PRE-PROCESSING INPUT TEXT: IMPROVING PRONUNCIATION FOR DUTCH TEXT-TO-SPEECH SYSTEMS

Dr. A.J. van Hessen [*], Drs. R. Jansen [**], Prof. Dr. Ir. L.C.W. Pols[+]
[*]*University of Twente, The Netherlands,* [**]*Comsys International BV, Huis Ter Heideweg 6, 3700 AJ Zeist, The Netherlands ,* [+]*Institute of Phonetic Sciences, University of Amsterdam, The Netherlands*

ABSTRACT

To improve pronunciation of Dutch Text-To-Speech Synthesisers, a processor was built that tries to detect problematic cases in input texts and solve these automatically if possible. Its primary task is to realise pronounceable forms for numbers that do not have a straightforward pronunciation: structural and contextual information is used in an attempt to determine to what category a number belongs. Once categorised, each number is expanded according to the pronunciation conventions of its category. Moreover, pre-processing of raw ASCII text to a more readable text format is performed. It can be said that this pre-processor is a useful aid in improving performance at run-time, for example during the reading out aloud of emails, although ambiguity and redundancy in the input text illustrate the need for improved semantic and syntactic parsing to approach human text interpretation skills.

1. INTRODUCTION

During the process of generating synthetic speech from text, one encounters various kinds of problems. One of the most serious problems in converting text to synthetic speech is that the standard written form of (any) language gives an imperfect and often distant rendition of the corresponding spoken forms. In a written text, there will often be occurrences of numerals, abbreviations, special symbols (such as %, @) et cetera. These form a serious obstruction to the generation of correct phonetic transcriptions. It is therefore necessary to pass the text through a pre-processing stage that converts all such occurrences to the appropriate pronounceable words, unless in particular cases it is better to remove them instead.

One of these "text normalisation" examples is the *TextScan* system [1], a pre-processing module for the Dutch Text-To-Speech system *Spraakmaker*. The purpose of this pre-processing mode is to perform a segmentation of the input text, and to convert anomalous symbol strings such as numerals and abbreviations into a lexical format. Since abbreviations ending in a period do not necessarily indicate a sentence ending, period disambiguation is performed, for the purpose of end-of-sentence detection.

Any reasonable pre-processing module must of course perform some disambiguation of the input text that it is expanding: for example, the string *Fl. 2,50* (2.50 Dutch guilders) contains information that 2,50 is a money amount, and therefore is expanded differently from the way in which 2,50 (=2.50) would be expanded. Also, the full stop here does not indicate a sentence ending, but an abbreviation of *gulden* (guilder). A simple substitution does not suffice here: *gulden* must be placed elsewhere in the string (between *2* and *50*).

In an attempt to tackle the problems illustrated above, a pre-processor was built that solves pronunciation difficulties [2]. It was designed to be used for the Fluent Dutch Text-To-Speech synthesiser developed by Arthur Dirksen [3], and tries to identify the nature of complex numbers (numbers that should not be pronounced as integers) by examining their structure and context in the sentence. An attempt is made to categorise them as a date, time, telephone number, bank account number, money amount, ordinal number, fraction, or an area code. Once identified, they are expanded to an appropriate pronounceable form. This is accomplished by inserting orthographic clues and phonetic phrasing cues.

Since this pre-processor is a useful aid in the reading out aloud of emails, special attention is paid to the construction of correct sentences. Full stops are inserted if necessary, and constructions with special characters are rewritten to satisfy the demands of the TTS system.

2. THE FLUENT DUTCH TEXT-TO-SPEECH SYNTHESISER

The Fluent Dutch Text-To-Speech synthesiser converts text to speech by using a Dutch diphone database and a MBROLA diphone synthesiser [4]. An outline of its structure is shown in Figure 1. First, the text is analysed with the use of three lexicons. The main lexicon is the basic lexicon that contains many Dutch words and some common English words, specified by their phonemic transcription. The user lexicon and extra lexicon can contain uncommon words to make the synthesiser suitable for use in specific applications. Words that occur in one of the dictionaries are given the indicated transcription, and numerals, special characters and words that do not appear in any of the dictionaries are converted to phonemic representations by using letter-to-sound rules. Also, pitch and phrase characteristics are determined. Text analysis transforms the text to a phonological representation containing information about phonemes and prosodic structure that can serve as input to the prosody rules. These convert the phonemic transcription to a phonetic transcription, consisting of allophones, specified by their duration and pitch points.

Based on this phonetic information, the MBROLA diphone synthesiser concatenates the appropriate diphones with their appropriate duration and pitch. This is done phrase by phrase, and produces the desired waveform.

3. IMPROVING THE PERFORMANCE OF GRAPHEME TO PHONEME CONVERSION FOR NUMBERS

The pronunciation of numbers by Text-To-Speech Synthesisers is a process that very often goes wrong. The synthesiser interprets any number as if it were an integer, which leads to a very

unorthodox way of pronouncing common digit sequences like telephone numbers, area codes, bank account numbers et cetera. The TTS itself accounts for a few special cases, such as time and date, but this only works when written in the exact format (e.g., a time is read for *12:30*, but not for *12.30* or *12:30:23*). Minor deviations in spelling will cause the TTS to resort to spelling mode.

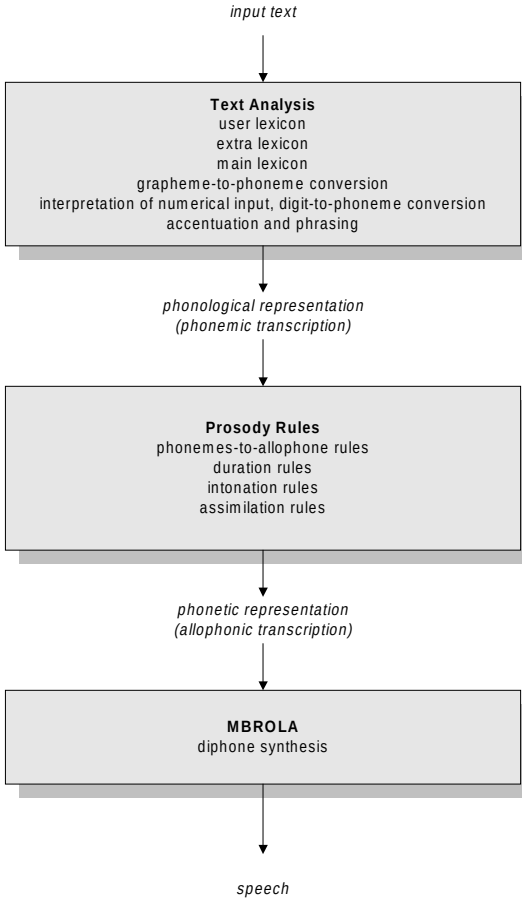

Figure 1: The Fluent Dutch Text-To-Speech Synthesiser

The pre-processor tries to solve this problem by examining if the numbers found in the text can be placed into categories that have equal pronunciation.
First, the text is scanned for numbers (in this case, a number is considered any sequence of characters that contains at least one digit). If a number is found, it undergoes a series of tests that try to determine what category the number belongs to. The different categories include:

1. Area code
2. Ordinal number
3. National telephone number
4. International telephone number
5. Amount of money
6. Time
7. Date
8. Fraction
9. Bank account number
10. Combination
11. Spelled number

A crucial element in successfully tagging the text is to allow for variability in structure of the number. For instance, a number may be placed between brackets, occur at the end of a phrase or sentence, or contain white spaces. When a number contains white spaces, it is difficult to determine where a number stops and the next one begins. This can be seen in expressions like *I have dialled 06-54645433 3 times.* Therefore, airtight number identification requires syntactic parsing of some form. The pre-processor discussed here does not use syntactic parsing, but structure analysis includes contemplating the possibility that a number exceeds word boundaries (for example, *020 6 454 828* will be recognised as a Dutch telephone number).

It should be noted that, since the pre-processor was designed for a Dutch TTS system, the number identification is based on Dutch number structures. Since each country has its own writing conventions, it would be an impossible task to cover for all those different styles. To illustrate this, *06/05/98* is interpreted as May 6th, 1998 in Dutch, whereas it means June 5th, 1998 in English. In this system, Dutch conventions have priority over the English conventions, except when this would lead to meaningless structures, say, a date like *5/14/98*.

If a number meets the criteria of one of the categories listed, a tag with the category's name is attached to it. Only one tag per number can be given, and overruling is impossible at this level. The examination process walks through the categories in the order shown above, so for instance if a number is found to be a date, it can not be tagged as a bank account number anymore. The only exceptions to this are fractions and spelled numbers. Fractions can be altered to dates if there is contextual evidence for this (for example *d.d. 6/10*), and any category can be changed to a spelled number if the context contains clues for this. However, since the tagging criteria show virtually no overlap, it is most likely for the tag to be correct when a number is tagged.
The second step in the process is the investigation of the contextual environment of the number. Whereas in the first step the only concern was the number itself, now it is being looked at as part of a sentence. In the event of letters occurring at word-initial or word-final position, they are peeled off, and the remainder of the number is being checked for categories again (which could even c over for typing mistakes!). If, after this process, still no tag has been found, the immediate surroundings of the number in the sentence are searched. If a word, occurring within a distance of two words from the number, could indicate that the number belongs to a certain category, and the number satisfies the minimal conditions for that category (for example, an *area code* number should be at least four digits long), the tag is given.

To scan the context words of a number, each category has its own dictionary containing characteristic words for that category. For example, *bellen* (call) is one of the words of the telephone dictionary. The dictionaries also contain information about whether the context words should precede or follow the number in order to be assigned to that number.

The dictionary that contains the most matches is assumed to be of the correct category. Also, each category has its own, predefined priority number. If two or more dictionaries find an

equal number of matches, the category that has the highest priority number prevails. Priority is based on three considerations:

- Given a lexical item of a specific category matching a context word, what is the probability that the number belongs to that category?
- What is the probability that a number of a particular category will be recognised from context but not from structure analysis?
- Suppose the wrong tag would be assigned, what would be the consequence for the eventual pronunciation of the number?

Spelled numbers get highest priority, since the presence of dictionary items for this category offers a fair chance that indeed a spelled number is present, and, more importantly, the number could not have been recognised earlier during the process. Furthermore, the rewriting procedure for pronunciation of a spelled number allows for any character combination without risk of losing information. Bank account numbers and telephone numbers are next, mainly because giro numbers and local telephone numbers contain no information and cannot be categorised in the initial tagging routine (but, on the other hand, would be understood if classified as a spelled number). Area codes are positioned at the bottom of the list since their structure is so well defined that they should be categorised straight away in the initial tagging routine. Date and time also receive low priority since their phonetic transcription routine is disastrous to numbers unjustly categorised this way.

The final step is to expand the tagged numbers to produce an input text for the TTS. Numbers that have not been tagged are assumed to be normal integers, and since the TTS pronounces them correctly, there is no need for further processing. The program scans through the text looking for a tag, and if one is found, the tagged number is cut out of the sentence, the tag is disposed of and the number undergoes the expanding process appropriate for the tag. This expanding process can take place at two levels: phonemic or orthographic. In the first case, the rewritten number is built up from small pieces of phonetically transcribed text, added as the process goes along. This is done to allow phonetic cues to be inserted in the text more easily, or to give the syllables the correct stress. For example, expanding ordinal numbers such as *12301ste* can be done by first obtaining the phonemic transcription of the integer and next change the ending / `?*en` / to /`*er-st@` /.

In the second case, numbers are rewritten orthographically. In this way, the program can make use of the ability of the TTS to pronounce certain strict “formats”, for instance the time format as shown above. By converting the number to the TTS format, with optional supplementary information, it can be transcribed as a whole.

In this way, a sentence at orthographic level is created that contains the entire text, constructed of sentences combined with expanded numbers and phonemic insertion tags.

The pre-processor creates alternative pronunciation directives that result in smoother pronunciation of most of the numbers. It is, however, not possible to capture all numbers into their appropriate categories, and sometimes numbers end up with a correct tag (through the context search) but still are pronounced incorrectly (for example, when a number is tagged during the context search while the structure can not be correctly rewritten in the expanding routine). An example of this is the *money* number *123.95 gulden* (123.95 guilders), where the full stop is not supported since commas are default in Dutch money amounts.

This problem can be overcome in two ways. The expanding routine can be refined to produce correct pronunciations for more different structures, or the tags should be redefined more accurately. This would involve defining more categories, and, for instance, the implementation of expanding directives within tags. Expanding directives are already successfully implemented in the SABLE SAYAS markup language [5], (that uses a similar tagging system to indicate pronunciation for numbers) and they mainly include word order directives (for example, the *date* tag can be refined with a Day-Month-Year mode type option, that indicates the structure of the date as, say, MDY or MD or DMY).

4. FORMATTING THE INPUT TEXT

Besides expanding numbers to a pronounceable form, special attention is paid to the construction of the sentence. Parsing at word level can influence the structure of the sentence: if a word is taken out of the sentence and placed back afterwards, an incorrect pronunciation might be generated. Consider the next sentence:

The number in this sentence is \insert=<phonrep>\ .

The part between the backslashes is a phonetic cue for the speech synthesiser. This is used in certain cases to ensure that correct pronunciation is generated. However, this causes the full stop indicating the sentence ending to become a stand-alone character. To solve this problem, which applies for all punctuation marks, the punctuation mark is replaced by a phrasing cue or, in the event of a full stop, a hard return.

In emails, many lines are terminated by a hard return instead of a full stop, as a result of writing habits of people. Moreover, in raw ASCII format, hard returns might be inserted at the end of a *terminal line* (typically 70 characters long). To ensure that correct sentences are formed when the mail is retrieved from the mail server, an extra end of sentence detection is carried out. Sentences are reconstructed, according to full stops (abbreviations taken into account), capitals, and the length of the line. A sentence ending is identified when a full stop is found, followed by a white space or a tab character, and a capital. A hard return also indicates a sentence ending, unless the length of the line exceeds 65 characters (since that might indicate a line ending) and the next line does not start with a capital.

A final pre-processing routine that improves pronunciation removes or inserts white spaces before special characters. Most special characters need to be isolated to ensure proper pronunciation: *45%* is spelled, hence a white space is inserted (*45 %*)

5. CONCLUSION

Pre-processing texts before feeding them to a Text-To-Speech synthesiser considerably improves pronunciation and can create the illusion of computer intelligence. Nevertheless, it does not account for all the problem cases that one encounters in texts.
When most common structures of numbers are known to a pre-processor, a fair deal of them can be identified correctly.

Performing a context scan can improve identification even more. What causes most problems occurring in number identification is the redundancy that is often used in text writing. Whereas humans usually have no problems in inferring the meaning of character strings, subconsciously using syntactic and semantic context information, computers need more information to determine the identity of a number. The pre-processor that is discussed here mainly relies on the use of clear structures, and can cover for some redundancy by doing a context search that tries to suggest semantic parsing. When texts are written without a lot of redundancy, most numbers are identified correctly.

Summarising, it can be stated that the pre-processor that is discussed here performs as it was intended to do. However, it seems safe to conclude that although the majority of cases that form an obstruction to the pronunciation generation process can be solved successfully, there seems to be a limit in pre-processing performance that cannot be exceeded unless some form of semantic and syntactic parsing is used to correctly identify all words in the text, in order to provide an appropriate pronunciation for them.

REFERENCES

[1] Y. v. Holsteijn, (1993): *TextScan: a preprocessing module for automatic text-to-speech conversion*. In: V. v. Heuven, L.C.W. Pols (eds.): Analysis and synthesis of speech: strategic research towards high-quality text-to-speech generation. *Speech Research*, Mouton de Gruyter, Berlin 1993.

[2] R. Jansen, (1998): *Pre-Processing Input Text: Improving Pronunciation for the Fluent Dutch Text-To-Speech Synthesiser*. Institute of Phonetic Sciences, University of Amsterdam, Comsys International B.V., Zeist, The Netherlands.

[3] *http://www.fluency.nl*

[4] *http://tcts.fpms.ac.be/synthesis/mbrola.html*

[5] R. Sproat, A. Hunt, M. Ostendorf, P. Taylor, A. Black, K. Lenzo, M. Edgington. 1998. *Sable: A standard for TTS Markup*. International Conference on Spoken Language Processing (ICSLP), Sydney, Australia, December 1998.