Plan-based *vs.* template-based NLG: a false opposition?

Kees van Deemter[†], Emiel Krahmer[‡] & Mariët Theune[‡]
ITRI[†], Brighton and IPO[‡], Eindhoven

January 11, 2001

Abstract

This paper uses the algorithm employed in a number of recent template-based NLG systems to challenge the wide-spread assumption that template-based methods are inherently less well-founded than plan-based methods.

Keywords: NLG paradigms, D2S, templates for NLG, plan-based NLG

1 Introduction: a caricature

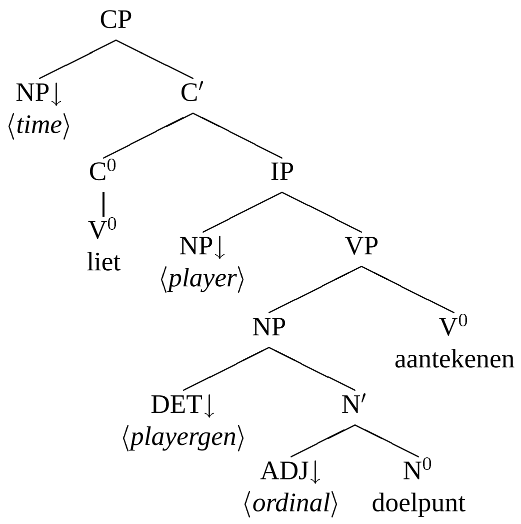
Natural Language Generation (NLG) systems are sometimes partitioned into two mutually exclusive, jointly exhaustive classes [1,11,13]: **(A)** theoretically well-founded systems, which embody generic linguistic insights and are, as a result, easily maintainable. Sometimes, the term '(full-blown) NLG' has been narrowed down to denote this class only; and **(B)** application-dependent systems which lack a proper theoretical foundation. These systems may be relatively easy to deploy but they are difficult to maintain. The following equalities tend to be stated or suggested: **A** = plan-based NLG systems; **B** = template-based NLG systems.[1] We will argue against these two identifications. We start out by sketching a class of systems that are template-based, while at the same time being as theoretically well-founded as any existing plan-based system.

[1]This identification may have originated when the term 'template' approach was used ('for lack of a better name') to refer to 'programs that simply manipulate character strings, in a way that uses little, if any, linguistic knowledge' [11]. In the present paper, 'template-based' will be taken to mean "making extensive use of a mapping between semantic structures and representations of linguistic surface structure that contain gaps".

2 NLG with syntactically structured templates

In this section a brief description of a data-to-speech method called D2S is given. D2S is the foundation of a number of language generation applications for different domains (Mozart compositions, soccer reports, route descriptions, train information) and languages (Dutch, English, German). As a running example we use the GoalGetter system which generates Dutch soccer reports. (See http://iris19.ipo.-tue.nl:9000/english.html for an on-line demonstration.) D2S consists of two modules: (1) a language generation module (LGM) which converts a typed data-structure into *enriched text*, i.e., a text annotated with information about the placement of accents and boundaries, and (2) a *speech generation module* (SGM) which turns the enriched text into a speech signal. Here we focus on the LGM and in particular on its use of syntactically structured templates, an example of which is given in Figure 1.

$S =$



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

$C =$ Known (*currentmatch.result*) $\wedge$ *currentgoal* $=$ First (*notknown,goallist*) $\wedge$
currentgoal.type $\neq$ *owngoal*

$T =$ goalscoring

Figure 1: Sample syntactic template from the GoalGetter system.
liet een doelpunt aantekenen (‘let a goal be noted’)
means *put a goal on the scoresheet*

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 condition on the applicability of σ and T is a set of topics. We discuss the four components in more detail, beginning with the *syntactic tree, S*. All interior nodes of the tree are labeled by non-terminal symbols, while the nodes on the frontier are labeled by terminal or non-terminal symbols, where the non-terminal nodes are the gaps which are open for substitution and are marked by a $\downarrow$. Many templates contain only one (group of) lexical node(s), which may be thought of as the head of the construction, while the gaps are to be filled by its arguments. An example is the template in Figure 1, whose head is the collocation *een doelpunt laten aantekenen* (put a goal on the scoresheet).

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, which generates the set of possible slot fillers. This process is handled by the function ApplyTemplate, shown on the left in Figure 2. 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 Express functions associated with the slots. Figure 2 (right) shows an example Express function, namely ExpressObject, which generates a set of NP-trees and is used to generate fillers for the $\langle$*player*$\rangle$ and $\langle$*playergen*$\rangle$ slots in the template of Figure 1. The first of the two, for example, leads to the generation of NPs such as 'Atteveld' (proper name), 'the defender Atteveld', 'Vitesse player Atteveld', 'Vitesse's Atteveld', etc., depending on the context in the which the NP is generated.[2] Once all the gaps in the template are filled, the set *all_trees* results. For each tree in this set, it is checked (*i*) whether it obeys Chomsky's Binding Theory and (*ii*) whether it is compatible with the Context Model, which is a record containing all the objects introduced so far and the anaphoric relations among them. From the resulting set of *allowed_trees*, one is selected randomly and returned to the main generation algorithm.

[2]For a more sophisticated version of the way in which nominals are generated in context, see [8].

```
$\boxed{\text{ApplyTemplate}(template)}$
allowed_trees ← {}
chosen_tree ← nil
all_trees ← FillSlots(template)
for each member $t_i$ of all_trees do
    if ViolateBindingTheory($t_i$) = false ∧
      Wellformed(UpdateContext($t_i$)) = true
    then allowed_trees ← allowed_trees ∪$t_i$
if allowed_trees = nil
then return false
else chosen_tree ← PickAny(allowed_trees) ∧
    return final_tree
```

```
$\boxed{\text{ExpressObject}(r, P, case)}$
PN, PR, RE ← nil
trees ← {}
PN ← MakeProperName (r)
PR ← MakePronoun (r, case)
RE ← MakeReferringExp (r, P)
trees ← PN ∪ PR ∪ RE
return trees
```

Figure 2: Functions ApplyTemplate (left) and ExpressObject (right).

be410pie The third ingredient of a syntactic template σ is C: *the Boolean condition*. A template σ is applicable if and only if its associated condition is true. Several kinds of conditions can be distinguished including, most notably perhaps, conditions on the knowledge state. An example is the condition saying 'X should not be conveyed to the user before Y is conveyed', which implies that the template can only be used if the result of the current match 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). Finally, each template σ contains a set of *topics* T, which the LGM algorithm uses to group sentences together into coherent chunks of text.

3 The caricature exposed

Taking our inspiration from D2S [4,6,7], we will argue that the caricature from the introduction is precisely this: a caricature. For starters, D2S' application across domains and languages (cf. Section 2), has revealed a remarkable genericity. Important parts of the system (e.g., the basic generation algorithm and such functions as ApplyTemplate and ExpressObject) turned out to be independent of application domain (Classical Music / Soccer games) and output language (English / Dutch). This is, of course, not true for the templates themselves, many of which have to be written anew for each new domain as well as for each language. Based on these experiences, however, it seems fair to say that D2S is as generic and maintainable as any plan-based system, which will have to adapt its grammar, for example, whenever a new application or a new output language comes along.

But is D2S also well-founded? This depends on what it means for an NLG system to be well-founded. If it means that every decision made by the system (e.g., expressing a proposition in one or in two sentences, using passive or active voice; lexical choice [2]) should be based on sound linguistic principles, then no NLG system we are aware of qualifies as being even remotely well-founded: the gap between raw data and text is bridged in ways that are often arbitrary. Many NLG systems use linguistic principles, but typically such sophistication is reserved for a few aspects of the generated text. D2S is no exception, as may be seen from Section 2. For example, D2S uses well-established rules for constraining the use of anaphors (see e.g., ViolateBindingTheory and Wellformed in ApplyTemplate), and a new variant of Dale and Reiter's algorithm [3] for the generation of referring expressions that takes contextual salience into account (MakeReferringExp in ExpressObject) [7]. Other choices (most notably, perhaps, the choice of a pool of templates from which the generator can pick a candidate) are made on less principled grounds. The main limiting factor for the deployment of linguistic rules in D2S is *not* that the method does not allow it, but simply that not enough good linguistic rules are known. In sum: D2S, though it is a template-based system, is as well-founded as any plan-based system.

In fact, we believe that the terminology itself is misleading. Few if any NLG systems are *plan-based* in the full sense in which this term is used in artificial intelligence: in NLG, there usually is no place for logical inference (e.g., avoiding a certain wording because of some explicitly represented common-sense knowledge) or even backtracking. (Whether or not this limitation reflects a property of *human* speaking and writing is a different matter.) *If*, as has become usual in NLG, the notion of planning is stretched to cover, say, Moore and Paris-style NLG [10], *then* the system described in Section 2 could be described as implementing a distributive, reactive ('situated') planner. (See also the Conclusion of this paper.)

It is worth noting that D2S rather resembles an approach to NLG that is sometimes omitted in discussions about practical versus applied systems, namely Tree Adjoining Grammar (TAG) (e.g., [5,9,14]). The trees in D2S are similar to the 'initial trees' of TAG. Joshi [5:234] points out that "The initial (…) trees are not constrained in any manner other than as indicated above. The idea, however, is that [they] will be *minimal* in some sense." The minimalism constraint is usually interpreted as: the tree should not contain more than the lexical head plus its arguments. The comparison with TAG-based NLG suggests that it is not the choice of a template-based approach that makes an NLG system theoretically unwell-founded, but the choice for nonminimal templates / elementary trees in these systems (or the use of canned text in plan-based systems, for that matter). Of course, non-minimal templates/ elementary trees are essential for the treatment of any phenomena where compositionality breaks down, such as idioms, special collocations, etc. (cf. the treatment of collocations in [14]). But, generally speaking, the larger the templates/elementary trees, the less systematic the treatment, the less insight it gives into the compositional structure of language, and the larger the number of templates/elementary trees needed. Unlike the earliest D2S-based NLG systems (e.g., [4]), GoalGetter can be argued to use templates that are minimal except where there is a good reason to make them larger [6].

4 Conclusion

We have argued against the caricature presented in Section 1, according to which template-based NLG systems are always linguistically less interesting than so-called plan-based systems. We have illustrated our claim by sketching a template-based generation system that is theoretically as well-founded as any plan-based system, as well as being practically useful (deployable, maintainable, etc.). Of course, there are genuine and interesting differences between the two paradigms. For example, template-based systems do not conform to the well-known pipeline model for NLG [12], which starts from the assumption that the entire semantic content of a discourse is known at the beginning of the pipeline – after which this content is processed by the next module and so on until the document comes out at the end of the pipeline. This could point the way to an understanding of what makes plan-based systems more suitable for one type of application and template-based systems for another. We hypothesize that their pipeline structure (in a different *jargon*, their top-down orientation) makes plan-based systems unsuitable for the modeling of 'spontaneous' types of speaking/writing, in which the speaker/writer does not always have a plan for the complete discourse before the first word is uttered. The incremental setup of D2S's language generation module, which lets templates 'fire' until a topic (e.g. the topic of goalscoring) is exhausted, without a preconceived plan about the order in which this must happen [4,6], illustrates how such a spontaneous manner of speaking/writing can be modeled using a template-based method.

References

1. S. Busemann and H. Horacek. A Flexible Shallow Approach to Text Generation. In *Proceedings of the 9th International Workshop on Natural Language Generation (IWNLG'98), Niagara-on-the-Lake*, 238-247, 1998.

2. L. Cahill. Lexicalisation in applied NLG systems. ITRI report ITRI-99-04, obtainable

via http://www.itri.brighton.ac.uk/projects/rags/, 1998.

3. R. Dale and E. Reiter. Computational Interpretations of the Gricean Maxims in the Generation of Referring Expressions. *Cognitive Science* 18, 233-263, 1995.

4. K. van Deemter and J. Odijk. Context Modelling and the Generation of Spoken Discourse. *Speech Communication* 21(1/2), 101-121, 1997.

5. A.K. Joshi. The Relevance of Tree Adjoining Grammar to Generation. In G. Kempen (ed.), *Natural Language Generation*, Martinus Nijhoff, Dordrecht, The Netherlands, 233-252, 1987.

6. E. Klabbers, E. Krahmer, and M. Theune. A Generic Algorithm for Generating Spoken Monologues. In *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP'98), Sydney*, 2759-2762, 1998.

7. E. Krahmer and M. Theune. Context Sensitive Generation of Descriptions. In *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP'98), Sydney*, 1151-1154, 1998.

8. E. Krahmer and M. Theune. Efficient Generation of Descriptions in Context. To appear in *Proceedings of* ESSLLI *workshop Generation of Nominals*, Utrecht, August 1999.

9. D. McDonald and J. Pustejovsky. TAG's as a Grammatical Formalism for Generation. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics (ACL'85)*, Chicago, 94-103, 1985.

10. J.D. Moore and C.L. Paris. Planning Text for Advisory Dialogues: Capturing Intentional and Rhetorical Information. *Computational Linguistics* 19(4), 652-694, 1994.

11. E. Reiter. NLG vs. Templates. In *Proceedings of the 5th European Workshop on Natural Language Generation (EWNLG'95), Leiden*, 95-106, 1995.

12. E. Reiter. Has a Consensus NLG Architecture appeared and is it Psychologically Plausible? *Proceedings of the 7th International Workshop on Natural Language Generation*, pp. 163-170, 1994.

13. E. Reiter and R. Dale. Building Applied Natural Language Generation Systems. *Natural Language Engineering* 3(1), 57-87, 1997.

14. M. Stone and C. Doran. Paying Heed to Collocations. In *Proceedings of the 8th International Workshop on Natural Language Generation (IWNLG'96), Herstmonceux*, 91-100, 1996.