A PARSER-BASED TEXT PREPROCESSOR FOR ROMANIAN LANGUAGE TTS SYNTHESIS

Dragoş BURILEANU, Claudius DAN, Mihai SIMA, Corneliu BURILEANU

«Politehnica» University of Bucharest, Faculty of Electronics and Telecommunications,
Blvd. Iuliu Maniu 1-3, Sector 6, Bucharest 77 202, ROMANIA.
E-mail: bdragos@mESsnet.pub.ro

ABSTRACT

Text preprocessing plays an important role in a text-to-speech (TTS) synthesis system. The correct detection and interpretation of input strings influence the overall system accuracy and contribute to the conversion of an unrestricted text into synthetic speech. This paper describes the design philosophy of a preprocessing module for a TTS system in Romanian language. The preprocessor is implemented using the standard *flex/bison* lexer and parser generators. The paper discusses the text preprocessing task and the major difficulties connected with Romanian language, proposes a set of definitions and rules, gives some implementation details and concludes with a few considerations about the TTS system and performances of the preprocessing module.

1. INTRODUCTION

For many years, natural sounding text-to-speech synthesis has been an important goal for speech researchers. However, even if many commercial TTS systems have now reached a satisfactory quality, we are still far from obtaining a machine able to read as fluent, intelligible and natural as a human speaker [3]. A lot of research is to be done to convert *any* text into natural speech in a given language.

Usually, a TTS system comprises a *natural language processing* stage, able to produce a phonetic transcription of the input text, together with information about intonation, stress and duration, as well as a *signal processing* stage, which transforms the symbolic information it receives into speech [1], [2], [4].

The basic modules of a (high quality) natural language processing stage are depicted in figure 1.

- The *text preprocessor* converts the input text into a standard orthographic form, suitable for further processing. All the words in the current sentence, including abbreviations, acronyms, number sequences, time expressions and dates, are identified and possibly transformed into full text.
- The *morpho-syntactic analysis* finds the part of speech categories for each word and also the syntactic structure of text sentences.
- The *letter-to-phone conversion* transforms the orthographic character sequences into phonetic ones (usually allophone sequences).
- The *prosodic analysis* assigns melodic contours to the sentence and duration values to each acoustic segment.

Usually, written texts are presented as strings of ASCII characters; they consist of orthographic words, and also of symbols such as punctuation marks, number strings, or mathematical operators. One may encounter numerals (for example, in Romanian, *al 2-lea – 'the 2nd'*, *24*, *24.530*, *2,453*), abbreviations (*Prof.*, *ms*, *ing.* – for *'engineer'*, *dl* – for *'mister'*), or acronyms (*IBM*, *S.R.L.*, *TTS*). These strings may be considered "anomalous" (from a linguistic point of view) relative to the majority of text words, and have to be converted into a suitable orthographic format before any other linguistic analysis. This conversion represents the task of the preprocessing module. This task also includes word and sentence boundary detection, as well as punctuation marks and special symbols processing [4], [6], [8], [9].

2. TEXT PREPROCESSING

Text preprocessing for a TTS system is a difficult task in any language. In Romanian language, the number of pronunciation problems and ambiguities is big enough; they include, for example, multiple functions for period, comma and colon, or different pronunciations for numerals – according to case, number and gender [2].

For our TTS system, we proposed a set of definitions and preprocessing rules, based on a detailed analysis of the most common use of punctuation marks, lower- and upper cases and digit sequences.

The basic proposed definitions are given bellow.

D1. A "word" is a sequence of lower case letters.

D2. An "expression" is a sequence of characters that contains one or more of the following categories: a letter sequence with at least one upper case; a digit sequence; punctuation marks; other special symbols.

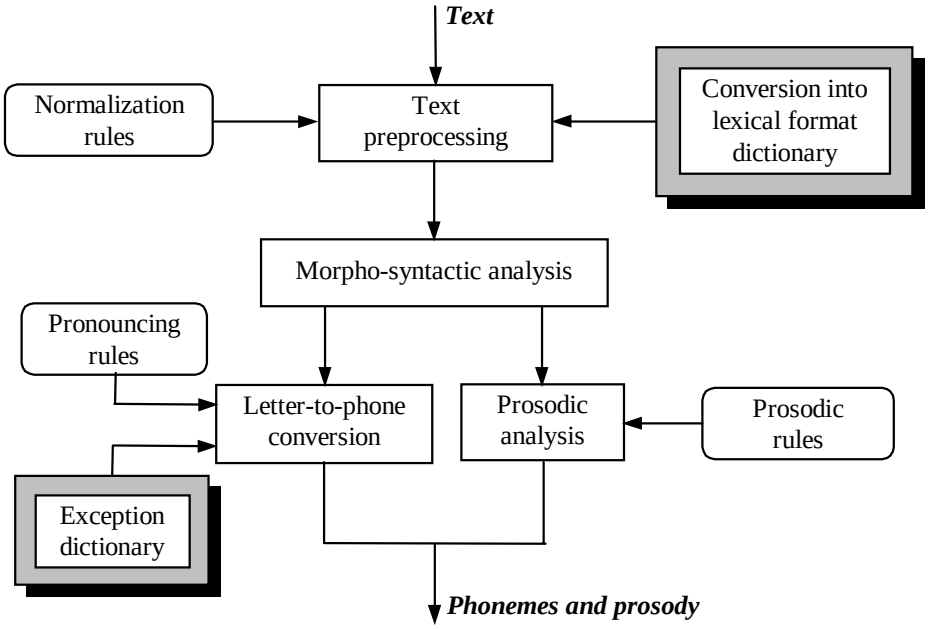

Fig. 1. The structure of the natural language processing stage in a TTS system

D3. An "extra-textual symbol" is a symbol performing a punctuation function.

D4. An "inter-textual symbol" is a symbol that belongs to an expression and helps to its spelling.

D5. "Expansion" is the process of conversion of an expression to its full textual form.

D5. An "ambiguous character sequence" is a sequence of characters that may affiliate with more than one linguistic class.

According with those definitions, we designed a preprocessing algorithm, which mainly consists of three basic steps:

I. Text segmentation

The input text is segmented into *character groups*, from left to right. We obtain strings of ASCII characters delimited by white space characters; punctuation is temporarily included into these groups.

II. Conversion of anomalous symbol strings into orthographic characters

We defined an original rule set for Romanian language regarding the detection of basic punctuation marks (*. , ? ! : ; ... - / ' " ( ) [ ] { }*) and mathematical operators, as well as numerals, abbreviations and acronyms expansion. The character groups obtained after the first step are transcribed into orthographic characters (when necessary), based on a crude contextual analysis at word/sub-word level, and a number of dictionaries for abbreviations and unusual acronyms.

III. Interpretation of certain punctuation marks

The text preprocessor also detects and memorizes the position of certain punctuation marks, in order to be used by morpho-syntactic and prosodic modules.

Next we give some fundamentals rules used for the correct detection and interpretation of periods.

Firstly, one must note that in Romanian language the period (.) may occur in abbreviations (e.g. *tel.*, *etc.*, *P.S.*), acronyms (*S.R.L. – for 'Ltd.'*), digit sequences (*1.234*, *1.234,567*, *21.03.1999*), in sequences of three periods indicating, for example, that a text fragment will be omitted (…), or may indicate the end of a sentence. Ambiguities created by period are a major problem for the preprocessing task, because it may represent either an "extra-textual" or an "inter-textual" symbol, or both. For example, the period after an abbreviation may also indicate the sentence-end.

The previous considerations suggest the procedure to be followed: *when a period embedded in a character group is detected, the context is investigated in order to take the appropriate decision*:

- if the period is surrounded by digits, the period is considered as an "inter-textual" symbol and further analysis is performed:
  - if another period, separated by two digits is detected, the sequence represents a date and is expanded accordingly (e.g. *21.03.1999 = douăzeci şi unu martie o mie nouă sute nouăzeci şi nouă*);
  - if the string of digits contains no other period or it contains other period(s) separated by three digits, it is a number (e.g. *1.234.567 = un milion două sute treizeci şi patru de mii cinci sute şaizeci şi şapte*);
- if there are three consecutive periods, it is an ellipsis and is replaced by the text '*puncte puncte*';
- if the period appears at the end of a sequence of lower/upper case letters and, possibly, other periods, the "expression" is looked-up in the corresponding dictionary and:
  - if the "expression" is found, the period is considered an "inter-textual" symbol and the

sequence is expanded to the form in the dictionary;
- if it is not found, but all letters in the "expression" are upper cases, it is an acronym and it is expanded to the sequential reading of their characters produced individually (e.g. *S.R.L. = se re le*);
- if none of the previous rules is meet, the period that should be in the final position of the "expression", is declared an "extra-textual" symbol and represents the sentence-end; its position is marked and sent to the morpho-syntactic and prosodic analysis modules.

But this analysis still can not solve the ambiguity of the case when the same period is used for both an abbreviation (or an acronym) and also comes at the end of a sentence (*etc.* and *ş.a. – 'and others'* are typical cases). In the above-mentioned situations, the period is memorized, in order to inform the subsequent modules of its status.

We must also emphasize that a special attention was paid to hyphens, mathematical expressions, proper names, and insertion of prosodic pauses for numerals spelling.

3. PREPROCESSOR IMPLEMENTATION

We implemented the preprocessor using the standard *lex*/*flex* and *yacc*/*bison* lexer and parser generators. These programs generate a very efficient C code for the preprocessor and also provide a standard and eloquent conversion rules description for the three tasks to be performed (described in Section 2).

The *flex*-generated lexer performs the input text segmentation and identifies "words" and "expressions" (see definitions in Section 2). "Expressions" are further separated according to their type (number of upper cases and inter-textual symbols), providing appropriate information to the parser. It also transforms numbers, dates and hours into their character sequences using separately developed C routines. Detection of such digit and punctuation strings that follow strict rules are best suited for lexer processing.

For examples, identification of integers, written with or without intermediate dots (e.g. *123456* ≡ *123.456*) is done by the *flex* code:

```
[0-9]+ {
    yylval.dval = atof(yytext);
    return INT;
  }

[0-9]{1,3}[.]([0-9]{3}[.])*[0-9]{3} {
    trimchar(yytext, '.');
    yylval.dval = atof(yytext);
    return INT;
  }
```

All "words" and "expressions" identified by the lexer are first looked up in abbreviation dictionaries according to their forms, and the corresponding information is provided to the parser. We use several dictionaries grouping abbreviations based on the modality they are written. For example, one dictionary contains only words that are written using lower case letters (e.g. *dna – 'misses'*), another contains words that use both lower- and upper case letters (e.g. measurement units like *Hz* and *mA*), another with dotted abbreviations (e.g. *P.S.*, *etc.*), etc. Using more than one dictionary and grouping words based on their respective writing manner accelerates the search process. For each abbreviation the dictionary indicates the correct spelling, including case, number and gender forms when needed. A similar processing sequence is used for acronyms.

The *bison*-generated parser identifies relationships between different types of words using the information provided by the lexer (numeral, abbreviation, acronym, etc.). This allows, for example, the correct spelling of a number followed by an abbreviated unit measure; note that in Romanian language, between a number greater than 20 and the numbered objects, an extra *'de'* must be inserted: *19 km = nouăsprezece kilometri* and *20 km = douăzeci de kilometri.* The number spelling also differs according to the gender of the object. The *bison* code that treats this situation is depicted next:

```
objects: number MEAS {
    fprintf(yyout, "%s",
       spellnum($1, NUMERAL, $2->gender));
    if($1 > 1)
     fprintf(yyout, "%s", $2->text_plur));
    else
     fprintf(yyout, "%s", $2->text_sing));
  }
  ;

number: INT  {$$ = $1;}
  | FLOAT {$$ = $1;}
  ;
```

Our `spellnum()` function (not shown here) is a C routine that translates a number into its correct sequence of characters.

Using *flex* and *bison* we were able to concentrate on specification issues, transferring parts of the C code generation task to the two programs. An important effort was devoted to balance the set of tasks to be done by each program.

An error recovery mechanism allows the preprocessor to be tolerant to some typical "syntax errors", like phrase beginning with a lower case letter, or end detection of input file before identifying a punctuation mark that concludes the phrase. Incorrect format for dates and numerals, as example, does not cause preprocessor abnormal termination; they are translated in their closest spelling form.

4. CONCLUSION

The TTS system built in our laboratory is based on a two level parser for preprocessing and morpho-syntactic/prosodic analysis, a powerful neural network letter-to-phone converter and a diphone-concatenation scheme; the system accepts Romanian language texts for input and generates good quality speech.

However, we must note that our preprocessor (together with linguistic analysis) can not solve all possible text ambiguities or ungrammatical constructions (even if, for example, unexpected special symbols are ignored).

We evaluated the overall system accuracy on a large set of newspaper and scientific texts; the number of errors due to the preprocessor module was under 1%.

The number of unusual character sequences in a text that must be translated into speech obviously depends on the type and subject of text and on the writing correctness [5], [7]. Since the writing rules are not widely known in detail by the majority of people, we think that a way to enforce correct writing is to include our preprocessor (or even our entire TTS system) in a spelling checker program that will let the user decide on any ambiguity.

ACKNOWLEDGMENT

We would like to thank *ECAS ELECTRO S.R.L. Bucharest* (*Electronic Components Distributor*) for partially funding the research activity that lead to the results presented in this paper.

REFERENCES

[1] BURILEANU, C., E. OANCEA, M. RADU, D. BURILEANU, J. ARHIP, and L. VASILESCU, "Text-to-Speech Synthesis for Romanian Language: Present and Future Trends". In *Recent Advances in Romanian Language Technology* (D. Tufis and P. Andersen - Eds.), Romanian Academy, Bucharest, pp. 189-206, 1997.

[2] BURILEANU, D., "Natural Language Processing for Speech Synthesis in Romanian Language". In *Proceedings of The 12th International Conference on Control Systems and Computer Science – CSCS12*, Bucharest, May 26-29, 1999.

[3] D'ALESSANDRO, C., M.G. RIZET, and P.B. DE MAREUIL, "Synthèse de la parole à partir du texte". In *Fondements et perspectives en traitement automatique de la parole* (H. Méloni - Coord.), Aupelf-Uref, pp. 81-96, 1996.

[4] DUTOIT, T., *An introduction to Text-to-Speech Synthesis*, Kluwer, 1997.

[5] FERRI, G., P. PIERUCCI, and D. SANZONE, "A Complete Linguistic Analysis for an Italian Text-to-Speech System". In *Progress in Speech Synthesis* (J.P.H. van Santen et al. – Eds.), Springer-Verlag, New York, pp. 123-138, 1997.

[6] FRIES, G., and A. WIRTH, "FELIX – A TTS System with Improved Preprocessing and Source Signal Generation". In *Proceedings of EUROSPEECH'97*, Rhodes, Greece, pp. 589-592, 1997.

[7] LIBERMAN, M.Y., and K.W. CHURCH, "Text Analysis and Word Pronunciation in Text-to-Speech Systems". In *Advances in Speech Signal Processing* (S. Furui and M. Sondhi – Eds.), Dekker, New York, pp. 791-831, 1992.

[8] LINDSTROM, A., and M. LJUNGQVIST, "Text Processing within a Speech Synthesis System". In *Proceedings of the ICSLP*, Yokohama, Japan, pp. 139-142, 1994.

[9] MOBIUS, B., R. SPROAT, J.P.H. VAN SANTEN, and J.P. OLIVE, "The Bell Labs German Text-to-Speech System: An Overview". In *Proceedings of EUROSPEECH'97*, Rhodes, Greece, pp. 2443-2446, 1997.