposition, of the dynamical decomposition when adding new words etc.

For each root we store on the hard disk the number of the corresponding ending set. To increase the efficiency, we divide the vocabulary on pages and use the hash table to access the page. Ending sets represented as bitmaps, the hash table and endings themselves are stored in RAM. Also in RAM is stored a small vocabulary of about 600 most frequently used Roumanian words. Now we use for MS-DOS implementaion only one page buffer, and we do not use any additional compression techniques. The vocabulary data base size is now about 1.2 Mbyte – less than 2 bytes per a word. As it may be compressed by usual archivers (LHA, ARJ, PKZIP) with the ratio of 0.5, we have reserves to gain more space.

As the matter of fact, we have two vocabularies: a constant one and a user's private one. The user's vocabulary may be expanded by the user.

3 Detecting similar words

This problem arises when we are to correct the misspelled word automatically, or when user asks suggestions for the word. For ROMSP we use two different techniques.

At first the spelling checker tries to correct the following errors: skipping a letter (or a space), permuting two letters, replacing a letter by another letter, inserting a letter. E.g., to correct skipping a letter we try to insert all letters from the alphabet (and a space) in all possible positions, etc.

We keep in RAM the triades table (about 4,000 bytes) representing all three-letter combinations existing in the words from the current

vocabulary. Every generated word is checked against this triades table, before to be checked against the vocabulary which may lead to the disk access. This makes the process speed quite satisfactory.

After this first stage, we use another algorithm to generate more suggestions and to try to correct more than one error. We use for the purpose a letter similarity matrix. We can use different letter similarity matrices in different cases. E.g., when correcting texts typed on the computer keyboard, a letter is "near" to another if their keys are neighboring. A text entered by a scanner represents another case with different criteria, etc.

Let M be the letter similarity matrix. Consider the following variation of the well-known algorithm for searching the maximal common subsequence for the two given words v and w (n and m letters long correspondingly).

Let us construct $n \times m$ matrix L, where by induction

$$L_{i,j} = \max(L_{i-1,j}, L_{i,j-1}, L_{i-1,j-1} + M_{v_i,w_j}),$$

and out-of-matrix $L_{0,j}$, $L_{i,0}$ are supposed to be zero. Taking $L_{n,m}$ as a criterion of the similarity we can get a rather good similarity function f (e.g., $f(w,v) = L_{n,m}/(n+m)$).

This algorithm uses too many calculations. Let us restrict our suggestion to a reasonable bound: the similar word should be found if there are no more than two mistakes. The possibility to correct three mistakes remains, but only in the case of "natural" mistakes. This restriction gives us a possibility to construct only 5 diagonals from matrix $L_{i,j}$, where the difference between i and j is at most 2, and $L_{i,j} = 0$ in other cases. Moreover, it is not necessary to calculate all diagonals. If the values are too small we can be sure that more than two mistakes were made and stop calculations.

4 Automatic word inflection

Special attention was given to nouns and adjectives declination, and verbs conjugation because they generate a lot of flexions (12 for nouns,

20 for adjectives, 35 for verbs). Pronouns, articles and numerals were entered in the vocabulary data base manually.

We grouped registered affixes for noun declination in 32 groups and has based the declination algorithm on this grouping. In some cases the group for the given noun can not be selected automatically, and we ask the user to select the group giving him/her examples. Some irregularities in letter alterations also imply a dialog with the user.

Adjectives were classified in 26 groups.

The generation of verb word-forms is based on the infinitive of the verb. Generating the inflectional paradigm for the verb is the most complicated part of our system. Verbs are divided into 5 types having their characteristic features. Each type is divided into some classes, characterized by affixes. Each class in its turn is divided into schemes, depending on specific letter alteration, subvariants of affixes, of word roots, etc. We have in total 56 schemes for verbs.

You can found detailed discussion and examples in [1, 2].

The word inflection program shows all generated forms on the screen, and the user can correct them before writing them to the vocabulary data base.

We have two variants of the word inflection program. The variant for internal use generates the log file registering chronologically all data base changes. The log function is disabled in the distributed variant.

5 The user interface

5.1 Visual and operational aspects

ROSMP is a commercial software product, so the friendly and comfortable user interface is the indispensable condition of its visual representation. We are strongly orientating to the inexperienced user. We use the standard CUI supported by Borland's Turbo Vision. The menu, dialogs, and the system behavior are as simple as possible. ROMSP is so simple that it has not the hypertext help. We found quite enough suggestions in the status line.

In the menu line we have the following items:

System – a submenu with two items: "About" and "Refresh display".

File – a dialog to start the file checking.

Directory – a dialog to change the current directory.

Word – a dialog to check the correctness of a single word.

Exit – to exit from the spelling checker.

Colors – a dialog to select the color palette (for color, black and white, or LCD screen).

As the user starts a file checking, he/she sees on the screen the text being scrolled. When a misspelled word is found, the scrolling stops, the word is marked, and a word correction dialog appears. It shows the message[1]: "Word ... was not found in the dictionary" (we had rejected messages like "Error" or "Wrong word" on psychological backgrounds). User has the following possibilities:

Add – add this word to a file to expand further his/her private vocabulary.

Good – consider this word to be good during the current session.

Suggestion – get the suggestion list of similar words from the vocabulary.

Edit – correct the word manually.

Mark – mark the word with the asterisk to better found it further.

Skip – skip the word.

Word – check the correctness of another single word (same dialog as from "Word" menu item).

Stop – stop spelling checking.

[1]Of course all menus, suggestions and dialogs are in Roumanian.

During manual correction, the user has a possibility to edit the whole visible screen, however not inserting and deleting lines and without block operations. The user can not scroll the edited text. We suppose that the text is prepared by a separate editor which is to be used for radical changes, so we had not included a full-scale editor as a part of the spelling checker.

After the manual correction the scanning restarts from the topmost change.

If the user stops ROMSP, it is possible to save partial corrections, or to keep the file untouched. Saving a file, ROMSP always keeps its backup copy.

5.2 Event stream synchronization

It is the internal problem of the event driven, object oriented implementation of the system. In our case we have three event streams.

The first event stream flows from user keyboard and mouse activity. The second stream is produced by the checked text file scrolling, and the third stream consists of words proposed as user wants a suggestion to correct an error. The streams are to be synchronized with screen scrolling and user actions (e.g., user does not want additional suggestions as the search continues). Turbo Vision has not any mean to maintain such synchronization. We were to program it from the very beginning.

We had previewed several approaches to the solution of the problem.

The first approach is based on the overwriting of the Turbo Vision's `GetEvent` method initiating other event streams checkup when no events exist from keyboard and mouse. This approach is the least abstract, and it was implemented in the distributed ROMSP version.

We are also experimenting with the second approach based on the event queue concept. We generalize the notion of the event to form the queue. We define the composition of events, obtaining recursive compound event structure. The queue is a new independent object built into the environment.

For detailed discussion see [5].

6 Supporting tools

We include here implementation support, data base management tools, coding schemes support, and distribution tools.

6.1 Implementation support

MS-DOS version of ROMSP was implemented on Borland Pascal with Objects 7.01 using Turbo Vision, with little Turbo Assembler programming. We were to develop a toolkit to support the implementation. These tools include many extensions for Turbo Pascal and Turbo Vision, and some changes in run-time library modules.

All extensions for Turbo Pascal are designed as separate compilation units (modules). We classified them as `PT` (Pascal Tools) and `VT` (Vision Tools). E.g., the `PTARITHM` unit includes arithmetical functions like `Max` and `Min` which lack in the Pascal standard and are usually absent in Pascal implementations. The `PTSTRING` unit includes additional string routines etc. The `VTKEYBD` unit replaces the Turbo Vision keyboard driver additionally supporting the PC AT extended keyboard, etc.

All run-time library modules containing visible English texts were changed to show the corresponding Roumanian texts. For example, the "OK" button in dialogs was replaced by "Gata" ("Ready"). These changes had sometimes implied the corresponding changes in the dialog element layout. In the previous example the string "Gata" is longer than "OK" so the button is to be wider.

Several errors were corrected in the Turbo Vision run-time library modules.

6.2 Vocabulary management

We use a separate interactive program to maintain the vocabulary data base. This program has the following functions:

- initialize the base (create a new empty base);
- check the base integrity;

- compress the base;
- add a word;
- add words from a file;
- delete a word;
- search for a word;
- search for similar words;
- make triades;
- show the hash table;
- show information about pages;
- show last words from each page;
- output words page by page;
- output words in the single list;
- add a synonym/translation for a word;
- add synonyms/translations from a file;
- delete a synonym/translation for a word;
- show synonyms/translations for a word;
- exit.

Two functions need the explanation – "compress the base" and "make triades".

The compressing means that all words stored into the user vocabulary that is not such effective are moved to the main vocabulary. See the above discussion on the effective vocabulary representation in the sec. 2.

"Triades" is the table of all three-letter combinations existing in the vocabulary entries. It is organized as the array of bitmaps. We use it when generating suggestions (see sec. 3 above).

"Pages" are not printing pages but those vocabulary data base pages described above in sec. 2.

Synonym and translation support is the experimental part of our project. Now it is not included into the distributed version.

The output list of words can be used later to reconstruct the whole vocabulary from scratch. We had also used it to collect some statistical information.

6.3 Coding schemes

Code pages 852 (MS-DOS) and 1250 (Microsoft Windows) support Roumanian extension of the Latin alphabet. Having the strong intention to follow these conventions, we had meanwhile met more than 15 other coding schemes and keyboard layout variants. We can mention the following reasons for the situation:

- Till now there are no national standards on the subject in Roumania and the Republic of Moldova.
- Typical MS-DOS software uses pseudograhpics character of the code page 437 and then conflicts with the code page 852.
- In the Republic of Moldova we are to use at once Roumanian and Russian languages, and English to communicate with software. The corresponding code pages (852, 866, and 437) also conflict.
- English, Roumanian and Russian typewriter and computer keyboard layouts pose most non-alphabetical characters on different keys. It is very unpleasant to find "?" or "+" on three different keys depending on the currently used language.

We use some unified internal code and convert user text to it and vice versa. Every distribution includes screen and keyboard drivers supporting the user's coding scheme and keyboard layout.

Due to known recommendation of the Roumanian Academy on letters Â and Î we have now two orthographical variants for many words. We support both variants through different vocabulary data bases.

6.4 Distribution

We use Turbo Pascal conditional compilation directives to generate the distributive version supporting the particular coding scheme and keyboard layout. When the user's coding is new, we extend our driver library with new modules and add a new condition in the corresponding Pascal module. A special condition switches to the compilation of the demonstrative version.

When the version is ready, a special program generates its distributive diskettes. It copies files on diskettes and calculates their CRC-32 check codes. When all files are placed on diskettes, the file names and the check codes are included in the Pascal source code of the installation program, the resulting text `SETUP.PAS` is compiled, the module `SETUP.EXE` is extended by its own CRC-32 and is copied to the diskette. The first diskette of the distribution set is thereby formed the last.

When installing ROMSP, the setup program performs the self-test checking its own CRC-32. Then the setup program asks the user about the disk and the directory to place the system, creates the directory and copies files to it. After the first diskette with the `SETUP.EXE` module, the order of diskettes is free. The setup program marks all installed files and asks for a new diskette until all files from the list were copied.

After copying files, the setup program tries to tune the MS-DOS starting scripts `CONFIG.SYS` and `AUTOEXEC.BAT`. The successful installation is concluded by reloading MS-DOS.

7 Further development

Except features mentioned above we had experimented with our system to test some variants of its possible development. We list here two more experiments:

- We had formed the vocabulary for the English language. Our system worked quite well.

- We tried to introduce the concept of user selected filters to read and to write the processed text. Filters may solve problems with coding schemes, and with texts prepared for a word processor and not being plain ASCII. We had experimented with AMI PRO texts.

General directions of further development are:

- Keeping more information for a word including style, semantics, synonyms, translations to other languages, grammatical correlations etc.

- Correcting not separate words but also word combinations using advanced NLP[2] techniques.

- Multilanguage system using several vocabularies.

- ROMSP for Windows; the problem of interacting with a lot of Windows word processors arises.

- Generalizing the word inflection program to became a tutorial one.

- Improving implementation, support and distribution tools.

- Using known compression techniques to reduce vocabulary file size, etc.

We also undertake investigations using the accumulated lexicographical material. E.g., we intend to create the Roumanian hyphenation table for TEX.

[2]Natural language processing.

Acknowledgements

ROMSP is the spelling checker for the Roumanian language developed at the Institute of Mathematics (Chișinău, Moldova). This article summarizes efforts of the ROMSP project team. Let us mention here other project participants (alphabetically): Dr. Elena Boian, Dr. Constantin Ciubotaru, Dr. Svetlana Cojocaru, Dr. Anna Danilchenco, Valentina Demidova, Michael Evstunin, Ludmila Malahova, Dr. Natalia Shvets, Ludmila Topal, Dr. Victor Ufnarovski, Tatiana Verlan.

We thank our Roumanian colleagues: Dr. Dan Tufiș (București) and Prof. Dr. Nicolae Curteanu (Iași) for the help and fruitful discussions.

We especially thank mr. Șerban Mihaescu (USA) for the delivered software.

References

[1] E. Boian, A. Danilchenco, L. Topal
The automation of speech parts inflexion process
Computer Science Journal of Moldova, vol. **1**, nr. 2 (2), 1993

[2] E. Boian, A. Danilchenco, L. Topal
Automation of word-forming process in the Roumanian language
Studies in Informatics and Control, vol. **3**, nr. 1, March 1994

[3] S. Cojocaru, M. Evstunin, V. Ufnarovski
Detecting and correcting spelling errors for the Roumanian language
Computer Science Journal of Moldova, vol. **1**, nr. 1 (1), 1993

[4] S. Cojocaru, M. Evstunin, V. Ufnarovski
Roumanian spelling-checker
Studies in Informatics and Control, vol. **3**, nr. 1, March 1994

[5] A. Colesnicov, L. Malahova
Event synchronization in object oriented environment – a case study
9th International Conference on Control Systems and Computer

Science CSCS9
București, România, 25–28 May 1993. Vol. **II**

[6] A. Colesnicov, L. Malahova
An environment for a language processing program – the Roumanian spell checker
First Conference on Applied and Industrial Mathematics. Oradea, România, 3–5 Sep. 1993

Lexicographical sources

[7] Dicționarul explicativ al limbii române
[The Roumanian language explanatory dictionary]
București, 1975 (in Roumanian)

[8] Supliment la dicționarul explicativ al limbii române
[Supplement to the Roumanian language explanatory dictionary]
București, 1988 (in Roumanian)

[9] F. Marcu
Mic dicționar de neologisme
[The small neologism dictionary]
București, 1985 (in Roumanian)

[10] A. Lombard, C. Gadei
Dictionnaire morphologique de la langue roumaine
[The morphological Roumanian language dictionary]
București, 1981 (in French)

[11] Dicționar ortografic cu elemente de ortoepie și morfologie
[The orthographical dictionary with elements of orthoepy and morphology]
Chișinău, 1991 (in Roumanian)

[12] F. Șuteț, E. Șoșa
Dicționar ortografic al limbii române
[The orthographical Roumanian language dictionary]
București, 1994 (in Roumanian)

[13] V. Bucur
Scurt dicționar de teorie a evidenței contabile rus-român și român-rus
[The short Russian-Roumanian and Roumanian-Russian dictionary on bookkeeping]
Chișinău, 1992 (in Russian and Roumanian)

[14] Mic dicționar rus-român de termini economici
[The small Russian-Roumanian economical terms dictionary]
Chișinău, 1991 (in Russian and Roumanian)

[15] C. Tănase
Dicționar de terminologie financiară rus-român
[The Russian-Roumanian financial terms dictionary]
Chișinău, 1992 (in Russian and Roumanian)

[16] M. Carauș
Mic dicționar de termini de economie
[The small economical terms dictionary]
Chișinău, 1990 (in Russian and Roumanian)

A.Colesnicov,
Institute of Mathematics,
Academy of Sciences of Moldova,
5 Academiei str., Kishinev,
277028, Moldova
phone: (373–2) 738058
e–mail: kae@math.moldova.su

Received 3 January, 1995