Natural Language Engineering **7** (1): 47–86. Printed in the United Kingdom

© 2001 Cambridge University Press

From data to speech: a general approach

M. THEUNE, E. KLABBERS, J. R. DE PIJPER, E. KRAHMER

IPO, Center for User-System Interaction,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
e-mail: {M.Theune, E.A.M.Klabbers, J.R.d.Pijper, E.J.Krahmer}@tue.nl

J. ODIJK

Lernout & Hauspie Speech Products, Flanders Language Valley 50, 8900 Ieper, Belgium
e-mail: Jan.Odijk@lhs.be

(*Received 11 November 1998; revised 10 November 2000*)

Abstract

We present a data-to-speech system called D2S, which can be used for the creation of data-to-speech systems in different languages and domains. The most important characteristic of a data-to-speech system is that it combines language and speech generation: language generation is used to produce a natural language text expressing the system's input data, and speech generation is used to make this text audible. In D2S, this combination is exploited by using linguistic information available in the language generation module for the computation of prosody. This allows us to achieve a better prosodic output quality than can be achieved in a plain text-to-speech system. For language generation in D2S, the use of syntactically enriched templates is guided by knowledge of the discourse context, while for speech generation pre-recorded phrases are combined in a prosodically sophisticated manner. This combination of techniques makes it possible to create linguistically sound but efficient systems with a high quality language and speech output.

1 Introduction

In this paper we present a generic system, D2S, which can be used for the construction of data-to-speech systems for various domains and languages. The most important characteristic of data-to-speech[1] is that it combines language and speech generation. Language generation is used to produce a natural language text expressing the system's input data, and speech generation is used to make this text audible. An obvious way of combining language and speech generation in a data-to-speech system is to incorporate them as two separate modules whose interface consists of plain text. A serious drawback of such an architecture is that valuable information for speech generation is lost (Pan and McKeown 1997; Zue 1997). For this reason,

[1] We prefer to use the term 'data-to-speech' rather than the more common 'concept-to-speech', as our system takes as input data retrieved from tables or databases, rather than some sort of semantic, or 'conceptual' representations.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

in D2S linguistic information provided by the language generation module is used for the reliable generation of prosodic markers, thus improving the prosodic quality of the system's speech output.

D2S is a *hybrid* system, in which some parts of the generation process are based on general, linguistic principles, whereas other generation tasks are carried out using less flexible, application-specific methods. One of the interesting features of the language generation module of D2S is that it does not follow the relatively common pipeline architecture for language generation (Mykowiecka 1991; Reiter 1994; Cahill, Doran, Evans, Mellish, Paiva, Reape, Scott and Tipper 1999) in which text and sentence planning precede linguistic realisation. In fact, it contains hardly any *global* text planning: sentences are generated from so-called *syntactic templates*. These are TAG-like syntactic structures, associated with conditions which determine when they can be used properly given the current state of the generation process. Speech generation in D2S can be done using two different speech generation methods: one based on pre-recorded phrases, which offers high speech quality but is very inflexible, and one based on diphone synthesis, which offers high flexibility but has a lower output quality. What the two have in common is that they can make use of the prosodic marking provided by language generation to determine prosody.

D2S was initially developed with the construction of the Dial Your Disc (DYD) system, which generates spoken monologues in English, giving information about recordings of compositions by Mozart (van Deemter, Landsbergen, Leermakers and Odijk 1994; Odijk 1995; van Deemter and Odijk 1997). D2S is also used for output generation in OVIS, a Dutch travel information system (Veldhuijzen van Zanten 1998; van Noord, Bouma, Koeling and Nederhof 1999). Earlier D2S-related papers concentrated on specific aspects of D2S, such as topic management (Odijk 1995), the use of context in language generation (van Deemter and Odijk 1997), the computation of prosody (Theune, Klabbers, Odijk and de Pijper 1997), and speech generation (Klabbers 1997). In the current paper we want to give a detailed overview of D2S as a whole, discussing in particular the link between language and speech generation.

In this paper, we illustrate D2S and the techniques it is based on using a data-to-speech system called GoalGetter, which generates spoken reports of football matches in Dutch. An interactive, on-line demonstration of the GoalGetter system can be found at http://iris19.ipo.tue.nl:9000/. Two systems with the same application domain as GoalGetter are SOCCER (André, Herzog and Rist 1988) and MIKE (Tanaka, Hasida and Noda 1998). However, both SOCCER and MIKE generate *commentaries*, spoken descriptions of image sequences of football scenes, whereas GoalGetter generates *summaries* of football matches, taking tabular information about the match as input. In that respect, GoalGetter is more like the STREAK system (Robin 1994; McKeown, Robin and Kukich 1995), which also generates sports summaries, though the sports domain is basketball instead of football. An important difference between GoalGetter and STREAK is that the latter produces only written, not spoken, output.

This paper is organised as follows. In section 2 we briefly describe a range of possible techniques for language and speech generation, and show where the

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

techniques used in D2S can be positioned along this range (in subsections 2.1 and 2.2 respectively). The role of prosody in data-to-speech generation is discussed in section 2.3. Section 3 provides a detailed description of D2S. We first show an example of the input and output of the GoalGetter system, and then explain the techniques that are used in the different modules of D2S, using examples from GoalGetter as an illustration (sections 3.1 to 3.3). We end with a discussion (section 4).

2 Techniques for language and speech generation

In this section we briefly describe a range of possible techniques for language (section 2.1) and speech generation (section 2.2). In both language and speech generation, two extremes can be distinguished. At one end of the spectrum we find simple, application-specific approaches which are generally inflexible and not reusable, and at the other end we have linguistically motivated approaches, which are more general and more flexible but may have practical drawbacks. The language and speech generation techniques employed in D2S aim at finding a balance between these extremes, achieving flexibility, efficiency and a good output quality. A key feature of D2S is that it provides a tight coupling between the language and speech generation modules in the form of prosody computation. In section 2.3 we give an overview of the types of information that are needed for adequate prosody computation.

2.1 Language generation

Natural Language Generation (NLG) is the process of automatically creating a natural language text on the basis of a non-linguistic information representation, for instance a table or a database record. The simplest 'language generation techniques' are based on pure string manipulation. An example is the use of so-called 'templates', string patterns that contain empty slots where other strings must be filled in. As Reiter (1995) points out, linguistic notions hardly play any role in such 'canned text' approaches, which are mainly used for very simple applications in a limited domain. On the other hand, there are more scientifically oriented approaches, where the generation process is (at least partly) guided by linguistic principles. Reiter and Dale (1997) distinguish the following basic tasks that an NLG-system should perform when going from its input data to a natural language text:

1. Content determination: deciding which information to express.
2. Discourse planning: ordering the information and determining the structure of the output text.
3. Sentence aggregation: deciding which information to put in one sentence.
4. Lexicalisation: choosing the right words to express the information.
5. Referring expression generation: creating phrases to identify domain entities.
6. Linguistic realisation: creating grammatical sentences.

Ideally, each of these tasks should be carried out in a theoretically well-founded manner. In practice, however, such sophistication is usually reserved for only a few

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

steps of the generation process. This limitation may have several causes. For instance, some NLG systems are aimed (for research purposes) at only one particular NLG task, e.g. linguistic realisation. Other tasks are then performed in a more *ad hoc* fashion (if they are performed at all). Another limiting factor for the deployment of linguistic rules in NLG is simply that not enough good linguistic rules are known yet (van Deemter, Krahmer and Theune 1999). Finally, 'linguistic' generation techniques may lack computational speed and efficiency (e.g. see Bateman and Henschel 1999), which makes them less attractive for use in applied NLG-systems.

However, the use of linguistic generation techniques also has practical advantages. First of all, output texts created using a 'canned text' approach tend to be rather simple (in particular at the discourse level) and show almost no variation.[2] In contrast, linguistic techniques allow for the generation of texts that are more complex and more coherent, making principled use of e.g., anaphora and rhetorical markers. Also, most canned text systems are entirely application-specific, and therefore not reusable. Linguistic methods are usually general in nature and therefore domain- and application independent. They are more flexible and easier to maintain. Generalising, we can say that canned text approaches offer ease of development, computational speed and efficiency at the cost of text quality, generality and flexibility, whereas for linguistic generation techniques the opposite holds.

It is clear that for the generation of all but the most simple texts, at least some linguistic knowledge is required. However, given the current state-of-the-art it is hardly possible to build text generation systems where each generation task is fully guided by linguistic principles. As a consequence, most applied NLG-systems can be characterised as *hybrid* systems, in the sense that some generation tasks are carried out on the basis of linguistic notions, whereas other tasks are performed using a non-linguistic method, e.g., by using ready-made text strings. An example is the IDAS system (Reiter and Mellish 1993), which incorporates those portions of text that are difficult to generate linguistically as 'canned text' in the output. Other recent hybrid generation systems are described in Carenini, Mittal and Moore (1994), Geldof and van de Velde (1997), White and Caldwell (1998), Busemann and Horacek (1998), and Reiter (1999).

The Language Generation Module (LGM) developed at IPO also employs a hybrid technique, using for instance well-established rules for the use of anaphors, the generation of referring expressions, and prosody computation, but performing linguistic realisation by means of hand-made sentence structures. The language generation technique of D2S is discussed in detail in section 3.1.

2.2 Speech generation

The most straightforward way to provide a system with speech output is to simply record all utterances that one wants the system to be able to pronounce, and then play them back as required. The obvious advantage is that perfect speech output

[2] Coch (1996) offers a discussion of the weak points of such texts, based on a formal evaluation of the quality of business reply letters, written by means of different techniques.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

quality can be achieved, limited only by the medium through which the speech is transmitted. The most apparent disadvantage is that this approach is impracticable in all but the simplest of applications, as it will work only for a limited number of sentences, which are exactly known beforehand.

The other extreme is to use full-fledged speech synthesis. This offers great flexibility and has none of the disadvantages inherent to the record-and-play-back scheme. It is not necessary to restrict the number of possible sentences that can be generated by the application, and addition of new material to be pronounced presents no problem. Unfortunately, there is a price to be paid for this flexibility. Although speech technology has reached the stage where synthesised speech has a high degree of intelligibility, in general the speech still sounds quite unnatural.

Another solution, somewhere between these two extremes, is phrase concatenation: entire words and phrases are pre-recorded, and these are played back in different orders to form complete utterances. Using this approach, a large number of utterances can be pronounced on the basis of a limited number of pre-recorded phrases, saving memory and disk space and increasing flexibility. The key merit of the technique is that it becomes possible to generate sentences that have never been produced as such by any human speaker, but with a quality approaching natural speech. However, this technique is practical only if the application domain is limited and remains rather stable. Commercial applications in which it is used are, for instance, travel information services (e.g. see Aust, Oerder, Seide and Steinbiss 1995), telephone banking systems, and market research tele-services.

In the conventional approach to phrase concatenation, all the necessary words and phrases are recorded only once, often pronounced in isolation. Speech is generated by concatenating these fragments to form the required utterance, which is then played. This approach (which we will refer to as 'conventional concatenation') presents two major problems:

- First, the *segmental quality* of the speech output is often suboptimal, because the recordings fail to be carefully controlled. The concatenative units are usually recorded in isolation, which causes mismatch in loudness, tempo and pitch between concatenated units, leading to disfluent speech. Phrases seem to overlap in time and create the impression that several speakers are talking at the same time, from different locations in the room. In order to disguise these imperfections, pauses are often inserted, which are very conspicuous and make the speech sound even less fluent.
- Second, the *prosodic quality* of the speech output is often suboptimal. In natural speech, the prosody of words in an utterance varies depending on several factors such as their position within the utterance, the syntactic structure of the utterance, and the discourse context. For instance, words expressing information that has been previously mentioned tend to be deaccented. (See section 2.3.) By recording all words and phrases in one prosodically neutral version, such contextual variation is not taken into account.

One simple application that does take some prosodic properties into account is the telephone number announcement system described in Waterworth (1983).

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

To increase the naturalness of the telephone number strings that are output by the system, digits are recorded in three versions with different intonation contours. There is a *neutral form*, a *continuant*, with a generally rising pitch, and a *terminator*, with a falling pitch contour. Most digits in a telephone number, e.g. *010 - 583 15 67*, are pronounced using the neutral form. However, the numbers occurring before a space, viz. *0, 3* and *5*, are pronounced using the continuant form to signal a boundary and to indicate that the utterance has not yet finished. The final *7* is pronounced with a terminator to signal the end of the string. Experiments showed that people preferred this method over the simple concatenation method.

Another application, a computer-assisted language learning program called Appeal, uses a more sophisticated form of word concatenation to deal with prosodic variations (de Pijper 1997). When making the recordings the words were embedded in carrier sentences to do justice to the fact that words are shorter and often more reduced when spoken in context. Only one version of each word was recorded, but when, during generation, the words are concatenated to form a text the duration and pitch of the words are adapted to the context using the PSOLA technique (Pitch Synchronous Overlap and Add (Moulines and Charpentier 1990)). This ensures a natural prosody, but the coding scheme may deteriorate the quality of the output speech to some extent.

Our approach to phrase concatenation can be seen as an extension to the simple concatenation approach. It is different from conventional concatenation in that (*i*) all concatenative units have been recorded embedded in carrier sentences, and (*ii*) like Waterworth's approach, it takes prosodic variation into account by recording different prosodic versions for otherwise identical words and phrases. No manipulation of the speech signal is required, thus retaining a natural speech quality. The technique is explained in detail in section 3.3.

2.3 Computation of prosody: the missing link

In the previous section we already briefly remarked that prosody depends on linguistic context. In text-to-speech systems, the linguistic context of a word or phrase must be obtained through linguistic analysis of the input text. Such an analysis may yield unreliable and incomplete results, which has a negative impact on the prosodic quality of the speech output. However, in data-to-speech the text which is to be made audible has been generated by the system itself, so information about linguistic context is present in the language generation component. To exploit this information, different solutions are possible. One solution is to have a monolithic architecture, where language and speech generation are closely integrated. This design may be efficient, because all relevant information is directly available, but it has the disadvantage that language and speech generation are so closely intertwined that it is impossible to reuse either component in another system. An example of such an intertwined architecture is the SSC (Speech Synthesis from Concept) system proposed by Young and Fallside (1979). In this system, the computation of prosody is done during speech generation. The two tasks are inseparable, and fully dependent

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

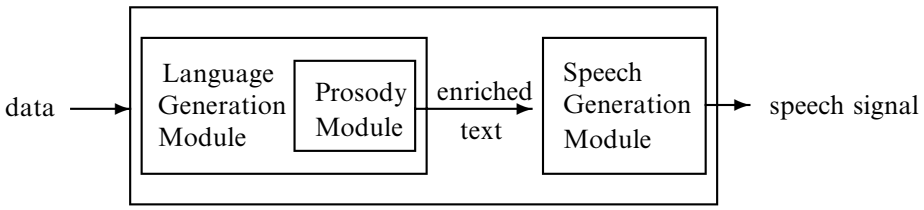


Fig. 1. Global architecture of D2S.

on the specific input (a full syntactic structure for an utterance) provided by the preceding language generation component.

An alternative solution, proposed by Pan and McKeown (1997) is to have an architecture in which language and speech generation are independent modules which are interfaced by a general prosodic component. The advantage of such an architecture is that the language and speech generation modules are reusable for different applications, and that the intermediate prosody component can in principle be used to couple different language and speech generation components. However, in practice the usability of such a prosodic component may be restricted by the variety of linguistic information provided by, and representation formalisms used in, current NLG-systems. Given this variety, a separate pre-processing module will be required for almost every NLG-system the prosodic component is to be coupled with.

Again, our approach is somewhere in between. In D2S, language and speech generation form separate, reusable modules, which are connected through a prosodic component (see figure 1). However, the prosodic component is not an independent module in the system, but is embedded in the language generation module, with which it shares a mutual knowledge source, containing information about the context. This information is used for computing the placement of pitch accents and phrase boundaries. The advantage of this architecture is that both language and speech generation are reusable: the language generation module can be used stand-alone (without speech output) or in combination with different speech output methods (discussed in Section 3.3). In their turn, the speech output methods currently employed in D2S can be used in other systems, given that the required prosodic mark-up is provided in the input. Only the prosody module, which is inherent to the LGM, cannot be ported to another system.

We now give a brief overview of the kind of linguistic information that is relevant for prosody computation. First, information about the preceding discourse should be available, because it is important for the placement of pitch accents. It is generally accepted that in Dutch, as in other Germanic languages, accent functions (among other things) as a marker of *information status*: words or phrases are usually accented if they express information that is assumed to be *new* to the hearer, whereas they are usually unaccented if they express *given* information, i.e., information that is assumed to be known to the hearer, for instance because it is has been mentioned previously in the discourse (Halliday 1967; Chafe 1976; Brown 1983). Experimental work has

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

shown that accenting new information, while leaving given information unaccented, facilitates comprehension (Bock and Mazzella 1983; Terken and Nooteboom 1987). The notion of information status also covers *contrast* of information: phrases expressing contrastive information are always accented, even if the information they express may be regarded as given (Chafe 1974; Hirschberg 1992; Prevost 1995; Theune 1999). Example (1) illustrates the importance of information status (accented words are printed in small capital letters):

(1) Last week, PRESIDENT CLINTON visited the FRENCH CAPITAL.
His WIFE doesn't LIKE France, but the PRESIDENT really LOVES it.

The second sentence of example (1) contains three instances of deaccentuation due to givenness: the words *his*, *France* and *it*. The pronoun *his* is not accented, because it refers to president Clinton who was mentioned in the preceding sentence and is therefore already 'known' to the hearer. The word *France* is not accented because the concept 'France' has been evoked in the previous sentence (by the use of the adjective *French*) and can therefore also be regarded as given. Finally, the pronoun *it* also refers to France and is therefore not accented either. On the other hand, the phrase *the president* in the second sentence expresses given information (like the pronoun *his*, it refers to Clinton) but, since it contrasts with the phrase *his wife*, the word *president* is still accented.

The placement of accents in a sentence is not only influenced by the preceding discourse, but also by the syntactic structure of the sentence. Generally, not all words within a phrase expressing new or contrastive information are accented; which words are, depends among other things on the syntactic structure of the phrase. In Dutch and English, it is usually the rightmost word in a phrase which is accented, unless factors like givenness override this default.

Syntactic information is not only relevant for the distribution of pitch accents, but also for the placement of phrase boundaries within an utterance. As shown by Pierrehumbert and Hirschberg (1990), Sanderman (1996) and others, the prosodic phrase structure of a sentence is co-determined by its syntactic structure. An example is shown below in (2) (adapted from Sanderman (1996)). To indicate that the PP *with the stick* modifies the VP, a phrase boundary (indicated by a slash) may be placed after the word *dog*, as in (2)a. If the PP modifies the NP (as in (2)b), such a phrase boundary is less appropriate.

(2) a The man hit [$_{NP}$ the DOG] / with the STICK
b The man hit [$_{NP}$ the DOG with the STICK]

This brief overview indicates that, to compute the placement of accents and phrase boundaries, it is important to have information about the discourse and about the syntactic structure of the generated sentences. In the next section, we discuss in detail how the language generation module of D2S makes such information available and how it is used for the computation of accents and phrase boundaries.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press



Fig. 2. Example Teletext Page, containing data from two football matches. (Arbiter = referee; Toeschouwers = spectators; Geel = yellow card).

3 Description of D2S

In this section we use examples from a data-to-speech system called GoalGetter to illustrate D2S and the techniques it is based on. As noted in the introduction, various applications have been developed on the basis of D2S. Of these, GoalGetter is the least complex one due to its limited domain. This makes it suitable for use as an example, but it will also leave some aspects of D2S (in particular the method of *topic management* employed in the LGM – see section 3.1.3) somewhat underexposed. This will be further discussed in the relevant sections below.

The GoalGetter system generates Dutch spoken summaries of football matches. The data which form the input for GoalGetter are automatically retrieved from *Teletext*, a system with which textual information is broadcast along with the television signal and decoded in the receiver. The information is distributed over various 'pages', each filling a screen, which are continuously refreshed and are also available via the Internet. Some pages contain textual information, e.g. news messages, and some contain tables, e.g. weather reports and sports results. Figure 2 shows an example Teletext page, which contains information about two football matches.

For each match on the Teletext page, information about the home team is shown on the left; information about the visiting team is shown on the right. Behind each team name, this team's result is shown. Below it, a list is given of all players who scored a goal for this team. The minute in which a certain player has scored is given between brackets behind the player's name. If the goal was not a 'normal' one, this is indicated using a specific marker; for instance, */pen* indicates a penalty. Then, the name of the referee (*arbiter*) and the number of spectators (*toeschouwers*) are given. Finally, for each team a list is given of all players who received a card. (If a player commits a minor offence, he receives a yellow card. For a major offence, he receives a red card and is sent off the field. Two yellow cards amount to one red card.)

An example output text, describing the first match of figure 2, is given in figure 3,

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

Output:

Go Ahead EAGLES / ging op BEZOEK bij Fortuna SITTARD // en speelde GELIJK ///
Het duel eindigde in TWEE // - TWEE ///
VIJFENVEERTIG honderd TOESCHOUWERS / kwamen naar 'de BAANDERT' ///
<*new-par*>
De PLOEG uit SITTARD / nam na ZEVENTIEN MINUTEN de LEIDING / door een TREFFER van HAMMING ///
EEN minuut LATER / bracht SCHENNING van Go Ahead EAGLES / de teams op GELIJKE HOOGTE ///
Na ACHTENVEERTIG minuten / liet de AANVALLER HAMMING / zijn TWEEDE doelpunt aantekenen ///
In de VIJFENZESTIGSTE minuut / bepaalde de Go Ahead EAGLES speler DECHEIVER de EINDSTAND / op TWEE // - TWEE ///
<*new-par*>
De wedstrijd werd GEFLOTEN door SCHEIDSRECHTER UILENBERG ///
Hij deelde GEEN RODE KAARTEN uit ///
MARBUS van Go Ahead EAGLES / liep tegen een GELE kaart aan ///

Translation:

Go Ahead EAGLES / visited Fortuna SITTARD // and DREW ///
The duel ended in TWO // - ALL ///
FOUR thousand FIVE hundred SPECTATORS / came to 'de BAANDERT' ///
<*new-par*>
The TEAM from SITTARD / took the LEAD after SEVENTEEN MINUTES / through a GOAL by HAMMING ///
ONE minute LATER / SCHENNING from Go Ahead EAGLES / EQUALISED the score ///
After FORTY-EIGHT minutes / the FORWARD HAMMING / had his SECOND goal noted ///
In the SIXTY-FIFTH minute / the Go Ahead EAGLES player DECHEIVER brought the FINAL SCORE to TWO // - ALL ///
<*new-par*>
The match was OFFICIATED by REFEREE UILENBERG ///
He did NOT issue any RED CARDS ///
MARBUS of Go Ahead EAGLES / picked up a YELLOW card ///

Fig. 3. Example output of the LGM. Accents are indicated by small capital letters, phrase boundaries by /, // or ///, and the start of a new paragraph by <*new-par*>.

together with its translation. The output text is given in enriched text format, i.e. including prosodic mark-up. Accented words are printed in small capital letters, and phrase boundaries of different strengths are indicated by a number of slashes (/, // or ///). The marker <*new-par*> indicates the start of a new paragraph.

In the following subsections, we follow the general architecture of D2S as shown in figure 1. First, in section 3.1 we describe the Language Generation Module (LGM) and illustrate its workings using an example sentence from figure 3. Then, in section 3.2 we explain how the prosodic markers are assigned by the prosodic component embedded in the LGM. Finally, in section 3.3 we discuss how these markers are used by the Speech Generation Module (SGM).

3.1 Language generation in D2S

The language generation module of D2S (from now on abbreviated as LGM) was designed for spoken information presentation in situations where the user is likely to hear several presentations in succession. Variation in the generated presentations is important here, and this is reflected in the architecture of the LGM, sketched in section 3.1.1. In particular, the LGM has no global text planner but instead makes use of a set of syntactic templates with conditions on their use (discussed

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

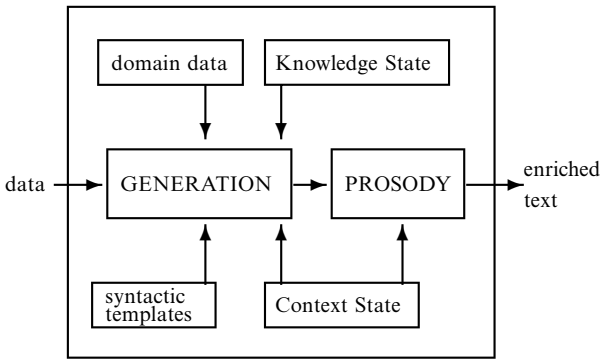


Fig. 4. The architecture of the Language Generation Module (LGM).

in section 3.1.2). Combined with the use of topic information to achieve coherence, these conditions act as a kind of local, reactive planner, as discussed in section 3.1.3. The selection of syntactic templates and the filling of their slots is discussed in section 3.1.4; finally, a detailed example is given in section 3.1.5.

3.1.1 Architecture

The general architecture of the LGM is depicted in figure 4. The module *Generation* contains the basic generation algorithm of the LGM (see figure 9). It takes data from outside the system as input; in GoalGetter these are data concerning the characteristics of a particular football match. Because the Teletext pages providing football results have a fixed format, a simple parser can be used to convert the information they contain into the typed data structures that are used by GoalGetter as a basis for generation. Figure 5 shows the LGM's input data structure for the first match of figure 2.

In addition to the input data, the LGM also uses *domain data*, i.e. a collection of relatively fixed background data on the relevant domain. In GoalGetter these are data about the football teams and their players, such as the home town of each team and the position and nationality of each player. These data are stored in typed data structures for the teams and their players. An example is the feature structure containing information about the team Fortuna Sittard, shown in figure 6. The domain data serve as a supplement to the system's input data from Teletext, and are used to achieve more variation in the generated texts by providing additional information about the players and teams that are mentioned. For instance, knowledge about the positions of the players provides the system with more possibilities for the generation of referring expressions than if only the Teletext data were available. Examples of the use of domain data in our example text (figure 3) are the reference to 'de Baandert' (the stadium of Fortuna Sittard); the reference to Fortuna Sittard as *the team from Sittard* and the second reference to Hamming as *the forward Hamming*. On the basis of only the input data shown in figure 5 (and derived from the first half of figure 2), 'de Baandert' could not have been referred to, and both teams and players could only be referred to using their proper names. Note that the

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

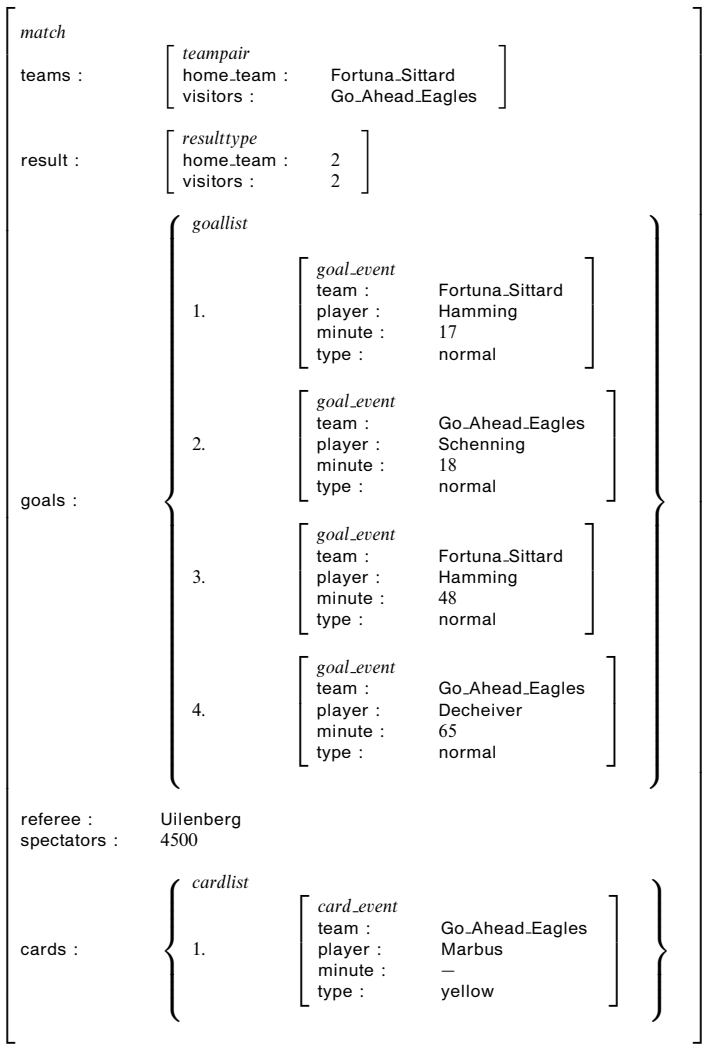


Fig. 5. Data structure representing the first match from figure 2.

domain data are not necessarily known to the user. They are mainly used to provide additional information about domain objects, not to identify them.

The module *Generation* additionally uses a collection of *syntactic templates* to express (parts of) the input data. Syntactic templates contain syntactic tree structures with open slots for variable information. The syntactic information from the templates[3] is used for prosody computation (see section 3.2), and for checking certain grammatical conditions (see section 3.1.4). Each syntactic template has a (complex) condition on its use, and the interplay between these conditions during generation

[3] Note that in the remainder of this paper, the word 'templates' is used to refer to *syntactic* templates of the kind used in the LGM, not the simple templates discussed in section 2.1.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

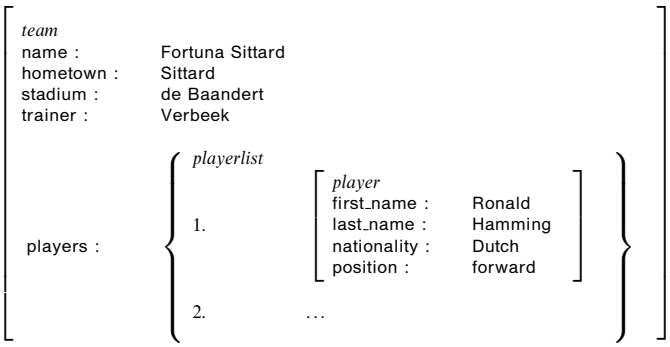


Fig. 6. Background data structure for the team Fortuna Sittard.

determines the structure of the generated text. The form and content of the syntactic templates and their use in generation are discussed in detail below.

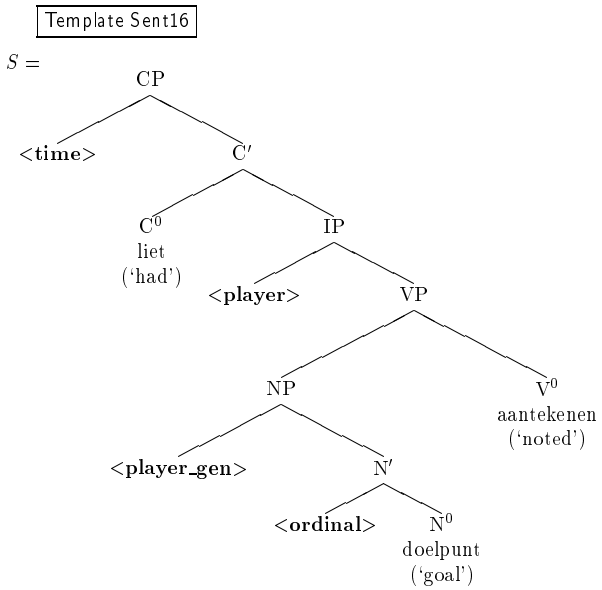
During generation, two records are kept. One of them is the *Knowledge State*. It records which parts of the input data structure have been expressed by the system (these are assumed to be *known* to the user) and which parts have not (these are assumed to be *unknown*). The Knowledge State takes the form of a labelling on all fields in the input data structure, indicating whether their values are known to the user. Initially, all fields in the input data structure are labelled 'unknown'. After generation of a sentence that expresses one or more fields of the input data structure, these fields are labelled as 'known'. The Knowledge State information is used to guide the selection of templates by the *Generation* module.

In addition to the Knowledge State, there is another record which is kept during generation: the *Context State*, which records various aspects of the linguistic context (i.e. the part of the text that has so far been generated). A central part of the Context State is the *Discourse Model*, which keeps track of the discourse objects that have been mentioned. The information in the Context State is used, among others, during the generation of referring expressions and the computation of prosody. For a detailed discussion of the modelling and use of contextual information in the LGM, see van Deemter and Odijk (1997). Finally, the *Prosody* component computes the prosodic features of each generated sentence.

3.1.2 Syntactic templates

One of the main characteristics of the LGM is the usage of *syntactic templates*. Each template can be used to express one or more parts of the system's input data structure. Figure 7 contains an example from GoalGetter, which has been used to generate the sixth sentence from the example text in figure 3. Formally, a syntactic template σ is a quadruple $\langle S, E, C, T \rangle$, where S is a syntactic tree (typically for a sentence) with open slots in it, E is a set of links to additional syntactic structures which may be substituted in the gaps of S, C is a (possibly complex) condition on the applicability of σ and a T is a set of topics. Let us discuss the four components of the syntactic templates in some more detail, beginning with the *syntactic tree* S.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press



$E =$ **time** $\leftarrow$ ExpressTime $(currentgoal.minute)$
player $\leftarrow$ ExpressObject $(currentgoal.player,\ nom)$
player_gen $\leftarrow$ ExpressObject $(currentgoal.player,\ gen)$
ordinal $\leftarrow$ ExpressOrdinal $(ordinalnumber)$

$C =$ Known $(match.teams) \wedge$
$currentgoal =$ First $(unknown,\ match.goals) \wedge$
GoalsScored $(currentgoal.player) > 1 \wedge$
$currentgoal.type \neq owngoal$

$T =$ 'game_course'

Fig. 7. Syntactic template used for the generation of the sixth sentence of figure 3, *Na achtenveertig minuten liet de aanvaller Hamming zijn tweede doelpunt aantekenen* ('After forty-eight minutes the forward Hamming had his second goal noted'). (CP = Complementiser Phrase, IP = Inflectional Phrase).

The syntactic tree structures in the templates bear a certain resemblance to the initial trees of Tree Adjoining Grammar (TAG) (Joshi 1987): all interior nodes of the tree are labelled by non-terminal symbols, while the nodes on the frontier are labelled either by terminal or non-terminal symbols, where the non-terminal nodes on the frontier are the gaps which are open for substitution. A notable difference with TAG trees is that the latter are generally 'minimal', i.e. only the head of the construction is lexicalised and the gaps coincide with the arguments of the head, whereas the syntactic trees in the templates may contain more words, often in order to express collocations (groups of words with a frozen meaning). Examples of collocations occurring in the GoalGetter templates are *een doelpunt laten aantekenen* ('have a goal noted') (as in Template Sent16) or *de leiding nemen* ('take the lead'). (See Klabbers, Krahmer and Theune (1998) for some further discussion.) The syntactic trees in the templates are given in full detail because during prosody computation (see section 3.2) they need to be converted into full metrical trees.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

The second element of a syntactic template is E: *the slot fillers*. Each open slot in the tree S is associated with a call of a so-called Express function, which generates the set of possible slot fillers for the given gap.

The third ingredient is *C: the condition*. A template σ is applicable if and only if its associated condition is true. Two kinds of conditions can be distinguished: (*i*) conditions on the Knowledge State and (*ii*) linguistic conditions. Conditions of the former type state things like 'X should not be conveyed to the user before Y is conveyed'. The first two (sub)conditions of Template Sent16 are of this kind. They state that the template can only be used if the teams involved in the match have been conveyed to the user (i.e. are known) and the current goal is the first one which has not been conveyed (is unknown). The first condition has to do with the desired global discourse structure: in GoalGetter, we have chosen the strategy of presenting general information first, and then giving further details, so we want the competing teams to be known to the user before describing who scored when. Therefore, the first condition checks if the teams field of the input match has been labelled 'known'. The second condition ensures that the template only expresses goals which have not been previously described: the function First takes the first *goal_event* from the goals list that is labelled 'unknown'. If there is no such goal (i.e. all goals have been described), the template is not applicable.

'Linguistic' conditions are related to the semantics/pragmatics of the sentence that can be generated from the template, and pose restrictions on the kind of input data to which the template can be applied. The two final conditions on Template Sent16 are of this type. The first of the two says that Sent16 is only applicable if the player of the current goal has scored more than once during the match. Because a sentence of the form *X had his first goal noted* creates the impression that player X has scored more than one goal, we do not want such a sentence to be used if X has actually scored only once. The final condition on Template Sent16 states that this template cannot be used if the current goal is an own goal. This restriction is added because using the phrase *having a goal noted* to describe an own goal would give rise to a false conversational implicature (Grice 1975) by creating the impression that the current goal is a normal goal when, in fact, it is not.

Finally, each template σ contains a set of one or more *topics* T. These are labels which globally describe what the syntactic template is about. The LGM algorithm uses the topic information to group sentences together into coherent chunks of text. Each topic has several templates associated with it, and each template is associated with one or more topics. This situation can be illustrated using the simple Venn diagram in figure 8, which represents the topics and templates of the GoalGetter system. In GoalGetter, which is a relatively small system, there are only three topics: (*i*) 'general' (giving global information about, for instance, the names of the opposing teams and the final result of the match), (*ii*) 'game_course' (giving information about events which occurred at a specific time during the match) and (*iii*) 'game_statistics' (giving details of the match that are not necessarily associated with a specific time, e.g. bookings of specific players). The GoalGetter system currently contains approximately thirty syntactic templates, not all of which are shown in figure 8. Some of these templates belong to more than one topic; for instance, information

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

about red cards (which cause a player to be sent off the field during the game) can either be expressed as part of the 'game_course' or as part of the 'game_statistics'. Similarly, information about the number of spectators of a match may be seen as part of either the 'general' information or the 'game_statistics'.

Not all of GoalGetter's templates are used in each text; for instance, if no own goals occurred in the match to be described, the templates that express the scoring of an own goal are not used. In addition, each piece of information from the input data structure can typically be expressed using more than one template. For instance, in GoalGetter there are four different templates available to convey information about the referee of a match. The selection of templates and the form of 'planning' used in the LGM are discussed below.

3.1.3 Topics, conditions and coherence

Given that we have a set of syntactic templates, which we can use to create sentences expressing parts of the input data, we need a method for combining these sentences into a coherent output text. Various approaches are possible here. For instance, we could write an explicit grammar which states where every sentence can occur. A different approach would be to make use of a form of text planning where, before linguistic realisation, the pieces of information to be conveyed are grouped in such a way that a coherent text results. In the LGM we take a different approach, starting as it were from the other side: instead of explicitly specifying in advance where in the output each sentence should occur, we assume that in principle each sentence can occur anywhere, but that conditions prohibit their use in some cases. With this approach, we try to achieve maximal variation in the output texts. Variation is of high importance, because we expect the users of typical D2S applications to listen to several texts in succession. If these texts do not show sufficient variation, this will presumably be slightly boring (Odijk 1995).[4]

So, we wish to generate texts that are both varied and coherent. We assume that there are two main factors determining the coherence of a text: (i) the information must be presented in a natural order, and (ii) the information must be presented in natural groupings. To ensure a natural grouping of the sentences in the output of the LGM, the topics associated with the templates are used. Each topic corresponds to a paragraph in the generated text, which contains only sentences that have been generated from templates belonging to that specific topic. In the GoalGetter example in figure 3, the first paragraph corresponds to the 'general' topic, the second paragraph is about the 'game_course' and the third about 'game_statistics'.

The ordering of the paragraphs in a text and of the sentences within a paragraph is determined by the conditions on the templates. A template can be used if it belongs to the topic of the current paragraph, and if its conditions evaluate to true given the current Knowledge State. If more than one template is applicable in the current state of the generation process (and this will often be the case), one is

[4] We conjecture that the pleasure derived from variation is proportional with the length of the generated text, and with the number of similar texts to be read or listened to.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

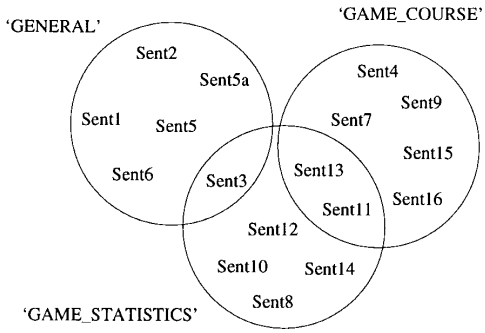


Fig. 8. Topics and templates.

chosen arbitrarily. After a sentence has been generated from the chosen template, the Knowledge State is updated and new templates become applicable. If there are no more applicable templates within the current topic, a new topic must be chosen. There is no *a priori* ordering on the topics; whether a new paragraph can be started given a topic T depends on the applicability of the templates within that topic. If there are no templates associated with T whose conditions evaluate to true in the current Knowledge State, T must be skipped until the Knowledge State has been sufficiently changed for some of its templates to be applicable. Below, we discuss the generation algorithm in detail and illustrate it using some examples.

One might argue that the conditions on the templates act as a distributive, reactive planner, in the sense that the conditions are spread across the templates and respond to the current stage of the generation process. This 'local condition' approach makes it possible to formulate certain general principles on the presentation of information (e.g. that global information is presented first) without having to specify exactly at which point in the output text each piece of information should be conveyed. This enables the system to achieve a high degree of variation in the generated texts, which is assumed to be pleasant for the hearer.

The GoalGetter system is not a typical D2S application in the sense that it does not fully exploit the possibilities for variation offered by the LGM's planning mechanism. The limited variation in the output of GoalGetter is a consequence of the relatively small amount of available data, in combination with the structured nature of a football report (most notably the chronological description of the course of the game). However, other D2S systems have a higher number of topics and templates and a less strict ordering among the topics, thus allowing for much more variation. For instance, the DYD system (van Deemter *et al.* 1994; Odijk 1995; van Deemter and Odijk 1997) has nine topics which each discuss a different aspect of a Mozart composition, and may occur in virtually any order, thus reflecting the associative process of describing a composition. In general, we can say that some types of output text (e.g. descriptions) offer more opportunities for variation than others (e.g. football reports). For each application, the desired amount of variation can be achieved by making the conditions on the templates either very global or

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

very strict. Some other templates and their conditions are shown in section 3.1.5. For a more detailed discussion of 'local planning' in the LGM, see Odijk (1995).

3.1.4 Algorithm

The LGM generation algorithm is shown in figure 9. Its input is formed by the set of topics (*all_topics*), the set of all syntactic templates (*all_templates*), the initial Knowledge State (*InitialKS*) and the initial Context State (*InitialCS*). In the initial Knowledge State it is recorded that all parts of the input data structure are marked as 'unknown', and the initial Context State is simply empty (no discourse entities have been introduced yet). The algorithm starts by initialising two variables, *relevant_topics* and *untried_topics*. The first variable, *relevant_topics*, contains the set of topics that have not yet been used as the basis for a paragraph (none of their templates has been applied yet). The second variable, *untried_topics*, contains the set of topics which the algorithm has not yet tried to use in its current generation round. Both variables are initialised as the set of all topics. Finally, the variables KS and CS are initialised to *InitialKS* and *InitialCS* respectively.

After initialisation the algorithm gets into its first **while** loop, walking through the set of *untried_topics*. As long as this set is not empty, the algorithm performs the following actions. It starts by randomly picking one of the *untried_topics* and makes it the *current_topic*. Using this topic, it will try to start a new paragraph. First, the variable *possible_templates* is instantiated as the set of those templates that are associated with the *current_topic* and whose conditions evaluate to true in the current Knowledge State. These are the templates that are currently applicable. (It may be that at this point, none of the templates associated with the *current_topic* are applicable yet; in that case, *possible_templates* is empty.) Then the variable *topic_successful* is set to false; this variable records if any template associated with the *current_topic* has been applied successfully (i.e. has produced a sentence). If *possible_templates* is not empty, the second **while** loop is entered. In this loop, the algorithm goes through the set of *possible_templates*, picking one of them at random (*chosen_template*) and trying to apply it by giving it as an argument to the function ApplyTemplate, which is discussed in detail below. If ApplyTemplate is not successful, it outputs nil. In that case, the algorithm removes the *chosen_template* from the set of *possible_templates*, picks another template from this set and tries to apply it. If the sentence produced by ApplyTemplate is not nil, the algorithm orders the Speech Generation Module to pronounce it. The Knowledge State is updated so that the data expressed by means of *chosen_template* are marked as 'known', and the Context State is updated by (among other things) adding the discourse entities mentioned in the *chosen_template* to the Discourse Model. *Topic_successful* is then set to true, since one of the templates from the *current_topic* has been applied successfully. Now, the set of *possible_templates* is re-instantiated given the updated Knowledge State. (Because the Knowledge State has changed by applying a template, new templates may have become applicable and others unapplicable.) From the new set of *possible_templates*, one template is picked and the procedure starts all over again. This continues until there are no applicable templates left

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

```
Generate(all_topics, all_templates, InitialKS, InitialCS)
relevant_topics ← all_topics
untried_topics ← all_topics
KS ← InitialKS
CS ← InitialCS
while untried_topics ≠ {}
do current_topic ← PickAny(untried_topics)
    possible_templates ← { t ∈ all_templates | current_topic ∈ Topic(t) ∧
                                               Conditions(t, KS) = true }
    topic_successful ← false
   while possible_templates ≠ {}
   do chosen_template ← PickAny(possible_templates)
       sentence ← ApplyTemplate(chosen_template, CS)
       if sentence = nil
       then possible_templates ← possible_templates − chosen_template
       else Pronounce(sentence)
            KS ← UpdateKS(KS, chosen_template)
            CS ← UpdateCS(CS, chosen_template)
            topic_successful ← true
            possible_templates ← { t ∈ all_templates | current_topic ∈ Topic(t) ∧
                                                       Conditions(t, KS) = true }
       endif
   endwhile
   if topic_successful = false
   then untried_topics ← untried_topics − current_topic
   else relevant_topics ← relevant_topics − current_topic
        untried_topics ← relevant_topics
        StartNewParagraph
   endif
endwhile
```

Fig. 9. The basic generation algorithm of the LGM.

within the *current_topic*. The algorithm then leaves the second **while** loop and checks if the *current_topic* has been successful. If not, it means that nothing has happened; no sentences have been generated. In that case, *current_topic* is removed from the set of *untried_topics*. The set of *relevant_topics* does not change, so that the *current_topic* may be tried again later. The algorithm now picks a new topic from the remaining *untried_topics*, and tries to start a paragraph with that topic.

If *topic_successful* is true, this indicates that a paragraph of one or more sentences has been generated, so the algorithm now has to start a new paragraph. The *current_topic* is removed from the *relevant_topics*, and the set of *untried_topics* is instantiated as the set of all *relevant_topics*. This means that all topics that were unsuccessful in a previous Knowledge State, can now be tried again as the basis for the new paragraph. The algorithm picks a new topic and continues until there are no *untried_topics* left. This concludes the discussion of the main generation algorithm; we now turn to the function ApplyTemplate, shown in figure 10.

ApplyTemplate attempts to generate a sentence from a template, given the current Context State. This is done by creating a set of all sentences that result from all relevant slot fillings of the template, and then filtering out those sentences that

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

```
ApplyTemplate(template, CS)
allowed_trees ← {}
sentence ← nil
all_trees ← FillSlots(template, CS)
for each member t_i of all_trees do
    if Violate_BT(t_i) = false
    then allowed_trees ← allowed_trees + t_i
    endif
if allowed_trees ≠ {}
then chosen_tree ← PickAny(allowed_trees)
     final_tree ← AddProsody(chosen_tree, CS)
     sentence ← Fringe(final_tree)
endif
return sentence
```

Fig. 10. The function ApplyTemplate.

violate the Binding Theory (Chomsky 1981). From the remaining sentences, one is picked at random. More specifically, ApplyTemplate works as follows. First, it calls the function FillSlots(*template, CS*) to obtain the set of all possible trees that can be generated from the template, using all possible combinations of slot fillers generated by the Express functions associated with the slots in the templates. Typically, there are several possible slot fillings for each slot in a template. For instance, a person may be referred to using a proper name, a definite description or a pronoun (if the Discourse Model, which is part of the Context State, contains an appropriate antecedent).[5] As a consequence, FillSlots typically returns a set of several trees, all expressing the same piece(s) of information albeit in different ways. For each tree in this set it is checked whether it obeys the Binding Theory, for instance to see if there are any non-pronouns in a bound position (Chomsky 1981); see the example below. The trees that are not in line with Binding Theory are filtered out, and the system arbitrarily selects one of the remaining trees. This tree is sent to the Prosody module, where its prosodic properties are computed using both syntactic and contextual information. (This is described in section 3.2). ApplyTemplate then returns the sentence consisting of the tree's terminal nodes plus their prosodic markings (i.e. the 'fringe' of the tree).

As with the random choice of topics and templates, the making of an arbitrary choice from the suitable trees is motivated by the need for variation within and between the generated texts. We are aware that this strategy still leaves room for improvement, for instance, because making a random choice is not a guarantee for optimal variation. Finally, note that the 'generate and test' strategy employed at several stages of the generation process does not lead to inefficiency, because the LGM is written in a programming language that uses *lazy evaluation*: expressions are only evaluated when necessary.

[5] Definite descriptions and pronouns are generated using a modified and extended version of Dale and Reiter's (1995) MakeReferringExpression algorithm for the generation of referring expressions. For details, see Krahmer and Theune (1999) and Theune (2000).

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

3.1.5 Example

We now illustrate the workings of the generation algorithm using (parts of) the text in figure 3 as an example. For ease of exposition, throughout this section we only refer to the English translation of the output generated by GoalGetter.

After initialisation of *relevant_topics* and *untried_topics* as the set {'game_course', 'game_statistics', 'general'}, and initialisation of the Knowledge State, the algorithm starts by randomly picking a topic from *untried_topics*. Let us assume the selected topic is 'game_course'. Now *topic_successful* is set to false (no sentence for this topic has been uttered) and the set *possible_templates* is constructed of all templates which are associated with 'game_course' and whose conditions are true given the current Knowledge State, which says that all parts of the input data structure are still unknown to the user. In this case, it turns out that the set of *possible_templates* is empty: there are no 'game_course' templates which are applicable in the initial Knowledge State, when no information about the match has been conveyed yet. This is because all templates in 'game_course' have as their condition that they can only be used if the competing teams are known to the user: before providing details about which player did what during the match, the teams should have been introduced. Because there are no applicable templates, the 'game_course' topic cannot be used for the first paragraph. This means that the attempt with this topic has finished without being successful, so after having removed 'game_course' from the *untried_topics* the algorithm starts a new generation round. Although 'game_course' remains a relevant topic (nothing has been said about it yet), this time the algorithm can only choose from the two topics which have not yet been tried, 'general' and 'game_statistics'. We assume that now the 'general' topic is picked.

For explanatory purposes, let us assume that this topic only contains the three templates that have been used to generate the first three sentences of the text in figure 3. (This is a severe simplification.) These templates are shown in an abbreviated form in figure 11; we call them Sent1, Sent2 and Sent3 respectively. Instead of showing the full syntactic trees of the templates, only the 'flat' sentences with the slots are shown (in translation); in addition, we left out the topic information and the calls of the Express functions, and slightly simplified the conditions.

For the 'general' topic, in the initial Knowledge State the set of *possible_templates* is not empty: it contains both Sent1 and Sent2, which do not require any information to be known before being applied. One of the two templates is chosen at random; this happens to be Sent1. This template can be successfully applied by filling the <team1> slot with the name of the visiting team and the <team2> slot with the name of the home team. (The ApplyTemplate function will be illustrated in more detail below.) After the template has been applied, the resulting sentence (the first sentence of Figure 3) is pronounced by the SGM, and the Knowledge State is updated with the information that the teams field of the match is now known to the user. The Context State is updated as well, among other things by extending the Discourse Model with discourse entities corresponding to the teams and the game. Then *topic_successful* is instantiated as true (the first sentence of a paragraph has been generated) and the set of *possible_templates* is computed anew, given the

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

Template Sent1

S = <**team1**> visited <**team2**> and drew.
C = Unknown (*match.teams*) ∧
match.result.home_team = *match.result.visitors*

Template Sent2

S = <**match**> ended in <**result**>.
C = Unknown (*match.result*)

Template Sent3

S = <**spectators**> visited <**stadium**>.
C = Unknown (*match.spectators*) ∧
Known (*match.teams*)

Fig. 11. Abbreviated templates from the 'general' topic. The syntactic structures, the topics, and the calls of the Express functions, used to fill the gaps, are omitted.

updated Knowledge State. Because the competing teams are now known, Sent1 can no longer be used. Sent2 is still applicable because the result of the match has not been explicitly conveyed. In addition, Sent3 has now become applicable because the teams are known. Therefore, *possible_templates* = {Sent2, Sent3}. Now assume that Sent2 is chosen to be applied. Its first slot is filled with an expression for the match. In this case, the definite description *the duel* is used, which is generated using the function ExpressObject, discussed below. The second slot of the template is filled with a tree expressing the result of the match. After application of this template, the result of the match is marked as known. This means that Sent2 is no longer applicable; the only template left is Sent3. After this template has been applied successfully (using the domain information that the stadium of Fortuna Sittard is called 'de Baandert'), no more applicable templates are left within the topic, so the paragraph is finished. 'General' is removed from the *relevant_topics*, and the *untried_topics* are instantiated as the *relevant_topics*, i.e. {'game_course', 'game_statistics'}. The algorithm now starts a new paragraph, which we will use to illustrate the working of ApplyTemplate and the Express functions.

For the new paragraph, the algorithm now picks 'game_course' from the *untried_topics*. Because the result of the match has been conveyed in the previous paragraph, this time there are several templates that can be applied given the current Knowledge State. One of them is picked at random and used to generate the fourth sentence of figure 3: *The team from Sittard took the lead after seventeen minutes through a goal by Hamming*. Consequently, the Knowledge State is updated to reflect the fact that information about the first goal has been conveyed. In addition, the Discourse Model is extended with entities corresponding to the phrases *the team from Sittard*, *Hamming* and *after seventeen minutes*. They receive an index indicating to which parts of the data structure they refer. Now the system goes on and attempts

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

to convey the second goal scoring event. It cannot use the same template as the one used for the fourth sentence, since the second goal scoring event of the current match does not make one team take the lead. Instead, a template is used that is applicable if the scores of two teams are equalised: *One minute later Schenning from Go Ahead Eagles equalised the score*. A noteworthy aspect of this sentence is the time expression that is used, viz. the expression *<N> minute/minutes later*. This expression can only be used if the Discourse Model contains an appropriate reference time, i.e. if the most recent time expression in the Discourse Model is an *explicit* one. The variable <N> is then a cardinal expression for an integer which equals the time value of the current event minus the time value of its *reference time*. Since the current Discourse Model contains an appropriate reference time entry, viz. the expression *after seventeen minutes*, the expression *one minute later* can be used here.

Now a sentence must be generated to describe the third goal of the match. To do this, Template Sent16 (shown in figure 7) is selected. As the reader can verify, all conditions associated with this template are met. After having selected Sent16, the system attempts to generate a sentence from it using the function ApplyTemplate from figure 10 which will now be illustrated in some detail. ApplyTemplate first calls FillSlots to obtain the set of all possible trees that can be generated from the template, using all possible combinations of slot fillers generated by the associated Express functions. Let us start with the first slot, <time>. The function ExpressTime can generate several time expressions, but one of them is not allowed given the current context: since the most recent time expression in the Discourse Model (*one minute later*) is not explicit, it cannot serve as a reference time for a second expression of the form '<N> minute/minutes later'. Such an expression is therefore not allowed here, and ExpressTime only returns two possible slot fillings: the explicit time expressions *in the forty-eighth minute* and *after 48 minutes* (which we take to be synonymous, although strictly spoken this is not true).

The second slot to be filled is the <player> slot. The function ExpressObject is called to generate an expression (in the nominative case) for the player who scored the third goal (Hamming). Again there are several options: a player can be described using a definite description (expressing the player's position or nationality attribute, or both; see figure 6) a pronoun, a proper name, or an appositive that combines a definite description and a proper name. However, as before, not all options are appropriate given the current context. Let us first consider the first two options: pronoun and definite description. Depending on the context, the MakeReferringExpression algorithm that is called in ExpressObject generates either a pronoun (if there is an appropriate antecedent in the Discourse Model) or a definite description. In the current example, a pronoun cannot be used: the antecedent (*Hamming* in sentence four) is not accessible due to the intervening reference to the player Schenning. Neither can a definite description be generated: referring to Hamming's nationality (Dutch), does not distinguish him from Schenning, who is also Dutch, and although Hamming's position (forward) does in fact distinguish him from Schenning (who happens to be a midfielder) we assume that the hearer is unaware of this. This means that describing Hamming as either *the Dutchman*, *the forward* or even *the Dutch forward* is insufficiently distinguishing. In contrast,

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

<**time**>	{*in the forty-eighth minute, after forty-eight minutes*}
<**player**>	{*Hamming, the forward Hamming*}
<**player_gen**>	{*his, Hamming's, the forward Hamming's*}
<**ordinal**>	{*second*}

Fig. 12. Possible slot fillings for Template Sent16.

the third option of using a proper name is always allowed (i.e. if there is a name available): disregarding stylistic considerations, a proper name can in principle be used in any context. The proper name *Hamming* therefore constitutes a possible slot filling. The fourth option, combining a definite description and a proper name to form an appositive, is also always available: the proper name distinguishes the described entity from other entities, and the definite description provides additional information about this entity. In our example, the appositive *the forward Hamming* is generated as a candidate description: since the property of being Dutch is not very interesting, being the default, only the position information is included in the description. In sum, the function ExpressObject returns two possible slot fillings for the <player> slot: the proper name *Hamming* and the appositive *the forward Hamming*.

We now turn to the third slot, <player_gen>. This slot must be filled with an expression for Hamming in the genitive case. Because there is an antecedent for this slot in the same sentence (i.e. the expression in the <player> slot), the MakeReferringExpression returns the genitive pronoun *his*. In addition, a proper name and an appositive are available, so for this slot ExpressObject returns a set of trees for *his*, *Hamming's* and *the forward Hamming's*.

Finally, the formation of the filler for the <ordinal> slot, expressing the number of goals the current player has scored so far, requires a bit of computation (not indicated in figure 7). This computation yields a positive integer (in the current example, 2), which must then be expressed by its corresponding ordinal (*second*).

Combining all different slot fillings (shown in figure 12), the function FillSlots returns a set of $2 \times 2 \times 3 \times 1 = 12$ trees that can be generated from Template Sent16 in the current context. For each tree in this set, it is checked whether it obeys the Binding Theory (Chomsky 1981). This test filters out the trees where either the proper name *Hamming's* or the appositive *the forward Hamming's* occupies the <player_gen> slot, because these expressions ('R-expressions' in the Binding Theory) are not free in this position, thus violating Principle C of the Binding Theory. The set of *allowed_trees* therefore contains trees for the following four sentences:

$$\left\{ \begin{array}{l} \textit{After forty-eight minutes Hamming had his second goal noted,} \\ \textit{After forty-eight minutes the forward Hamming had his second goal noted,} \\ \textit{In the forty-eighth minute Hamming had his second goal noted,} \\ \textit{In the forty-eighth minute the forward Hamming had his second goal noted} \end{array} \right\}$$

From these remaining trees, one is selected arbitrarily (in this case the second one), and sent to the Prosody module, where its prosodic properties are computed as described in the next section. The fringe of the resulting tree (i.e. the sentence

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

enriched with prosodic markers) is returned to the main algorithm, where the sentence is sent to the SGM to be pronounced, and the Knowledge State and the Context State (including the Discourse Model) are updated accordingly. Here we leave our illustration of the language generation algorithm, and continue by discussing prosody computation in D2S.

3.2 Prosody computation in D2S

In this section, we show how the Prosody module determines the location of accents and phrase boundaries in a generated sentence on the basis of both syntactic and semantic information (see section 2.3). We use our earlier example sentence *Na achtenveertig minuten liet de aanvaller Hamming zijn tweede doelpunt aantekenen* ('After forty-eight minutes the forward Hamming had his second goal noted') as an illustration. The prosodic rules described in this section are independent of domain and language, within the class of Germanic languages (e.g. English, Dutch, and German). The same set of rules has been used for prosody computation in both GoalGetter and the DYD-system, which differ with respect to language (Dutch versus English) and domain (football versus Mozart).

3.2.1 Overview

Since accentuation is relevant for the placement of phrase boundaries, but not vice versa, the Prosody module starts with computing the accentuation pattern of each sentence, using an algorithm that is based on a version of Focus-Accent Theory (Baart 1987) proposed by Dirksen (1992) and Dirksen and Quené (1993). In Focus-Accent Theory, binary branching metrical trees are used to represent the semantic and syntactic prominence of nodes with respect to pitch accent. In our implementation, the metrical tree of a sentence is based on the sentence's syntactic tree.[6] It is constructed by converting the syntactic tree to a tree that is at most binary-branching and marking its nodes with *focus* markers and *weak* or *strong* labels. The focus markers indicate information status. Words and phrases that express new or contrastive information are considered to be in focus, whereas words and phrases that express 'given' information are considered to be out of focus. The weak/strong (w/s) labels in the tree represent the structural prominence of the nodes with respect to accentuation. In other words, the focus marking indicates

[6] Note that having such a direct link between 'traditional syntactic structure' and intonational structure is somewhat controversial. Steedman (1990, 1996) and others have pointed out that prosodic structure does not always adhere to the traditional subject–predicate division of a sentence. An example from Steedman (1996) is the following:

(3) Q: Well, what about MARY? What does SHE admire?
A: MARY admires / MUSICALS

(The placement of accents and phrase boundaries in this example is Steedman's; the notation is our own.) The prosody module of the LGM would not generate a phrase boundary in the above example (but it would generate the same accentuation pattern).

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

goal_event	
team :	Go_Ahead_Eagles
player :	Schenning
minute :	18
goaltype :	normal

Een minuut later bracht Schenning van Go Ahead Eagles de teams op gelijke hoogte.
('One minute later, Schenning from Go Ahead Eagles equalised the score.')

goal_event	
team :	Fortuna_Sittard
player :	Hamming
minute :	48
goaltype :	normal

Na achtenveertig minuten liet de aanvaller Hamming zijn tweede doelpunt aantekenen.
('After forty-eight minutes, the forward Hamming had his second goal noted.')

Fig. 13. Data structures expressed by the example sentence and its predecessor.

which words and phrases should or should not receive an accent, while the w/s labelling determines where in each constituent an accent may land. The first is a matter of discourse semantics, while the second is a matter of syntax.

3.2.2 *Focus and information status*

The focus properties of the nodes in the metrical tree are determined as follows. First, the Prosody module adds an initial, preliminary focus marking to the tree. In this initial state, all major constituents (i.e. all maximal projections of the form XP) are, by default, assumed to be in focus and are marked [+F]. The other nodes are initially not specified for focus. After the initial, default assignment of focus markers has taken place, the system tries to determine the information status of the words or phrases in the tree.

First of all, it tries to determine which words and phrases are *contrastive*. The method used to determine contrast of information is described in Theune (1997a, 1997b, 2000). It is based on a comparison of the data structure expressed by the current sentence with the data structure expressed by its predecessor. It is checked if the two data structures are of the same type (e.g. two *goal_events*), and if so, which of their attributes have different values. The words and phrases of the current sentence that express those differing values are marked as being in focus due to contrast. An important advantage of using data structures as the basis for assigning contrastive accents within the LGM is that it allows for the detection of contrast in cases where there is no syntactic or semantic parallelism between sentences. We can illustrate this using our earlier example sentence, the sixth sentence from figure 3. Figure 13 shows the *goal_events* expressed by this example sentence and the preceding sentence. These are the second and third goals of the example match (see figure 5).

As can be seen in figure 13, except for the type attribute all attributes of the *goal_event* expressed by the example sentence have different values from that of the preceding sentence. This means that the phrases in the example sentence that express the values of those attributes should receive contrastive accent. These are the AP *achtenveertig* ('forty-eight', expressing the value of the minute attribute) and the NP *de aanvaller Hamming* ('the forward Hamming', expressing the value of the player

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

attribute). These phrases are marked [+C] to indicate that they are in focus due to contrast. If a constituent expresses contrastive information, its focus marking cannot be changed, even if it might be regarded as given. Examples like (1) in section 2.3 show that contrast overrides givenness. Even 'unaccentable' words like determiners may receive an accent if they are used contrastively.

Next, the system determines which words or phrases in the tree express given information, and which words are 'unaccentable' (e.g. certain function words). On the basis of this information, the initial placement of focus markers in the tree may be altered (except for the [+C] markers). The focus value of a node is changed to [-F] in three cases: (*i*) if the node directly dominates a word or phrase expressing given information, (*ii*) if it directly dominates an 'unaccentable' word and (*iii*) if all the nodes it dominates are marked [-F].

Information from the LGM's Context State is used to determine whether a word or phrase expresses given information. The rules for determining givenness are based on the theory proposed by van Deemter (1994), who, like Chafe (1976), distinguishes *object-givenness* and *concept-givenness*. A word or phrase is object-given if it refers to a discourse entity (e.g. a player) that has already been mentioned, and it is concept-given if it expresses a concept (e.g. 'scoring a goal') which has already been evoked earlier in the discourse. In D2S, the Discourse Model can be used to check object-givenness, because it records which entities have so far been referred to by the system. In our example sentence, the NP's *de aanvaller Hamming* ('the forward') and *zijn* ('his') are object-given, because their referent, the player Hamming, was referred to two sentences earlier (see figure 3). The focus marking of the second NP (*zijn*) is therefore changed to [-F], but the marking of the first NP does not change because it is marked as contrastive.

Concept-givenness is determined by checking if words or phrases are synonymous or identical to words or phrases that were used earlier in the text, or if the concept they express *subsumes* another concept that was expressed earlier. Currently, this is checked using an application-specific list of synonymous and subsuming words and phrases. An example of concept-givenness through subsumption can be found in the seventh sentence of the example text in figure 3: the concept 'player', which is referred to in the appositive NP *de Go Ahead Eagles speler Decheiver* ('the Go Ahead Eagles player Decheiver'), subsumes the concept 'forward' mentioned in the preceding sentence, and is therefore regarded as given. (Since a forward is a player, we assume that mentioning the concept 'forward' automatically activates the subsuming concept 'player'.) In our system, subsumption is currently determined using a small hand-crafted subsumption hierarchy, which is application-specific. In addition, application-specific lists of synonyms are used, recording for instance that the words *treffer* and *doelpunt* are synonyms for *goal*. Following Hirschberg (1992), we relate givenness to topic structure, assuming that items only remain given within one topic, which corresponds to one paragraph in the output of D2S.

The example sentence contains two cases of concept-givenness. First, the word *minuten* ('minutes') expresses the same concept as *minuut* ('minute'), occurring in the previous sentence, and is therefore defocused. Second, the concept expressed by the collocation *een doelpunt laten aantekenen* ('having a goal noted') subsumes the

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

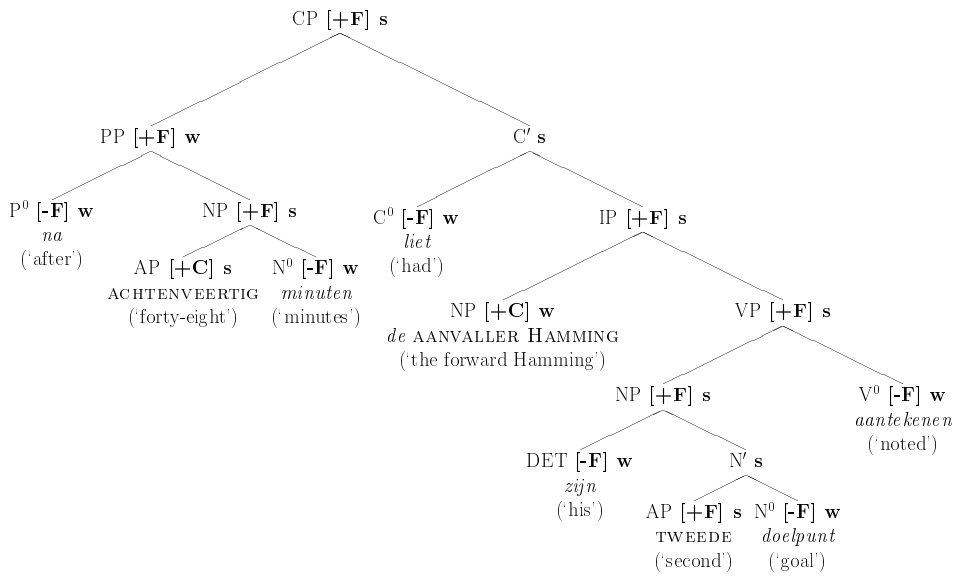


Fig. 14. Final metrical tree of the sixth sentence of figure 3.

concept expressed by the collocation *op gelijke hoogte brengen* ('equalise'), which occurred in the previous sentence. The words *doelpunt*, *liet* and *aantekenen* are therefore regarded as expressing given information and marked as [-F].

3.2.3 Weak and strong nodes

The weak/strong labelling of the metrical tree nodes, which ultimately determines on which words an accent will land, depends both upon the structure of the tree and on the focus properties of its nodes. In Dutch, like in English, normally the left node of two sisters is weak and the right node is strong. If the structurally strong node is marked [-F] while the structurally weak node is not, the weak/strong labelling is switched. In figure 14, showing the complete metrical tree of the example sentence, this has occurred in three cases: (*i*) for the AP *achtenveertig* and the defocused N^0 *minuten*, (*ii*) for the AP *tweede* and the defocused N^0 *doelpunt*, and (*iii*) for the NP *zijn tweede doelpunt* and the defocused V^0 *aantekenen*.

When the metrical tree is complete, the focus markers indicate which constituents should be accented, and the weak/strong labelling indicates on which words the accent may land. The actual accentuation algorithm can therefore be very simple: each node that is marked [+F] launches an accent, which trickles down the tree along a path of strong nodes until it lands on a terminal node, dominating a word. In the example, the accents launched by CP, IP and VP all coincide with the accent launched by the NP node of *zijn tweede doelpunt*, finally landing on the word *tweede*. Since the nodes dominating *liet* and *aantekenen* are weak, no accent trickles down to them, and because they are marked [-F] they do not launch an accent themselves. The PP node dominating the phrase *na achtenveertig minuten* does launch an accent,

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

which trickles down to the NP *achtenveertig minuten*, where it coincides with the accent launched by the NP itself. Within the NP, the accent goes to the left because the right node dominating *minuten* has been defocused, so it ends up on the word *achtenveertig*. Finally, the appositive, contrastive NP *de aanvaller Hamming* consists of two NP's (not shown in the tree due to space restrictions), both of which launch an accent that trickles down to their head nouns.

3.2.4 Phrase boundaries

After accentuation, phrase boundaries are assigned. Currently, three phrase boundary strengths are distinguished.[7] The strongest of the three is the *sentence-final* boundary (///). Next comes the *major* boundary (//), which follows words preceding a punctuation symbol other than a comma (e.g. ';') and sentence-internal clauses (i.e. a CP or IP within a sentence). Finally, a *minor* boundary (/) follows words preceding a comma and constituents meeting the following conditions: (i) the constituent has sufficient length (more than four syllables), (ii) the constituent on its right is an I', a C' or a maximal projection, and (iii) both constituents contain at least one accented word. This is a slightly modified version of a structural rule proposed by Dirksen and Quené (1993). In our present example only the PP *Na achtenveertig minuten* and the NP *de aanvaller Hamming* meet this condition and are therefore followed by a minor phrase boundary. Since the example sentence contains no punctuation and consists of just one clause, the only other phrase boundary is the sentence-final one.

3.2.5 Evaluation

The accentuation algorithm of D2S was formally evaluated for Dutch in a small-scale experiment (Nachtegaal 1997). In the experiment, recordings were made of non-professional speakers of Dutch who read aloud the plain text versions of texts generated by the LGM of GoalGetter. The texts contained sentences which were structurally similar to those of the example text given in Figure 3. 'Expert listeners' were presented with the recordings and indicated on which words they heard an accent. The accentuation patterns produced by the speakers were then compared to those generated by the system. The results of this comparison showed that the number of words on which the accentuation by GoalGetter deviated from the accentuation by the speakers was very small. For details, see Nachtegaal (1997).

For the algorithm determining the placement of phrase boundaries no formal evaluation has taken place yet. However, we have informally compared the general prosodic quality of the output of D2S with the output of the two best text-to-speech systems currently available for Dutch (according to the anonymous evaluation in Sluyter, Bosgoed, Kerkhoff, Meier, Rietveld, Sanderman, Swerts and Terken (1998)).

[7] In longer texts, containing more complicated constructions, it might be desirable to distinguish more levels. Sanderman (1996) proposes a boundary depth of five to achieve more natural phrasing.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

One of the two systems employs the same speech synthesis as used in GoalGetter (see section 3.3.1); the other system employs a different kind of speech synthesis. We used the two text-to-speech systems to pronounce some texts generated by the GoalGetter system (plain text version). We then compared the prosodic quality of the speech output to that produced by GoalGetter. We observed two main flaws, displayed by both text-to-speech systems, which made their output sound somewhat less natural than that of GoalGetter. First, the placement of phrase boundaries by the two text-to-speech systems was less adequate than in D2S: several obvious phrase boundaries were missing (e.g. between conjugated clauses), or misplaced (e.g. between an adjective and the NP it modified). Secondly, both systems failed to perform deaccentuation even in the simplest cases, like the second occurrence of the word *kaart* ('card') in the following example:

(4) BLOM gaf COCU een GELE KAART.
VOS kreeg een RODE kaart / ?? VOS kreeg een RODE KAART
Translation:
BLOM handed COCU a YELLOW CARD.
VOS received a RED card. / ?? VOS received a RED CARD.

3.3 Speech generation in D2S

The D2S system currently has two different output modes available in the SGM. One is phonetics-to-speech synthesis and the other is phrase concatenation. These modes are discussed in more detail below.

3.3.1 Phonetics-to-speech

Phonetics-to-speech generates speech not from unrestricted text, as in text-to-speech, but from a phonetic transcription with prosodic annotations. This means that prosody computation and grapheme-to-phoneme conversion must be done prior to speech generation. We have already seen how the Prosody module of the LGM generates prosodic markers, thus producing 'enriched text'. For the benefit of phonetics-to-speech, this enriched text must be converted into a phonetic transcription. Because the LGM generates an orthographic representation with a unique phonetic representation, it is possible to do errorless grapheme-to-phoneme conversion by lexical lookup instead of rules. The speech output is generated by concatenating diphones, small speech segments consisting of the transition between two adjacent phonemes. A complete diphone inventory for a language covers all possible transitions between any two sounds of that language. The phonetics-to-speech system CALIPSO, developed at IPO, uses a method called *phase synthesis*, which combines the advantages of PSOLA and mixed-excitation LPC to achieve an output quality that is quite high (Gigi and Vogten 1997). In two anonymous tests concerning subjective evaluation under telephone conditions, CALIPSO was judged favourably on several aspects, including general quality, intelligibility and voice pleasantness (Rietveld, Kerkhoff, Emons, Meijer, Sanderman and Sluijter 1997; Sluijter *et al.* 1998).

Current diphone synthesis systems reach a high degree of intelligibility. However,

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

recent evaluations show that when synthetic speech is sent through a telephone channel, intelligibility decreases significantly. In GSM (mobile phone) conditions, intelligibility drops even further (Rietveld *et al.* 1997). Furthermore, naturalness still leaves a great deal to be desired. Still, it was implemented in the D2S system because it offers unlimited flexibility. In addition, it allows for the testing of the prosody assignment algorithm used in the system, because the prosodic realisation of synthesised speech can be controlled to a large extent.

To achieve more natural sounding speech output, we are currently concentrating on the improvement of a few specific aspects of the diphone synthesis system, such as the occurrence of audible discontinuities at diphone boundaries (Klabbers 1997) and duration control (Klabbers 2000).

3.3.2 Phrase concatenation using prosodic variants

Unlike speech synthesis, phrase concatenation offers a speech quality that is close to that of natural speech. Therefore, we chose this technique as the primary technique for speech generation in D2S.

Our advanced phrase concatenation technique (Klabbers 2000) can be seen as an extension to the simple concatenation technique. It resembles the technique that Waterworth used in the telephone announcement system (Waterworth 1983), in that several prosodic variants of otherwise identical words and phrases are used. An important difference is that Waterworth's approach is specifically aimed at the pronunciation of telephone numbers, whereas our approach is far more general and completely domain-independent. Our phrase concatenation technique is similar to the one used in the Appeal system (de Pijper 1997) in that the phrases are embedded in (dummy) carrier phrases during recording. An important difference is that the phrases are recorded in different prosodic versions. As a consequence, our method requires no additional manipulation or coding of the recordings. This results in a speech quality that approaches that of natural speech.

Our use of several prosodic variants relates especially to the slots in the templates. The fixed parts of the syntactic templates, corresponding to the carrier sentences can usually be recorded as a whole, and in only one version. Sometimes, it is more convenient to split the carrier into two or more phrases, if parts of the carrier occur in several other carrier sentences as well. The prosody of the slots in the templates however, is most crucial, because there the variable (and usually most important) information is inserted. To find out which prosodically distinct versions we need for these slot fillers we analysed texts generated by the LGM which were made audible through our phonetics-to-speech system (CALIPSO). The intonation rules in this system are based on the IPO Grammar of Intonation (Collier and 't Hart 1981; 't Hart, Collier and Cohen 1990), which describes the intonation of a sentence in terms of pitch movements. It assigns (combinations of) pitch movements on the basis of combinations of accents and boundaries. After analysis we came up with six different prosodic realisations, one for each context described in terms of prosodic markers. Stylisations of these prosodic realisations are depicted in figure 15 and are explained below.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

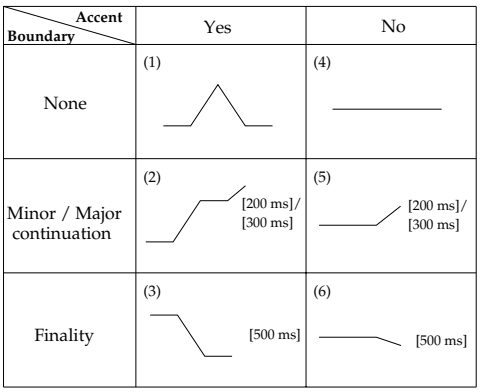


Fig. 15. Stylised examples of the different prosodic versions that are needed. Two factors determine their pitch and pausing: the accentuation and the position relative to a minor/major/final phrase boundary. The pauses are indicated between brackets.

1. An accented slot filler which does not occur before a phrase boundary is produced with the most frequently used pitch configuration, the so-called (pointed) *hat pattern*, which consists of a rise and fall on the same syllable. This contour corresponds to the prosodically neutral version used in many other phrase concatenation techniques.
2. An accented slot filler which occurs before a minor or a major phrase boundary is most often produced with a rise to mark the accent and an additional continuation rise to signal that there is a non-final boundary. A short pause follows the constituent, which is 200 ms in length in case of a minor boundary (/) and 300 ms in case of a major boundary (//).
3. An accented slot filler which occurs in final position receives a final fall. It is followed by a longer pause of 500 ms.
4. Unaccented slot fillers are pronounced on the declination line without any pitch movement associated with them.
5. Unaccented slot fillers occurring before a minor or a major phrase boundary only receive a small continuation rise. This prosodic situation does not occur very often. The LGM usually puts a minor or major phrase boundary immediately after an accented word. Again, a 200 ms or 300 ms pause is inserted.
6. Unaccented slot fillers in a final position are produced with final lowering, i.e. a declination slope that is steeper than in other parts of the utterance. They are followed by a 500 ms pause.

When recording the material for the phrase database, the slots in the carrier sentences were filled with dummy words so that the fixed phrases to be stored in the database could be excised easily. In this way, the effect of co-articulation at the word boundaries was minimised. Fade-in and fade-out was applied to all material in the phrase database to avoid clicks in concatenation. The slot fillers, such as player names and time expressions, were embedded in dummy sentences that provide the right prosodic context. The sentences were constructed in such a way

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

Table 1. *Composition of the GoalGetter phrase database. For each concept it is indicated how many types there are in the database, how many prosodic variants are recorded of each concept, and the resulting number of tokens*

Concept	Number of types	Number of prosodic variants	Number of tokens
Carrier phrase	378	1	378
Player name	407	6	2442
Team name	18	6	108
Trainer name	18	6	108
Place name	18	6	108
Stadium name	18	6	108
Number in score[8]	20	2	40
Number in time expression	200	2	400
Total:	1022		3692

as to make the speaker produce the right prosodic realisation naturally. We used a female semi-professional speaker. She received no specific instructions about how to produce the sentences. The recordings were made in a sound-treated room using two high-quality microphones which were positioned on either side of the speaker, a fixed distance away from the mouth. The speech was recorded on a DAT-tape using a 48 kHz sampling frequency. The speech signal was stored on an SGI workstation in mono with sampling frequency of 16 kHz. The concatenative units were excised manually and sentences were generated to check for large differences in loudness to be corrected. Only one or two recording sessions were required. The manual excision of all the concatenative units was the most time-consuming task.

The GoalGetter phrase database consists of 3692 concatenative units that can be divided into different categories as listed in Table 1. As can be seen, the player names (407 different names) form the bulk of the data, especially since they have been recorded in six prosodic versions. In conventional phrase concatenation, the size of the database would be 1022 units, the total number of types as indicated in Table 1. Recording additional prosodic variants increases the size of the database with a factor 3.6 to 3692, the total number of tokens in the table. IPO's phrase concatenation method has also been employed in OVIS, a spoken dialogue system that provides train travel information. In this system, D2S is used for output

[8] 'Number in score' and 'number in time expression' are listed as different concepts, since the numbers are pronounced differently. We assume (taking a safe margin) that no numbers higher than twenty occur in the score. In GoalGetter the score is always expressed as *number* - *number*, where both numbers are accented and are separated by a major phrase boundary. Since the score is always mentioned at the end of a sentence (i.e. before a final boundary), this means that only two prosodic versions need to be recorded: variant two for the first number and variant three for the second number (see Table 15). In contrast, the numbers in a time expression are never followed by a phrase boundary, so for these numbers only the prosodic versions one and four from Table 15 are relevant.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

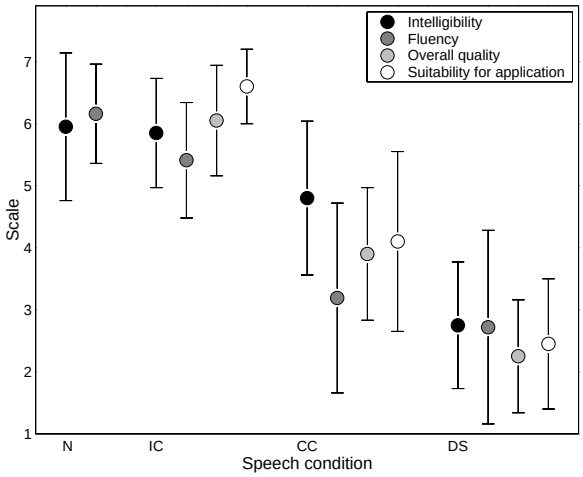


Fig. 16. Average quality ratings for intelligibility, fluency, overall quality and suitability for the application; N = natural speech, IC = IPO's phrase concatenation, CC = commercial phrase concatenation, DS = IPO's diphone synthesis.

generation. The phrase database of OVIS, which includes the names of all 382 Dutch train stations, contains less than 3000 tokens.

To concatenate the proper words and phrases, an algorithm has been designed that performs a mapping between the enriched text, i.e. text with accentuation and phrasing markers, as provided by the LGM, and the pre-recorded phrases that have to be selected. The different prosodic variants are chosen on the basis of the prosodic markers. The algorithm recursively looks for the largest phrases to concatenate into sentences. It works from left to right. First, it tries to find the string of N words that contains the entire sentence. If it is present, it is retrieved and can be played. If not, the string comprising the first $N-1$ words is looked up. This process continues until a matching phrase is found. Then the remaining part of the sentence undergoes the same procedure, until the entire sentence can be played.

3.3.3 Evaluation

The IPO phrase concatenation method has been evaluated in a formal listening experiment (Klabbers 2000), in which it was compared to (i) natural speech output, (ii) a conventional concatenation approach (as often used in commercial applications), and (iii) diphone synthesis. Twenty naive subjects rated twenty different messages on intelligibility, fluency, overall quality and suitability for the application on a 7-point scale. The results are summarised in figure 16. The results show that the IPO phrase concatenation compares well to natural speech on both intelligibility and fluency, and scores very well on overall quality and suitability. Overall quality and suitability for application were not tested for natural speech, but we may safely assume that this form of speech output would receive near maximal scores on these dimensions.

The conventional concatenation approach scores significantly less on all dimensions than IPO's phrase concatenation, indicating that it sounds less natural than

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

is sometimes assumed (Sluyter *et al.* 1998). The evaluation results indicate that it is worth the extra effort to take a prosodically sophisticated approach to phrase concatenation. Diphone synthesis scores worst on all dimensions. However, in applications where the vocabulary is large, or changes frequently, phrase concatenation will be infeasible and speech synthesis will be the only option available.

4 Discussion

We have presented a generic data-to-speech system, D2S, in which there is a tight coupling between the language generation and the speech generation modules. Language and speech generation in D2S is done by means of techniques which incorporate linguistic insights while achieving practical usability. For language generation, we use a hybrid technique where the use of syntactically enriched templates is constrained by local conditions on the discourse context, while for speech generation we combine pre-recorded phrases in a sophisticated manner, taking prosodic variations into account. Speech generation can also be achieved by phonetics-to-speech synthesis, which offers greater flexibility but a less natural speech quality. The coupling between the language and speech generation modules is brought about by the computation of prosody by the LGM. This ensures that the syntactic, semantic and discourse knowledge captured in the LGM can be used in speech generation, without having the SGM compute the required information anew. Since linguistic analysis of the generated texts would provide less reliable information than is available from the LGM, our approach also allows us to achieve a better prosodic quality of the system's output than could be obtained by simply feeding the outcome of language generation into a text-to-speech system.

D2S is a practically useful system which can serve as a basis for a wide range of applications. Systems developed on the basis of D2S can be run efficiently on PC/Windows and on Unix platforms. With respect to language generation, porting D2S to a new application mainly involves constructing a set of syntactic templates, designing a structure representing the input data and, optionally, adding a domain database; all other parts of the LGM are application independent. The work to be done on speech generation depends on the chosen output mode. If phonetics-to-speech is chosen, an off-the-shelf speech synthesis program may be used and only an application-specific lexicon for grapheme-to-phoneme conversion has to be made. Alternatively, an automatic grapheme-to-phoneme converter can be used. Currently, such systems achieve an error-rate of approximately 2% (van den Bosch 1997); for names, however, hand-made lexicon entries are still required. The use of phrase concatenation gives rise to more work, because it involves the making of recordings, and the excision of the required phrases. This higher work-load is compensated by a more natural sounding speech output (Klabbers 2000).

A major advantage of the use of syntactic templates in the LGM of D2S is that there are no restrictions on the complexity of the sentences that can be generated, nor on the type of information that can be expressed. The variation in the generated texts, achieved by the use of local conditions on the templates, is an important feature of the system, especially in the light of applications like DYD and GoalGetter, where

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

the user is likely to hear a number of generated texts in succession. In addition, the 'local condition' approach seems to be quite suitable for language generation in a dialogue situation, as it may be seen as a form of reactive planning. A version of D2S which can be used in dialogue systems has recently been developed. It is used for the generation of system output in the OVIS system, a spoken dialogue system that gives information about public transport in the Netherlands. In OVIS, planning is performed by the Dialogue Management module, described in Veldhuijzen van Zanten (1998). This module provides the LGM with conceptual representations of the messages to be generated. The use of syntactic templates by the LGM fits in well with such an architecture.

Although D2S is presented in this paper as one integrated system, the techniques described here can also be used independently. A variant of the phrase concatenation method used in D2S has been employed in a new version of the German train information system described in Aust *et al.* (1995), while the LGM of D2S has been used for the generation of English and German route descriptions in the VODIS project, which is a European project aimed at the development of a speech interface for a car navigation system (Pouteau and Arévalo 1998).

Finally, we would like to conduct a formal evaluation of the general prosodic quality of the output of D2S as well as the quality of the texts generated by the LGM. Evaluation of the prosodic quality may be done by formally comparing the output of D2S with that of the best text-to-speech systems available for Dutch. Informal comparison has so far given encouraging results. Evaluation of the LGM is a more complicated matter. As Dale and Mellish (1998) have pointed out, evaluation of natural language generation systems is still in its infancy, and there are no well-established evaluation methods in this area. An evaluation method which seems promising is the one adopted by Coch (1996) and Lester and Porter (1997). They compared computer-generated texts to texts from human authors by having a panel of judges, who did not know the source of the texts, rate their quality on several dimensions. However, see Dale and Mellish (1998) for a discussion of some problems related to such a 'black box' evaluation.

In addition to having separate evaluations of the LGM, the prosody module and the SGM, it would also be interesting to see an evaluation of D2S as a whole. However, as data-to-speech systems are obviously even more difficult to evaluate than systems generating only written output, at present we may have to stick to a 'glass box' evaluation of the system.

Acknowledgements

The authors wish to thank Jan Landsbergen and two anonymous reviewers for their useful comments on an earlier version of this paper. Authors Theune and Klabbers carried out their research within the framework of the Priority Programme Language and Speech Technology (TST), which is sponsored by NWO (Netherlands Organisation for Scientific Research). Jan Odijk currently works at Lernout & Hauspie Speech Products, but the work for this publication was carried out when he was employed at Philips Research Laboratories, Eindhoven, The Netherlands.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

References

André, E., Herzog, G. and Rist, T. (1988) On the simultaneous interpretation of real world image sequences and their natural language description: The system SOCCER. In: Y. Kodratoff (ed.), *Proceedings 8th European Conference on Artificial Intelligence (ECAI'88)*, pp. 449–454. London: Pitman.

Aust, H., Oerder, M., Seide, F. and Steinbiss, V. (1995) The Philips automatic train timetable information system. *Speech Communication*, **17**: 249–262.

Baart, J. L. G. (1987) *Focus, Syntax and Accent Placement*. PhD thesis, University of Leiden.

Bateman, J. and Henschel, R. (1999) From full generation to 'near-templates' without losing generality. In: T. Becker and S. Busemann (eds.), *Proceedings KI'99 Workshop 'May I Speak Freely?'*, pp. 13–18. DFKI Saarbrücken.

Bock, J. and Mazzella, J. (1983) Intonational marking of given and new information. *Memory and Cognition*, **11**(1): 64–76.

Bosch, A. van den (1997) *Learning to Pronounce Written Words. A Study in Inductive Language Learning*. PhD thesis, Maastricht University.

Brown, G. (1983) Prosodic structure and the given/new distinction. In: D. R. Ladd and A. Cutler (eds.), *Prosody: Models and Measurements*, pp. 67–77. Berlin: Springer-Verlag.

Busemann, S. and Horacek, H. (1998) A flexible shallow approach to text generation. *Proceedings 9th International Workshop on Natural Language Generation (IWNLG'98)*, pp. 238–247. Niagara-on-the-Lake, Ontario.

Cahill, L., Doran, C., Evans, R., Mellish, C., Paiva, D., Reape, M., Scott, D. and Tipper, N. (1999) In search of a reference architecture for NLG systems. *Proceedings 7th European Workshop on Natural Language Generation (EWNLG'99)*, pp. 77–85. (KIT Report 97, Toulouse, France.)

Carenini, G., Mittal, V. O. and Moore, J. D. (1994) Generating patient-specific interactive natural language explanations. *Proceedings 18th Annual Symposium on Computer Applications in Medical Care (SCAMC'94)*. Washington, DC.

Chafe, W. L. (1974) Language and consciousness. *Language*, **50**: 111–133.

Chafe, W. L. (1976) Givenness, contrastiveness, definiteness, subjects, topics and points of view. In: C. N. Li (ed.), *Subject and Topic*, pp. 25–55. New York: Academic Press.

Chomsky, N. (1981) *Lectures on Government and Binding*. Dordrecht: Foris.

Coch, J. (1996) Evaluating and comparing three text-production techniques. *Proceedings 16th International Conference on Computational Linguistics (COLING-96)*, pp. 249–254. Copenhagen, Denmark.

Collier, R. and 't Hart, J. (1981) *Cursus Nederlandse Intonatie*. Leuven: Acco.

Dale, R. and Mellish, C. (1998) Towards evaluation in natural language generation. In: A. Rubio, N. Gallardo, R. Castro and A. Tejada (eds.), *Proceedings 1st International Conference on Language Resources and Evaluation*, pp. 555–562. Paris: European Language Resources Association.

Dale, R. and Reiter, E. (1995) Computational interpretations of the Gricean maxims in the generation of referring expressions. *Cognitive Science*, **18**: 233–263.

Deemter, K. van. (1994) What's new? A semantic perspective on sentence accent. *J. Semantics*, **11**: 1–31.

Deemter, K. van, Krahmer E. and Theune, M. (1999) Plan-based vs. template-based NLG: A false opposition? In: T. Becker and S. Busemann (eds.), *Proceedings KI'99 Workshop 'May I Speak Freely?'*, pp. 1–5. DFKI Saarbrücken.

Deemter, K. van, Landsbergen, J., Leermakers, R. and Odijk, J. (1994) Generation of spoken monologues by means of templates. In: L. Boves and A. Nijholt (eds.), *Proceedings 8th Twente Workshop on Language Technology (TWLT 8): Speech and Language Engineering*, pp. 87–96. Enschede, The Netherlands.

Deemter, K. van and Odijk, J. (1997) Context modeling and the generation of spoken discourse. *Speech Communication*, **21**(1/2): 101–121.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

Dirksen, A. (1992) Accenting and deaccenting: A declarative approach. *Proceedings 14th International Conference on Computational Linguistics (COLING-92)*, pp. 865–869. Nantes, France.

Dirksen, A. and Quené, H. (1993) Prosodic analysis: The next generation. In: V. van Heuven and L. Pols (eds.), *Analysis and Synthesis of Speech: Strategic Research Towards High-Quality Text-to-Speech Generation*, pp. 131–144. Berlin: Mouton de Gruyter.

Geldof, S. and van de Velde, W. (1997) An architecture for template based (hyper)text generation. *Proceedings 6th European Workshop on Natural Language Generation (EWNLG'97)*, pp. 28–37. Leiden, The Netherlands.

Gigi, E. and Vogten, L. (1997) A mixed-excitation vocoder based on exact analysis of harmonic components. *IPO Annual Progress Report*, **32**: 105–110. Eindhoven: IPO.

Grice, H. P. (1975) Logic and conversation. In: P. Cole and J. Morgan (eds.), *Syntax and Semantics: Vol. 3, Speech Acts*, pp. 43–58. New York: Academic Press.

Halliday, M. A. K. (1967) Notes on transitivity and theme in English. *J. Linguistics*, **3**: 199–244.

Hart, J. 't, Collier, R. and Cohen, A. (1990) *A Perceptual Study of Intonation: An Experimental Phonetic Approach to Speech Melody*. Cambridge: Cambridge University Press.

Hirschberg, J. (1992) Using discourse context to guide pitch accent decisions in synthetic speech. In: G. Bailly, C. Benoît and T. R. Sawallis (eds.), *Talking Machines: Theories, Models and Designs*, pp. 367–376. Amsterdam: Elsevier.

Joshi, A. (1987) An introduction to Tree Adjoining Grammars. In: A. Manaster-Ramer (ed.), *Mathematics of Language*, pp. 87–114. Amsterdam: John Benjamins.

Klabbers, E. (1997) High-quality speech output generation through advanced phrase concatenation. *Proceedings COST Workshop on Speech Technology in the Public Telephone Network: Where are We Today?*, pp. 85–88. Rhodes, Greece.

Klabbers, E. (2000) *Segmental and Prosodic Improvements to Speech Generation*. PhD thesis, Eindhoven University of Technology.

Klabbers, E., Krahmer, E. and Theune, M. (1998) A generic algorithm for generating spoken monologues. *Proceedings 5th International Conference on Spoken Language Processing (ICSLP'98)*, pp. 2759–2762. Sydney, Australia.

Krahmer, E. and Theune, M. (1999) Efficient generation of descriptions in context. In: R. Kibble and K. van Deemter (eds.), *Proceedings of the Workshop on the Generation of Nominals, ESSLLI'99*. Utrecht, The Netherlands.

Lester, J. and Porter, B. (1997) Developing and empirically evaluating robust explanation generators: The KNIGHT experiments. *Computational Linguistics*, **23**(1): 65–102.

McKeown, K., Robin, J. and Kukich, K. (1995) Generating concise natural language summaries. *Infor. Process. and Manage.*, **31**(5): 703–733.

Moulines, E. and Charpentier, F. (1990) Pitch synchronous waveform processing techniques for text-to-speech synthesis using diphones. *Speech Comm.*, **9**(5/6): 453–467.

Mykowiecka, A. (1991) Natural-language generation – an overview. *Int. J. Man-Machine Stud.*, **34**: 497–511.

Nachtegaal, D. (1997) *An Evaluation of GoalGetter's Accentuation*. Report 1142, IPO, Eindhoven.

Noord, G. van, Bouma, G., Koeling, R. and Nederhof, M. (1999) Robust grammatical analysis for spoken dialogue systems. *Natural Lang. Eng.*, **5**(1): 45–93.

Odijk, J. (1995) Generation of coherent monologues. In: T. Andernach, M. Moll and A. Nijholt (eds.), *CLIN V: Proceedings of the 5th CLIN Meeting*, pp. 123–131. Enschede, The Netherlands.

Pan, S. and McKeown, K. R. (1997) Integrating language generation with speech synthesis in a concept to speech system. In: K. Alter, H. Pirker and W. Finkler (eds.) *Proceedings Workshop on Concept-to-Speech Generation Systems, ACL/EACL'97*, pp. 23–28. Madrid, Spain.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

Pierrehumbert, J. and Hirschberg, J. (1990) The meaning of intonational contours in the interpretation of discourse. In: P. R. Cohen, J. Morgan and M. E. Pollack (eds.), *Intentions in Communication*, pp. 271–311. Cambridge: MIT Press.

Pouteau, X. and Arévalo, L. (1998) Robust spoken dialogue systems for consumer products: A concrete application. *Proceedings 5th International Conference on Spoken Language Processing (ICSLP'98)*, pp. 1231–1234. Sydney, Australia.

Prevost, S. (1995) *A Semantics of Contrast and Information Structure for Specifying Intonation in Spoken Language Generation*. PhD thesis, University of Pennsylvania.

Pijper, J. R. de. (1997) High quality message-to-speech generation in a practical application. In: J. van Santen, R. Sproat, J. Olive and J. Hirschberg (eds.), *Progress in Speech Synthesis*, pp. 575–589. New York: Springer-Verlag.

Reiter, E. (1994) Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? *Proceedings 7th International Workshop on Natural Language Generation (IWNLG'94)*, pp. 163–170. Kennebunkport, USA.

Reiter, E. (1995) NLG vs. templates. *Proceedings 5th European Workshop on Natural Language Generation (EWNGL'95)*, pp. 95–106. Leiden, The Netherlands.

Reiter, E. (1999) Shallow vs. deep techniques for handling linguistic constraints and optimisations. *Proceedings KI'99 Workshop 'May I Speak Freely?'*. DFKI Saarbrücken.

Reiter, E. and Dale, R. (1997) Building applied natural language generation systems. *Natural Lang. Eng.*, **3**(1): 57–87.

Reiter, E. and Mellish, C. (1993) Optimizing the costs and benefits of natural language generation. *Proceedings 13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, pp. 1164–1169. Chambery, France.

Rietveld, T., Kerkhoff, J., Emons, M. J. W. M., Meijer, E. J., Sanderman, A. A. and Sluijter, A. M. C. (1997) Evaluation of speech synthesis systems for Dutch in telecommunication applications in GSM and PSTN networks. In: G. Kokkinakis, N. Fakotakis and E. Dermatas (eds.), *Proceedings ESCA 5th European Conference on Speech Communication and Technology (Eurospeech'97)*, pp. 577–580. Rhodes, Greece.

Robin, J. (1994) Automatic generation and revision of natural language report summaries providing historical background. *Proceedings 11th Brazilian Symposium on Artificial Intelligence*. Fortaleza, Brazil.

Sanderman, A. (1996) *Prosodic Phrasing: Production, Perception, Acceptability and Comprehension*. PhD thesis, Eindhoven University of Technology.

Sluijter, A., Bosgoed, E., Kerkhoff, J., Meier, E., Rietveld, T., Sanderman, A., Swerts, M. and Terken, J. (1998) Evaluation of speech synthesis systems for Dutch in telecommunication applications. In *Proceedings 3rd ESCA/COCOSDA International Workshop on Speech Synthesis*, pp. 213–218. Jenolan Caves, Australia.

Steedman, M. (1990) Structure and intonation in spoken language understanding. *Proceedings 28th Annual Meeting of the Association for Computational Linguistics (ACL'90)*, pp. 9–17. Pittsburgh, USA

Steedman, M. (1996) Representing discourse information for spoken dialogue generation. *Proceedings International Symposium on Spoken Dialogue, ICSLP'96*. Philadelphia, USA.

Tanaka, K., Hasida, K. and Noda, I. (1998) Reactive content selection in the generation of real-time soccer commentary. *Proceedings 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics (COLING-ACL'98)*, pp. 1282–1288. Montreal, Canada.

Terken, J. and Nooteboom, S. (1987) Opposite effects of accentuation and deaccentuation on verification latencies for given and new information. *Lang. & Cognitive Process.*, **2**: 145–163.

Theune, M. (1997*a*) Contrastive accent in a data-to-speech system. *Proceedings 35th Annual Meeting of the Association for Computational Linguistics and the 8th Conference of the European Chapter of the Association for Computational Linguistics (ACL/EACL'97)*, pp. 519–521. Madrid, Spain.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press

Theune, M. (1997*b*) GoalGetter: Predicting contrastive accent in data-to-speech generation. In: K. van Deemter, J. Landsbergen, J. Odijk and G. Veldhuijzen van Zanten (eds.), *CLIN VII: Proceedings 7th CLIN Meeting*, pp. 177–190. Eindhoven, The Netherlands.

Theune, M. (1999) Parallelism, coherence, and contrastive accent. *Proceedings 6th European Conference on Speech Communication and Technology (Eurospeech'99)*, pp. 555–558. Budapest, Hungary.

Theune, M. (2000) *From Data to Speech: Language Generation in Context*. PhD thesis, Eindhoven University of Technology.

Theune, M., Klabbers, E., Odijk, J. and de Pijper, J. R. (1997) Computing prosodic properties in a data-to-speech system. In: K. Alter, H. Pirker and W. Finkler (eds.) *Proceedings Workshop on Concept-to-Speech Generation Systems, ACL/EACL'97*, pp. 39–45. Madrid, Spain.

Veldhuijzen van Zanten, G. (1998) Adaptive mixed-initiative dialogue management. *Proceedings IEEE 4th Workshop on Interactive Voice Technology for Telecommunications Applications (IVTTA'98)*, pp. 65–70. Turin, Italy.

Waterworth, J. A. (1983) Effect of intonation form and pause durations of automatic telephone number announcements on subjective preference and memory performance. *Appl. Ergonomics*, **14**(1): 39–42.

White, M. and Caldwell, T. (1998) EXEMPLARS: A practical, extensible framework for dynamic text generation. *Proceedings 9th International Workshop on Natural Language Generation (IWNLG'98)*, pp. 266–275. Niagara-on-the-Lake, Ontario.

Young, S. and Fallside, F. (1979) Speech synthesis from concept: A method for speech output from information systems. *J. Acoustical Society of America*, **66**: 685–695.

Zue, V. (1997) Conversational interfaces: Advances and challenges. *Proceedings 5th European Conference on Speech Communication and Technology (Eurospeech'97)*, pp. 9–18. Rhodes, Greece.

https://doi.org/10.1017/S1351324901002625 Published online by Cambridge University Press