An Advanced NLP Framework for High-Quality Text-to-Speech Synthesis

Catalin Ungurean
Faculty of Electronics, Telecommunications and IT
University "Politehnica" of Bucharest
Bucharest, Romania

Dragos Burileanu
Faculty of Electronics, Telecommunications and IT
University "Politehnica" of Bucharest and
Romanian Academy Center for Artificial Intelligence
Bucharest, Romania
dragos.burileanu@upb.ro

***Abstract*—In order to build a TTS (Text-to-Speech) synthesis system one must provide two key components: a NLP (Natural Language Processing) stage, which essentially operates on the input text, and a speech generation stage to produce the desired output. These two distinct levels must exchange both data and commands to produce intelligible and natural speech. As the complete TTS task relies on many distinct scientific areas, any achievement toward standardization can minimize the effort and increase the dynamic of the results.**

This paper gives an overview of the NLP stage in the TTS system for Romanian language built by our collective, and describes the integration into the system of SSML (Speech Synthesis Markup Language), as a nowadays well recognized standard for TTS document authoring and inter-modules communication.

Keywords - TTS; NLP; SSML; diacritic restoration; lexical stress assignment; syllabification; letter-to-phone conversion

I. INTRODUCTION

Our research collective is currently developing a high-quality TTS synthesis system in Romanian, based on acoustic segment concatenation and multiple instances of non-uniform speech units. However, building such a system for an under-resourced language such as Romanian involves many theoretical and technical challenges, especially at the NLP level, which is highly language-specific. The NLP stage must extract from the input text sufficient linguistic information as an essential condition to obtain a true naturalness of the synthesized speech [1].

The main work was split in two different directions and a successive number of achievements has been accomplished, regarding both NLP and speech generation techniques. Thus, we implemented different linguistic processing methods at the NLP level and also two different speech engines, using first TD-PSOLA (Time Domain – Pitch Synchronous Overlap Add) [1] and afterwards HNM (Harmonic plus Noise Model) [2].

The link between the NLP and the speech generation stages can be done by means of a markup language, either proprietary or standard. As SSML already became a *de facto* standard for TTS, we finally built both stages in such a manner that it can easily integrate most important SSML elements and attributes.

A text in a given language is written within a standard orthographic and grammatical rules set and there are a great number of constraints that the NLP stage tries to decode. For example, a good TTS system must first transform the grapheme string into a adequate phonetic representation. At the same time, from the input text one should extract enough prosody information for the following TTS stages so as the synthesized speech message could sound natural and correctly transmit the original input message. Consequently, the NLP stage has mainly a double role: the phonetic transcription of the input text and the assembly of contextual and phonetic–prosodic information.

Our TTS system's most important NLP sub-stages are: diacritic restoration, preprocessing and normalization (which directly operate on the input text), syllabification, letter-to-phone (L2P) conversion, and lexical stress positioning. Other modules, such as homograph extraction, corpora size reduction, proper name database phonetic translation, or lexicon assembly are not discussed here. As we will show further in this paper, we obtained results that are superior to those reported in the literature for the NLP tasks in Romanian.

An important idea that we want to highlight is that a good quality TTS system needs to accept an unrestricted text as input, while the synthesized message must sound highly natural. Therefore, the NLP endeavor must describe a language by collecting enough relevant linguistic information, so as the resulting language model to be able to offer, at run time, properly decoded phonetic and prosodic aspects from the input text. In Romanian, this language modeling is not an easy task, neither technically nor theoretically. It mostly suffers from the lack of a comprehensive linguistic framework specially dedicated to TTS, being able to compare to the state of the art in this field. As Romanian has its own particularities, it is not always possible to benefit from the achievements of the other, possibly better represented languages. A NLP framework for a Romanian TTS system would assume the existence of a complete set of linguistic studies and instruments such as phonetically translated dictionaries, speech labeled corpora, acoustic, phonetic, and prosodic related studies, etc.

Obviously, a powerful NLP stage alone does not guarantees a very good text pronunciation, but at the same time a poor

978-1-4577-0441-3/11/$26.00 ©2011 IEEE

phonetic and prosodic representation of the input text can significantly lessen the quality of the synthesized speech. For example, even the only commercial TTS system in Romanian, namely the Polish system *IVONA TTS* [3], clearly suffers from a lack of linguistic expertise in the NLP stage building.

For this reason, our team made great efforts to improve continuously all the NLP modules, by using those methods which can lead to the best possible results, while the lack of enough adequate language resources (as mentioned above) is a difficult obstacle to overcome.

The paper is structured as follows: in Section II we present the best results reported in the literature so far, at different NLP sub-stages, and describe how the training and testing corpora for different modules in our NLP stage were built; in Section III we briefly explain the most important NLP algorithms developed and discuss the main results obtained; Section IV describes the philosophy of integrating the SSML within our TTS system, and the final section is reserved for conclusions.

II. NLP ISSUES

A. NLP Methods and Results for Romanian TTS

Nowadays achievements in the field of NLP for the TTS synthesis task have been accomplished through different approaches which can be grouped in the following three main categories: rule-based, statistical, and hybrid. We briefly present the most significant results in Table I (in terms of WER – Word Error Rate), emphasizing on those obtained for Romanian language; they will be used in Section III for comparison with our own results.

Because the methods applied in our TTS system basically rely upon large collections of Romanian texts, both for the training and the testing sides of the algorithms, this section will explain further this fundamental aspect from the perspective of our implementation, which is mainly *n*-gram based; we will emphasize only the following tasks: diacritic restoration, syllabification, lexical stress positioning, and letter-to-phone conversion.

TABLE I. DIFFERENT NLP APPROACHES RESULTS

NLP Field	Method	Language	WER (%)	Reference
Diacritic restoration	Statistical	Romanian	2.6	[4]
Diacritic restoration	Statistical	Czech, Hungarian, Polish, Romanian	3	[5]
L2P	Rule-based	Romanian	4.79	[6]
L2P	Rule-based	Romanian	22 - 5.2	[7]
Lexical stress assignment	Rule-based	Romanian	6	[8]
Syllabification	Rule-based	Romanian	15	[9]
Syllabification	Hybrid	Romanian	10.55	[6]

B. Building the Corpora for Diacritic Restoration

One of the most important drawbacks encountered in our work, from the perspective of an *n*-gram statistical NLP approach, is the lack of training and testing data. Other languages such as English or French may benefit from large collection of texts. As we started the exertion with an *n*-gram type NLP algorithm for a diacritic restoration task, built at word level, we encountered a well known phenomenon which is *data sparseness*. This usually happens when there is not sufficient training data. The corpus size grew by adding more text, extracted from newspapers, literature and Internet, and at present we have a medium-sized corpus of almost 10 millions of words. This dimension is only apparently large, compared with other word-level corpora, built for high-resourced languages. In fact, we have already observed that for our *n*-gram greedy algorithm the corpus size is insufficient even for a history of 3. For example, the German Reference Corpus has 4 billion words, the Oxford English Corpus has almost 2 billions, and the Russian Corpus has 150 millions of words that are automatically lemmatized and POS-/grapheme-tagged.

From this (offline) training corpus we extracted *unigrams*, *bigrams* and *trigrams*, the last being built with the final letters of every training word. Also, from the same data, we enlarged a first version of our Romanian electronic dictionary (which initially had roughly 65,000 words), with many directly extracted inflected forms. This database served for the diacritic restoration algorithm. At present the dictionary counts more than 330,000 different words and it was manually corrected.

We tested our method with three different Romanian written test sets (see Table II), arbitrary extracted from the European Parliament Proceedings Parallel Corpus [10]. The final performance was calculated as the average of the three results.

C. The Corpora for Lexical Stress Assignment

In many languages, lexical stress is a fundamental component for the pronunciation of a word; hence its correct placement is a major task in prosody prediction and generation for high-quality TTS synthesis systems. The Romanian language has no regular position for lexical stress, as for example Hungarian (the stress is always on the first syllable) or French (the stress is always on the last syllable). For this reason the stress resolution task is not at all straightforward.

The dictionary obtained from the diacritic restoration work was further prepared by manually placing lexical stress marks before the stressed vowels. Then, we freed this resulted corpus (named T) of homographic forms by automatically eliminating words having two or more stressed variants; the homographs extracted from T were used to build a distinct dictionary. In addition, the monosyllabic words (considered as carrying no stress) were discarded by a syllabification algorithm. Finally, the T dictionary was randomly divided into three different parts T1, T2, and T3, according to Table III.

TABLE II. DIACRITIC RESTORATION TEST SETS

Test set	TS1	TS2	TS3
Words	119,696	716,693	2,151,278

TABLE III. THE CORPORA FOR LEXICAL STRESS ASSIGNMENT

Corpus	Name	Size (%) of T	Size (in words)
Training	T1	75	234,700
Held-out	T2	10	31,300
Testing	T3	15	47,000

T1 was used to train the *n*-gram models, built at the character level, T2 has the role of held-out data, and T3 was used for testing. Evaluation was performed by automatically comparing the results obtained in different testing conditions on the T3 corpus without stress marks and on the complete (annotated) T3 corpus.

D. The Corpora for Automatic Syllabification

This algorithm works also on a statistic model built at character level. We started the experiments with an initially hand-made dictionary which has 11,000 correctly syllabified words, from previously endeavors of our team. Although we initially expected the algorithm to saturate more slowly, we noticed that the results could be further improved with more training data. We continued the experiments, and extended the training dictionary, in a sequential manner. Our final corpus is composed of 56,284 correctly syllabified words, from which we extracted a 20% test set.

E. L2P Testing Corpus

We tested the proposed L2P conversion algorithm on a manually-corrected test corpus which has 11,819 different words transformed in their phonetic forms. As the method is a rule-based one, no further data was necessary except the correctly syllabified words which are provided by the automatic syllabification module.

III. MAIN RESULTS OF THE PROPOSED NLP ALGORITHMS

A. The Diacritic Restoration Algorithm

Our statistical method, described in detail in [11], works on a successive multi-level technique and uses a minimal filtering principle, each level trying to eliminate the variants (generated by the algorithm) which are considered to be wrong. Another issue which we consider to be new is that one of the processing levels was built with *n*-grams extracted from the termination of words (the last four characters), as it was experimentally noticed that most of the diacritic ambiguities appear in this area. Due to the size characteristics of our corpus, we used word level bigrams, and trigrams obtained with the final four characters of each word. A final unigram level directly calculates the probability of the remaining variants and returns the one with the maximum value.

Our studies showed that in Romanian the most significant diacritic restoration problems are those created by an ambiguity between the characters *a* or *ă* in the final position of the words. This phenomenon appears most frequent in some short function words as *ca* (to) – *că* (that), *sa* (his) – *să* (to), *sau* (or) – *său* (his). Another very important errors' class is represented by feminine nouns which can occur in an articulated or non-articulated form: *casa* (the house) – *casă* (house). The best accuracy results were obtained for *i/î* class – 99.93%, while the worst results were obtained for the *a/ă/â* ambiguity class – 98.22% (both calculated at character level).

We consider that our results are superior to those reported in the literature (see Table I in Section II – *A*) and can be reported as a 1.4% WER and 99.6% accuracy at character level.

B. The Lexical Stress Assignment Algorithm

We proposed a statistical approach to lexical stress assignment for Romanian language TTS synthesis [12]. The method is essentially based on *n*-gram language models built at character level and uses a modified *Katz backoff smoothing* technique to solve the problem of data sparseness during training. Monosyllabic words were considered as carrying no stress and were separated by an automatic syllabification algorithm. The best result of 0.89% WER was obtained for a history length of eight characters. We further analyzed the errors returned and observed that almost 60% of these have the output probabilities with rather similar values. For the TTS task it proved to be useful to let unstressed the words where the returned probabilities are comparable within a threshold β, even if the overall accuracy slightly decreases.

It must be mentioned that in the present version of the TTS system, the lexical stress is positioned at the level of syllables to provide the proper prosody information for the speech generation stage. This position is further aligned during the L2P algorithm with the correct phonetic transcription.

C. The Syllabification Algorithm

The third NLP sub-stage that we are discussing here is a hybrid syllabification module. Automatic syllabification is generally necessary for correct prosody generation, but we also used it in the lexical stress assignment and L2P algorithms, as it is shown in this paper (Section III – *B* and *D*).

The syllabification algorithm is built using a mixed strategy, i.e. rules and statistics. We applied a minimal set of seven general rules, followed by a statistical approach, using character-based *n*-grams [13]. The best results were obtained for a history of 5 and are presented in Table IV.

Without applying the rule set, the results are inferior (the WER is about 5%) and the algorithm becomes time consuming for long words, as the number of possible generated variants increases exponentially. On the contrary, by applying only a general rule set, the results are dramatically lower, decreasing to an average of 25% WER, as reported in [14]. By using the mixed strategy we obtained an overall result of 2.96% WER.

TABLE IV. SYLLABIFICATION ALGORITHM RESULTS

n-Gram Length (history)	WER (%)
2	28.62
3	15.54
4	6.30
5	**2.96**
6	3.92

D. The L2P Algorithm

The phonetic transcription module is essential for any TTS system, as the pronunciation of words significantly differs from their spelling in many languages. In particular, the lack of detailed linguistic and phonetic studies and the numerous pronunciation difficulties, make this task very difficult for the Romanian language.

Since we have previously solved both automatic syllabification and lexical stress assignment, the idea of further using information from these two modules, normally appeared. To better understand how the proposed L2P method works, after a correct syllabification process, we first figure out in Table V the possible pairs of vowels which can occur before the phonetic conversion; we mention that in Table V *h* marks a possible hiatus, ↓ and ↑ mark descendant or ascendant diphthongs.

It must be mentioned that the L2P conversion in Romanian is not straightforward; several graphemes have multiple phonetic values, and also many difficulties arise from diphthongs or triphthongs. Despite these facts, we observed that eliminating ambiguous hiatus versus diphthong/triphthong occurrences, by a correct syllabification process, could lead to a clear, unambiguous condition.

A similar analysis was made for triphthongs resulting in an overall 45 rules-set for the L2P conversion from graphemes, while an ambiguity still remains for the vowels pair "*iu*" inside the same syllable, that could ambiguously be treated both as an ascendant or as a descendant diphthong [13]. At this point, we introduced lexical stress information for a further disambiguation process, as the presence of lexical stress near any of these vowels can successfully solve the L2P conversion. Although we took into consideration lexical stress positioned at the level of vowels, we must emphasize that for the final conversion, the entire stressed syllable is marked.

Our approach resulted in a 3.01% WER, where any single wrongly converted grapheme was taken into account as being an error. A detailed analysis of the L2P conversion results, made for every incorrectly-returned phone proved that a number of phonetic conversion errors can still be considered as acceptable as the perceived quality of the synthesized speech is almost not affected by these transcription errors. This observation led to a final L2P conversion result of about 2% WER.

TABLE V. AMBIGUOUS VOWELS PAIRS WITH DIPHTHONG AND HIATUS

	a	**e**	**i**	**o**	**u**	**ă**	**î/â**
a	*h*	*h*	*h* ↓	*h*	*h* ↓		
e	*h* ↑	*h*	*h* ↓	*h* ↑	*h* ↓		*h*
i	*h* ↑	*h* ↑	*h* ↓	*h* ↑	*h* ↑ ↓		*h*
o	*h* ↑	*h*	*h* ↓	*h*	*h* ↓		*h*
u	*h* ↑	*h* ↑	*h* ↓	*h*	*h*	*h* ↑	*h* ↑
ă			*h* ↓	*h*	*h* ↓		
î/â			*h* ↓		*h* ↓		

IV. INTEGRATING SSML INTO THE TTS SYSTEM

In older TTS systems it was common for each developer to have its own type of inter-modules communication language, but the main disadvantage was that the proprietary signaling messages would mean nothing to a different TTS system. In order to approach this drawback, W3C elaborated SSML recommendations, which are designed to provide an XML-based markup language for assisting the generation of synthesis speech in a variety of applications. SSML offers a large spectrum of voice control instructions as input for a TTS. Thus, speech attributes like prosody (pitch, durations, pauses and intensity), style or even overall voice can be easily modified at run time.

XML-based markup languages provide relatively high level markup functionality for speech synthesis input, and aim giving to a non-expert user the possibility to add information into a text in order to improve the way it is spoken. They are (at least in principle) independent of any particular TTS system which is assumed to parse the markup enriching its input and translate the information contained in it into a system-internal data representation format which in most cases is not XML-based.

The advantages of using SSML are well explained in [15]. The authors describe a unified framework for a multilingual TTS which uses SSML specifications as interface and suggest that such an approach brings standardization, interoperability, multilinguality, and extensibility.

As TTS society nowadays affinity is toward the use of XML as a *de facto* internal language, we have also augmented our TTS achievements with SSML standard specifications. More precisely, in this paper we show how our NLP stage is used for SSML authoring, in a way that provides the speech generation stage with the structural and production information required to create natural speech.

A. How to Embed the SSML Stack into a TTS Application

A TTS engine must be able to parse and translate markup commands into its internal language in order to control the synthesis process. Different lexicons with phonetic translations can be used, domain oriented or language oriented. IPA (International Phonetic Alphabet) is established as a *must have* attribute but other proprietary phonetic encodings are admitted. Even though we have used a proprietary phonetic code, in a final release IPA code will also be available.

The SSML stack embeds a number of distinct levels which are mapped on those generally recognized for a TTS system [16]. A text document provided as an input to the synthesis processor may be produced automatically or by human authoring, or through a combination of these methods. For example, in a typical multi-device voice access to web content, the web server has also the role of a HTML-to-XML transcoder which produces both visual and audio representations of web content, by means of HTML and VXML documents. The following are the six major processing steps undertaken by a synthesis processor to translate raw or (partially) marked-up text input into automatically generated

voice output: *XML parse*, *structure detection*, *text normalization*, *text-to-phone*, *prosodic analysis*, *and waveform generation*.

B. The Proposed *SSML Implementation*

In Fig. 1 we show how the SSML elements parse through the complete NLP stage in our TTS system. In our application, SSML strings stop before unit selection. Further SSML standard development or different proprietary implementation could continue XML string beyond this level.

To obtain a modular approach of the SSML stack, we separated the application task from the speech generation one, using SSML to exchange data and controls, both internally between different NLP modules or externally between the two already mentioned tasks. In our model the application is built upon the previously described NLP algorithms. Each distinct NLP module will add more significance to the textual or SSML string provided as an input in an incremental manner, aiming to obtain an appropriate phonetic and prosodic representation of the starting textual message. Such an approach can offer more flexibility because both the NLP and the speech generation engine can be independently replaced or upgraded anytime, or even distinct NLP levels can be easily replaced.

1) XML parse

To exemplify how our SSML stack is produced, we chose the following sentence:

D-le John, este ora 14:00, de ce nu mergeți la școală?
(Mr. John, is 14:00 o'clock, why don't you go to school?)

XML parse level must produce a well-formed SSML document, with a proper header (*xml version, lang* and *encoding*) and elements being added (<*speak*>). Raw text is converted to UTF-8, as an internal encoding for all NLP tasks and for SSML elements:

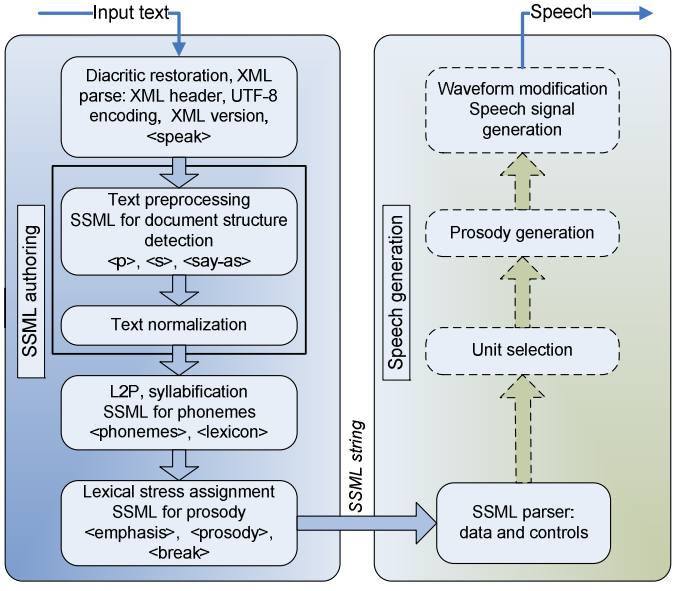

Figure 1. SSML implementation for NLP framework

```
<?xml version="1.0" encoding="UTF-8"?>
<speak version="1.1"
    xmlns="http://www.w3.org/2001/10/synthesis"
    xmlns:xhtml="http://www.w3.org/1999/xhtml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
http://www.w3.org/TR/speech-synthesis11/synthesis.xsd"
xml:lang="ro">

    D-le John, este ora 14:00, de ce nu mergeți la școală ?

</speak>
```

2) Embedding the SSML with the text preprocessor

Our preprocessor realizes two of the SSML tasks: *structure detection* and *text normalization*.

A. The *structure detection* means the extraction of the document structure, i.e. paragraphs and sentences. All the punctuation marks are detected, interpreted and preserved by the preprocessor. Many special written constructions (e.g., abbreviation, numerals, time, phone numbers, etc.) are detected and augmented using SSML <*say-as*> element:

```
<p><s>
   <say-as interpret as="abbreviation">d-le
         john, este ora
    <say-as interpret as="time"> 14:00,
         de ce nu mergeți la școală?
</s></p>
```

where <*p*> and <*s*> are the markup symbols for paragraph and sentence delimiters.

B. The *text normalization* stage will transform all the elements previously marked with <*say-as*> elements into the corresponding spoken forms. The previously text will be expanded as follows:

```
<p><s>
   domnule john, este ora paisprezece, de ce nu mergeți la
școală?
</s></p>
```

3) Using SSML to embed phones

Once the synthesis processor has determined the set of tokens to be spoken, it must derive pronunciations for each token, by means of a *text-to-phone* level.

SSML offers two distinct elements to parse phone strings:

- *phoneme* allows a phone sequence to be provided for any token or token sequence;
- *lexicon* is used for external definitions.

As we previously mentioned, our L2P method works together with the syllabification and lexical stress assignment algorithms. The syllabification will also be used to emphasize the corresponding stressed syllable for prosody generation. As L2P data output, we used the <*phoneme*> SSML markup element, with the *ph* attribute for the phonetic transcription.

We exemplify the phonetic transcription for the Romanian word *mergeți* (you go).

<*phoneme ph='mer7e#i'*>mergeți</*phoneme*>

Due to specific language constraints of our automatically built L2P conversion, we have stored the correspondence between graphemes and phones in a lexicon for some exception words (most of them require English pronunciation). A SSML lexicon must respect some constraints, described in PLS (Pronunciation Lexicon Specification) format, using a pair of *grapheme* and *phoneme* tags as it is exemplified below for the word *John* which does not respect Romanian phonology model:

```
<lexeme>
        <grapheme>John</grapheme>
        <phoneme>7on</phoneme>
</lexeme>
```

4) Using SSML to embed prosody

Our NLP modules offer the phonetic transcription of the input texts, together with various prosodic marks (lexical stress positions, sentence type, punctuation, etc.), syllable delimiters, and also other linguistic information used in the unit selection procedure. This information is provided at the NLP output and further used by the speech generation stage. For example, the stressed syllable is underlined for every polysyllabic word using the *emphasis* element, as shown below for the word *domnule* (mister), which is converted as *'dom-nu-le* (with lexical stress on the first syllable):

```
<emphasis level="moderate">
        <phoneme ph='dom'>dom</phoneme>
</emphasis>
<phoneme ph='nule'>nule</phoneme>
```

As one can see the lexical stress markers are highlighted using <*emphasis*> SSML elements for the corresponding syllable.

For this research phase, our prosodic model [17] is based on a specific set of linguistic knowledge sources (available at the output of the NLP stage): word segmentation, phones and their corresponding durations, syllabification, lexical stress positions, punctuation information, utterance type (i.e., declarative, interrogative, exclamatory, or imperative), etc.

V. CONCLUSIONS

This paper described the most important elements of an advanced NLP framework for a Romanian TTS synthesis system. The results are mainly achieved in a statistical manner, but rule-based or even hybrid approaches are also used. An important characteristic which has not been emphasized is that our NLP approaches, besides very good results, is essentially text independent. In other words, it can successfully solve the problem of lexical stress positioning, syllabification and L2P conversion for any unseen word that respects Romanian lexicology and phonological rules. For diacritic restoration, as we used a word level approach, the solution basically relies on a closed but large start dictionary, from where we statistically chose the best variants, but the algorithm will not stop for unseen words.

Our modular approach resulted in a very flexible way of embedding different techniques, such as SSML for document authoring and inter-modules communications, which seems to become universally recognized as a *must have* standard for TTS synthesis.

REFERENCES

[1] D. Burileanu, "Basic research and implementation decisions for a text-to-speech synthesis system in Romanian," in International Journal of Speech Technology, vol. 5, no. 3, Kluwer, Dordrecht, pp. 211-225, Sept. 2002.

[2] D. Burileanu, C. Negrescu, and M. Surmei, "Recent advances in Romanian language text-to-speech synthesis," in Proceedings of the Romanian Academy, Series A, vol. 11, pp. 92–99, 2010.

[3] www.ivona.com

[4] D. Tufis and A. Chitu, "Automatic diacritics insertion in Romanian texts," Proc. COMPLEX'99, Pecs, Hungary, pp. 185-194, June 16-19, 1999.

[5] R. Mihalcea and V. Nastase, "Letter level learning for language independent diacritics restoration," Proc. CoNLL 2002, Taipei, Taiwan, pp. 105-111, 2002.

[6] S. A. Toma, E. Oancea, and D. P. Munteanu, "Automatic rule-based syllabification for Romanian," in From Speech Processing to Spoken Language Technology, Publishing House of the Romanian Academy, Bucharest, pp. 87-94, 2009.

[7] M. A. Ordean, A. Saupe, M. Ordean, M. Duma, and Gh. C. Silaghi, "Enhanced rule-based phonetic transcription for the Romanian language," Proc. of the 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, pp. 401-406, Sept. 2009.

[8] E. Oancea and A. Badulescu, "Stressed syllables determination for Romanian words within speech synthesis applications," in International Journal of Speech Technology, vol. 5(3), pp. 237–246, 2002.

[9] O. Buza and G. Toderean, "Syllable Detection for Romanian Text-to-Speech Synthesis," Proc. of the 6th International Conference on Communications, Bucharest, pp. 135-138, June 2006.

[10] http://www.statmt.org/europarl/

[11] C. Ungurean, D. Burileanu, V. Popescu, C. Negrescu, and A. Dervis, "Automatic diacritic restoration for a TTS-based e-mail reader application," in UPB Scientific Bulletin, Series C, vol. 70, Issue 4. Bucharest: Politehnica Press, pp. 3-12, 2008.

[12] C. Ungurean, D Burileanu, C. Negrescu, and A. Dervis, "A statistical approach to lexical stress assignment for TTS synthesis," in International Journal of Speech Technology, Springer Netherlands, vol. 12(2-3), pp. 63-73, 2009.

[13] C. Ungurean, D Burileanu, V. Popescu, and A. Dervis, "Hybrid syllabification and letter-to-phone conversion for TTS synthesis," in UPB Scientific Bulletin, Series C, 2011, in press.

[14] A. Stan, J. Yamaghishi, S. King, and M. Aylett, "The Romanian speech synthesis (RSS) corpus: building a high quality HMM-based speech synthesis system using a high sampling rate," in Speech Comm., vol. 53, pp. 442–450, 2011.

[15] W. Zhiyong, C. Guangqi, M. Helen, and C. Lianhong "A unified framework for multilingual text-to-speech synthesis with SSML specification as interface," in Tsinghua Science & Technology, vol. 14(5), pp. 623-630, Oct. 2009.

[16] www.w3.org/TR/speech-synthesis11/

[17] D. Burileanu and C. Negrescu , "Prosodic modeling for an embedded TTS system," Proc. of the 14th European Signal Processing Conference (EUSIPCO), Florence, Italy, Sept. 4-8, 2006.