Real *vs.* template-based natural language generation: a false opposition?

Kees van Deemter*
University of Brighton

Emiel Krahmer†
Tilburg University

Mariët Theune‡
University of Twente

This paper challenges the received wisdom that template-based approaches to the generation of language are necessarily inferior to other approaches as regards their maintainability, linguistic well-foundedness and quality of output. Some recent NLG *systems that call themselves 'template-based' will illustrate our claims.*

1 Introduction

Natural Language Generation (NLG) systems are sometimes partitioned into application-dependent systems which lack a proper theoretical foundation, on the one hand, and theoretically well-founded systems which embody generic linguistic insights, on the other. Template-based systems are often regarded as automatically falling into the first category. We will argue against this view. First, we describe the received view of both template-based and 'standard' NLG systems (section 2). Then we describe a class of recent template-based systems (section 3) that will serve as a basis for a comparison between template-based and other NLG systems with respect to their potential for performing NLG tasks (section 4). We ask what the real difference between template-based and other systems is, and argue that the distinction between the two is becoming increasingly blurred (section 5). Finally, we discuss the implications of engineering shortcuts (Mellish 2000) and corpus-based methods (section 6).

2 Templates vs. real NLG: the received view

Before we can argue against the distinction between template-based and 'real' NLG systems, we should first sketch how these two classes are commonly understood. It is surprisingly difficult to give a precise characterization of the difference between them (and we will later argue against the usefulness of such a characterisation), but the idea is the following. **Template-based** systems are natural language generating systems that map their non-linguistic input *directly* (i.e., without intermediate representations) to the linguistic surface structure (cf., Reiter and Dale (1997), p.83-84). Crucially, this linguistic structure may contain gaps; well-formed output results when the gaps are filled or, more precisely, when all the gaps have been replaced by linguistic structures that do not

* Information Technology Research Institute, University of Brighton, United Kingdom, E-mail: Kees.van.Deemter@itri.brighton.ac.uk.
† Communication and Cognition / Computational Linguistics, Faculty of Arts, Tilburg University, Tilburg, The Netherlands, E-mail: E.J.Krahmer@uvt.nl.
‡ Parlevink Language Engineering Group, Computer Science, University of Twente, The Netherlands,E-mail: Theune@cs.utwente.nl.

contain gaps. (Canned text is the borderline case of a template without gaps.) Adapting an example from Reiter and Dale (1997), a simple template-based system might start out from a semantic representation saying that the 306 train leaves Aberdeen at 10:00 am:

$$\text{Departure}(\text{train}_{306},\text{location}_{abdn},\text{time}_{1000}),$$

and associate it directly with a template such as

[train] *is leaving* [town] *now*,

where the gaps represented by [train] and [town] are filled by looking up the relevant information in a table. Note that this template will only be used when the time referred to is close to the intended time of speaking; other templates must be used for generating departure announcements relating to the past or future. 'Real' or, as we shall say, **standard** NLG systems, by contrast, use a less direct mapping between input and surface form (Reiter (1995), Reiter and Dale (1997)). Such systems could start from the same input semantic representation, subjecting it to a number of consecutive transformations until a surface structure results. Various NLG submodules would operate on it (determining, for instance, that 10:00 am is essentially the intended time of speaking), jointly transforming the representation into an intermediate representation like

$$Leave_{present}(train_{demonstrative}, Aberdeen, now),$$

where lexical items and style of reference have been determined while linguistic morphology is still absent. This intermediate representation may in turn be transformed into a proper sentence, for example: *This train is leaving Aberdeen now*. Details vary; in particular, many systems will contain more intermediate representations.

Template-based and standard NLG systems are said to be 'Turing equivalent' (Reiter and Dale (1997)), i.e., each of them can generate all recursively enumerable languages. However, template-based systems have been claimed to be inferior with respect to maintainability, output quality and variation, and well-foundedness. Reiter and Dale (1997) state that template-based systems are more difficult to maintain and update (p.61), and that they produce poorer and less varied output (p.60, 84) than standard NLG systems. Busemann and Horacek (1998), p.238, go even further by suggesting that template-based systems do not embody generic linguistic insights. Consistent with this view, template-based systems are sometimes overlooked. In fact, the only current textbook on NLG (Reiter and Dale (2000)) does not pay any attention to template-based generation, except for a passing mention of the ECRAN system (Geldof and Van de Velde (1997)). Another example is a recent overview of NLG systems in the RAGS project (Cahill et al. (1999)). The selection criteria employed by the authors were that the systems had to be fully implemented, complete (i.e., generating text from non-textual input), and accepting non-hand-crafted input; although these criteria appear to favour template-based systems, none of the 19 systems investigated were template-based. In what follows, we claim that the two types of systems have more in common than is generally thought, and that it is counterproductive to treat them as distant cousins instead of close siblings. In fact, we will argue that there is no crisp distinction between the two.

3 Template-based NLG systems in practice

In recent years, a number of new template-based systems have seen the light, including TG/2 (Busemann and Horacek (1998)), D2S (van Deemter and Odijk (1997), Theune et al. (2001)), EXEMPLARS (White and Caldwell (1998)), YAG (McRoy et al. (2003)), and

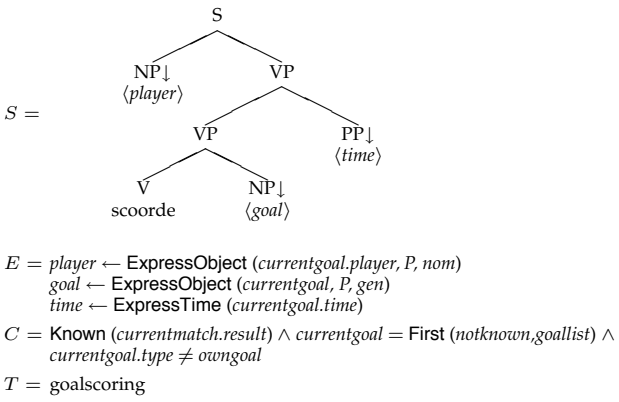

Figure 1
Sample syntactic template from the GOALGETTER system.

XTRAGEN (Stenzhorn (2002)). Each of these systems represents a substantial research effort, achieving generative capabilities beyond what is usually expected from template-based systems, yet they call themselves template-based, and they clearly fall within the characterization of template-based systems offered above.

In this paper we shall draw on our own experiences with a data-to-speech method called D2S. D2S has been used as the foundation of a number of language generating systems, including GOALGETTER, a system that generates soccer reports in Dutch.[1] D2S consists of two modules: (1) a *language generation module* (LGM) and (2) a *speech generation module* (SGM) which turns the generated text into a speech signal. Here we focus on the LGM and in particular on its use of syntactically structured templates to convert a typed data-structure into a natural language text (annotated with prosodic information). Data structures in GOALGETTER are simple representations describing lists of facts, such as:

goal-event	
TEAM	Ajax
PLAYER	Kluivert
MINUTE	38
GOAL-TYPE	penalty

Besides goal events, there are several other types of events, such as players receiving yellow or red cards. Figure 1 shows a simple template, which the LGM might use to express the above fact as, for instance, *Kluivert scored a penalty in the 38th minute*.

Formally, a syntactic template $\sigma = \langle S, E, C, T \rangle$, where S is a syntax 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 of a template in more detail, starting with the *syntax tree*, S. All S's interior nodes are labeled by non-terminal symbols, while the nodes on the frontier are labeled by terminal or non-terminal symbols: the non-terminal nodes ('gaps') are open for substitution and they are marked by a $\downarrow$. 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 (ExpressTime, ExpressObject, etc.), which generates a set of expressions that can be used to fill the slot. Figure 2 (right) shows an example Express function, namely ExpressObject, which

1 See `http://www.cs.utwente.nl/~theune/GG/GG_index.html` for some example reports.

```
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 chosen_tree
```

```
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).

generates a set of NP-trees and is used to generate fillers for the ⟨*player*⟩ and ⟨*goal*⟩ slots in the template of Figure 1. The first, for example, leads to the generation of NPs such as 'Kluivert' (proper name), 'the forward Kluivert', 'Ajax player Kluivert', 'Ajax' Kluivert', 'the striker', 'he' etc., depending on the context in which the NP is generated.

Figure 2 (left) shows the function ApplyTemplate, which handles the choice between all possible combinations of slot fillers. ApplyTemplate first calls FillSlots to obtain the set of all possible trees (*all_trees*) that can be generated from the template, using all possible combinations of slot fillers generated by the Express functions associated with the slots. For each tree in this set, it is checked (*i*) whether it does not violate a version of the Chomskyan 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 (using the function PickAny) and returned to the main generation algorithm. The random choice option was chosen to maximize the *variety* of sentences produced by the system.

The mechanisms described so far take care of sentence planning and language realization. Text planning is performed by components C and T. C is a *Boolean condition*. A template σ is only applicable if its associated condition is true. An example is the condition from Figure 1 saying that the template can only be used if the result of the current match has been conveyed to the user (i.e., is known) and the current goal is the first one which has not been conveyed (i.e., is not known). To cater for aspects of text planning that allow a less knowledge-intensive approach, GOALGETTER associates every template with a set of *topics* T, which the LGM algorithm uses to group sentences together into coherent chunks of text. For example, any template associated with the topic of goal scoring can 'fire' throughout the opening paragraph of the report.

4 Template-based NLG: deep or shallow?

How do template-based systems measure up against the criteria mentioned in section 2? When dealing with this question, we will be interested as much in what could be done *in principle* as in what has been achieved in practice. After some preliminary remarks, we will focus on the criterion of linguistic well-foundedness.

It is far from obvious that template-based systems should always score low on **maintainability**. Several template-based systems such as TG/2, EXEMPLARS, and XTRAGEN have been reused for generation in different languages or in different domains (cf. Kittredge et al. (1994)). In the case of D2S, the basic generation algorithm and such functions as ApplyTemplate and ExpressObject have been used for different application domains (music/soccer games/route descriptions/public transport) and different lan-

guages (English/Dutch/German); D2S has been used for the generation of both monologues and dialogue contributions (Van Deemter and Odijk (1997) and Theune et al. (2001)). When applying a template-based system to a new domain or language, many of the templates will have to be written anew (much like new grammar fragments need to be developed for standard NLG systems), but the underlying generation mechanisms generally require little or no modification.

As for the **output quality** and **variability** of the output, if template-based systems have the same generative power as standard NLG systems (Reiter and Dale (1997)), there cannot be a difference between the types of output that they are able to generate in principle. The fact that templates can be specified by hand gives template-based systems an advantage in cases where good linguistic rules are not (yet) available or for constructions which have unpredictable meanings or highly specific conditions of use (Stone and DeCarlo (**?**)). Some template-based systems have variability as one of their central design specifications: current D2S-based systems rely mainly on random choice to achieve variation, but more context-sensitive variations (e.g., varying the output depending on user characteristics) can also be achieved through the use of parametrized templates (XTRAGEN) or template specialization hierarchies (EXEMPLARS).

The most crucial question, in our view, is whether a template-based NLG system can be linguistically **wellfounded** (or 'deep' in terms of Busemann and Horacek (1998)), in the sense that the choices inherent in its mapping from input to output are based on sound linguistic principles. To judge the well-foundedness of template-based systems, let us look at the different types of decisions that an NLG system needs to make, as distinguished by Cahill et al. (1999) and Reiter and Dale (2000).

Content Determination During *Content Determination*, it is decided what information is to be conveyed. Since content determination precedes language generation proper it is clear that, in *principle*, template-based systems can treat it in the exact same ways as standard NLG systems. In *practice*, template-based systems tend to take their departure from 'flat data' (e.g., database records), whereas standard systems often use richer input, where some decisions concerning the linguistic structure of the output (e.g., decisions about quantificational or rhetorical structure) have already been made. To the extent that this is the case, the 'generation gap' to be bridged by template-based systems is actually wider than the one to be bridged by standard NLG systems.

Referring Expressions As for the generation of referring expressions, template-based systems vary widely: the simplest of them (e.g., MS WORD-based systems for mail merge) can fill their gaps with only a limited number of phrases, but more sophisticated systems (called 'hybrid' systems in Reiter (1995)) have long existed, which effectively use standard NLG to fill their gaps. Recent systems have moved further in this direction. D2S, for example, uses well-established rules for constraining the use of anaphors (see e.g., the Chomskyan ViolateBindingTheory and Wellformed in ApplyTemplate), and a new variant of Dale and Reiter's (1995) algorithm for the generation of referring expressions that takes contextual salience into account (MakeReferringExp in ExpressObject) (Krahmer and Theune (2002)). A similar range of approaches can be found among NLG systems that are not template-based; in fact, several systems from the RAGS inventory do not really address referring expression generation at all (Cahill et al. (1999)).

Aggregation Aggregation is an NLG task where differences between the two types of systems may be expected. After all, every template contains a 'fixed' part, and surely, this part cannot be re-combined with other parts? The reality is slightly more complex. The GOALGETTER system, for instance, uses the following approach: in order to gen-

erate a subject-aggregated sentence of the form 'A and B got a red card', a separate template is called of the form 'X got a red card' [syntactic structure omitted], subject to conditions requiring that the gap X be filled with an appropriate conjoined noun phrase, referring to the set $\{A, B\}$. Other approaches are possible. For example, the system could first generate 'A got a red card' and 'B got a red card', then aggregate these two structures (whose syntactic and semantic structure is known) into the desired conjunctive structure (van Deemter and Odijk) (1997). Whether a system is able to perform operations of this kind does not depend on whether the system is template-based, but on whether it possesses the required syntactic and semantic information.

Lexicalisation The same point is relevant for Lexicalisation. Let us suppose (perhaps rather charitably, Cahill et al. (1999)) that a variety of near-synonymous verbs are present in the lexicon of the NLG system, e.g. 'give', 'offer', 'donate', 'entrust', 'present to', etc. How would a standard NLG system choose between them? Typically, the system does not have a clue, because our understanding of the differences between these verbs is too imperfect. (The input to the system might prejudge such decisions by pairing each of these verbs with different input relations, but that would be cheating.) As with the previous tasks, it is not clear that standard NLG systems are in a better position to perform them than template-based ones: the latter could use templates that vary in the choice of words, and stipulate that they are applicable under slightly different conditions (cf., the use of specialization hierarchies in EXEMPLARS). The condition C for 'X kicked the ball in the net', for example, (as opposed to 'X scored', or 'X nudged the ball in') might require that the ball did not touch the ground after departing the previous player.

Linguistic Realisation It is here that the most obvious differences between standard and template-based approaches appear to exist. Many template-based approaches lack a general mechanism for gender, number and person agreement, for example. Systems in the D2S tradition avoid errors by letting functions like `ExpressObject` use handmade rules, but this approach becomes cumbersome when coverage increases; generalisations are likely to be missed and portability is reduced, for example if different templates are used for 'John walks' and 'John and Mary walk'. One should not, however, let one's judgement depend on accidental properties of one or two systems: nothing keeps the designer of a template-based system from adding morphological rules, witness systems like YAG (McRoy et al. 2003) and XTRAGEN (Stenzhorn (2002)). The YAG system, for example, allows subject and verb of a template to be underspecified for number and person, while using Attribute Grammar rules to complete the specification: returning to the example above, the number attribute of 'John and Mary' is inferred to be PLURAL (unlike e.g., 'John and I'); a *subject-verb agreement* rule makes the further inference that the verb must be realised as 'walk', rather than 'walks'.

5 Templates: an updated view

A new generation of systems that call themselves template-based have blurred the line between template-based and standard NLG. This is not only because some systems *combine* standard NLG with templates and canned text (Piwek (2003)), but also because modern template-based systems tend to use *syntactically structured* templates, and allow the gaps in them to be filled *recursively* (i.e., by filling a gap, a new gap may result). Some 'template-based' systems, finally, use *grammars* to aid linguistic realisation. These developments call into question the very definition of 'template based' (section 2), since the systems that call themselves template-based have come to express their nonlinguistic input with varying degrees of directness.

'Template-based' systems vary in terms of linguistic coverage, the amount of syntactic knowledge used, and the number of steps involved in filling the templates, among other things. Here, we highlight one particular dimension, namely the size of (the fixed part of) the templates. A comparison with Tree Adjoining Grammar- (TAG) based approaches to NLG may be useful (Joshi (1987), see also Becker (2002)). Joshi (1987:234) points out that "The initial (...) trees are not constrained in any other manner than [...]. The idea, however, is that [they] will be *minimal* in some sense." Minimality is usually interpreted as saying that a tree should not contain more than the lexical head plus its arguments. Initial trees may be likened to templates. Non-minimal templates / elementary trees are essential for the treatment of idioms and special collocations. Generally speaking, however, 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. Again, the history of D2S is instructive: the earliest D2S-based NLG system (DYD, van Deemter and Odijk (1997)), used long templates, but the majority of the templates in GOALGETTER are minimal in the sense explicated above (Theune et al. (2001)).

6 Discussion: Shortcuts and Statistics in NLG

Let us compare our views with those of Mellish (2000). Mellish points out that NLG systems often use *shortcuts*, whereby one or more modules are trivialised, either by bypassing them (and the representations that they create) or by letting their operations be dictated by what the other modules expect (e.g., Lexical Choice may be trivialised by using a one-to-one mapping between semantic relations/predicates and lexical items). Mellish argues that shortcuts have a legitimate role in practical NLG when linguistic rules are missing, provided the existence of the shortcuts is acknowledged: even though they lead to diminished generality and maintainability, the unavailability of 'deep' rules means that there is no alternative (yet). For instance, there is little added value in using abstract representations from which either a passive or an active sentence can be generated if we are unable to state a general rule that governs the choice, in which case one can be forgiven for explicitly specifying which sentences should be active and which ones passive, avoiding a pretense of linguistic sophistication. It is shortcuts of this kind that a template-based system is well placed to make, of course. But, crucially, template-based systems do not *have* to use shortcuts any more than standard NLG systems: where linguistic rules are available, both types of systems can use them, as we have seen.

Another response to the absence of linguistic rules is the use of statistical information derived from corpora, as is increasingly more common in realisation, but also for instance in aggregation (e.g., Walker et al. (2002)). The point we want to make here, however, is that 'template-based' systems may profit from such corpus-based approaches just as much as 'standard' NLG systems. The approach of Langkilde and Knight (1998), for example, where corpus-derived n-grams are used for selecting the best ones from among a set of candidates produced by overgeneration, can also be applied to template-based systems (witness the mixed template/stochastic system of Galley et al. (2001)).

We have argued that systems that call themselves template-based can, in principle, perform all NLG tasks in a linguistically wellfounded way, and that more and more actually implemented systems of this kind deviate dramatically from the stereotypical systems that are often associated with the term *template*. Conversely, most standard NLG systems perform many NLG tasks in a less than wellfounded fashion (e.g., relying heavily on shortcuts, and non-transparent ones at that). We doubt that there is still any important difference between the two classes of systems, since the variation within each of them is as great as that between them.

Acknowledgments

This is a remote descendant of a paper presented at the workshop *May I Speak Freely?* (Becker & Busemann (1999)). We thank three reviewers for comments.

References

Becker, Tilman. 2002. Practical, Template–Based Natural Language Generation with TAG. In *Proc. of TAG+6*, Italy.

Becker, Tilman and Stephan Busemann, editors. 1999. *"May I Speak Freely?". Between Templates and Free Choice in Natural Language Generation. KI-99 Workshop*. DFKI, Germany.

Busemann, Stephan and Helmut Horacek. 1998. A flexible shallow approach to text generation. In *Proc. 9th International Workshop on Natural Language Generation*, pages 238–247, Canada.

Cahill, Lynn, Christy Doran, Roger Evans, Chris Mellish, Daniel Paiva, Mike Reape, and Donia Scott. 1999. In search of a reference architecture for NLG systems. In *Proc. 7th European Workshop on Natural Language Generation*, pages 77–85, France.

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

Galley, Michel, Eric Fosler-Lussier, and Alexandros Potamianos. 2001. Hybrid natural language generation for spoken dialogue systems. In *Proc. 7th European Conference on Speech Communication and Technology*, Denmark.

Geldof, Sabine and Walter van de Velde. 1997. An architecture for template based (hyper)text generation. In *Proc. 6th European Workshop on Natural Language Generation*, pages 28–37, Germany.

Joshi, Aravind. 1987. The relevance of Tree Adjoining Grammar to generation. In Gerard Kempen, editor, *Natural Language Generation*, pages 233–252, Martinus Nijhoff, Leiden, The Netherlands.

Kittredge, Richard, Eli Goldberg, Myunghee Kim, and Alain Polguère. 1994. Sublanguage engineering in the FOG system. In *Fourth Conference on Applied Natural Language Processing*, pages 215–216, Germany.

Krahmer, Emiel and Mariët Theune. 2002. Efficient context-sensitive generation of descriptions in context. In Kees van Deemter and Rodger Kibble, editors, *Information Sharing*, pages 223 – 264, CSLI Publications, CSLI, Stanford, Ca.

Langkilde, Irene and Kevin Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proc. of the ACL*, pages 704–710, Montreal, Canada.

McRoy, Susan W., Songsak Channarukul, and Syed S. Ali. 2003. An augmented template-based approach to text realization. *Natural Language Engineering*, 9(4):381–420.

Mellish, Chris. 2000. Understanding shortcuts in NLG systems. In *Proc.Ìmpacts in natural language generation: NLG between technology and applications'*, pages 43–50, Dagstuhl, Germany.

Piwek, Paul. 2003. A flexible pragmatics-driven language generator for animated agents. In *Proc. of EACL03 (Research Notes)*, pages 151–154, Budapest, Hungary.

Reiter, Ehud. 1995. NLG vs. Templates. In *Proc. 5th European Workshop on Natural Language Generation*, pages 95–105, Leiden, The Netherlands.

Reiter, Ehud and Robert Dale. 1997. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87.

Reiter, Ehud and Robert Dale. 2000. *Building Natural Language Generation Systems*. Cambridge University Press.

Stenzhorn, Holger. 2002. A natural language generation system using XML- and java-technologies. In *Proc. 2nd Workshop on NLP and XML*, Taipei, Taiwan.

Theune, Mariët, Esther Klabbers, Jan-Roelof de Pijper, Emiel Krahmer, and Jan Odijk. 2001. From data to speech: a general approach. *Natural Language Engineering*, 7(1):47–86.

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

Walker, Marilyn, Owen Rambow, and Monica Rogati. 2002. Training a sentence planner for spoken dialogue using boosting. *Computer Speech and Language*, 16:409–433.

White, Michael and Ted Caldwell. 1998. EXEMPLARS: A practical, extensible framework for dynamic text generation. In *Proc. 9th International Workshop on Natural Language Generation*, pages 266–275, Canada.