

Efficient Parsing of Romanian Language for Text-to-Speech Purposes

Andrei Şaupe¹, Lucian Radu Teodorescu¹, Mihai Alexandru Ordean¹,
Răzvan Boldizsar¹, Mihaela Ordean¹, and Gheorghe Cosmin Silaghi²

¹ iQuest Technologies, Cluj-Napoca, Romania
{Andrei.Saupe, Lucian.Teodorescu, Mihai.Ordean,
Razvan.Boldizsar, Mihaela.Ordean}@iquestint.com
² Babeş-Bolyai University of Cluj-Napoca, Romania
Gheorghe.Silaghi@econ.ubbcluj.ro

Abstract. This paper presents the design of the text analysis component of a TTS system for the Romanian language. Our text analysis is performed in two steps: document structure detection and text normalization. The output is a tree-based representation of the processed data. Parsing is made efficient with the help of the Boost Spirit LL parser [1], the usage of this tool allowing for a greater flexibility in the source code and in the output representation.

1 Introduction

Converting words from written form into speakable forms is a non-trivial process and it strongly influences the performance of a text-to-speech (TTS) system [2]. The text analysis component of a TTS system is generally responsible for determining the document structure, the conversion of non-orthographic symbols and the parsing of the language structure and meaning. It indicates all the knowledge about the text and reveals out the message that is not specifically phonetic or prosodic in nature and encodes it in a format easy to use in speech synthesis. The core element of the text analysis component is the text parser which translates and disambiguates the human language and conveys the meaning among a potentially unlimited range of possibilities.

In this paper the focus is on the text analysis component of a TTS system tailored for the Romanian language. As we are on the early stages of the development of a TTS system for the Romanian language, our goal is to produce an efficient parser of the written text and proper data structures to allow the tuning of the core unit selection component of our TTS system.

For the Romanian language, Burileanu et al. [3] presents a text preprocessor based on *lex/flex* and *yacc/bison* lexer and parser generators, including facilities like conversion of anomalous symbol strings into orthographic characters and interpretation of certain punctuation marks. Buza et al. [4] presents another text parser focused toward syllable detection, using the same *lex* tool as the basic lexer. R. Sproat [5] indicates the weighted finite-state transducer approach as being applied also for Romanian, but without entering the implementation details

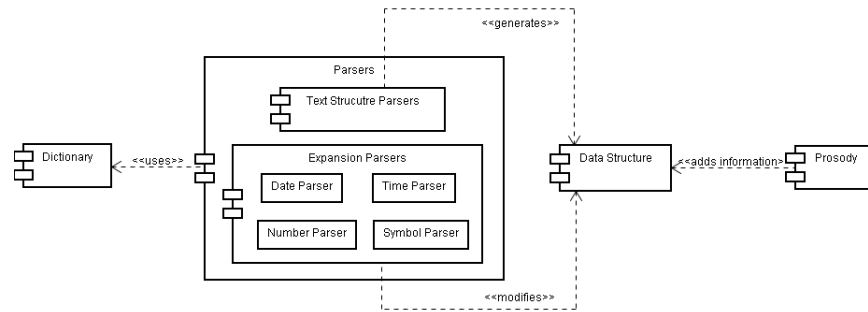


Fig. 1. The structure of the **Text Preprocessing** component

of the Romanian language. We go beyond this previous research by enhancing our parser with further computational efficiency and better output representation to allow a unit selection approach for the core speech synthesis. Our parser uses the power of the Boost Spirit LL parser [1] from the C++ Boost Framework³ [6], targeting rather ambiguous EBNF grammars, completely embedded in the source C++ code. Further, we resolve more text ambiguities than Burileanu et al. [3].

This paper is organized as follows. Section 2 details the text analysis component, emphasizing the document structure detection and text normalization. Section 3 presents the performance of our text processing component and discuss some specific features. Section 4 concludes the paper.

2 The Text Processing Component

In this section we enter the details of the **Text Processing** component of our TTS system, depicted in figure 1. This component works directly on the raw text and produce a suitable internal representation. It is further structured in several smaller sub-components: the **Dictionary**, containing the common words and exceptions; the **Parsers**, containing the parsing logic, taking as input the text and constructing the internal data structures representation of the text; the **Data Structure**, representing the storage of the processed text; and the **Prosody** component which analyzes the data structure and adds prosodic information accordingly.

The **Text Processing** component intensively uses the Boost Spirit LL parser [1]. Boost Spirit is an object-oriented recursive descent parser implemented using template meta-programming techniques in C++. It allows the programmer to describe the grammar in way similar with EBNF [7], while still writing C++ code. We design EBNF-like grammar specifications (presented further in this section) for matching the input text. When such a specification matches different predicates (word, phrases, paragraphs etc.), different methods are activated

³ <http://www.boost.org> (consulted on 20 March 2009)

for saving the matched structure and for building the internal tree structure representation of the text.

Building a big grammar specification for the Romanian language like Bureleanu et al. [3] allows to process all the text in only one traversal, but with the cost of a high coupling, high maintenance and low extensibility of the code. We improve this by employing a two phase parsing described in subsections 2.1 and 2.2, which better structures the data representation. In the first phase, we identify the text structure and in the second, we enter the ambiguities details and normalize the text. Thus, for almost similar processing costs, we succeed to write independent grammar rules for text structure and for normalization, which overcome the drawback of high coupling inside the big grammar of the Romanian language.

We construct a tree representation of the text, facilitating ease traversal and other general functionalities. We design two sorts of nodes: *structure nodes* and *expansion nodes*. The structure nodes are used to model the input text in paragraphs, phrases, words, etc. The expansion nodes are used in text normalization, to replace new text for the original input: E.g. if the raw input text is "2 e un număr", number '2' is an expansion node of type **Number** for which we will create the replacement text "doi". Figure 2 represents the tree structure produced by the parsing, together with the tree representation of the following Romanian sequence: "... astăzi este 2.10.2010, dar ...". To construct this tree structure representation, we basically employ two types of parsers: for text segmentation and for expansion of words. The text segmentation parser identifies the text structure while the expansion parsers are used on the structure nodes to identify special cases like numbers, time, date, currency, etc. They also attach an expansion node to the structure node when necessary.

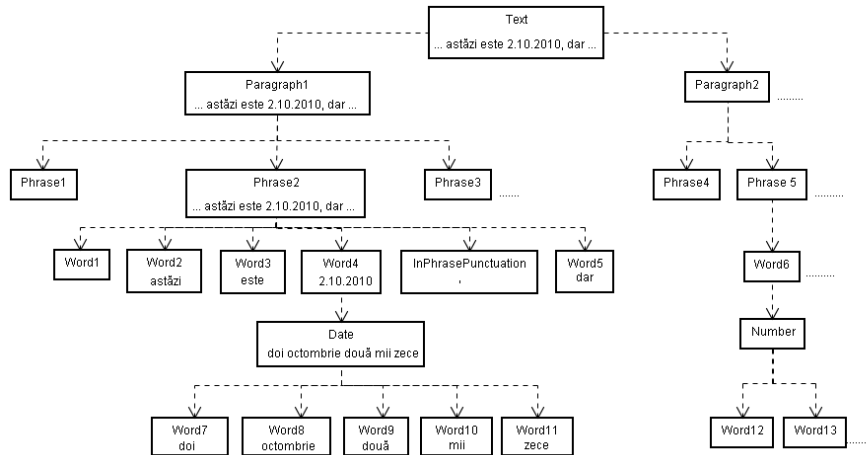


Fig. 2. The tree representation of a text

A paragraph is composed by a succession of phrases. Phrase boundaries (PHS) are signaled by terminal punctuation from the set: {., !, ?} followed by white spaces.

A phrase is a set of words, IPP (in phrase punctuation), SC (special characters) and PHS (phrase separators). The phrase could start with a word, several IPP or SC, could contain one or more words, IPP and SC separated by one or more WS (word separators) and could end with zero or more PHS separated by SS (sign separators). If the phrase starts with phrase separators, we encounter the case in which the paragraph is formed only by phrase separators. Another exception is the one where the first phrase starts with some phrase separators. In this situation, an empty phrase is made containing only that phrase separator. Because the parsing is performed in a greedy descend manner, any other phrase separators that are in front of a phrase will be taken by the phrase separators predicate of the previous phrase. If more phrase separators are encountered at the end of the phrase, they will be all considered to depend on that phrase.

A word could contain any character except special characters and word separators but could not start or end with any of the characters relevant for the other predicates: WS, PAS, PHS, IPP, SC. Special characters (SP) like '@' are considered as individual non words that will be expanded in the second phase of the text analysis. In-phrase punctuation (IPP) are also considered independently of words and they will modify the previous and next word token properties. In this way, a web page (e.g.) "www.dlalex.com" will constitute one word and will allow a later processing module to identify it as a web page.

2.2 Text Normalization

In this subsection we describe how the text analysis component further expands and normalizes each identified expansion node. Our text normalization deals with abbreviations, acronyms, number formats, phone numbers, dates, times, money and currency, account numbers, ordinal numbers, cardinal numbers, domain specific tags like mathematical expressions and chemical formulas and miscellaneous formats. No ambiguous text remains unexpanded after the text normalization. Even if the text does not match any normalization rule, it is expanded up to its letter decomposition. We will detail only the abbreviation, the number formats and the date and time, as they are the most important text normalization issues. After text normalization, we further perform the phonetic transcription and syllable construction for those words which are not identified in the main dictionary. These later steps are not in the scope of this paper.

An abbreviation is marked by a final dot ".". When a "." is encountered by the text parser, the previous word is looked-up in the dictionary for an expansion. If the word is found to be an abbreviation, it is expanded accordingly with the dictionary and the phrase ended by the dot of the abbreviation is linked with the next phrase in the case there is no additional punctuation mark after.

Next, we depict the EBNF-like syntax for numbers, date and time.

```

number ::= numberWS | numberWNS
numberWNS ::= (digit, {digit}) | (digit, {digit}, DES, digit, {digit})
numberWS ::= (digit | digit, digit | digit, digit, digit, {DIS, digit,
digit, digit}) | (digit | digit, digit | digit, digit, digit, {DIS, digit,
digit, digit}, DES, digit {digit})
DES ::= ","
DIS ::= "."
date ::= day, DMYS, month, (DMYS, year) | month, MDYS, day,
(MDYS, year) | day, DS, month | month, DS, year
day ::= ("0"|"1"|"2"|"3"), digit where day is less then 32 and greater then 0
month ::= (("0"|"1"), digit ) | writtenMonth where month is less then 13
and greater then 0
year ::= (digit - "0", digit) | (digit - "0", digit, digit, digit)
DMYS ::= "."
MDYS ::= "/"
DS ::= DMYS | MDYS
time ::= (hour, TS, minute) | (hour, TS, minute, TS, second)
hour ::= ("0"|"1"|"2"), digit where hour is less then 25
minute ::= ("0"|"1"|"2"|"3"|"4"|"5"), digit where minute is less then 60
second ::= ("0"|"1"|"2"|"3"|"4"|"5"), digit where second is less then 60
TS ::= ":"

```

Generic number formats or cardinal numbers are mainly used in simple counting or the statement of amounts. A complex number format can be divided in several cardinal number tokens and treated individually. This is usually the case for most complex number formats but not all; for example the month of a date is better to be expanded in the word specifying the month than in a number. Generic number formats in the Romanian language are fitting in two categories: number formats that use the digit separator '.' for grouping groups of tree digits (**numberWS**) and the ones that do not use this kind of separators (**numberWNS**). For both categories the numbers can be real numbers which means that they can have decimals. A decimal is separated from the rest of the number by the decimal separator '.'. Algorithm 1 presents the verifications performed when expanding the generic numbers.

The rule for the date numbers includes two formats: the US form (month day year) and the Romanian form (day month year). We also treat the case when only the day and month are specified or only the month and year. The time format is recognized by the hour and minutes separated by the time separator (TS) and eventually followed by the seconds.

3 Evaluation

In this section we evaluate our **Text Processing** component against various inputs. We designed three test cases, covering spoken and scientific language. The first text case consists of full page excerpt from a Romanian literature book. The second test case is selected from a scientific paper from biology. To further stress the **Text Processing** component, the third test case represents the selected

Algorithm 1 Expanding the generic numbers

```

1. Check number type
if number token  $n$  has DES separator then
  separate number token  $n$  in integer token  $i$  and decimal token  $d$ 
  perform decimal expansion
else
  set number token  $n$  as integer token  $i$ 
  perform integer expansion
end if
2. Decimal expansion
if decimal token  $d$  starts with digit 0 then
  perform decimal expansion digit by digit
else
  perform decimal expansion from right to left in groups of tree digits
end if
3. Integer expansion
if integer token  $i$  starts with digit 0 then
  perform integer expansion digit by digit
else if integer token  $i$  has DIS separator then
  apply integer expansion from right to left in groups of tree digits separated by DIS
else
  perform integer expansion from right to left in groups of tree digits
end if
4. Advance to next number and GOTO step 1.

```

text from the biological scientific paper appended with several paragraphs full of terms to be disambiguated. Table 1 summarizes the results. We counted how many paragraphs, phrases and words were correctly identified or disambiguated by our parsers. Table 1 depicts both the percentage and the successful hits.

We note that the parser gives a good performance in identifying the text structure. For specific excerpts (test cases 2 and 3), paragraphs and phrases were fully identified. A worse performance is reported only when identifying phrases on the book excerpts (spoken language). But, even in this case, words were identified with a pretty good accuracy. As properly identifying the words is our main target, because speech synthesis will be done mainly at the word level, we can conclude that our Text Processing component fulfill the requirements of a TTS system.

Further, we should note that using the Boost Spirit technology allowed us to write small pieces of grammar specification, easily embedded into the C++

Table 1. Performance evaluation of the **Text Processing** component

Test case	Paragraphs	Phrases	Words
1. Book	100%, (5/5)	90.56%, (48/53)	97.45%, (996/1022)
2. Scientific paper	100%, (10/10)	100%, (32/32),	99.30% (713/718)
3. Scientific paper plus extra	100%, (12/12)	100% (37/37)	96.85% (863/891)

source code. Thus, we opted for a higher degree of transparency and independence between the source code and the grammar specification. As opposed to Burileanu et al. [3] which implemented a highly coupled parser, our approach allows us to quickly write new pieces of grammar specification and embed them into the system, when some exceptions are encountered during the TTS system usage.

4 Conclusion

In this paper we present the design of the text analysis component of a concatenative TTS system for the Romanian language. As our overall goal of the TTS system is a high speech accuracy through unit selection, we employed a two phases text analysis. First, we present the design of a parser that identifies the text structure. Next, we enter the details of the text normalization component that performs abbreviation disambiguation, number, date and time expansion and other normalization specific issues. Text processing is made efficient with the help of the Boost Spirit parser [1], targeting a tree-based data representation.

As a further work, we will describe the letter-to-sound conversion and the main unit selection component of our TTS system, emphasizing the use of intelligent tools to achieve a good speech accuracy.

Acknowledgements. iQuest TTS system is partially supported by the Romanian Authority for Scientific Research under contract INOVARE no. 186/2008.

References

1. de Guzman, J., Kaiser, H., Nuffer, D.: Spirit. <http://spirit.sourceforge.net> (consulted on 18 March 2009)
2. Huang, X., Acero, A., Hsiao-Wuen, H.: Spoken Language Processing: A Guide to Theory, Algorithms, and System Development. Prentice Hall (2001)
3. Burileanu, D., Dan, C., Sima, M., Burileanu, C.: A Parser-Based Text Preprocessor for Romanian Language TTS Synthesis. In: Proceedings of the 6th European Conference on Speech Communication and Technology (Eurospeech '99). Volume 5., Budapest, Hungary (September 1999) 2063–2066
4. Buza, O., Todorean, G., Bodo, A.Z.: Syllable detection for romanian text-to-speech synthesis. In: Sixth International Conference on Communications COMM'06, Bucharest, June, 2006. (2006) 135–138
5. Sproat, R.: Multilingual text analysis for text-to-speech synthesis. *Natural Language Engineering* **2**(4) (1996) 369–380
6. Karlsson, B.: Beyond the C++ Standard Library: An Introduction to Boost. Addison Wesley, (2005)
7. : ISO/IEC 14977 : 1996(E), Extended BNF. ISO (1996)