Spraakmaker: a text-to-speech system for the Dutch language

H.C. van Leeuwen and E. te Lindert

Abstract

Spraakmaker integrates a number of modules resulting from the combined efforts of several institutes in the Netherlands which aim at doing research in the field of automatic text-to-speech conversion. Spraakmaker is intended as a demonstration model for Dutch as well as a flexible tool for future research. In its design, Spraakmaker differs from most other existing text-to-speech systems. This paper first discusses the general language-independent framework on which Spraakmaker is built, which concerns the data structure used to store relevant data for the text-to-speech process and the general means of control. Then Spraakmaker is discussed as a specific implementation, which illustrates the use of the data structure and how linguistic modules operate on this structure.

Introduction

Under the auspices of the nationally co-ordinated research programme 'Analysis and Synthesis of Speech' (ASSP), a large number of projects have been launched in the period 1985-1990, covering the whole spectrum of text-to-speech conversion for the Dutch language. One of the aims of the programme was the 'production of operational text-to-speech systems as an intermediate between research and applications.' Spraakmaker is the text-to-speech system for Dutch which resulted from this programme and it integrates the linguistic modules resulting from some of the ASSP projects.

Spraakmaker is a system which is built on a certain framework which can be characterized by a certain language-independent architecture. In what follows, the term *Spraakmaker* refers to the Dutch text-to-speech system as a whole, whereas its English equivalent *Speech Maker* refers to the language-independent framework. In this paper, first the general framework Speech Maker will be discussed, followed by Spraakmaker, the specific implementation for the Dutch language.

Speech Maker: a framework for text-to-speech systems

Most text-to-speech (TTS) systems consist of a sequential processing structure and a linear data representation (Carlson & Granström, 1976; Kerkhoff, Wester & Boves, 1984; Kulas & Rühl, 1985; Allen, Hunnicutt & Klatt, 1987; Van Rijnsoever, 1988). A sequential processing structure means the various modules (such as the grapheme-to-phoneme conversion unit or the intonation contour generation unit) are called upon in a certain temporal order and do not interact. A linear data representation means that the information which is transferred from one module to the other is coded in a linear way, often by a string of characters. Such a string may contain different pieces of information. For instance, a character string presented for grapheme-to-phoneme conversion may contain information on sentence accent, morphological structure and orthography. The Dutch word 'partijvoorzitterschap' (English: 'party chairmanship') can be represented for instance as *'part'ij#voor%zitt%er%schap'*, where '' denotes word stress, '#' a morphological boundary between compounds and '%' a morphological boundary between affixes or between a stem and an affix.

In Speech Maker the sequential processing structure is maintained, but the linear data representation is replaced by a multi-level, synchronized data structure. In this structure, different types of information (e.g., morphology, orthography, pronunciation, etc.) are represented on different levels. The information on the different levels is synchronized by so-called sync marks, which are placed between the data items on each level. For instance, 'partijvoorzitterschap' can be represented in Speech Maker as depicted in Figure 1.

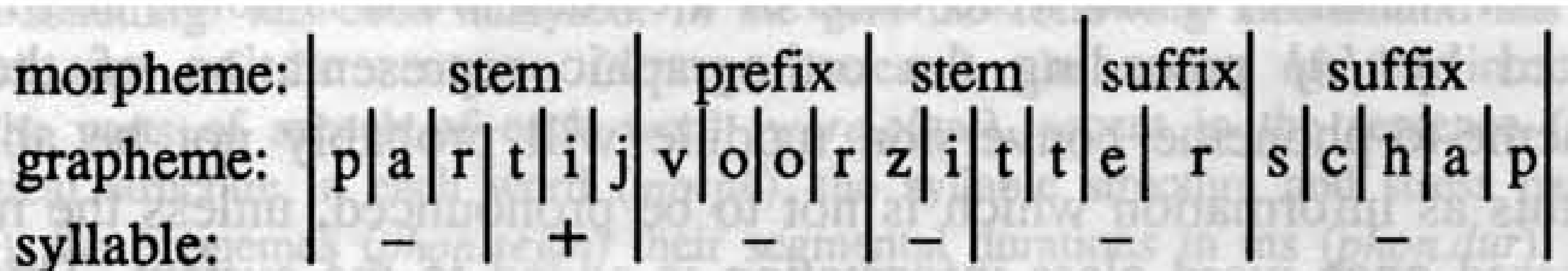
morpheme: | stem | prefix | stem | suffix | suffix |
grapheme: | p | a | r | t | i | j | v | o | o | r | z | i | t | t | e | r | s | c | h | a | p |
syllable: | – | + | – | – | – | – |

Figure 1: An example of how the word 'partijvoorzitterschap' is represented in Speech Maker. Between each data item (e.g., a letter, or a morphological type), a sync mark (a vertical bar) is placed. If the sync marks are displayed exactly beneath one another, the levels are synchronized at that point. In this representation, the morpheme types are coded directly (instead of indirectly by the different morphological boundary types), and word stress (as attribute of the syllable structure) is indicated by a more insightful code ('+' and '–' instead of ''). Since syllable structure and morphological structure do not have a strictly hierarchical relation, they are displayed here beneath and above the grapheme level, so that both structures are clearly visible.

This somewhat more elaborate manner of data representation, which was first introduced by Hertz, Kadin and Karplus (1985), is chosen mainly for two reasons.

(1) We believe that this representation is more transparent. Each level represents a different type of information. The symbols used at a certain level need not necessarily be different from those used at the other levels, simply because the level itself disambiguates. For instance, the presence of sentence accent (i.e., which words are to receive accent in the sentence) and word stress (which syllable carries word-internal stress) may both be indicated by '+'. In a linear string, one would have to use different symbols for both types of information.

Moreover, text-to-speech systems are still to be improved, and many aspects of text-to-speech conversion may have to be changed. More information of a linguistic nature will become available to the system, which will be increasingly difficult to code transparently in a linear string. For instance, word class determination is closely related to morphological analysis. Word class is needed for syntactic analysis, which in turn is needed for determination of sentence accent and pause position. Explicitly separating the different types of information will considerably enhance its transparency to the user. And we believe a transparent data representation increases comprehension of the system's internal state. This in turn will increase the speed with which a developer can make new modules or improve existing ones, given the correct tools to interact with such a structure.

(2) Separate modules in the TTS system become more independent of each other. A certain module is developed at a certain time. It is designed to deal with a certain kind of input. When a new module is added to a TTS system in which a linear output string from one module serves as the input to another module, it is quite possible that the latter will be confronted with input it cannot deal with. For instance, if the word class of a certain word is *noun*, and if this is represented by *[n]* preceding the orthographic representation of the word, the grapheme-to-phoneme conversion module will probably not be able to recognize this as information which is not to be pronounced, unless the module is also altered once word class information is added to the system. This is undesirable, since it means that each module in the entire TTS system may need to be adjusted when the system is expanded. In the case of a multi-level data structure, new types of information can simply be ignored.

The multi-level synchronized data structure

The multi-level synchronized data structure is the heart of Speech Maker. All data transferred between modules pass through this structure. The (linguistic) data in the structure are the only data directly accessible to the user. Owing to its two-dimensional aspects, this multi-level synchronized data structure has been called *grid.*

The abstraction supported by the grid is that of a finite set of *streams* (in this we follow the terminology introduced by Hertz et al. (1985)). A stream is a data structure in which one level of information is represented. The number and nature of streams can be chosen freely, so long as each stream is identified by a unique name. For instance, in Figure 1 a grid is depicted which contains three streams, viz. the morpheme, the grapheme and the syllable stream. In Figure 2, which will be explained in more detail presently, a larger grid is depicted which contains eight streams: sentence, int_phrase, word, morpheme, syllable, grapheme, phoneme and pitch.

To each stream, one or more *attributes* can be connected. Attributes serve to specify different aspects of a certain level of information. For instance, the word level may contain information on parts of speech (noun/verb/...) and on whether or not the word is to receive accentuation. These aspects both concern the word level and are therefore coded as different attributes of the word stream. For example, in Figure 2 the word stream has two attributes, 'class' and 'accent', in which the parts of

sentence:	declarative																				
int_phrase:																					
word.class:	det		noun			verb				p				det		noun					
.accent:	−		+			−				−				−		+					
morpheme:	stem		stem			stem				stem				stem		stem				suffix	
syllable:	−		+			+				+	−			−		+				−	
grapheme:	d	e	b	a	l	v	l	o o	g	o	v	e	r	d	e	s	c h	u	t t	i	n g
phon.segm:	d	@	b	A	L	v	l	o	x	o	v	@	r	d	@	s	x	U	t	I	N
.dur:	46	69	39	54	100	154	46	123	69	123	62	54	46	46	54	77	77	63	54	63	126
pitch.type:	0			1	&													A	0		
.anchor:	−			v.o.	−													v.o.	−		
.onset:	−			−70	−													−20	−		
.dur:	−			120	−													120	−		
.exc:	−			6.0	−													−6.0	−		

Figure 2: An example of the grid when all analysis modules have operated. The sentence 'De bal vloog over de schutting' has been analysed. In the grid the following information has been stored: the scope and type of the sentence (*sentence* stream), the scope of the intonation phrase (*int_phrase*), the parts of speech of each word (*word.class*), accent in the sentence (*word.accent*), the morphological structure in each word (*morph*), the syllable structure and stress pattern of each word (*syllable*), the phonemes (*phon.segm*) their segmental durations in ms (*phon.dur*), and the relevant pitch movement parameters (*pitch*). Here '0' denotes low declination, '1' a pitch-lending rise, '&' high declination and 'A' a pitch-lending fall. The pitch-lending rise is anchored at the vowel onset (v.o.), starts 70 ms before the vowel onset, has a duration of 120 ms, and has an excursion of 6.0 semitones.

speech and sentence accents are stored, respectively. The number and nature of attributes can also be chosen freely, the names of which must be unique within the stream. If there is only one attribute connected to a stream, the identification of the attribute may be omitted since, in that case, the stream identification is sufficient.

A stream is a sequence of *tokens* separated by *sync marks*. A token contains one or more fields, one field for each attribute that has been defined for the stream. For example, in Figure 2 each token of the pitch stream consists of five elements, the second having the value: (1, v.o., -70, 120, 6.0). When the grid is displayed, the fields of a token are printed above one another. On each line, the fields of a specific attribute are printed. Such lines are called *substreams*, as they represent part of a level.

A sync mark serves both to separate tokens from each other in an individual stream, and to synchronize tokens in one stream with those in other streams. Two (sequences of) tokens in different streams are synchronized if the sync marks in which they are enclosed are printed above each other. So, for instance, in Figure 2 the token sequence 's c h u t t' in the grapheme stream is synchronized with the token sequence 'stem' in the morpheme stream, and the graphemes 'c h' are synchronized with the phoneme 'x'.

The general architecture of Speech Maker is as follows: centrally, the grid contains all data relevant to the derivation of speech from text. On the one hand, all linguistic modules operate on the grid, that is, all modules collect their input data from the grid (except for the first, which reads the text from some input device) and write their results back into the grid (except for the last, which produces speech). On the other hand, a user interface is available to the linguist developer to control Speech Maker, with which, among other things, one can inspect and change the contents of the grid at strategic points in the derivation. The general architecture of Speech Maker is depicted in Figure 3 (where a specific implementation of linguistic modules is already shown).

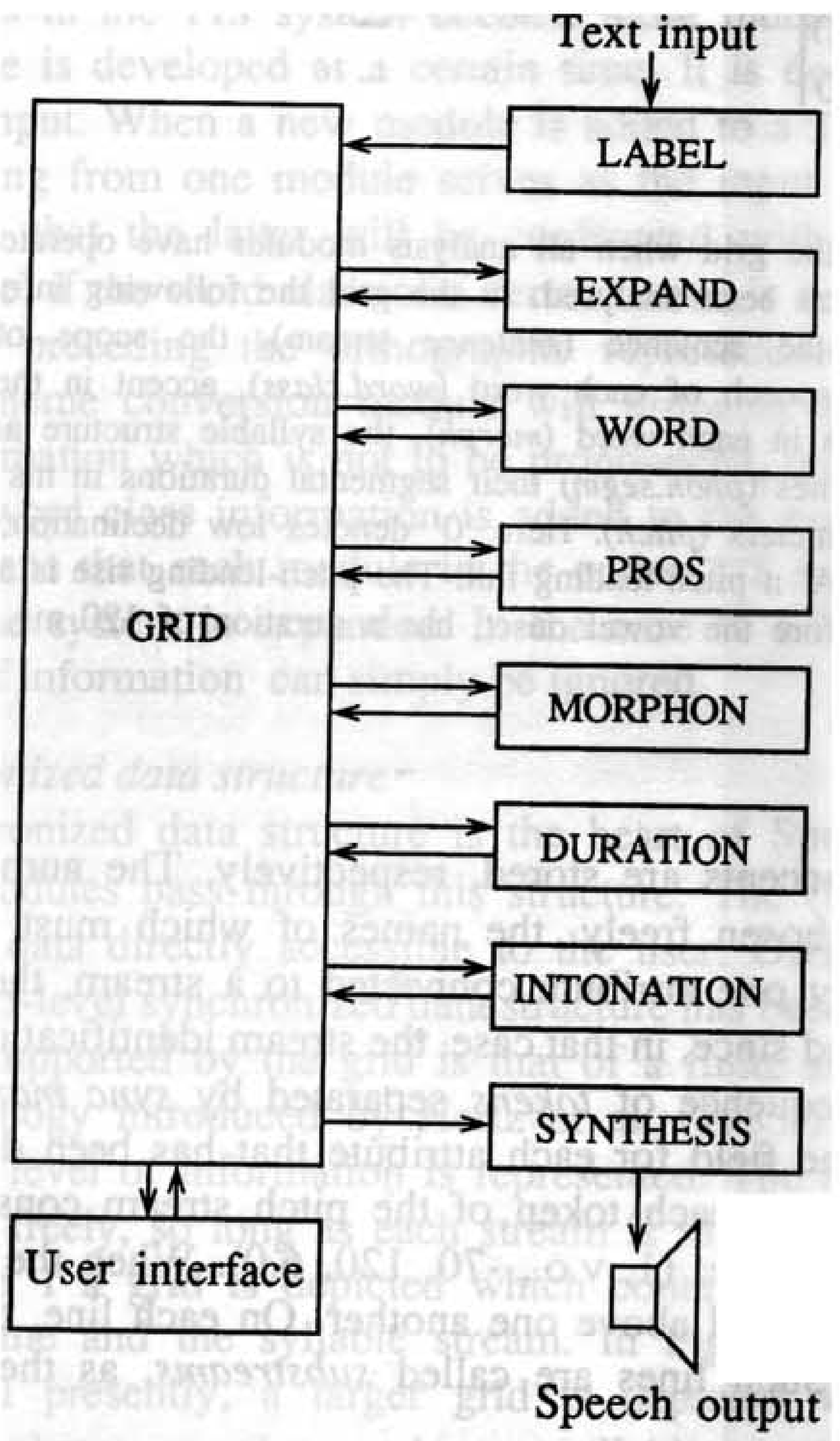

Figure 3: General architecture of Speech Maker. The eight linguistic modules are called from top to bottom and communicate via the multi-level synchronized data structure, called *grid*.

Control of Speech Maker
By means of the user interface, a linguist who develops linguistic modules can control Speech Maker. This encompasses three main functions which will be discussed in order: inspection of the data in the grid, manipulation of these data and control of the data flow within Speech Maker.

Inspection: After application of each linguistic module the linguist can inspect the grid to see whether the module has produced the desired results. Since, in a large text, the grid contains too much information to be grasped by a user in one view, one can select specific portions of the grid, for instance one sentence at a time, or specific streams. However, not only can the results of linguistic modules be inspected, but also the derivation process inside such a module can be inspected if the module supports such an option. Of course, this facility must be provided by the constructor of the module, but once such a provision has been made, it is also included in Speech Maker.

Manipulation: Besides inspecting the momentary status of the grid, one can also manipulate the information in the grid. In fact, when one inspects the grid, one invokes a specialized 'grid editor'. With this tool, one can insert, delete or modify tokens, and modify the synchronization relations between the streams. The grid editor can be an important tool in the development of linguistic modules, since one can prepare (and store) a desired input status of the grid independently of whether previous linguistic modules have operated satisfactorily. On the other hand, when all modules operate satisfactorily, Speech Maker can be used as a stimulus generator for all kinds of perceptual experiments concerning artificial speech, since it is very easy to modify parameters and store produced utterances.

Control of flow: In normal operation, Speech Maker's input is obtained from a terminal or file, and its output is speech or a waveform written to a file. However, for development and testing purposes one can select a portion of the normal processing trajectory. For instance, to test a specific module, one can instruct the system to stop processing after this module and display the grid information for inspection, and simultaneously write it to file. If the results of this module are satisfactory, these stored grids can serve as input for a testing session of the next module.

A second facility is the possibility of interchanging linguistic modules. For instance, when two alternative modules have been developed, it is possible to exchange the one module for the other, run a new session with the same input and compare the results of the two sessions. This facility operates under the condition that alternative modules require input which is present in the grid, and produce similar types of output, so that subsequent modules can operate without adjustment.

These three types of interaction provide a linguist with the most important facilities to manipulate Speech Maker's behaviour. The user interface has been implemented in X-windows, so that the user has a graphics-oriented tool to control the system in an interactive session. However, all functions can also be achieved by typed-in commands. Thus, also in noninteractive sessions, such as batch jobs, the system can be instructed to operate as desired.

Spraakmaker: a text-to-speech system for Dutch

To build a text-to-speech system with Speech Maker, two things must be defined: a specific implementation of the grid (i.e., the number and nature of the streams) and the linguistic modules. This section describes how this has been done for Spraakmaker. The majority of the linguistic modules in Spraakmaker already existed, so they were determined historically. As to the implementation of the grid, it was decided to include only one stream per type of information for reasons of efficiency and transparency. For instance, the orthography is represented in the grapheme stream. If an amount of money, such as 'fl. 2,50', is encountered in the text, this will be stored in the grapheme stream. During the derivation, however, this may be altered to the normalized string 'twee gulden vijftig' (two guilders fifty) which overwrites the original information in the grapheme stream. The pronunciation, however, will be represented on a separate level (the phoneme stream) since this is essentially a different type of information. The streams which are distinguished in Spraakmaker are discussed below, followed by the linguistic modules of the system.

Spraakmaker's grid

The most important streams in Spraakmaker, in which the information transferred to the linguistic modules is stored, are:

(a) Sentence stream: indicates the beginning and end of the sentence and its type (declarative/interrogative).

(b) Intonation stream: indicates the beginning and end boundaries of an intonation phrase. In the phase of synthesizing speech, a pause will be added between the intonation phrases and a declination reset will occur at the new phrase.

(c) Word stream: indicates the beginning and end of a word and contains all information concerning words. Several substreams are introduced to represent the different aspects of a word. The most important substreams are:
- word.type substream: indicates whether a word is lexical (the 'normal' words, to be dealt with by the morphological analyser), or a numerical expression, or an acronym (to be dealt with by specialized rule-based expansion modules).
- word.class substream: gives the part of speech (noun/verb/...) of the word.
- word.accent substream: indicates whether or not the word is to receive sentence accent.

(d) Morphological stream: indicates the morphological structure of a word and the types (prefix/stem/suffix) of the morphemes.

(e) Syllable stream: gives the syllable structure of a word and indicates whether or not the syllable bears stress.

(f) Grapheme stream: gives the orthography of a word.

(g) Phoneme stream: gives the phonemic representation of the word. Attached to each phoneme is an indication of its duration.

(h) Pitch stream: gives the type of pitch movement according to the Dutch intonation grammar of 't Hart, Collier and Cohen (1990). There are four additional substreams which specify the movement parameters:

- pitch.anchor substream: gives the anchor of the pitch movement (vowel onset/end of voicing).
- pitch.onset substream: gives the timing onset relative to the anchor (e.g., a pitch-lending rise in Dutch usually starts 70 ms before vowel onset).
- pitch.duration substream: gives the duration of the pitch movement.
- pitch.excursion substream: gives the excursion of the pitch movement (for instance in semitones).

When all analysis modules have operated, the grid will roughly resemble Figure 2 for the input sentence 'De bal vloog over de schutting.' (The ball flew over the fence).

Spraakmaker's linguistic modules
The second aspect of Spraakmaker's implementation is the succession of modules which operate on the grid. Figure 3 shows the current configuration of Speech Maker. The majority of the modules add information to the grid; only two of them (EXPAND and MORPHON) overwrite existing information. The modules are presented in the order in which they are called.

(1) LABEL: reads text from a file or from the terminal. It marks the beginning and the end of sentences, words and graphemes and labels sentences and (groups of) words in the sentence. A wide range of labels is used (abbreviations, telephone numbers, amounts of money, etc.). The grapheme, word.type and sentence streams in the grid are initiated by this module.

(2) EXPAND: deals with the words that have received a special label, like abbreviation, telephone number, etc. and expands the words into their normalized form. After the application of this module only three output types are distinguished: lexical, numerical, or acronym. This label determines the path which is chosen in the next module, since different pronunciation rules apply to each type.
For example, after application of LABEL, 'tel. 050-123456' is represented in the grid as:

word.type:	abbr	tel_nr
grapheme:	tel	050 - 123456

EXPAND will alter these data into:

word.type:	lex	d	d	d	d	d
grapheme:	telefoon	0	50	12	34	56

where 'd' (from 'digit') represents a numerical expression.

(3) WORD: deals with several aspects of grapheme-to-phoneme conversion on the word level. This comprises phonemic representation, word stress, morphological and syllable structure and word class. Note that these attributes relate to different streams in the grid. These different types of output are included in one

module for efficiency reasons as it is relatively easy to determine them at the same time. Words marked as lexical items by LABEL/EXPAND must be analysed morphologically to arrive at the correct pronunciation. For this purpose a morpheme lexicon is needed, and once one has a lexicon, one can also store phoneme representation, syllable and stress information. By combining the morphemes one can also determine the word class. Nonlexical items (words marked as numbers and acronyms by LABEL/EXPAND) are dealt with by specialized, rule-based modules. The input of WORD is the orthography (the grapheme stream), the output is written into the phoneme, syllable, morphology and word.class streams.

(4) PROS: determines sentence accents and the beginning and end of intonation phrases. It takes its input from the word.class substream and writes its output to the word.accent and intonation phrase (sub)streams.

(5) MORPHON: adjusts the phonemic representation. It deals with phonological effects which surpass morpheme and word boundaries (e.g., 'reduction' ↔ 'reduce', which cannot be dealt with by WORD). These effects are dealt with by this phonological module. Input and output are both the phoneme and syllable streams.

(6) DURATION: determines durations of the phonemes. It takes the phonemes, syllable structure, word stress and sentence accent as input. Its output is written to the duration stream.

(7) INTONATION: determines the relevant pitch movement parameters. Input: sentence range, intonation phrase, sentence accent, word stress; the output is written to the pitch movement stream.

(8) SYNTHESIS: determines the speech waveform. As input it takes the phonemes, syllable structure, segmental duration, intonation phrase and pitch movement parameters. The output (a sampled-data file) can be written to a file, and/or is sent to a loudspeaker via a DA converter.

Concluding remarks

With this, most of the important aspects of the general framework, Speech Maker, and the specific implementation, Spraakmaker, have been discussed. One important aspect of Speech Maker has not been mentioned, however. This concerns a specialized rule formalism called SMF (Speech Maker Formalism), with which a linguist can directly manipulate the grid by means of rules. In their layout, the rules reflect the organization of the grid, that is, the succession of informational units (tokens) in time is reflected horizontally, and the synchronization between units of different informational types vertically. By means of such rules a 'program' can be built which specifies a sequence of actions operating on the grid, which alter the information in the grid, or add information to it. Such a program forms a specific linguistic module such as those which are depicted in Figure 3. The advantages of writing a module in SMF are that one does not need a specific interface between the grid and the linguistic module (which is the case for most of the current modules) and that the knowledge concerning a specific linguistic process is specified in a compact and transparent code. Since a detailed discussion of SMF would exceed the space available in this paper, more information can be found in Van Leeuwen and Te Lindert (1991).

In conclusion, Spraakmaker is a text-to-speech system with a flexible underlying framework, Speech Maker. The kernel of this framework is the grid database, in which relations between data are expressed in an advantageous way. A user interface enables the user to control the system and a formalism allows algorithmic manipulation of the database. Spraakmaker is a specific implementation for the Dutch language which comprises eight major linguistic modules.

References

Allen, J., Hunnicutt, S. & Klatt, D. (1987) *From Text to Speech: The MITalk system.* Cambridge: Cambridge University Press.

Carlson, R. & Granström, B. (1976) A text-to-speech system based entirely on rules. *Proceedings of ICASSP 76*, 686-688.

Hart, J. 't, Collier, R. & Cohen, A. (1990) *A perceptual study of Intonation.* Cambridge: Cambridge University Press.

Hertz, S.R., Kadin, J. & Karplus, K. (1985) The Delta rule development system for speech synthesis from text. *Proceedings of the IEEE, 73(11)*, 1589-1601.

Kerkhoff, J., Wester, J. & Boves, L. (1984) A compiler for implementing the linguistic phase of a text-to-speech conversion system. In: H. Bennis, W.U.S. van der Kloecke (Eds): *Linguistics in the Netherlands*. Dordrecht: Foris Publications, 111-117.

Kulas, W. & Rühl, H.W. (1985) Syntex – unrestricted conversion of text-to-speech for German. In: R. de Moris and C.Y. Suen (Eds): *New Systems and Architectures for Automatic Speech Recognition and Synthesis*. Berlin: Springer, 517-535.

Leeuwen, H.C. van & Lindert, E. te (1991) Speech Maker: text-to-speech synthesis based on a multilevel, synchronized data structure. *Proceedings of ICASSP 91*, pages unknown.

Rijnsoever, P.A. van (1988) A multi-lingual text-to-speech system. *IPO Annual Progress Report, 23*, 34-40.