A GENERIC ALGORITHM FOR GENERATING SPOKEN MONOLOGUES

Esther Klabbers, Emiel Krahmer and Mariët Theune

IPO, Center for Research on User-System Interaction, Eindhoven, The Netherlands
{klabbers/krahmer/theune}@ipo.tue.nl

ABSTRACT

The defining property of a Concept-to-Speech system is that it combines language and speech generation. Language generation converts the input concepts into natural language, which speech generation subsequently transforms into speech. Potentially, this leads to a more 'natural sounding' output than can be achieved in a plain Text-to-Speech system, since the correct placement of pitch accents and intonational boundaries —an important factor contributing to the 'naturalness' of the generated speech— is co-determined by syntactic and discourse information, which is typically available in the language generation module. In this paper, a generic algorithm for the generation of coherent spoken monologues is discussed, called *D2S*. Language generation is done by a module called *LGM* which is based on TAG-like syntactic structures with open slots, combined with conditions which determine when the syntactic structure can be used properly. A speech generation module (SGM) converts the output of the LGM into speech using either phrase-concatenation or diphone-synthesis.

1. INTRODUCTION

This paper describes the underlying algorithms of *D2S*, which embodies a generic architecture for the generation of coherent spoken monologues from concepts. D2S is to a large extent domain and language independent. It was originally developed for the *Dial Your Disc* (*DYD*) system, which generates English monologues about Mozart compositions derived from information found in a database (van Deemter et al. 1994, Odijk 1995, van Deemter and Odijk 1997). More recently it formed the core of the *GoalGetter* system, which produces Dutch soccer reports on the basis of Tele-Text pages (Theune et al. 1997a), and of the *VODIS* system, which outputs English and German route descriptions on the basis of a 'trip table' (Krahmer et al. 1998). For the sake of illustration, we take the GoalGetter system as our running example.[1]

Essentially, D2S consists of two big modules: a *language generation module* (called *LGM*) which converts a typed data-structure into a so-called *enriched text*, i.e., a text annotated with information about the placement of accents and boundaries, and a *speech generation module* (called *SGM*) which turns the enriched text into a speech signal. One of the interesting features of LGM is that it does not follow the relatively common pipeline architecture for language generation in which text and sentence planning precede linguistic realization. In fact, LGM contains hardly any *global* text planning. The only assumption is that a text consists of one or more paragraphs, each paragraph in turn consisting of one or more sentences. The 'lack' of global text planning is compensated for at sentence level: sentences are generated from so-called *syntactic templates*, which combine TAG-like syntactic structures with conditions which determine when the syntactic tree can be used properly. The generation strategy employed by LGM may be characterized as '*survival of the fittest sentence(s)*': each generated sentence leads to an update of the context model, and the conditions on the templates determine which syntactic structure(s) are suitable given the new state of the context model. If there are several currently suitable templates —and this is typically the case— LGM makes a non-deterministic choice among them. The advantage of this method is that a given input will lead to a different output text each time it is fed into the system. This variability is assumed to be more 'pleasant' for the hearer. The output of LGM is fed into the speech generation module (SGM). Ideally, a method for generating speech in D2S should be *flexible*: it should be able to deal with the variability and the prosodic annotations in the LGM-output. Moreover, it should yield *high quality* speech output. Since no existing method fully satisfies both requirements, we adapted two speech generation methods to suit our needs: *phrase concatenation* and *speech synthesis*. Phrase concatenation scores high on naturalness, but less on flexibility; for speech synthesis the opposite holds. The remainder of this paper mimics the bipartite structure of D2S: in section 2 we take a closer look at LGM, while in section 3 we go into SGM.

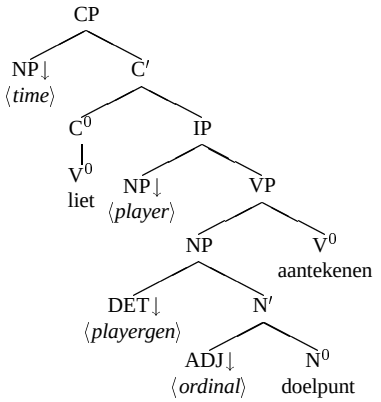
2. LANGUAGE GENERATION

I. Syntactic Templates One of the main characteristics of LGM is the usage of *syntactic templates*. Figure 1 contains an example from GoalGetter. Formally, a syntactic template $\sigma = \langle S, E, C, T \rangle$, 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 (typically NPs and PPs) 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. The trees have the form of an initial tree in *Tree Adjoining Grammar* (TAG, Joshi 1987): all interior nodes of the tree are labeled by non-terminal symbols, while the nodes on the frontier are labeled either by terminal or non-terminal symbols, where the non-terminal nodes on the frontier are the gaps which are open for substitution and are marked by a $\downarrow$. Like in the Lexicalized version of TAG (LTAG, Schabes 1990), we demand that the frontier contains at least one terminal, lexicalized node. Unlike in LTAG, we do not have the stronger requirement that *at most* one terminal, lexicalized node is allowed. Still, many templates of the GoalGetter system contain only one (group of) lexical node(s), which may be thought of as

[1] On line demonstration: http://iris19.ipo.tue.nl:9000/english.html.

Template Sent16b

$S =$



$E =$ *time* ← ExpressTime (*currentgoal.time*)
player ← ExpressObject (*currentgoal.player, P, nom*)
playergen ← ExpressObject (*currentgoal.player, P, gen*)
ordinal ← ExpressOrdinal (*ordinalnumber*)

$C =$ Known (*currentmatch.result*) ∧
currentgoal = First (*notknown,goallist*) ∧
GoalsScored (*currentgoal.player*) > 1 ∧
currentgoal.type $\neq$ *owngoal*

$T =$ goalscoring

Figure 1: Sample template from the GoalGetter system.

the head of the construction, while the gaps are to be filled by its arguments. In the other templates, more material is lexicalized for reasons of efficiency. Typical about the GoalGetter templates is the high number of collocations: groups of words with a frozen meaning, such as *doelpunt laten aantekenen* (have a goal noted) in Template Sent16b. 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 some Express function (discussed below), which generates the set of possible slot fillers for the given gap. The third ingredient is *C: the Boolean 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. Examples of the latter kind are the conditions that Template Sent16b cannot be used to describe an own goal and that the player of the current goal must have scored more than once. Conditions of the former type state things like 'X should not be conveyed to the user before Y is conveyed'. Thus, Template Sent16b can only be used if the result of the match currently being described has been conveyed to the user (i.e., is known) and the current goal is the first one which has not been conveyed (is not known). These conditions 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. One advantage of this strategy is that it carries over immediately to dialogues, in which there can be no pre-planning. Finally, each template σ contains a set of *topics* T. As we shall see below, the LGM algorithm uses the topic information to group sentences together into coherent chunks of text.

II. The Generation Algorithm Let us now consider an example to illustrate the working of the LGM generation algorithm, shown in Figure 2. Its input is formed by the set of topics *(all_topics)* and the set of templates *(all_templates)*. The GoalGetter system uses three topics, namely 'goalscoring', 'cards' and 'general', and approximately 30 templates, each associated with one or more topics. After initialization, the algorithm randomly picks a topic from *all_topics*, say 'goalscoring'. A set is constructed of all templates which are

Generate*(all_topics, all_templates)*

relevant_topics, untried_topics ← *all_topics*
templates ← {}
sentence_uttered, topic_successful, topic_finished ← **false**
current_topic, chosen_template ← nil
while *untried_topics* $\neq$ {}
do *current_topic* ← PickAny*(untried_topics)* ∧
  topic_successful ← **false**
  while *topic_finished* = **false**
  do *templates* ← { *t* ∈ *all_templates* | *current_topic* ∈ Topic *(t)* ∧
                Cond*(t)* = **true** }
    while *(sentence_uttered* = **false**) ∧ (*templates* $\neq$ nil)
    do *chosen_template* ← PickAny*(templates)* ∧
      sentence_uttered ← ApplyTemplate*(chosen_template)* ∧
      templates ← *(templates* \ *chosen_template)*
    endwhile
    if *sentence_uttered* = **false**
    then *topic_finished* ← **true** ∧
        if *topic_successful* = **true**
        then *relevant_topics* ← (*relevant_topics* \ *current_topic*) ∧
            untried_topics ← *relevant_topics* ∧
            StartNewParagraph
        else *untried_topics* ← (*untried_topics* \ *current_topic*)
        endif
    else *topic_successful* ← **true**
    endif
    sentence_uttered ← **false**
  endwhile
endwhile

Figure 2: The basic generation algorithm of LGM.

associated with this topic and whose conditions are true given the current knowledge state. In the case of 'goalscoring', the set turns out to be empty: there are no 'goalscoring' templates which are applicable when no information about the match has been conveyed. This means that the topic has finished without being successful, and the algorithm starts a new generation round with another topic, choosing from the two topics which have not yet been tried, 'general' and 'cards'. Assume that now 'general' is selected. For this topic, the set of appropriate templates is not empty: there are several 'general' templates for sentences introducing the soccer match, which can be used when the knowledge state is still empty. One of these is randomly selected, and an attempt is made to generate a sentence from it using the function ApplyTemplate (to be discussed below). If the attempt fails, other templates are tried until a sentence has been uttered. If it succeeds, the current topic is regarded as successful (a sentence is generated) but unfinished (other sentences may follow) and the algorithm tries to apply a new template within the current topic, taking into account that the knowledge state changed when the previous sentence was generated. In this way, sentences are generated until there are no usable templates left within the topic. Then the topic is finished and removed from the set of relevant topics. A paragraph break is realized, and the generation algorithm starts a new round with a new topic.

Assume that after the 'general' topic is finished, the algorithm once again tries the topic 'goalscoring', which has been included again in the set of untried topics. Because general information about the match, including the result, has been conveyed in the previous paragraph, this time there are several appropriate templates. Assume that Template Sent16b is one of them since the first goal is scored by the player Kluivert, who has scored

```
ApplyTemplate(template)
all_trees, allowed_trees ← {}
chosen_tree, final_tree, sentence ← nil
all_trees ← FillSlots(template)
for each member t_i of all_trees do
    if Violate_BT(t_i) = false ∧
       Wellformed(UpdateDiscourseModel(t_i)) = true
    then trees ← trees ∪ t_i
    endif
if allowed_trees = nil
then return false
else chosen_tree ← PickAny(allowed_trees) ∧
    UpdateContext(chosen_tree) ∧
    final_tree ← AddProsody(chosen_tree) ∧
    sentence ← Fringe(final_tree) ∧
    Pronounce(sentence) ∧
    return true
endif
```

```
ExpressObject(r, P, case)
PN, PR, RE ← nil
trees ← {}
PN ← MakeProperName (r)
PR ← MakePronoun (r, case)
RE ← MakeReferringExpression (r, P)
trees ← PN ∪ PR ∪ RE
return trees
```

Figure 3: Some functions used in the generation process.

more than once during the match, and that it is this template which happens to be chosen. Then ApplyTemplate, shown in Figure 3, 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 Express functions associated with the slots. Figure 3 shows the function ExpressObject, which generates a set of NP-trees and is used to generate fillers for the ⟨*player*⟩ and ⟨*playergen*⟩ slots in Template Sent16b. It has as input the entity to be expressed, a list of 'preferred attributes' (used in MakeReferringExpression, see Krahmer and Theune 1998, *these proceedings* for more details) and the case of the NP to be generated. The functions called by ExpressObject return phrases referring to the relevant entity using a proper name, a pronoun, and a definite description respectively. The outputs (if any) of these functions are gathered and returned. For the ⟨*player*⟩ slot in Sent16b, ExpressObject will return the set containing the proper name *Kluivert* and the pronoun *hij* ('he'). No definite description is returned, since at this point not even including the values for all the attributes in list P (e.g., team, position, nationality, etc.) is sufficient to single out Kluivert from the other players. For ⟨*playergen*⟩, which requires an expression in genitive case, ExpressObject returns trees for *Kluiverts* and *zijn* ('his') (Dutch does not allow definite description in genitive case). For the ⟨*ordinal*⟩ and ⟨*time*⟩ slots, other Express functions are used, which we assume return trees for *eerste* ('first') and *na vijf minuten* ('after five minutes') respectively. The set returned by FillSlots then contains trees for the following sentences:

{ *Na vijf minuten liet Kluivert Kluiverts eerste doelpunt aantekenen,*
Na vijf minuten liet hij Kluiverts eerste doelpunt aantekenen,
Na vijf minuten liet Kluivert zijn eerste doelpunt aantekenen,
Na vijf minuten liet hij zijn eerste doelpunt aantekenen. }

(English: *After five minutes* {*Kluivert / he*} *had* {*Kluivert's / his*} *first goal noted.*) For each tree in this set, it is checked (*i*) whether it obeys Chomsky's Binding Theory and (*ii*) whether it can be used to update the Discourse Model, which is a record containing all the objects which have been introduced so-far and the anaphoric relations (if any) among them. The first test filters out the first two sentences because the proper name *Kluiverts* which occupies the ⟨*playergen*⟩ slot is not free in this position, thus violating Principle C of the Binding Theory. The second test is failed by the fourth tree, since the Discourse Model contains no antecedent for the pronoun *hij* in the ⟨*player*⟩ slot. The remaining tree is selected and the context state, including the Discourse Model and the knowledge state, is updated with the information from this tree. Then its prosodic properties are computed by AddProsody (see Theune et al. 1997b for details): first it assigns pitch accents to the words VIJF, MINUTEN, KLUIVERT, EERSTE and DOELPUNT, each expressing information which is new to the discourse. The phrase *zijn* (referring back to Kluivert) is deaccented due to givenness, the phrase *liet aantekenen* is not accented due to structural constraints. Subsequently, intonational boundaries are added to the resulting tree: a minor boundary (/) is added after the time expression, and a major boundary (///) at the end of the sentence, giving the following result:

(1) Na VIJF MINUTEN / liet KLUIVERT zijn EERSTE DOELPUNT aantekenen ///

Finally, the prosodically annotated sentence (the fringe of the final tree) is sent to the SGM to be pronounced.

3. SPEECH GENERATION

I. Phrase Concatenation As said, two methods are available to convert the prosodically annotated text into a speech signal: phrase concatenation and speech synthesis. Concatenation of prerecorded words and phrases is very common in information systems and forms a good combination with template-based language generation. However, the following aspects have to be kept in mind. First, most information systems have fixed sentences with only one or two slots. LGM generates much more flexible output because it in general has many (syntactic) templates with many slots, which requires smaller building blocks of the concatenation method (the words and phrases). This leads to a smaller inventory size. Moreover, simply making one recording of each relevant word and phrase does not result in natural output, since variation in accentuation and phrasing (e.g., deaccentuation of given items) cannot be dealt with. We solve this by using several prosodic versions for slot fillers. Depending on the prosodic annotations in LGM's output, the correct phrases are selected. There are six different versions available depending on accentuation [±accent] and phrasing [phrase-medial, phrase-final, sentence-final]. The appropriate pitch patterns needed were elicited from the speaker by recording the words in contexts that closely match the actual situation. Finally, in order to get a good output quality it is important that the recordings are sufficiently well controlled. If this is not the case, differences in loudness, speaking rate and pitch patterns occur, which are often disguised by inserting longer pauses between the building blocks, thus hindering the fluency of the output speech. Given that our building blocks are smaller than usual (single words make up 75% of the GoalGetter database), extra care is taken. Speaking rate and intonation are controlled by recording all building blocks in context. Moreover, after recording, some effort was put into refining the building blocks, by removing

splutters and spikes in the speech signal and altering the volume at some points. The advantage of this approach is that the output sounds very natural and comprehensible. The disadvantage is that the database construction is very time-consuming. Moreover, the vocabulary should be of medium size and should remain stable, so that a minimum of recording sessions is required. More extensive information about this method can be found in Klabbers (1997).

II. Speech Synthesis Synthetic speech is generated via our diphone synthesizer SPENGI, using a method called *phase synthesis* (Gigi and Vogten 1997), which combines the advantages of PSOLA and mixed-excitation LPC to achieve an output quality that is quite high. In a subjective evaluation under telephone conditions, it was judged favorably on several aspects, including general quality, intelligibility and voice pleasantness (Rietveld et al. 1997). It uses a special strategy to determine the relative contribution of periodic and noise components of the synthesized signal, based on a very accurate pitch synchronous analysis of the amplitude and phase of the harmonic components of the input signal and a sophisticated determination of the '*factor of noisiness*'. In the case of D2S, SPENGI is used as a phonetics-to-speech system. The output of LGM is transformed into a phonetic transcription by consulting a lookup table. The accent and phrase boundary markers are copied into this transcription and serve as direct input to the intonation rules. With SPENGI, the relevant diphones are concatenated and rules are applied to control duration and intonation. The intonation, in terms of pitch movements is assigned per intonation phrase, according to the theory of 't Hart, Collier and Cohen (1990). For each pitch pattern several realizations are possible, chosen at random in order to achieve a certain amount of variation in the final speech output. Example (1) contains two intonation phrases, the first, *na* VIJF MINUTEN, contains two accented words before a minor phrase boundary. In one realization of such a configuration, the first accented word receives an accent-lending rise ('1') and the second a (pointed) hat pattern ('1&A'). The phrase boundary is marked by a continuation rise ('2') and a subsequent 400-ms pause. The second intonation phrase (*liet* KLUIVERT *zijn* EERSTE DOELPUNT *aantekenen* ///) contains three words which need to be accented before a major boundary. This can be configured by a succession of two accent-lending rises and an accent-lending fall ('A'). A final fall ('a') and a 800-ms pause signal the end of the sentence. This yields the intonation contour depicted in figure 4. The resulting

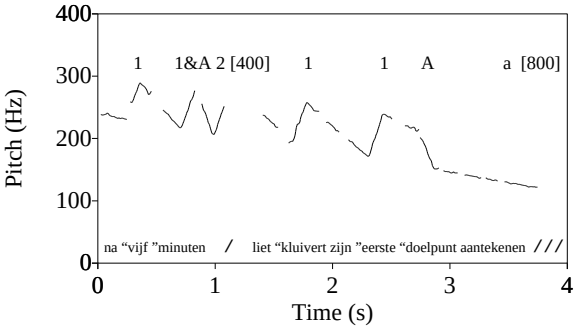


Figure 4: Example intonation contour for sentence (1).

speech thus sounds quite natural where the intonation is concerned, as LGM provides a reliable indication of prosody. However, on other levels, such as the segmental level, the quality of diphone synthesis can still be improved. One common problem with diphone synthesis is the occurrence of audible discontinuities at diphone boundaries. The investigation of this problem is presented in Klabbers and Veldhuis (1998, *these proceedings*).

Acknowledgments Klabbers and Theune did their work within the framework of the Priority Programme Language and Speech Technology (TST), sponsored by NWO (The Netherlands Organization for Scientific Research). Krahmer was partly supported by the Language Engineering/Telematics Applications Program, project LE-1 2277 (VODIS).

4. REFERENCES

1. van Deemter, K., Landsbergen, J., Leermakers, R., and Odijk, J., "Generation of Spoken Monologues by Means of Templates," *Proceedings of TWLT 8*, Enschede, University of Twente, 87-96, 1994.
2. van Deemter, K., and Odijk, J., "Context Modelling and the Generation of Spoken Discourse," *Speech Communication 21(1/2)*,101-121, 1997.
3. Gigi, E. and Vogten, L., "A Mixed-excitation Vocoder Based on Exact Analysis of Harmonic Components," *IPO Annual Progress Report, Eindhoven* Volume 32, 105-110, 1997.
4. 't Hart, H., Collier, R., and Cohen, J., *A Perceptual Study of Intonation*, Cambridge University Press, Cambridge, 1990.
5. Joshi, A., "An Introduction to Tree Adjoining Grammars," *Mathematics of Language*, A. Manastar-Ramer (ed.), John Benjamins, Amsterdam, 1987.
6. Klabbers, E., "Speech Output Generation in GoalGetter," *Papers from the Seventh CLIN Meeting, Eindhoven*, 57-68, 1997.
7. Klabbers, E. and Veldhuis, R., "On the Reduction of Concatenation Artefacts in Diphone Synthesis," 1998, *these proceedings*.
8. Krahmer, E., Landsbergen, J., Odijk, J. *A Guided Tour Through LGM. How to Generate Spoken Route Descriptions*. IPO Report 1182, 1998.
9. Krahmer, E. and Theune, M., "Context Sensitive Generation of Referring Expressions," 1998, *these proceedings*.
10. Odijk, J., "Generation of Coherent Monologues," *Papers from the Fifth CLIN Meeting*, Enschede, University of Twente, 123-131, 1995.
11. Rietveld, T., et al., "Evaluation of Speech Synthesis Systems for Dutch in Telecommunication Applications in GSM and PSTN Networks," *EUROSPEECH'97*, Rhodes, Greece, 577-580, 1997.
12. Schabes, Y., *Mathematical and Computational Aspects of Lexicalized Grammars*, Ph.D. thesis, University of Pennsylvania, 1990.
13. Theune, M., Klabbers, E., Odijk, J. and de Pijper, J.R., *From Data to Speech: A Generic Approach*, IPO Manuscript 1202, 1997a, http://www.tue.nl/ipo/people/theune/manuscript.ps.Z.
14. Theune, M., Klabbers, E., Odijk, J., and de Pijper, J.R., "Computing Prosodic Properties in a Data-to-Speech System," *Workshop on Concept-to-Speech Generation Systems*, (E)ACL, Madrid, 39-46, 1997b.